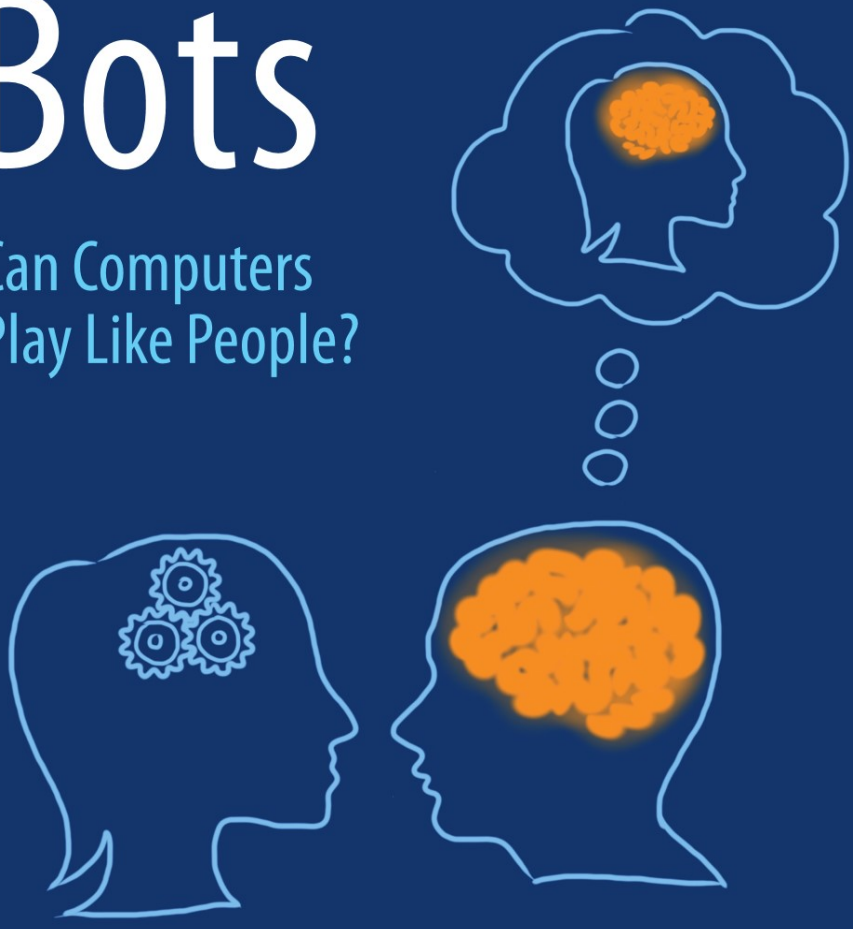
Philip Hingston *Editor*

Believable Bots

Can Computers Play Like People?



Springer

Believable Bots

Philip Hingston
Editor

Believable Bots

Can Computers Play Like People?

Springer

Editor
Philip Hingston
School of Computer and Security Science
Edith Cowan University
Mount Lawley, WA
Australia

ISBN 978-3-642-32322-5 ISBN 978-3-642-32323-2 (eBook)
DOI 10.1007/978-3-642-32323-2
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2012949379

ACM Computing Classification (1998): I.2, K.8, J.7

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The idea that we humans would one day share the Earth with a rival intelligence is as old as science fiction. That day is speeding towards us. Our rivals (or will they be our companions?) will not come from another galaxy, but out of our own strivings and imaginings. The bots are coming; chatbots, robots, gamebots.

Some of the early arrivers already build our cars, and soon may drive them for us. They reside in our mobile phones and try to answer our questions, and might one day arrange our diaries and help refill our fridges. We play games with them for fun. Ray Kurzweil [1] claims that we are merging with them and that we will together become a new, enhanced species.

Will we welcome them, when they come? Will bots have human friends? Will we grant them rights?

Future human-bot relations may depend on whether or not we see them as being like ourselves. And that is what the chapters of this book are about—what does it take for a bot to be *believable*—by which we mean, to seem like one of us? Can humanness be simulated? Can it be faked?

In the first chapter of this book, Weiss and Tscheligi present a series of case studies on humanoid robots, investigating the role of sociality in human-bot relations. In the following chapters, Bailey and her co-authors describe their psychosocial behaviour model and its implementation for computer game bots, showing how they can create bots that interact with a player in engaging ways. In *Actor Bots*, Arinbjarnar and Kudenko also consider computer game bots, viewing them as virtual actors in a virtual theatre, using belief networks to enable bots to behave believably within the game narrative. Morie et al. also focus on bots interacting with humans in virtual worlds which they call *embodied conversational agents*.

The next few chapters are about bots that masquerade as human players, in a competition for computer game bots which is modelled on the famous Turing Test. In 1950, Alan Turing [2] proposed a test in which a computer program converses with a reasonable person, and tries to convince that person that they are, in fact, conversing with a woman. Turing's assertion was that if a computer could play this game as well as a man could, then this would be evidence that the computer was intelligent. This test has been an enduring challenge and a controversy for

generations of computer scientists and philosophers. Since 1990, the test has been realised as an annual competition for chatbots—the Loebner Prize [3]. Taking inspiration from the Turing Test and the Loebner Prize, this book's editor organises and runs the annual BotPrize competition, a Turing Test for computer game bots [4]. In the BotPrize competition, humans play a computer game with an assortment of human and bot opponents, and try to tell which are human and which are not.

Entrants to the BotPrize have refined existing methods and developed new ones to try to win the prize. In Chap. 5, Schrum, Karpov and Miikkulainen describe the workings of their bot UT^2, which is based on *neuroevolution*, a powerful machine learning method combining the principles of Darwinian evolution with artificial brain models (neural networks). The next chapter by Karpov, Schrum and Miikkulainen explains how their bot imitates human players to solve complex navigation problems in the game. In A *Machine Consciousness Approach to the Design of Human-like Bots*, Arrabales et al. introduce their CERA-CRANIUM cognitive architecture, inspired by cognitive theories of consciousness. CERA-CRANIUM was at the heart of the bot they used to win the award for the most human-like bot at the 2010 BotPrize competition. The following chapter by Arrabales, Ledezma and Sanchis uses their *ConsScale* framework to measure and analyse the cognitive levels of a number of state-of-the-art computer game bots.

The BotPrize chapters are about bots that play a First-Person Shooter—a genre in which "interaction" between players (human or bot) takes the form of virtual combat. The final few chapters look at bots in games with different interaction modes. Jacek Mańdziuk and Przemysław Szałaj consider Realtime Strategy games, and propose a distinct personality system to operate alongside traditional reasoning and tactics. Togelius et al. in their chapter, take a step back to ask how best to judge how believable a bot is. For example, is the judgement better made by an external observer or by an active participant interacting with the bot? This question is examined using a Turing Test based on bots for Super Mario Bros., a well-loved and venerable platform game. Kemmerling et al. tackle Diplomacy, a game in which planning and negotiation are key skills. Finally, Muñoz, Gutierrez and Sanchis apply imitation learning to create a human-like bot for a car racing game.

We hope the stories in this book will start the reader thinking … Must robots and other bots be like us to be accepted as a part of our cultures and lives? Or would it be better if they were obviously and unmistakably different from us? If we want them to be like us, is it enough that they are believable—that they seem like us—or must they really *be* like us in their inner hearts, possessing consciousness, emotions, intellect, craving social interaction and meaning? What kinds of technologies would it take to implement these things in a computer program, and how could we tell whether we had implemented the real thing, or merely a facsimile of it? This book does not provide the answers—but perhaps it is a place to start.

Perth, Western Australia, March 2012 Philip Hingston

Acknowledgements The editor would like to thank colleagues and students for many conversations about believable bots, especially all those who have taken part in the BotPrize competitions. I am very grateful also to 2K Australia, which has been a steadfast financial and moral supporter of the BotPrize since its inception, especially Studio General Manager Anthony Lawrence.

References

1. Kurzweil, R: The Singularity Is Near: When Humans Transcend Biology. Penguin (Non-Classics), New York (2006)
2. Turing, A: Computing machinery and intelligence. Mind **59**(236), 433–460 (1950)
3. The Loebner Prize website, http://www.loebner.net/Prizef/loebner-prize.html
4. Hingston, P: A Turing Test for computer game bots. IEEE Trans. Comput. Intell. AI Games. **1** (2009)

Contents

Chapter 1
Rethinking the Human–Agent Relationship: Which Social Cues Do Interactive Agents Really Need to Have?

Astrid Weiss and Manfred Tscheligi

Abstract This chapter discusses the potential meaning of the term social in relation to human–agent interaction. Based on the sociological theory of object-centred sociality, four aspects of sociality, namely forms of grouping, attachment, reciprocity, and reflexivity are presented and transferred to the field of human–humanoid interaction studies. Six case studies with three different types of humanoid robots are presented, in which the participants had to answer a questionnaire involving several items on these four aspects. The case studies are followed by a section on lessons learned for human–agent interaction. In this section, a "social agent matrix" for categorizing human–agent interaction in terms of their main sociality aspect is introduced. A reflection on this matrix and the future (social) human–agent relationship closes this chapter.

1.1 Introduction

Several studies in the research fields of Human–Computer Interaction (HCI) and Human–Robot Interaction (HRI) indicate that people tend to respond differently towards autonomous interactive systems than they do towards "normal computing systems" [8]. In the 1990s, the Media Equation Theory already revealed that people treat media and computing systems in a more social manner, like real people and places [31]. However, not only the responses differ, also the expectations vary and tend into a more social direction, the more anthropomorphized the system design is [26]. For instance, when an inexperienced user has to interact with a robot for

A. Weiss (✉) · M. Tscheligi
HCI & Usability Unit, ICT & S Center,
Sigmund-Haffner Gasse 18, 5020 Salzburg, Austria
e-mail: astrid.weiss@sbg.ac.at

M. Tscheligi
e-mail: tscheligi@cure.at

P. Hingston (ed.), *Believable Bots*, DOI: 10.1007/978-3-642-32323-2_1,

the first time, the first impression of the robot is paramount to successfully initiate and maintain the interaction [24]. Thus, it is important that the robot's appearance matches with its task to increase its believability. Exploratory studies in the research field of HRI indicate that people have very clear assumptions that anthropomorphic robots should be able to perform social tasks and follow social norms and conventions [26]. These assumptions about the relation between social cues and anthropomorphic design for interactive agents can also be found on the side of developers and engineers: The Wakamaru robot, developed by Mitsubishi Heavy Industries, for instance was designed in a human-like shape on purpose, as it should (1) live with family members, (2) speak spontaneously in accordance with family member's requirements, and (3) play its own role in a family (http://www.wakamura.net).

But what do we actually mean when we are talking about social cues in the human–agent relationship? If we have a look at the WordNet[1] entry for the term "social" we find a wide variety of meanings, such as "social" relating to human society and its members, "social" in terms of living together or enjoying life in communities and organized groups and "social" as relating to or belonging to the characteristic of high society. However, the term social can also relate to non-human fields, such as the tendency to live together in groups or colonies of the same kind–ants can be considered to be social insects. All these meanings indicate that the term "social" has a very broad meaning in everyday language, but there is also a lack of definition for the term social in human–agent interaction. In HCI and HRI we can find several research topics which are related to "social interaction", such as social software, social computing, CMC (Computer-Mediated Communication), CSCW (Computer-Supported Collaborative Work), the above mentioned Media Equation Theory, and research on social presence and social play.

In traditional psychology "social" is mainly understood as interpersonal interaction. If we take this definition as the starting point, the question arises, if we can consider human–agent interaction as interpersonal. An experiment by Heider and Simmel in 1944 demonstrated for the first time that animated objects can be perceived as social if they move at specific speeds and in specific structures [17]; thus a perception of agency and interpersonality in the interaction with animated agents can be assumed.

In the following, we will present an overview of related literature on the topic of social human–agent interaction and subsequently go into detail regarding how the concepts of believability and sociality interrelate. We will present the concept of object-centred sociality [25] as theoretical baseline to derive four general aspects for social interaction with agents, namely forms of grouping, attachment, reciprocity, and reflexivity. Based on these four aspects, a questionnaire was developed, which was used in six user studies with humanoid robots that will be described subsequently. The results and lessons learned of theses studies will lead to a "social agent matrix"

[1] WordNet is an online lexical reference system, developed at Princeton University. Its design is inspired by current psycholinguistic theories of human lexical memory. English nouns, verbs, adjectives and adverbs are organized into synonym sets, each representing one underlying lexical concept (http://wordnet.princeton.edu/).

which allows the categorization of (autonomous) interactive agents in terms of their sociality. This chapter ends with a reflection on the social agent matrix and the future (social) human–agent relationship.

1.2 Believability, Sociality, and Their Interrelations

Research on believability of media content and computer agents has got a long history. Believability has often been the object of research, especially in the field of communication science, like for instance the comparison of radio, TV, and newspaper content, as e.g. Gaziano et al. [15] as well as recent research including online media as e.g. Abdulla et al. [2].

Hovland et al. showed in their study that believability consists of the two main principles *trustworthiness* and *expertise* [18]. Based on this research, Fogg and Tseng [13] investigated to what extent believability matters in Human Computer Interaction. Hereby, they suggested a set of basic terms for assessing computer believability. Bartneck [4] adapted Fogg's and Tseng's concept of believability [13] to his model of convincingness. He could show that convincingness and trustworthiness highly correlate.

One of the most famous experiments regarding the believability and persuasiveness of computing systems was conducted by Weizenbaum who employed a virtual agent called ELIZA [45]. This agent was able to simulate a psychotherapist and to keep the conversation going by passively asking leading questions. Throughout the study the participants did not notice that they were actually speaking to a computer program. At the end of the experiment, it was disclosed that participants thought that they were talking to a human and that the conversational partner appreciated their problems. A higher degree of sociality perception can hardly be achieved by a computing system.

A similar study was conducted by Sundar and Nass in which people favoured to interact with the computer over the human interaction partner [35]. Recent studies confronted their participants with questions regarding trustworthiness and believability of screen characters and fully embodied robots (see e.g. [4, 32]). In order to assess robots in terms of believability, scales were either adapted as done by Shinozawa et al. [33] or newly developed as by Bartneck et al. [5]. Similar scales were often used to assess the believability of an information source (TV, newspapers, web pages, etc.) or an individual agent.

Shinozawa et al. could show that the 3D model of a robotic agent was rated higher in terms of source believability by the means of McCroskey's believability scale [28] than a 2D on-screen agent. Kidd also used a believability scale [6], which was originally developed for media assessment (such as the McCroskey's scale), to rate a robot's believability [23]. Hereby he found out that women tended to rate believability higher than men. Additionally, people trusted robots that were physically present to a greater extent, which is similar to the findings of Shinozawa et al.

The physical appearance of robots plays also a huge role in social HRI. Powers and Kiesler found out that certain physical attributes of the robot could change the human's mental model of the robot. Moreover, they could show that people's intentions are linked to the traits of the mental model [3]. For instance, certain physical characteristics, such as the robot's voice and physiognomy, created the impressions of a sociable robot, which in turn predicted the intention to accept the advice of the robot and changed people's perception of the robot's human likeness, knowledge and sociability.

Furthermore, it makes a difference if a robot recommends something or not. Imai and Narumi conducted a user study with a robot that gave recommendations [19]. The participants in the experiment inclined to want what the robot recommended, e.g. this cup of tea is delicious and the other is not. The aspect of recommendation in human–agent interaction is tightly interwoven with trustworthiness and believability [13].

Another approach researched by Desai et al. is to design an interface where the user can adjust the level of trust in a robot so it can decide more autonomously [11]. For instance, if the user controls a robot with a joystick and the robot notes many errors made by the user, the robot suggests the user to switch to a higher level of trust. In such a level the robot could make more decisions without asking the human operator.

However, in how far can a believable computing system, a robot or any kind of anthropomorphic agent be considered to be social? As the research field of robotics and Artificial Intelligence heads towards a direction where engineers develop robots following anthropomorphic images, there seems to be an area in which social interaction between humans becomes comparable to social interaction between a human and a machine. Some researchers even go beyond anthropomorphic images and work on artificial behaviour traits by developing cognitive systems (see for instance Chap. 8), which already passed the false belief test (Leonardo, [34]) and the mirror test (Nico, [1]). According to technology assessment research it is hoped (and feared) that humanoid robots will in future act like humans and be an integral part of society [42]. Such cognitive systems have a model of the self and a model of the others. By continuously updating and relating these models to each other, cognitive systems can "socially" interact. We are convinced that these circumstances require subsequent research on social acceptance in human–robot interaction in specific and human–agent relations in general.

1.3 The Concept of Sociality in Human–Agent Interaction

Reviewing the state-of-the-art literature on social HRI research (a good overview can be found in [14, 44]) and looking on the data gathered during the case studies presented later in this chapter (see Sect. 1.4), it becomes obvious that people believe in anthropomorphic agents to be social actors. However, what could be the basis to assess the degree of sociality of interactive computing systems? One approach can be found in Chap. 8, entitled ConsScale FPS: Cognitive Integration for Improved Believability in Computer Game Bots. For ConsScale, social refers

to the Theory of Mind (ToM) cognitive function, i.e. being able to attribute mental (intentional) states to oneself and to other selves. In other words, artificial agents which are capable of attributing intentional states, could be perceived as social.

But, what behavioural embodiments of cognitive processes are perceived as social by humans? Studies in the area of sociology of knowledge point into the direction that humans tend to express similar behaviours towards artefacts and objects as towards humans, such as talking to a robot or expressing emotions. Knorr-Cetina [25] speaks of a "post-social" world in this context, in which non-human entities enter the social domain. According to her, these "post-social" interactions with objects include four aspects that explain their sociality towards humans: forms of grouping, attachment, reciprocity, and reflexivity.

1.3.1 Forms of Grouping

To form a group is a core element of human social behaviour. The aspect "forms of grouping" describes the fact that humans define their own identity by sharing common characteristics with others and by distinguishing themselves from other groups. Knorr-Cetina could show that humans ascribe personal characteristics also to objects with which they often closely interact and somehow form a group with these objects [25]. The arising question is: will humans show similar behaviours like that when cooperating with an anthropomorphic agent?

1.3.2 Attachment

Attachment as a psychological concept explains the bond a human infant develops to its caregiver [7]. However, Knorr-Cetina could show that humans also develop an emotional bond to objects (e.g. "my favourite book") [25] . In general, the idea of emotional attachment towards technology already found its way into HCI and HRI research [21, 22, 36]. For human–agent interaction, attachment could be understood as an affection-tie that one person forms between him/herself and an agent. One main aspect of emotional attachment is that it endures over time. Thus, Norman explains emotional attachment as the sum of cumulated emotional episodes of users' experiences with a computing system in various contextual areas [30]. These experiences are categorized into three dimensions: a visceral level (first impression), a behavioural level (usage of the device), and a reflective level (interpretation of the device).

1.3.3 Reciprocity

In accordance to Gouldner, reciprocity can be seen as a pattern of social exchange, meaning mutual exchange of performance and counter-performance and as a general

moral principle of give-and-take in a relationship [16]. In HRI research, the aspect of reciprocity became relevant with the increase of social robot design [20]. Humans can quickly respond socially to robotic agents (see e.g. [31]), but the question arising is, if humans experience "robotic feedback" as adequate reciprocal behaviour. Thus, in a human–agent relationship reciprocity could be understood as the perception of give-and-take in a human–agent interaction scenario.

1.3.4 Reflexivity

Reflexivity describes the fact that an object or behaviour and its description cannot be separated one from the other, rather they have a mirror-like relationship. Reflexivity can be a property of behaviour, settings and talk, which make the ongoing construction of social reality necessary [27]. In human–agent interaction reflexivity could be understood as the sense-making of a turn taking interaction of which the success depends on the actions of an interaction partner.

Thus, our assumption is that focussing on one of these aspects in the interaction design of a human–humanoid interaction scenario, increases the perception of sociality on the user-side. In other words, from an actor-centred sociology perspective of sociality (and subsequently the degree of believability that an interactive agent displays) is not manifested in its appearance (visual cues), but in its interaction (behavioural cues).

1.4 The Case Studies

In the following, we present our insights gained in six human–humanoid interaction studies. Four of these studies are laboratory-based case studies, whereas two of them were conducted with the humanoid HRP-2 robot [37, 38] and the other two with the humanoid HOAP-3 robot [40, 43]. The other two studies were field trials, both of which were conducted with the anthropomorphically designed ACE robot [39, 41]. In all these studies we investigated the sociality in the interaction with these robots on a reflective level by means of questionnaire data and additional observational data of the two field trials.

1.4.1 Investigating Social Aspects in Human–Humanoid Interaction

To gather quantitative data on the social aspects about the perception of humanoid robots, a questionnaire consisting of 13 items which had to be rated on a 5-point Likert scale, ranging from 1 = "strongly disagree" to 5 = "strongly agree",

Table 1.1 13 statements on social aspects, which had to be rated on a scale from 1 = "strongly disagree" to 5 = "strongly agree"

Item	Social aspect
Robots will be part of our everyday work	Forms of grouping
Robots will be an important part of our society	Forms of grouping
I would like to collaborate with robots	Forms of grouping
Robots will have a similar importance as human colleagues	Forms of grouping
Robots and humans will make a good team	Forms of grouping
I would not like to imagine a world in which robots were not used	Attachment
I can imagine taking a robot into my heart	Attachment
It would feel good if a robot was near me	Attachment
I can imagine building a special relationship with robots	Attachment
The interaction with robots will be a mutual experience	Reciprocity & reflexivity
I can imagine that I will care for the wellbeing of a robot	Reciprocity & reflexivity
The relationship with robots will be based on the principle of give and take	Reciprocity & reflexivity
Humans and robots will be interdependent	Reciprocity & reflexivity

was developed. In this questionnaire the aspects reciprocity and reflexivity were combined into one concept, as the reflected sense-making of the give-and-take interaction was assessed as successful task completion. The users filled in the questionnaire after they completed the tasks together with the robot. All items are presented in Table 1.1.

Prior to and directly after the interaction with the humanoid robot the study participants were asked to fill in the Negative Attitude towards Robots Scale (NARS) to gain insights on the question, if the interaction with the robot changed their general attitude towards robots. This questionnaire is based on a psychological scale to measure the negative attitudes of humans towards robots. It was originally developed by Nomura et al. [29]. This questionnaire tries to visualize which factors prevent individuals from interacting with robots. The questionnaire consists of 14 questions which have to be rated on a 5-point Likert scale ranging from 1 = "strongly disagree" to 5 = "strongly agree". The 14 questions build three sub scales: S1 = negative attitude toward situations of interaction with robots; S2 = negative attitude toward social influence of robots; S3 = negative attitude toward emotions in interaction with robots.

1.4.2 First Study with the HRP-2

The first case study was carried out as a Wizard-of-Oz user study [9], based on a mixed-reality simulation. The simulation was based on a 3D model of the HRP-2 robot and implemented with the Crysis game engine (more details on the technical

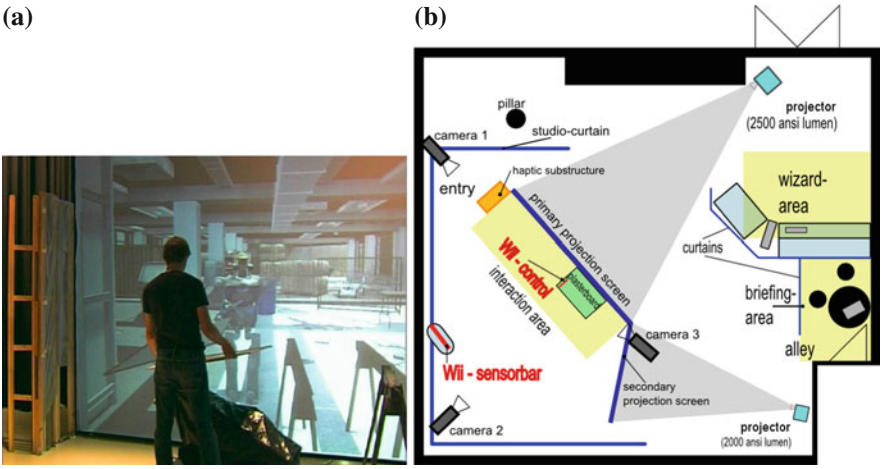


Fig. 1.1 Study setting: first HRP-2 study. **a** Study setting: participant, **b** Study setting: implementation

implementation can be found in [38]) which was controlled by a hidden human wizard during the interaction trials. The human–humanoid collaboration was based on the task of carrying and mounting an object together, whereas the object (a board) existed in the virtual as well as in the real world and built the contact point for the interaction. The research question was: "How do differently simulated feedback modalities influence the perception of the human–robot collaboration in terms of sociality?". The four experimental conditions were: Con0: interaction without feedback, Con1: interaction with visual feedback (blinking light showing that the robot understood the command), Con2: interaction with haptic feedback, and Con3: interaction with visual and haptic feedback in combination. The study was conducted with 24 participants in August 2008 at the University of Applied Sciences in Salzburg, Austria.

1.4.2.1 Study Setting

This user study was based on one task which had to be conducted together with the simulated robot via a mobile board as "input modality" (see Fig. 1.1). The task was introduced by the following scenario:

> Imagine you are working at a construction site and you get the task from your principal constructor to mount a gypsum plaster board together with a humanoid robot. You can control the robot with predefined voice commands, to carry out the following action sequences.

Task:

Lift, move, and mount a gypsum plaster board together with a humanoid robot. This task consists of the following action sequences:

1. Start the interaction by calling the robot.
2. Lift the board together with the robot.
3. Move the board together with the robot to the right spot.
4. Tilt the board forward to the column together with the robot.
5. Tell the robot to screw the board.

1.4.2.2 Findings on Social Aspects

The questionnaire data analysis revealed the mean values regarding the items on forms of grouping, attachment, and reciprocity & reflexivity, depicted in Table 1.2, which indicate that the aspect forms of grouping was perceived most intensely (summative overall factor rating: *mean*: 3.10, *SD*: 0.91). However, the statement about the importance of a humanoid robot as future working colleague was rated rather low. In terms of attachment, the participants rated the item "1 AT" the highest, indicating that they could imagine that robots will enter a special role in society in future (summative overall factor rating: *mean*: 2.21, *SD*: 0.92). The aspect reciprocity & reflexivity was even rated slightly better than attachment (summative overall factor rating: *mean*: 2.92, *SD*: 0.98), whereas the highest rated item was "1 RE", which demonstrates the importance of turn taking for simulated sociality. Moreover, an ANOVA on the overall factor rating revealed that the experimental conditions influenced the results of the aspect forms of grouping ($F(3, 20)6.26$, $p < 0.05$). A post-hoc test (LSD) showed that in Con1 (interaction with visible feedback) forms of grouping was rated significantly lower than in all other conditions–another support for the importance of turn taking translated to multimodal feedback on the side of the agent. The NARS questionnaire did not reveal any significant changes due to the interaction with the virtual HRP-2 robot in any of the three attitude scales.

1.4.3 Second Study with the HRP-2

Similar to the previously described case study on the simulation of HRP-2 (see Sect. 1.4.2), the task in this case study was to carry an object together with the robot, but the focus was to (1) investigate differences in the human–humanoid collaboration, between a tele-operated and an autonomous robot and (2) cultural differences in the perception between Western and Asian participants. The research questions were "How does the participant experience the relationship towards (1) the autonomous HRP-2 robot and (2) the tele-operated HRP-2 robot?" and "Is there a difference in

Table 1.2 Questions on social aspects: first HRP-2 study (1 = "strongly disagree" to 5 = "strongly agree"; N excluding missing answers)

No.	Question	N	Mean	SD
1 FG	Robots will be part of our everyday work	22	4.05	1.00
2 FG	Robots will be an important part of our society	22	3.50	0.96
3 FG	I would like to collaborate with robots	23	3.48	1.12
4 FG	Robots will have a similar importance as human colleagues	22	1.73	0.99
5 FG	Robots and humans will make a good team	21	3.19	1.17
1 AT	I would not like to imagine a world in which robots were not used	19	2.84	1.21
2 AT	I can imagine taking a robot into my heart	23	2.26	1.32
3 AT	It would feel good if a robot was near me	24	1.92	0.93
4 AT	I can imagine building a special relationship with robots	23	1.91	1.16
1 RE	The interaction with robots will be a mutual experience	22	3.45	1.34
2 RE	I can imagine that I will care for the wellbeing of a robot	24	2.96	1.46
3 RE	The relationship with robots will be based on the principle of give and take	23	2.26	1.39
4 RE	Humans and robots will be interdependent	23	3.13	1.33

terms of sociality perception, between participants with (1) Asian origin and (2) Western origin?". A total of 12 participants (6 Asian, 6 Western) took part in this study, which was conducted together with CNRS/AIST at Tsukuba University, Japan and the Technical University Munich, Germany in October 2009.

1.4.3.1 Study Setting

This study was based on direct human–humanoid interaction, in which the HRP-2 robot acted partly autonomously and was partly remotley controlled by an human operator located in Germany. The participants (acting as the human operator in Japan), had to lift, carry, and put down a table collaboratively with the HRP-2 robot. During lifting and putting down the table, HRP-2 was tele-operated. During carrying the table, HRP-2 was walking autonomously. The study was based on the following scenario:

> Imagine you are working at a construction site and you receive a task from your principal constructor: carrying an object from one place to another together with a humanoid robot which is partly acting autonomously and partly operated by a human expert operator.

Fig. 1.2 Study setting: second HRP-2 study **a** Action sequence 1, **b** Action sequence 2, **c** Action sequence 3

Task:

The task is to carry a table from place A to place B together with the humanoid robot HRP-2. This task is split into four action sequences:

- Action sequence 0: The robot is sent to the table by the principal instructor
- Action sequence 1: Lift the table together with HRP-2
- Action sequence 2: Walk together with HRP-2 from place A to place B
- Action sequence 3: Put down the table together with HRP-2

The interaction between the human and the robot took place in three sequences 1–3 (see Sect. 1.2). Sequence 0 does not require any interaction between the human and the robot.

1.4.3.2 Findings on Social Aspects

The questionnaire data analysis revealed the mean values regarding the items on forms of grouping, attachment, and reciprocity & reflexivity, depicted in Table 1.3, which indicate that the aspect forms of grouping was perceived most intensely, similar to the first HRP-2 study (summative overall factor rating: $mean : 3.64, SD : 0.52$). However, again the statement about the importance of a humanoid robot as a future working colleague ("4 FG") was rated rather low. In terms of attachment, the participants rated the same item as in the previous study ("1 AT") the highest, but compared to the previous study with the simulated robot, the participants rated items on attachment in general better for the embodied agent (summative overall factor rating: $mean : 3.23, SD : 0.69$). Similarly, the aspect reciprocity & reflexivity was rated better than in the first HRP-2 study (summative overall factor rating: $mean : 3.18, SD : 0.59$). However, in the second HRP-2 study the items "1 RE" and "4 RE" were rated equally high. This could be due to the fact, that the robot and the user were directly linked through the table during the interaction and that the robot was tele-operated during lifting and putting down the table, which directly demonstrated the interdependence of every single move. Significant differences in the perception of sociality due to Western or Asian origin could not be

Table 1.3 Questions on social aspects: second HRP-2 study (1 = "strongly disagree" to 5 = "strongly agree", N excluding missing answers)

No.	Question	N	Mean	SD
1 FG	Robots will be part of our everyday work	11	3.64	0.92
2 FG	Robots will be an important part of our society	12	4.25	0.62
3 FG	I would like to collaborate with robots	12	3.92	0.90
4 FG	Robots will have a similar importance as human colleagues	12	2.25	1.14
5 FG	Robots and humans will make a good team	12	4.17	0.72
1 AT	I would not like to imagine a world in which robots were not used.	12	3.50	0.80
2 AT	I can imagine taking a robot into my heart	12	2.92	1.17
3 AT	It would feel good if a robot was near me	12	3.08	0.90
4 AT	I can imagine building a special relationship with robots	12	3.42	1.00
1 RE	The interaction with robots will be a mutual experience	11	3.45	1.13
2 RE	I can imagine that I will care for the well-being of a robot	12	3.42	1.00
3 RE	The relationship with robots will be based on the principle of give and take	11	2.27	1.27
4 RE	Humans and robots will be interdependent	11	3.45	1.13

identified in this study. The analysis of the NARS questionnaire revealed a decrease in all three scales through the interaction with the HRP-2 robot. However, only the scale "Negative Attitude toward Social Influence of Robots" decreased significantly ($t(11)=2.88$, $p < 0.05$).

In particular in the comparison of the first HRP-2 study (embodied robot) and the second HRP-2 study (virtual robot) we have to consider the notion of a co-production between the embodiment of the robot and the perception of sociality in the human–agent relationship. The virtual HRP-2 robot understood voice commands immediately (as long as the command was correctly uttered by the participant) due to the fact that the robot was wizarded behind the scenes. Moreover, the virtual robot moved quicker and more smoothly than the embodied one, due to the simulation basis. These facts should have increased the perception of its sociality, whereas on the other hand the missing embodiment and immersion in the interaction may have lowered it again.

1.4.4 First Study with HOAP-3

In this user study, the participants had to conduct two "learning by demonstration" tasks with the robot. For the first task they had to teach the arm of the robot to (1)

push a box, and for the second one (2) to close a box. Twelve participants took part in this study conducted together with the Learning Algorithms and Systems Laboratory, EPFL at Lausanne, Switzerland, in August 2008. The research question of this study was the following: "How do novice users experience the collaboration with the humanoid robot HOAP-3 in terms of sociality, when the interaction is based on learning by demonstration?"

1.4.4.1 Study Setting

This user study was based on two tasks that the participants had to conduct together with the HOAP-3 robot (see Figs. 1.3 and 1.4). The tasks were introduced by the following scenario:

> Imagine you are working at an assembly line in a big fabrication plant. A new robot is introduced, which should support you in completing tasks. You can teach the robot specific motions by demonstrating them (meaning moving the robot's arm like you expect it to move it later on its own); the robot will repeat the learned motion. You can repeat this demonstration-repetition cycle as long until you are pleased with the result.

Task 1:

This task is to teach the robot to push this box from its working space into yours on its own. The task is split up into the following action sequences:

1. Show the robot the specific task card by putting it on the table in front of the robot (move it around until the robot recognizes it).
2. Demonstrate to the robot to push the box with its right arm, by putting the box very close in front of the robot and moving its arm.
3. Let the robot repeat what it learned.
4. (If necessary) repeat sequences 2 and 3 until you are pleased with the way the robot pushes the box.

- The interaction with the robot is based on speech commands. Just follow the commands of the robot and answer to them with yes or no (or any other answer proposed by the robot).
- You only need to teach the right arm of the robot by moving its elbow.

Task 2:

This task is to teach the robot to close this box on its own. The task is split up into the following action sequences:

1. Show the robot the specific task card by putting it on the table in front of the robot (move it around until the robot recognizes it).

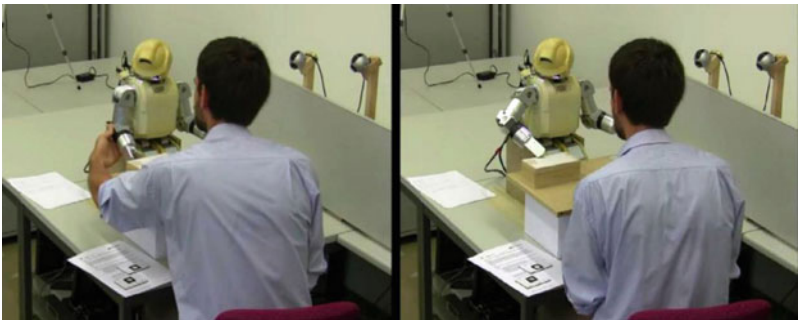


Fig. 1.3 First HOAP-3 study: study setting task 1

Fig. 1.4 First HOAP-3 study: study setting task 2

2. Demonstrate the robot to close the box, by putting the box very close in front of the robot and moving its arm.
3. Let the robot repeat what it learned.
4. (If necessary) repeat sequences 2 and 3 until you are pleased with the way the robot closes the box.

- The interaction with the robot is based on speech commands. Just follow the commands of the robot and answer to them with yes or no (or any other answer proposed by the robot).
- You only need to teach the right arm of the robot by moving its elbow.

As the pre-test of the user study showed that the tasks are experienced as different in their level of difficulty (task 1 was estimated more difficult than task 2) the order of the tasks was counterbalanced between the participants to reduce a potential learning effect.

Table 1.4 Questions on social aspects: first study with HOAP-3 (1 = "strongly disagree" to 5 = "strongly agree", N excluding missing answers)

No.	Question	N	Mean	SD
1 FG	Robots will be part of our everyday work	12	3.75	0.62
2 FG	Robots will be an important part of our society	11	3.18	1.33
3 FG	I would like to collaborate with robots	11	3.55	0.82
4 FG	Robots will have a similar importance as human colleagues	12	1.83	1.12
5 FG	Robots and humans will make a good team	12	3.67	0.78
1 AT	I would not like to imagine a world in which robots were not used	12	3.25	1.14
2 AT	I can imagine taking a robot into my heart	12	2.25	1.22
3 AT	It would feel good if a robot was near me	11	2.82	1.17
4 AT	I can imagine building a special relationship with robots	12	2.58	1.08
1 RE	The interaction with robots will be a mutual experience	10	3.10	1.10
2 RE	I can imagine that I will care for the well-being of a robot	11	3.55	0.82
3 RE	The relationship with robots will be based on the principle of give and take	12	3.08	1.51
4 RE	Humans and robots will be interdependent	11	2.82	0.75

1.4.4.2 Findings on Social Aspects

The questionnaire data analysis revealed the mean values regarding the items on forms of grouping, attachment, and reciprocity & reflexivity, depicted in Table 1.4, which indicate that the aspect forms of grouping was again perceived most intensely (summative overall factor rating: *mean*: 3.18, *SD*: 0.68). The item "5 FG" was rated second best after item "1 FG", which shows that team work was experienced even more in a learning by demonstration scenario than in a "pure" collaboration task. In terms of attachment, the participants rated the same item ("1 AT") the highest as in the two previous studies. The higher rating for item "3 AT" could be explained by the smaller size and "cuteness" of the HOAP-3 robot compared to the HRP-2 (see also the results of the second study with HOAP-3; summative overall factor rating: *mean*: 2.74, *SD*: 0.94). Regarding the aspect of reciprocity & reflexivity, the item "2 RE" was rated best, which indicates that learning by demonstration plus the "cuteness aspect" fosters a willingness for caring about the agent (summative overall factor rating: *mean*: 3.15, *SD*: 0.54). The NARS questionnaire revealed a significant decrease for the scale "Negative Attitude toward Social Influence of Robots" ($t(11) = 3.17$, $p < 0.05$), showing that the participants rated this scale significantly lower after interacting with HOAP-3.

1.4.5 Second Study with HOAP-3

This user study was also conducted with the HOAP-3 robot. However, the main difference was that the interaction with the robot was remote-controlled via a computer interface, so this scenario provided no direct contact interaction with the robot. The participants had to conduct two tasks via the computer interface: (1) move the robot through a maze and find the exit, and (2) let the robot check all antennas and detect the broken one. Twelve participants took part in this study that was conducted together with the Robotics Lab at the University Carlos III, Madrid, Spain, in September 2008. The research question of this study was: "How do novice users experience the collaboration with the humanoid robot HOAP-3 when interacting via a computer interface?"

1.4.5.1 Study Setting

This user study was based on two tasks that the participants had to conduct via a computer interface with the HOAP-3 robot (see Fig. 1.5). The first task was introduced by the following scenario:

> Your space shuttle has been hit by an asteroid and you were forced to an emergency landing. Your communication and internal ship monitoring system does not work, probably due to a damage caused by the crash. The good news is that you have the necessary material to replace the broken antenna for your communication system to send for help. As you are the only human survivor of the ship and the environment could possibly be dangerous for human beings, you decide to let this dangerous work be done by the ship's robot HOAP-3. At first you have to navigate HOAP-3 to the exit of the shuttle.

Task 1:

Help the robot to find its way through the corridor and find the door to the outside. The task is to move the robot by means of the computer interface. It is completed if you see the door through the interface and say "door found". The interaction with the robot is based on a computer interface, with which you can control the robot.

The second task was introduced by the following scenario:

> After you have accomplished the first task to get the robot HOAP-3 out of the shuttle, you now have to help your HOAP-3 to find the broken antenna. The problem is that your shuttle has several antennas of different shape and colour and you cannot distinguish the defected one from the others by sight, but HOAP-3 can. Inside the robot there is a mechanism which enables the robot to detect the malfunctioning parts.

Task 2:

Your task is to control the movements of HOAP-3 again, while it is processing and checking the different antennas. In this task, you have to help the robot to recognize the broken antenna. If HOAP-3 has recognized a malfunctioning device, it will put a square around it on the interface. The task is finished if you recognize the broken antenna through the interface and say "broken antenna recognized".

1.4.5.2 Findings on Social Aspects

The questionnaire data analysis revealed the mean values regarding the items on forms of grouping, attachment, and reciprocity & reflexivity, depicted in Table 1.5, which indicate that the aspect forms of grouping was again perceived most intensely (summative overall factor rating: *mean*: 3.43, *SD*: 0.62), closely followed by attachment (summative overall factor rating: *mean*: 3.31, *SD*: 0.52), and reciprocity & reflexivity (summative overall factor rating: *mean*: 3.15, *SD*: 0.80). The items "3 FG" and "5 FG" were rated equally high, indicating the team building aspect of the "collaborative explorer task " in this study. Similarly, as in the first HOAP-3 study, the robot was rated high in terms of attachment (see item "3 AT" and "3 AT"), which could be again due to the "cuteness aspect". In terms of reciprocity item "3 RE" was rated best. This could be due to the fact that this item was the only one which did not directly address mutuality, which was hard to perceive in an interaction scenario without direct contact interaction with the robot. Regarding the attitude towards interacting with the robot, the NARS questionnaire revealed a decrease for all three scales, but only statistically significant for the scale "Negative Attitude toward Emotions in Interaction with Robots"($t(11) = 2.25$, $p < 0.05$). The participants rated this scale significantly lower after interacting with HOAP-3.

1.4.6 First and Second Study with ACE

The ACE robot (Autonomous City Explorer Robot) is a robot with the mission to autonomously finds its way to pre-defined places, through proactive communication with passers-by.

In the first study (see Fig. 1.6), the ACE robot moved remote-controlled via the Karlsplatz, a highly frequented public place in Munich, which is situated at the end of a shopping street, with access to local transportations (metro). Although the robot was remote-controlled for security reasons, the illusion of an autonomous system was preserved as the operator was hidden from the pedestrians. Three researchers accompanied the experiment: one conducted the unstructured observation and two the interviews. The study lasted for two hours.

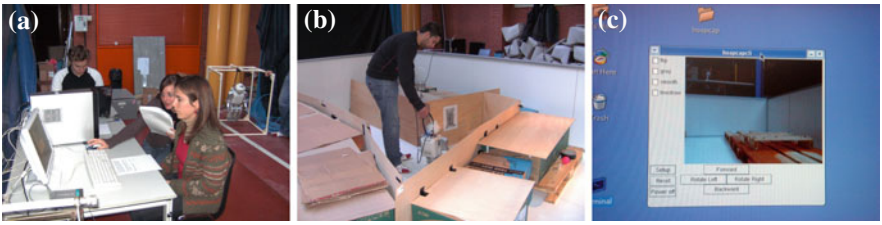


Fig. 1.5 Second HOAP-3 study: **a** study setting: participant, **b** study setting: maze, **c** computer interface

Table 1.5 Questions on social aspects: second HOAP-3 study (1 = "strongly disagree" to 5 ="strongly agree", N excluding missing answers)

No.	Question	N	Mean	SD
1 FG	Robots will be part of our everyday work	12	4.00	1.13
2 FG	Robots will be an important part of our society	12	3.25	1.14
3 FG	I would like to collaborate with robots	12	4.25	0.62
4 FG	Robots will have a similar importance as human colleagues	12	2.33	1.16
5 FG	Robots and humans will make a good team	12	4.25	0.45
1 AT	I would not like to imagine a world in which robots were not used	11	2.91	1.38
2 AT	I can imagine taking a robot into my heart	11	3.82	1.25
3 AT	It would feel good if a robot was near me	12	3.83	0.94
4 AT	I can imagine building a special relationship with robots	10	2.50	1.51
1 RE	The interaction with robots will be a mutual experience	12	3.33	1.16
2 RE	I can imagine that I will care for the well-being of a robot	12	4.00	1.28
3 RE	The relationship with robots will be based on the principle of give and take	12	2.42	1.24
4 RE	Humans and robots will be interdependent	11	2.73	1.49

In the second study (see Fig. 1.6), the ACE robot had to move autonomously from the Odeonsplatz to the Marienplatz and back by asking pedestrians for directions. The pedestrians could tell ACE where to go by first showing it the right direction by pointing and then show ACE the right way (e.g. how far from here) on the map on its touch-screen. The development team of the robot stayed near it because of security reasons, but stayed invisible from pedestrians because of the well-frequented environment. The study lasted for five hours and was accompanied by four researchers, one of whom performed the unstructured passive observation and the other three conducted the interviews.

Fig. 1.6 First **(a)** First ACE study and second **(b)** second ACE study with ACE

Table 1.6 Questions on social aspects: first ACE study (1 = "strongly disagree" to 5 = "strongly agree", N excluding missing answers)

No.	Question	N	Mean	SD
1 FG	I feel accompanied in the presence of ACE	18	4.39	0.85
1 AT	I would trust ACE if it gave me an advice	18	3.56	0.92
2 AT	I would follow the advice of ACE	18	3.72	0.96
1 RE	I perceive ACE as a "Social Actor"	18	1.94	1.31
2 RE	Humans and ACE robots will be interdependent from each other in future	18	2.56	1.29

In both settings, the participants had the possibility to interact with the robot via its touch-screen (in the first study to get more information about the robot itself and on the Karlsplatz, in the second study to show the robot the way on the map). In the first study, 18 participants filled in the social interaction questionnaire and 52 participants in the second study.

1.4.6.1 Findings on Social Aspects

Due to the public setting of the investigations, specialized questionnaires were used in the ACE studies, which were, however, trying to incorporate additional items on the concept of sociality (the complete questionnaires can be found in [37]). The Tables 1.6 and 1.7 show the items and the according results for the two studies.

In terms of the attitude towards interacting with the robot, the observational data of the first ACE study revealed the behaviour pattern of "investigating the engineering

Table 1.7 Questions on social aspects: second ACE study (1 = "strongly disagree" to 5 = "strongly agree", N excluding missing answers)

No.	Question	N	Mean	SD
1 FG	ACE needs my support to carry out its task	48	3.83	1.62
2 FG	I felt threatened because of the presence of ACE	52	4.52	1.26
3 FG	ACE will have a similar relevance in future like human colleagues	52	1.83	1.22
4 FG	ACE and me would compose a good team	49	3.57	1.31
1 AT	I would trust ACE if it gave me an advice	52	3.25	1.47
2 AT	I would follow the advice of ACE	51	3.27	1.36
3 AT	I can imagine to develop a special relationship to ACE	51	2.41	1.61
4 AT	I can imagine to take ACE into my heart	51	2.27	1.47
1 RE	ACE reacted on my behaviour	48	3.31	1.72
2 RE	Humans and ACE robots will be interdependent from each other in future	51	3.02	1.70
3 RE	The interaction of ACE should be a give and take	51	4.02	1.36
4 RE	The interaction with ACE was like give and take	41	3.15	1.68

of ACE" six times, which could be observed only for male pedestrians. For the second field trial, the analysis of the observational material showed that people were very curious towards this new technology and many people stated surprise in a positive way: "It is able to go around me. I would not have thought that". Unfortunately, some people also seemed scared. However, most of the time curiosity prevailed over anxiety (people not only watched, but also decided to interact with ACE). This could probably be due to the so called "novelty effect". In the first field trial, the participants rated the item on forms of grouping rather positive (*mean*: 4.39, *SD*: 0.85). The observational data revealed the interesting finding that pedestrians built "interaction groups", meaning that a group of 10–15 strangers stood in front of ACE and each member of this "coincidental" group stepped forward to interact with ACE, while the rest of the group waited and watched the interaction. In the second field trial, the participants rated the aspect of forms of grouping in the street survey rather positive (*mean*: 3.45, *SD*: 0.12). This aspect was rated significantly better by those participants who actually interacted with ACE ($t(35.96) = 4.09$, $p < 0.05$).

In the first field trial, the participants rated the aspect attachment rather positive for both items ("1 AT" and "2 AT"). However, in the second field trial, the participants rated the overall aspect of attachment rather low (*mean*: 2.85, *SD*: 0.14), which is due to the rating of item "3 AT". The items on trusting an advice of the robot ("1AT" and "2 AT") were rated similarly like in the first study. Interestingly, participants younger than 50 years rated this aspect significantly lower than older participants ($t(43.80) = -2.36$, $p < 0.05$). Moreover, in the second field trial, some people showed companion like behaviour towards the robot (e.g. "Let's have a look … Oh yes … come on … take off"). The robot was directly addressed like a social actor.

However, not everyone addressed the robot in second person, even if they were standing right in front of it; a behaviour which would be considered extremely impolite when interacting with a human and an indicator that the robot was not perceived as a partner by everyone.

Regarding reciprocity & reflexivity the participants of the first field trial rated the items rather low ("1 RE" and "2 RE"). A reason for this could be that only 54 % of the participants experienced the robot as interactive, due to its limited interaction possibilities in the first study set-up. In the second field trial, the participants rated the aspect of reciprocity & reflexivity in the street survey rather positively (*mean*: 3.50, *SD*: 0.16). This aspect was significantly better rated by those participants who actually interacted with ACE ($t(24.85) = 2.44$, $p < 0.05$). Furthermore, reciprocity & reflexivity was the only factor that was significantly better rated by men than by women ($t(19.99) = -2.42$, $p < 0.05$). Moreover, participants younger than 50 years rated this aspect significantly lower than older participants ($t(37) = -3.27$, $p < 0.05$).

1.5 Lessons Learned for Interaction Scenarios with Anthropomorphic Agents

The main goal of the comparison of the six case studies was to explore participants' perception of sociality during the interaction with humanoid robots. Overall, the studies have shown how deeply the social interaction paradigm is embedded within the interaction with anthropomorphic robotic systems. All participants in the case studies were novice users, they had no pre-experiences with robots at all and they received no other information on how the interaction with the robots works, than that given in the scenario and task description. Nevertheless, in all laboratory-based case studies (studies 1–4) almost all participants completed the task successfully (task completion rate >80 %), which indicates the high degree of sense-making on the user side and reflexivity on the robot side. An overview of all results in given in Table 1.8.

But what can we learn from that for future interaction scenarios with anthropomorphic agents (physical and virtual)? The first study with the HRP-2 robot showed the importance of multimodal feedback and turn taking to perceive the sociality aspect forms of grouping. It moreover showed that even a virtual screen representation of a humanoid robot can be perceived as social actor, as long as it proactively reacts to the actions of the user. The second study with the HRP-2 robot, revealed that a humanoid robot can even be a mediator for sociality between two humans, who are not collocated in the same room during the interaction. The robot served as embodiment for a feeling of social presence in a way that the participants experienced a grouping with the robot, as well as with the human operator behind the scene.

The first study with the HOAP-3 robot showed that an anthropomorphic design which is perceived as "cute" can foster the social aspect attachment. Moreover, a

Table 1.8 Questions on social aspects: comparison of the results of all 6 case studies (1 = "strongly disagree" to 5 = "strongly agree"): Case 5 and 6 had different questions than the other studies, see the corresponding section for details

No.	Study 1: HRP-2	Study 2: HRP-2	Study 3: HOAP-3	Study 4: HOAP-3	Study 5: ACE	Study 6: ACE
1 FG	m = 4.05 (1.00)	m = 3.64 (0.92)	m = 3.75 (0.62)	m = 4.00 (1.13)	m = 4.39 (0.85)	m = 3.83 (1.62)
2 FG	m = 3.50 (0.96)	m = 4.25 (0.62)	m = 3.18 (1.33)	m = 3.25 (1.14)		m = 4.52 (1.26)
3 FG	m = 3.48 (1.12)	m = 3.92 (0.90)	m = 3.55 (0.82)	m = 4.25 (1.14)		m = 1.83 (1.22)
4 FG	m = 1.73 (0.99)	m = 2.25 (1.14)	m = 1.83 (1.12)	m = 2.33 (1.16)		m = 3.57 (1.31)
5 FG	m = 3.19 (1.17)	m = 4.17 (0.72)	m = 3.67 (0.78)	m = 4.25 (0.45)		
1 AT	m = 2.84 (1.21)	m = 3.50 (0.80)	m = 3.25 (1.14)	m = 2.91 (1.38)	m = 3.56 (0.92)	m = 3.25 (1.47)
2 AT	m = 2.26 (1.32)	m = 2.92 (1.17)	m = 2.25 (1.22)	m = 3.82 (1.25)	m = 3.72 (0.96)	m = 3.27 (1.36)
3 AT	m = 1.92 (0.93)	m = 3.08 (0.90)	m = 2.82 (1.17)	m = 3.83 (0.94)		m = 2.41 (1.61)
4 AT	m = 1.91 (1.16)	m = 3.42 (1.00)	m = 2.58 (1.08)	m = 2.50 (1.51)		m = 2.27 (1.47)
1 RE	m = 3.45 (1.34)	m = 3.45 (1.13)	m = 3.10 (1.10)	m = 3.33 (1.16)	m = 1.94 (1.31)	m = 3.31 (1.72)
2 RE	m = 2.96 (1.46)	m = 3.42 (1.00)	m = 3.55 (0.82)	m = 4.00 (1.28)	m = 2.56 (1.29)	m = 3.02 (1.70)
3 RE	m = 2.26 (1.39)	m = 2.27 (1.27)	m = 3.08 (1.51)	m = 2.41 (1.24)		m = 4.02 (1.36)
4 RE	m = 3.13 (1.33)	m = 3.45 (1.13)	m = 2.82 (0.75)	m = 2.73 (1.49)		m = 3.15 (1.68)

Table 1.9 Social agent matrix: completed in accordance to the empirical results of the 6 case studies

Social aspect	Direct interaction	Mediated interaction
Forms of grouping	First HRP-2 study First HOAP-3 study Second HOAP-3 study First ACE study	Second HRP-2 study
Attachment		
Reciprocity Reflexivity	Second ACE study	

learning by demonstration scenario based on interactive tutelage is perceived as reciprocal and reflexive. The second HOAP-3 study also supported the assumption that a design which is perceived as "cute" fosters attachment, however, in a completely different situation in which the user had no direct contact interaction with the robot, but remote-controls it. Furthermore, this study could show that cooperative problem solving offers a suitable basis to foster the aspect forms of grouping.

The two studies with the ACE robot showed that sociality is also perceived if the perspectives are inverted and the robot proactively starts an itinerary request and is dependent on the users' input to achieve its goal. The importance of interactivity for the perception of reciprocal behaviour became obvious in the comparison of the two studies, as reciprocity was rated lower in the first study. The second study could also show the significant impact on the actual interaction (comparing pure observation) on the perception of sociality, in specific the aspects forms of grouping and reciprocity. This finding is also supported by summarizing the results of the NARS questionnaire. A significant change on the general attitude towards robots due to the interaction with them could be identified twice in a decreased rating of the "Negative Attitude toward Social Influence of Robots" and once of the "Negative attitude toward Emotions in Interaction with Robots".

Upon reflecting how the results for the four aspects of sociality affect not only the interaction with physical agents, but also with virtual ones, we can come up with a "social agent matrix", which distinguishes between direct human–agent interaction (see case studies 1, 3, 4, 5, and 6) and human–human interaction mediated by an agent (see case study 2). This matrix allows a categorization of agents in accordance to their "focus of sociality". According to the empirical data gained in the studies, the matrix in Table 1.9 should be filled in.

There has been an inscription of certain notions in the embodiment of the robots, as well as the human–robot interaction scenarios as a whole, which had an influence on participants' perception of sociality. Even though different scenarios have been performed with different robots, the aspect forms of grouping was rated best in all case studies except one, which indicates a relatively low

influence of the anthropomorphic embodiment of the robot. Besides the fact that questionnaire and item design/formulation issues could be the reason for this result, we have to act on the assumption of a co-production between the embodiment of the robot, the interaction scenario, and the perception of sociality on the human–agent relationship. If the embodiment of the robots and the interaction scenarios were more distinctive, the perception of sociality would probably not have been the same. For instance, the underlying narration of the learning by demonstration scenario with HOAP-3 or the mixed-reality scenario with the virtual HRP-2 robot points more into the direction of reflexivity as key social aspect than forms of grouping.

1.6 Reflection on the Relevance of Social Cues in Human–Agent Relationship

Upon reflecting on the four aspects of sociality, it becomes obvious that designing for the social as an end in itself cannot be the overall solution for establishing a working human–agent relationship. Rather sociality as design paradigm has to be interpreted as a modular concept in which the most relevant aspect has to be chosen to increase sociality for a specific interaction scenario. Designing agents in an anthropomorphic appearance increases the degree of being perceived as a social actor, as Fogg already shows in his work "Computers as Persuasive Social Actors" [12]. However, anthropomorphic design carries a lot of baggage with it, in terms of specific expectations of end-users, such as intelligence, adaptation towards user behaviour, and following social norms and human–oriented perception cues [10].

To our conviction, the one (most important) social aspect for the functionality of the agent has to be identified first and builds the basis for the design. For instance reflexivity could be most important while playing a game against an agent, as the moves of the user are always a reflection on the agent's gameplay. Forms of grouping will be most relevant in cooperative tasks, like solving a problem/task together as a team (e.g. a wizard agent who guides through an installation process). Reciprocity will be highly relevant in care-giving (comparable to the "Tamagotchi Effect") or persuasive tasks, in which the user expects an adaptation from the agent's behaviour due to his/her adaptation of behaviour. Attachment will be relevant in all cases in which we want the user to establish a long-term relationship with the agent. However, in this case variations in the agent's behaviour are highly relevant to ensure that the interaction does not become monotonously and annoying.

If we have again a look at all the studies presented in this chapter, how could we increase the degree of sociality for the specific interaction scenarios? In accordance to our underlying assumption on the "one (most important) social aspect rule", the social agent matrix for the studies should look like Table 1.10.

Thus, if we aim for a user-centred design approach of an agent, the first relevant question is to identify which social aspect it should predominantly depict

Table 1.10 Social agent matrix: completed in accordance to the assumption "one (most important) social aspect" should be fostered

Social aspect	Direct interaction	Mediated interaction
Forms of grouping	Second HOAP-3 study	Second HRP-2 Study
Attachment		
Reciprocity	First ACE study Second ACE study	
Reflexivity	First HOAP-3 study First HRP-2 study	

Table 1.11 Social interface matrix: a categorization of all kinds of interactive computing systems in terms of their degree of sociality

Social aspect	Human–system interaction	Human–human interaction
Forms of grouping	Virtual agents (e.g. virtual sports trainer)	CSCW systems (e.g. wikipedia) CMC systems (e.g. skype)
Attachment	Virtual pets (e.g. Tamagotchi) care robots (e.g. Paro robotic seal)	Social networking systems (e.g. Facebook)
Reciprocity	Multiplayer games (e.g. WoW)	Polite computing (e.g. install wizard characters)
Reflexivity	Recommender systems (e.g. Amazon)	Intelligent, adaptive systems (e.g. chess computer)

and to focus on that in the interaction and screen/embodiment design. However, user perception of sociality seems to focus more on the interaction scenario of the agent, than on its pure appearance, even though an anthropomorphic appearance fosters the perception of social abilities of an agent. As the theory of object-centred sociality already suggests (and also the Media Equation Theory [31] and the experiment of Heider and Simmel [17] point in that direction), the interaction with all kinds of agents can be perceived as social, also with a zoomorphic or a functionally designed agent. The most important thing to consider is that the interaction in itself is a social one and to focus on the main specific aspect of its sociality to increase the social perception on the user side. One could go even one step further and propose a "social interface matrix" which categorizes all kinds of interactive computing systems, irrespectively of their embodiment and design, in terms of their degree of sociality; some self-explaining examples are presented in Table 1.11.

1.7 Conclusion and Outlook

Table 1.11 demonstrated the power of the "post-social world" [25] in which non-human entities enter the social domain. It shows that human–system interaction can be social in many types of interaction paradigms, going far beyond the interac-

tion with anthropomorphic agents. However, in the design of the future human–agent relationship it will be crucial to identify whether the underlying interaction paradigm is a "post-social" one. There is no need to socialize every computing system we are using, e.g. word processing or mobile phones, with the assumption that this would improve the human–system relationship.

A tendency can be observed that more and more "post-social" interaction design strategies are pursued, primarily focussing on anthropomorphic appearance, however, as the case studies presented in this chapter could show, it is more about interaction design than appearance design to foster the social human–agent relationship. Not only humanoid robots could be perceived as social actors in collaboration scenarios, but also functionally designed robots in a factory context, as long as the interaction is designed in a social manner.

In a next step, we want to explore the impact of appearance on the perception of sociality further, by transferring the concept of the four sociality aspects to human–robot interaction with functionally designed robots in an industrial setting. We believe that future interaction scenarios for human–robot collaboration in the factory context need to combine the strengths of human beings (e.g. creativity, problem solving capabilities, and the ability to make decisions) with the strengths of robotics (e.g. efficient fulfilment of reoccurring tasks). Single robotic work cells are not sufficient for sustainable productivity increase. Thus, we want to use the "social agent matrix" to inform the interaction scenarios of joint human–robot collaboration in turn taking tasks, to explore if and how sociality may positively influence performance measures.

Acknowledgments This work was conducted in the framework of the EU-funded FP6 project ROBOT@CWE. Moreover, the financial support by the Federal Ministry of Economy, Family and Youth and the National Foundation for Research, Technology and Development is gratefully acknowledged (Christian Doppler Laboratory for "Contextual Interfaces").

References

1. Gold, K., Scassellati, B.: A Bayesian robot that distinguishes "self" from "other". In: Proceedings of the 29th Annual Meeting of the Cognitive Science Society (CogSci2007). Psychology Press, New York (2007)
2. Abdulla, R.A., Garrison, B., Salwen, M., Driscoll, P., Casey, D.: The Credibility of Newspapers, Television News and Online News. In: Journalism and Mass Communication, Miami (2002)
3. Powers, A., Kiesler, S.: The advisor robot: tracing people's mental model from a robot's physical attributes. In: Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human–Robot Interaction (HRI'06), pp. 218–225. ACM, New York (2006). doi:10.1145/1121241.1121280
4. Bartneck, C.: Affective expressions of machines. In: Extended Abstracts on Human Factors in Computing Systems (CHI'01). pp. 189–190. ACM, New York (2001). doi:10.1145/634067.634181
5. Bartneck, C., Croft, E., Kulic, D.: Measuring the anthropomorphism, animacy, likeability, perceived intelligence and perceived safety of robots. In: Metrics for Human–Robot Interaction Workshop in Affiliation with the 3rd ACM/IEEE International Conference on Human–Robot Interaction (HRI 2008), Technical Report 471, pp. 37–44. University of Hertfordshire, Amsterdam 2008

6. Berlo, D.K., Lemert, J.B., Mertz, R.J.: Dimensions for evaluating the acceptability of message sources. Public Opin. Q. **46**, 563–576 (1969)
7. Bowlby, J.: The nature of the child's tie to his mother. Int. J. Psychoanal. **39**, 350–373 (1958)
8. Cramer, H.: People's responses to autonomous and adaptive system. Ph.D. thesis, University of Amsterdam (2010)
9. Dahlbäck, N., Jönsson, A., Ahrenberg, L.: Wizard of Oz studies: why and how. In: Proceedings of the 1st International Conference on Intelligent User Interfaces (IUI'93), pp. 193–200. ACM, New York (1993)
10. Dautenhahn, K.: Design spaces and niche spaces of believable social robots. In: Proceedings of the 11th IEEE International Workshop on Robot and Human Interactive Communication (RO-MAN2002), pp. 192–197 (2002)
11. Desai, M., Stubbs, K., Steinfeld, A., Yanco, H.: Creating trustworthy robots: lessons and inspirations from automated systems. In: Proceedings of the AISB Convention: New Frontiers in Human–Robot Interaction (2009)
12. Fogg, B.J.: Computers as persuasive social actors. In: Fogg, B.J. (ed.) Persuasive technology. Using Computers to Change What we Think and do, Chap. 5, pp. 89–120. Morgan Kaufmann Publishers, San Francisco (2003)
13. Fogg, B.J., Tseng, H.: The elements of computer credibility. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'99), pp. 80–87. ACM, New York (1999) doi:10.1145/302979.303001
14. Fong, T., Nourbakhsh, I., Dautenhahn, K.: A survey of socially interactive robots. Robot. Auton. Syst. **42**, 143–166 (2003)
15. Gaziano, C., McGrath, K.: Measuring the concept of credibility. J. Q. **63**(3), 451–462 (1986)
16. Gouldner, A.W.: The norm of reciprocity: a preliminary statement. Am. Sociol. Rev. **25**, 161–178 (1960)
17. Heider, F., Simmel, M.: An experimental study of apparent behavior. Am. J. Psychol. **57**, 243–249 (1944)
18. Hovland, C.I., Janis, I.L., Kelly, H.H.: Communication and Persuasion. Yale University Press, New Haven (1953)
19. Imai, M., Narumi, M.: Immersion in interaction based on physical world object. In: Proceedings of the 2005 International Conference on Active Medial Technology (AMT2005), pp. 523–528 (2005)
20. Kahn P.H. Jr., Freier, N., Friedman, B., Severson, R., Feldman, E.: Social and moral relationships with robotic others? In: Proceedings of the 13th IEEE International Workshop on Robot and Human Interactive Communication (RO-MAN2004), pp. 545–550 (2004)
21. Kaplan, F.: Free creatures: the role of uselessness in the design of artificial pets. In: Proceedings of the 1st Edutainment Robotics Workshop (2000)
22. Kaplan, F.: Artificial attachment: will a robot ever pass the Ainsworth's strange situation test? In: Proceedings of the IEEE-RAS International Conference on Humanoid Robots (Humanoids 01), pp. 125–132 (2001)
23. Kidd, C.D.: Sociable robots: the role of presence and task in human–robot interaction. Master's thesis, Massachusetts Institute of Technology (2003)
24. Kiesler, S.B., Goetz, J.: Mental models of robotic assistants. In: CHI Extended Abstracts, pp. 576–577, Minneapolis (2002)
25. Knorr-Cetina, K.: Sociality with objects: social relations in postsocial knowledge societies. Theor. Cult. Soc. **14**(4), 1–30 (1997)
26. Lohse, M., Hegel, F., Wrede, B.: Domestic applications for social robots—a user study on appearance and function. J. Phys. Agents **2**, 21–32 (2008)
27. Lynch, M., Peyrot, M.: Introduction: a reader's guide to ethnomethodology. In: Qualitative Sociology, Springer (2005)
28. McCroskey, J.C., Hamilton, P.R., Weiner, A.N.: The effect of interaction behavior on source credibility, homophily, and interpersonal attraction. Human Commun. Res. **1**, pp. 42–52 (1974)
29. Nomura, T., Kanda, T., Suzuki, T.: Experimental investigation into influence of negative attitudes toward robots on human–robot interaction. AI Soc. **20**(2), 138–150 (2006)

30. Norman, D.A.: Emotional Design: Why We Love (or Hate) Everyday Things. Basic Books, New York (2004)
31. Reeves, B., Nass, C.: The Media Equation: How People Treat Computers, Televisions, and New Media Like Real People and Places. Cambridge University Press, New York (1996)
32. Reichenbach, J., Bartneck, C., Carpenter, J.: Well done, robot!—the importance of praise and presence in human–robot collaboration. In: Dautenhahn, K. (ed) The 15th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN 2006), pp. 86–90. Hatfield (2006). doi:10.1109/ROMAN.2006.314399
33. Shinozawa, K., Reeves, B., Wise, K., Lim, S., Maldonado, H.: Robots as new media: a cross-cultural examination of social and cognitive responses to robotic and on-screen agents. International Communication Association (2002)
34. Smith, L., Breazeal, C.: The dynamic life of developmental process. Dev. Sci. **10**(1), 61–68 (2007)
35. Sundar, S.S., Nass, C.: Source orientation in human–computer interaction. In: Communication Research, vol. 27, pp. 683–703. Sage Journals (2000). doi:10.1177/009365000027006001
36. Wehmeyer, K.: Assessing users' attachment to their mobile devices. In: Proceedings of the International Conference on the Management of Mobile Business (2007)
37. Weiss, A.: Validation of an evaluation framework for human–robot interaction. The impact of usability, social acceptance, user experience, and societal impact on collaboration with humanoid robots. Ph.D. thesis, University of Salzburg (2010)
38. Weiss, A., Bernhaupt, R., Schwaiger, D., Altmaninger, M., Buchner, R., Tscheligi, M.: User experience evaluation with a Wizard of Oz approach: technical and methodological considerations. In: Proceedings of the 9th IEEE-RAS International Conference on Humanoids Robotics (Humanoids 2009), pp. 303–308 (2009)
39. Weiss, A., Bernhaupt, R., Tscheligi, M., Wollherr, D., Kühnlenz, K., Buss, M.: A methodological variation for acceptance evaluation of human–robot interaction in public places. In: Proceedings of the 17th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN 2008), pp. 713–718 (2008)
40. Weiss, A., Igelsböck, J., Pierro, P., Buchner, R., Balaguer, C., Tscheligi, M.: User perception of usability aspects in indirect HRI—a chain of translations. In: Proceedings of the 19th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN 2010), pp. 574–580 (2010)
41. Weiss, A., Igelsböck, J., Tscheligi, M., Bauer, A., Kühnlenz, K., Wollherr, D., Buss, M.: Robots asking for directions—the willingness of passers-by to support robots. In: Proceedings of the 5th ACM/IEEE International Conference on Human Robot Interaction (HRI'10), pp. 23–30. ACM, New York (2010)
42. Weiss, A., Igelsböck, J., Wurhofer, D., Tscheligi, M.: Looking forward to a "Robotic Society"?—Imaginations of Future Human–Robot Relationships. Special issue on the Human Robot Personal Relationship Conference in the International Journal of Social Robotics (2010)
43. Weiss, A., Igelsböeck, J., Calinon, S., Billard, A., Tscheligi, M.: Teaching a humanoid: a user study on learning by demonstration with HOAP-3. In: Proceedings of the 18th IEEE International Symposium on Robot and Human Interactive Communication (Ro—Man2009), pp. 147–152 (2009)
44. Weiss, A., Tscheilig, M.: Special issue on robots for future societies evaluating social acceptance and societal impact of robots. Int. J. Soc. Robot. **2**(4), 345–346 (2010)
45. Weizenbaum, J.: Eliza—a computer program for the study of natural language communication between man and machine. Commun. ACM **9**(1), 36–45 (1966). doi:10.1145/365153.365168

Chapter 2
Believability Through Psychosocial Behaviour: Creating Bots That Are More Engaging and Entertaining

Christine Bailey, Jiaming You, Gavan Acton, Adam Rankin and Michael Katchabaw

Abstract Creating believable characters or bots in a modern video game is an incredibly challenging and complex task. The requirements for achieving believability are steep, yet the improvements in player engagement, immersion, and overall enjoyment and satisfaction with their experience make this a worthy problem in desperate need of further examination. In this chapter, we examine how the implementation of psychosocial behaviour in bots can lead to believability, and how this translates into new and exciting possibilities for advancing the state-of-the-art in this area.

2.1 Introduction

The field of Artificial Intelligence in games is a broad, yet demanding area of study. In [62], artificial intelligence is defined as being concerned with thought processes, reasoning, and behaviour as it applies to human performance as well as ideal intelligence, or rationality. Artificial Intelligence in games differs from traditional Artificial Intelligence in that it is optimized not for the simulation of human performance

C. Bailey · J. You · G. Acton · A. Rankin · M. Katchabaw (✉)
Department of Computer Science, The University of Western Ontario,
London, ON N6K 4K8, Canada
e-mail: katchab@csd.uwo.ca

J. You
e-mail: jyou23@csd.uwo.ca

G. Acton
e-mail: gacton@csd.uwo.ca

A. Rankin
e-mail: arankin@csd.uwo.ca

C. Bailey
e-mail: cdbailey@csd.uwo.ca

P. Hingston (ed.), *Believable Bots*, DOI: 10.1007/978-3-642-32323-2_2,

or rationality, but for entertainment and increased immersion in the game world [58, 73].

One of the more active areas of research in game artificial intelligence in both industry and academia is the creation of more believable characters or bots [11, 24–26, 30, 31, 37, 54, 59]. This is only natural, as players of modern video games increasingly expect Artificial Intelligence that is dynamic, is able to react to unexpected events, and behaves believably [17, 68–70]. This is particularly the case in character and story-driven genres, such as role-playing games and action-adventure games, but is also true of open world games and others in which individual characters or populations of characters are important to the overall player experience.

The state-of-the-art in the creation of believable decision making processes for bots has reached out beyond computer science using models from psychology (personality, emotions, appraisal theory) and sociology (social appraisal variables, relationships and role theory), as these elements have been recognized as foundations for believable behaviour [40]. Formalizing and encoding psychosocial elements for use in games has proven to be challenging, yet rewarding in terms of the believable bot behaviour that can be achieved through their use, and the richer and more engaging experiences that can be delivered to players as a result [4, 80].

This chapter provides a thorough and detailed treatment of this topic, delving into both theoretical and practical aspects of providing psychosocial bot behaviour in modern video games. The remainder of this chapter is sectioned as follows.

Background: Introducing the reader to this area, providing motivation for the use of psychosocial elements in defining more believable behaviour, and discussing the current state-of-the-art.

Introduction to Psychosocial Behaviour: A brief overview of relevant aspects of psychology and sociology and how these elements impact decision making processes and behaviour.

A Framework for Psychosocial Behaviour: An examination of how to formally model and represent various psychosocial elements for use in video games.

Implementation and Results: With the framework in mind, this section discusses practical aspects of implementing the framework and delivering psychosocial behaviour in a game.

Performance Optimization: Given that games are real-time and interactive by nature, the use of psychosocial behaviour raises performance considerations that must be dealt with [56].

Concluding Remarks: A summary of what has been covered, as well as a discussion of open problems and opportunities for further studies in this area.

2.2 Background

The literature in this area is rich with work taking various approaches to exploring aspects of psychosocial behaviour for believable bots. While this work provides many valuable lessons of importance to this area, the majority of this work has limitations

or lacks necessary elements for realistic and believable psychosocial behaviour. This section provides an overview of this background work.

Bot behaviour is often handled using finite state machines to represent state of mind [11, 24], or scripting to hard-code behaviour in each possible game situation [8]. Since the main goal for game artificial intelligence is to support the designers in providing a compelling game experience, supporting interactivity, player choice, and replayability [58], these commonly used approaches are too limiting to provide truly believable behaviour.

One of the factors in creating a compelling experience is suspending the player's disbelief. The Autonomy Interaction Presence (AIP) Cube model (as cited by [11]) states that the requirements for suspension of disbelief are: Autonomy of the bots, Interaction between the bots and the player, and the bots' Presence in the game world. The work in [11] also implicates personality, emotion, self-motivation, the illusion of life (which includes goals, plans, reactions to the environment, human-like restrictions, and many other behaviours which create the appearance of life), change, and social relationships as being important to character believability. This supports the commonly referenced definition of believability proposed by Loyall [40].

Some relevant readings in the area of affective computing and emotion synthesis include Funge's definition of emotion as the sum of past events [24], the discussion in [11] of character emotion in the context of games, and Picard's discussion of shifting emotions [52], and Picard in [52] and the overview in [11] of the field of affective computing. Personality in agents is defined as the agent's pattern of behaviours or interactions with its environment [37]. Crawford [15] outlines one possible personality model for bots in games that include intrinsic variables (i.e. integrity, virtue, intelligence, and so on), mood variables, relationship variables (beliefs about another's intrinsic variables), and the readiness to change the previous two variables. Isbister discusses the social psychology research and discusses the traits of agreeableness and dominance and how they can be used to form many different personalities [35].

Reputation systems—such as those in Fable [3], Thief: Deadly Shadows [34], and Ultima Online [29]—refer to systems that typically manage bot opinions of the player [3, 45], which are formed immediately and globally among all bots upon certain player actions [3, 10, 45, 29]. Reputation systems typically do not maintain individual opinions [3, 45], nor opinions about other bots, though they may maintain group opinions [3, 29, 34]. The work in [30] is an exception to this, however, providing a more flexible and realistic method of modeling reputation.

Some social science concepts of interest in video game research include an agent's roles [31, 35], cultures and subcultures [35], norms, values, worldview [31], and goals [35]. Some papers have attempted to address these [31, 35]. The area of social agents, or Socially Intelligent Agents (SIAs) [11, 55] would appear to have much relevance to this research, however many of these "social" agents do not exhibit realistic social behaviour [13, 31, 32, 35, 77]. Many "social" agents implement only communicative behaviour that is used in a multi-agent problem-solving context to reduce resource usage and increase efficiency. Some relevant research in this area includes Tomlinson and Blumberg's remembered interaction histories between agents [72], Prendinger

et al.'s change of attitudes and familiarity assessment between agents [55], Cole's comparison of opinion flow in a multi-agent system to flocking behaviour [14], and Guye-Vuilleme's high level architecture for social agents [31].

Finally, a particularly interesting approach to problems of providing unique and immersive experiences lies in emergent gameplay [51, 66, 78]. Until recently, emergent gameplay is known to have been used in bot behaviour only in relatively simple situations and behaviours [38, 51, 66, 69, 78], usually to deal with emergent properties of the game world and the objects contained within it. Emergence, however, had not been used to implement complex psychological states or social relationships. The foundations for the approach to believable bot behaviour discussed in this chapter come from our own previous work in [4], which first attempted to apply emergence to the creation of realistic psychosocial behaviour. This work was extended in [80] to provide better and more complete psychosocial modeling, and again in [56] to integrate more rigorous proactive planning elements to complement the more reactive behaviour present in our earlier work.

2.3 Introduction to Psychosocial Behaviour

The requirements for believability for bots in modern video games tend to be quite steep [40]. They include elements such as personality, emotion, self-motivation, social relationships, consistency, the ability to change, and the ability to maintain an "illusion of life", through having goals, reacting and responding to external stimuli, and so on [40]. While crafting a game artificial intelligence capable of achieving all of this is a daunting task indeed, one can see that at its core, a complete and integrated psychosocial model is necessary, operating in a dynamic fashion [4].

In this section, we examine the needs of bots from both psychological and sociological perspectives, and briefly discuss key relevant works in these areas.

2.3.1 Psychological Foundations

Elements of a bot's psychology that form the basis for its behaviour can include a model of emotion, a personality model, needs, and even values and a worldview [7]. Since there are different theoretical models of emotions and personality [23], and different game worlds may require different needs, values, worldview and cultural constructs, it is necessary to have a flexible approach to modeling psychological traits. For brevity, we will restrict our discussion here to personality and emotion, as these tend to be the most commonly used and referenced psychosocial elements in the literature [4].

Numerous models for personality have been developed over the years. One is the Myers-Briggs Type Indicator [44], which is based on four pairs of traits that are considered complementary and distinct that measure: how a person relates to

others (either by Extraversion or Introversion), how a person takes in information (either by Sensing or Intuition), how a person makes decisions (either by Thinking or Feeling), and how a person orders their life (either by Judging or Perceiving). Another popular model that has been advanced by many researchers is known as the Five Factor Model (FFM), the Big Five, or OCEAN, which assesses personality in five independent dimensions: Openness, Conscientiousness, Extraversion, Agreeableness and Neuroticism [74]. This work has origins that can be traced back to a sixteen factor model created by Cattell [12]. The PEN model [22], on the other hand, is comprised of just three personality dimensions, and is based on psychophysiology: Psychoticism, Extraversion, and Neuroticism. A slightly different perspective is offered in Reiss' model of basic desires [57], where personality is defined primarily by a set of tendencies and motivators that inspire or lead to action: power, curiosity, independence, status, social contact, vengeance, honour, idealism, physical exercise, romance, family, order, eating, acceptance, tranquillity and saving. Other work in this area has examined linkages amongst these and other models [1, 42], and while connections and correlations exist, there are still fundamental differences between these models, and no single complete over-arching model.

Similarly, several models of emotion have been formulated and studied. One of the more popular models was formulated by Ekman [18], defining six basic emotions: anger, disgust, fear, joy, sadness, and surprise. Another popular model is the OCC model, consisting of twenty-two emotions [47]. Scaled down versions of this model also exist [46]. Numerous other models exist, including Smith and Ellsworth's Emotion Appraisal Model [65], Mehrabian's PAD Emotional State Model [43], and models put forward by Tomkins [71], Plutchik [53], and Parrott [50]. Again, while there is overlap between these models, these models have many differences and were defined with different purposes in mind, such as facial expression, relation to adaptive biological processes, action readiness, and so on.

When applying these various psychosocial models to the creation of bots for games, researchers have followed one of two paths. In the first case, researchers have selected one of the models that they believe best suits their needs (or one each of personality and emotion), in the hopes that this will be sufficient and that their bots will not suffer from missing what is offered by the other models. This was done in work such as [5, 6, 19, 20, 75, 81] and our own previous work in [4]. In the second case, researchers have instead constructed their own custom models, borrowing aspects from the common standard models in an ad hoc fashion, as none of these models on their own meet their needs. This was done in work such as [27, 33, 48, 59, 61, 64]. Unfortunately, to date, there has been no work towards integrating the various models together or enabling their interoperability within agents or game characters, despite the potential benefits of leveraging the respective advantages of these well-studied and time-tested models all at once. An approach to doing this, however, is presented in Sect. 2.4.

2.3.2 Sociological Foundations

Humans are social beings, and our social relationships, culture, values, and norms all play an important part in defining who we are. As a result, believable bots must be socially situated, and able to reason in social terms [16, 28, 31, 41, 60]. Bots armed with social knowledge will have greater believability as they interact with players and each other [31, 60]. Psychology alone does not describe important aspects of human behaviour including social constraints, moral standards and routine behaviours [31], and so some notion of sociology is also required in bots. Unfortunately, social bots are particularly difficult to create as they must reproduce behaviours that are complex, reflecting a number of intricacies of human behaviour [31]. This has led to a number of creative approaches attempting to create believable social bots.

An important social element within bots is social relationship. While they may be simply defined as a link between two people, social relationships are clearly identified with additional information including social variables, emotions, and roles. Isbister notes that relationships are in part defined by two primary social variables being agreeableness and dominance [35]. Dominance is tied to the concept of social power, while agreeableness helps define the type of relationship as friendly, unfriendly, and neutral [35]. Our own earlier work in [4] implemented Isbister's social variables through a rule-based emergent social system that allowed for the varied interaction among its bots based on their social variables, emotional state and personality factors.

Social relationships have also been bolstered with quantitative social emotions in order to give feeling to the bots' relationships. The OCC cognitive models [47] have been used to define relationships with variables such as the degree to which a bot likes or dislikes another, the degree to which a bot is familiar with another, and the degree to which a bot has formed a cognitive unit with the other [60]. Several systems have implemented this approach to relationships. Some use this existing information in the causal attribution of blame for an event [28], as well as defining when another is friend or foe. These social emotions may update using a rule-based approach during the appraisal of the event. If a bot is perceived to have caused a harmful event, for example, a rule will convert emotional reaction such as fear to an increased dislike for the bot [27].

Another view of society and social relationships is through the concept of roles. Based on role theory, a role defines the relationship of a person to another person, group or even object [9]. Roles are often thought of as a concept used in theatre, where an actor plays an assigned part [9]. In reality, a role is more intuitively defined, with an individual assuming potentially multiple roles, depending on social context, relative role importance, and a variety of other factors. A role does more than simply formalize a relationship; it also brings together desires (goals), intentions, and beliefs [49]. Applying this to the creation of believable bots, roles provide a formalized description of what is expected from a bot in various social contexts [31, 49]. This provides significant advantages for designers as roles can be reused, and are intuitively understood by non-experts [31, 49]. For these reasons, we ultimately adopt a role-based approach in this chapter for social modeling, as discussed in Sect. 2.4.

2.4 A Framework for Psychosocial Behaviour

The fundamental design of a psychosocial bot behaviour framework is based on the core ideas presented earlier in this chapter. To provide dynamic and unscripted behaviour, this framework relies on various elements of emergence. Recall that emergence focuses on a bottom-up view of the game world, where simple component-level behaviours interact to form complex system-level behaviour. This is done by making every object in the world self-centred, goal-directed, and responsible for its own needs and responses. Emergent systems do not focus on algorithmic behaviours, but rather very simple stimulus responses at the component level [78]. It is important to note, however, some structure to cognitive processes is required for a bot; otherwise, pure emergence can lead to bots that appear to be too reactive or impulsive, with no long-term goals or planning [80].

To extend the ideas behind emergent systems into a psychosocial context, simple psychosocial objects must be defined that can react to psychosocial stimuli and other psychosocial objects, and be able to maintain their own attributes, needs, and responses to the dynamic game environment. Such psychosocial objects and stimuli can include emotions, personality traits, roles, bots, groups, or static objects of import to bots (such as possessions). In this way, simple component-level psychosocial behaviours will be defined that can interact in complex and interesting ways.

With this in mind, we first discuss the modeling of bots from a psychosocial perspective that is necessary to enable the mechanisms for emergence presented. Following this, we then describe the logic and mechanisms required to support emergent psychosocial behaviour.

2.4.1 Psychosocial Modeling

As discussed above, the work of Loyall [40] identified several psychological and sociological elements required to support believable behaviour in bots. Chief amongst these elements were personality, emotion, and some form of social network connecting bots with one another in various social contexts [40]. Consequently, for brevity, we will focus our discussion in this section on these elements, but our approach to modeling is general and applicable to other psychosocial elements as well [80].

2.4.1.1 Modeling Personality and Emotion

Selecting individual existing approaches to psychosocial modeling, as discussed in Sect. 2.3.1, is problematic as there is no single universal approach that captures all characteristics and possibilities to be modeled. As a result, a flexible approach to multi-model integration was adopted in our earlier work in [80], and provides

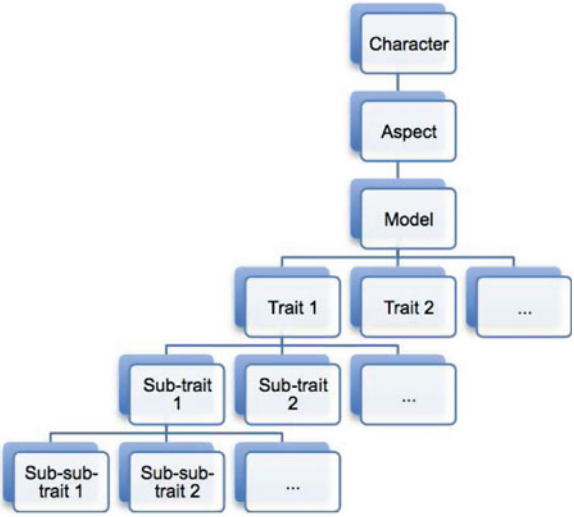


Fig. 2.1 Template for vertical scaling in a model

a method of accessing traits across model boundaries through both vertical and horizontal scaling of models.

The process of vertical scaling in psychosocial models refers to a hierarchical decomposition of the model into a collection of various traits, sub-traits, sub-sub-traits, and so on, typically in a tree or graph like fashion. Higher levels of the trait hierarchy would represent more abstract traits, while lower levels of the hierarchy would contain more concrete or more detailed traits. These hierarchies of traits have been suggested in the literature, but do not appear directly in most of the core models [50, 63, 67].

A template for vertical scaling in a model is shown in Fig. 2.1. At the highest level of the hierarchy is the character itself. Below that would be the aspect of the character being modeled below, such as personality or emotion. Below the aspect would be the model in question, and below that would be the traits, sub-traits, and additional refinements necessary to fully and completely describe the model. Doing so allows designers and developers to explore depth, subtle nuances, and detailed elements of a particular character with respect to a given aspect and model. (As we will demonstrate shortly, this vertical scaling also enables horizontal scaling, and the integration of multiple models with overlapping or otherwise similar features.)

For example, consider the emotion model of Parrott [50] with primary, secondary, and tertiary emotions. A partial set of traits for a particular bot, Alice, using only Parrott's emotion model could be visualized in Fig. 2.2.

Naturally, to define a character and drive believable behaviour from them, we need to assign values to the particular traits. (For instance, if a character is feeling anger, we must specify how angry that particular character is.) Even on this seemingly simple point, the literature in the area is fairly divided, with some researchers favouring

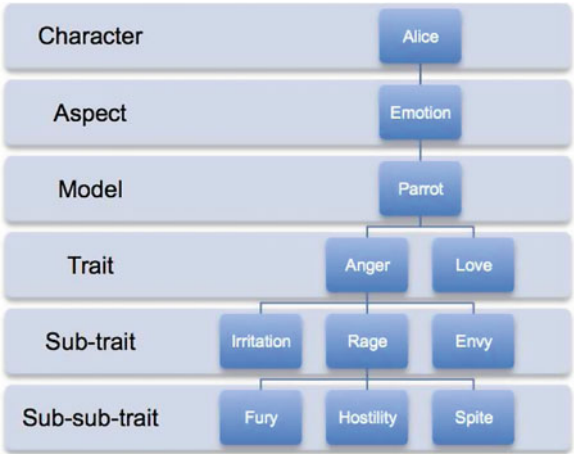


Fig. 2.2 Partial trait model for bot alice

discrete values (such as on or off, or present or not present), and others favouring a continuous scale (such as between 0.0 and 1.0 or −1.0 and 1.0). For our purposes, we have chosen to use a continuous scale between −1.0 (strongly negative) and 1.0 (strongly positive) for primitive trait values, as this would accommodate the majority of other approaches with the least difficulty. (For example, a discrete present or not present trait could be modeled by values of 1 and 0 respectively.) Some models like Reiss' [57] require compound traits with multiple values (such as a set point or target and a current value, in this case), but these approaches can typically be supported using a small set of primitive trait values [80].

To maintain consistency within the model, weights can be assigned to the linkages between a trait and its sub-traits to determine the contribution of each sub-trait towards the parent trait and the distribution from the parent trait to each sub-trait. With these weights in place, adjustments to lower-level traits made manually during production by game designers or developers, or at run-time by the game itself, can then propagate to higher-level traits, and vice-versa.

For example, consider the personality scenario in Fig. 2.3, showing the FFM trait of extraversion from [74] decomposed into just two of the possible sub-traits identified in [63]. In this example, the Outgoing trait accounts for 70 % of the extraversion trait value, while Independent accounts for 30 %. The character in question is very independent (0.9) and quite outgoing (0.8), so we would expect that the character would also be highly extraverted. Given these contributions and the sub-traits in Fig. 2.3, the value for extraversion can be computed as $(0.7 \times 0.8) + (0.3 \times 0.9) = 0.83$.

With this approach to vertically scaling psychosocial models, we can flexibly provide and automatically manage a great deal of depth and detail on the various aspects of non player characters. Through this scaling to sub-traits and sub-sub-traits, and the weightings applied in the process, we can now turn our attention

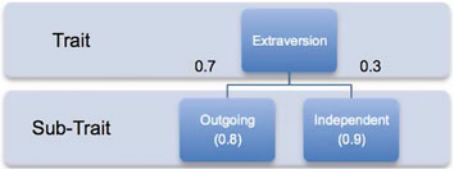


Fig. 2.3 Applying weights to traits and sub-traits

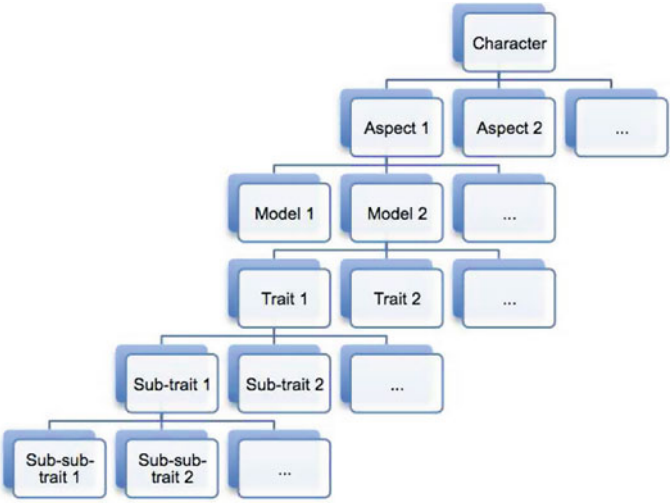


Fig. 2.4 Template for horizontal scaling across models

to horizontal scaling. While vertical scaling of a psychosocial model adds depth, horizontal scaling adds breadth, in this case integrating various models together and providing interoperability. Depending on designer and developer requirements, horizontal scaling can integrate entire models or only certain traits and sub-traits from models. Furthermore, additional traits from outside traditional psychosocial models can also be integrated, and other changes can be made to tailor and tune the resulting model to a great degree.

Conceptually, this can be as simple as the template for horizontal scaling shown in Fig. 2.4, demonstrating the integration of multiple aspects and multiple models for each aspect. When integrating multiple aspects, such as personality and emotion, with only one model per aspect, this is a relatively simple process as the aspects themselves are usually independent and do not contain conflicting traits or other potential issues.

For example, we can perform a relatively simple integration of personality and emotion using the FFM model [74] for modeling personality and Ekman's model [18] for representing emotion. A partial set of traits for a particular character, Bob, integrating these aspects and models could be visualized in Fig. 2.5.

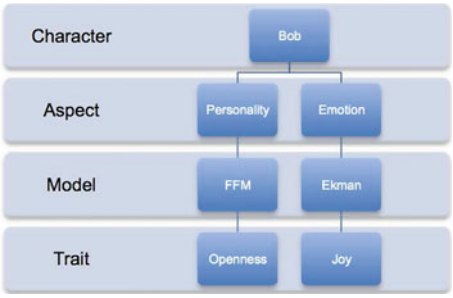


Fig. 2.5 Partial trait model for bot Bob

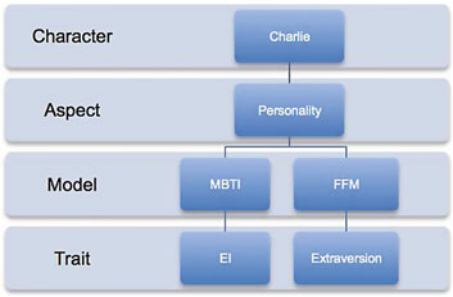


Fig. 2.6 Partial trait model for bot Charlie

The situation gets significantly more complicated, however, when combining multiple models under a single aspect because of the potential overlap and conflict in the models. Consider, for example, the partial set of traits for a character Charlie, whose personality we want to model to leverage elements of both MBTI [44] and FFM [74], as shown in Fig. 2.6. This combination of models can be useful, for example, when character behaviour is guided through FFM traits, and career and professional orientation are determined through MBTI [36, 44].

In the example in Fig. 2.6, both models have traits to represent the concepts of extraversion and introversion. (There are also other sources of overlap and potential conflict between these models, as discussed in [42], but the concepts of extraversion and introversion are the most highly correlated, pose the most obvious problem, and are the easiest to discuss here.) If these overlapping traits are not synchronized with one another, character behaviour could become erratic and inconsistent, thereby destroying any player immersion or suspension of disbelief [11, 39], which is highly undesirable in the game. Without any linkage between traits across models, synchronization is unfortunately difficult and error-prone, as changes and updates made manually by designers and developers must occur multiple times, and those that occur at run-time during the game must occur to all affected traits in tandem. (Doing so is complicated even further when the connections and correlations between traits across models are less obvious and better hidden within the models.)

To solve the above problem and ensure model consistency and believable behaviour when scaling horizontally to accommodate additional psychosocial models, we

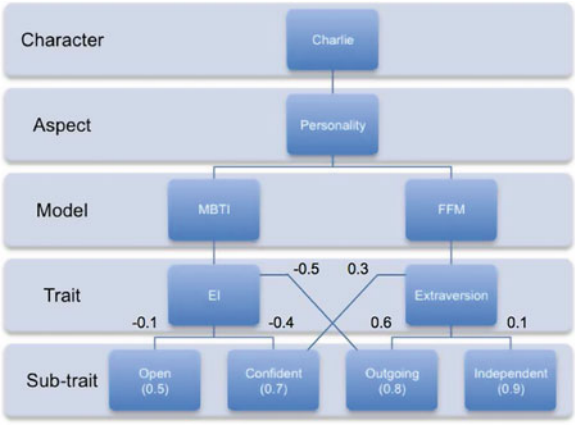


Fig. 2.7 Expanded partial trait model for bot Charlie

can take advantage of the vertical scaling discussed in the previous section. Overlap and conflicts occur between models because of elements in common somewhere in the trait hierarchies of the models. Otherwise, the models would be independent, and there would be no problems or difficulties in the first place.

By hierarchically decomposing model traits into sub-traits, sub-sub-traits, and so on, we can uncover the common elements between the models. Instead of duplicating the common elements in each trait hierarchy, we instead use only a single element with separate links into the higher levels of the corresponding trait hierarchies. By doing things in this fashion, we can take advantage of the common elements to link the various models together and synchronize them with one another.

For example, recall the partial set of traits for the character Charlie given in Fig. 2.6. While there is a strong correlation between the respective extraversion and introversion elements between the MBTI and FFM models used in this case [42], it is not perfect, suggesting that we cannot simply use a common extraversion trait to replace the corresponding traits in each model. Consequently, we can decompose these traits into their respective sub-traits, and potentially lower-level traits beyond that point as necessary. In the process of doing so, we can also assign weights to the linkages between the higher-level and lower-level traits in the hierarchies to indicate relative contribution and distribution, as we did in the previous section. When we do this, we end up with the expanded partial trait model shown in Fig. 2.7. (Note that we are following the same simplified decomposition as in Fig. 2.3 for the extraversion trait of FFM, and that decompositions and weightings were made for illustrative purposes in this example, and do not necessary reflect how scaling should actually be done.) In this example, we can make note of several things:

- Not every sub-trait needs to be linked to both trait hierarchies. This is only logical, since different models focus on different things. In this example, the sub-trait Open

is only associated with EI of the MBTI model, and the sub-trait Independent is only associated with Extraversion in FFM.
- The weights assigned to links between the traits and sub-traits differ between the two models. Again, this makes sense since different models emphasize and consider traits differently.
- The weights assigned can be negative, as was done with sub-traits linked to the EI trait of the MBTI model. In this case, doing so makes sense, since strongly negative values of EI indicate a highly extraverted individual and strongly positive values indicate a highly introverted individual.

With trait hierarchies linked in this fashion, the models can remain synchronized and consistent. Additional models and traits can be added for further horizontal scaling, building additional linkages as necessary.

When performing horizontal and vertical scaling, it is important to base decisions in hierarchical decomposition, linkages, and weight assignment on scientific study, such as the work in [42, 63, 67], and the literature in this area. Even a rigorous approach, however, will not be able to handle all integration scenarios, as all the background research to do so does not exist, or has yet to be completed. In such cases, we must use our best judgment given the research results that are available, but leverage the flexibility and openness in our approach to allow easy changes, revisions, and additions in the future.

2.4.1.2 Modeling Social Information

As discussed in Sect. 2.3.2, we adopt a role-based approach to social modeling in this chapter. As noted in [31], there is little to no standardization in social models in the literature, and so we initially developed our own model for roles in [2].

Roles, in essence, are used to clearly define the relationship of a bot to its environment, including the player, other bots, and other things in the world around it. As such, they are used to contain the social knowledge used by the bot in its decision making processes. This includes desires, beliefs, and intentions as in other work [31, 49] as well as role personalization through reference to emotional state and personality [2]. This personalization enables prioritization of roles based on social context and bot psychology. Unlike other approaches, multiple roles can be attached to an individual, even conflicting roles, as such conflicts can be resolved using elements of personalization and prioritization. Consequently, roles are dynamic and can be added, removed, changed, disabled, and re-enabled over time on a bot-by-bot basis.

Figure 2.8 presents the key components of a role. Each of these components, as well as other meta-information required to support a role, is discussed below in further detail.

Name: The name of the role. Simply, it is a label used to refer to a role, and is usually human-friendly to allow designers to more readily make decisions about its attachment to and usage by bots in the game. For example, one role could be "Mother".

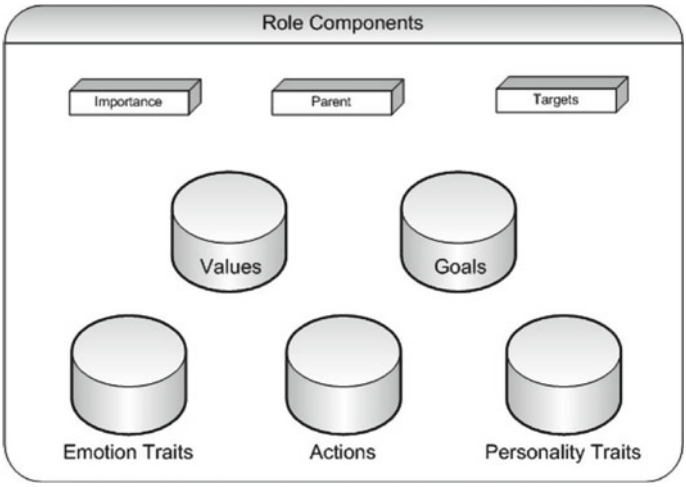


Fig. 2.8 Components of a role

Bot: The bot to whom this role is attached. This is assigned when the role is added to the bot and allows the role to tap into and access other bot-specific information and internal state. Continuing the above example, any female bot with children could have the "Mother" role attached to it, and this element of the role refers back to the appropriate "Children" bots.
Target: Who or what are the targets of the role. Typically this will be another bot, a group, or even the bot itself. The target can be any other object within the world. Targets are assigned to the bot during the role's instantiation. Continuing the above example, other bots designated as children of a female bot would be the targets in that bot's "Mother" role.
Goals: Goals are used to capture desires associated with a role. In essence, a role's goals are world states achievable through actions that can be taken by the bot. Typically, although not always, goals are related to the targets of the role either directly indirectly. Continuing the above example, the mother in the "Mother" role could have a goal stating that the targets (her children) are in a happy emotional state.
Values: Each role can also be composed of values or beliefs about the world that produce emotional reactions to events and generate information relating to the consequence of actions. Values, norms, or a world view make up a bot's belief system and play an important part in many decision making processes including appraisal and planning. Continuing the above example, the mother in the "Mother" role could have a belief that she should not physically strike her children in disciplining them.
Emotion Traits: A role's emotions define how the bot feels towards the role's targets. These emotions play a critical role in defining how a bot will interact with the target, and also contribute towards the bot's general mood, which govern how it behaves in general. Continuing the above example, the mother in the "Mother" role could feel love for her children, and happiness to be with them. Not only does this impact

her actions with her children, but her happiness will carry over into her mood and interactions with other people. Likewise, the removal of her children will decrease her happiness, and their absence could have a noticeable impact on her mood and interactions with others.

Personality Traits: A role may need access to the attached bot's personality, as these traits could impact decision making processes involving the role. Furthermore, similar to emotions above, there may be alterations of personality traits imposed by the presence of a role that might not otherwise occur naturally in the bot. Continuing the above example, a mother in the "Mother" role could have a heightened sense of responsibility and a more caring demeanour. (Of course, she might not, which could lead to interesting consequences.)

Actions: The actions defined within the role are role specific actions used during the planning process to create a plan to meet the goals of the role. Actions will use the role's emotions and personality traits to modify their utility during the planning process. Continuing the above example, a mother in the "Mother" role could have access to actions to feed and clothe her children, which are likely not present in many of the other roles in the game.

Importance: The importance of the role to the bot defines, at a high level, how critical the role is to the agent. This importance impacts goal and action selection of the bot, ultimately determining its behaviour. It also influences how events associated with the role are committed to emotional memory, and how the emotions of the role impact general mood. Continuing the above example, the mother in the "Mother" role could also be an employee in an "Employee" role and have to balance the needs of her job against the needs of her children. The importance of each role determines the balance. If her "Mother" role is more important, she will sacrifice aspects of her job in support of her children. Likewise, the happiness derived from her children will outweigh negative feelings that come from a hard day of work. Role importance is dynamic, and can be influenced by emotion, personality, social context, and other factors.

Parent: Roles can be hierarchical in nature, with child roles inheriting aspects of parent roles. This allows for a more modular, reusable role structure. Continuing the above example, the "Mother" role could have a "Parent" role as its parent. Social knowledge and information common to both a "Mother" and a "Father" role could be encapsulated in this "Parent" role, with only specific differences appearing in the particular child roles.

2.4.2 The Mechanics of Psychosocial Behaviour

With models for psychological and sociological elements defined in the previous section, we can now turn our attention to using these models to create believable psychosocial behaviour in bots. In this section, we present two mechanisms that work in tandem: a stimulus-response system to produce reactive actions, and a

goal-oriented utility-based planning system to produce proactive actions. These systems are derived from our earlier work in this area [2, 4, 56, 80].

2.4.2.1 A Stimulus-Response System

As mentioned above, one form of action that is required as part of believable bot behaviour is the reactive action, or simply reaction. In this case, the bot is responding to a stimulus in the environment and carrying out an action with minimal or no formal planning involved in the process. To support this, a stimulus-response system can be used. We created one using principles of emergence in our earlier work in this ares [4] and then extended this approach in [80].

Before delving into the specifics of the stimulus-response system, it is necessary to understand how emergent systems operate. Emergent systems use simple component-level behaviours that can interact to form complex system-level behaviour [66, 78]. These component-level behaviours operate as stimulus-responses, in that every entity in the world monitors its own perceptions of stimuli and are responsible for responding to those stimuli [78]. Such systems have been used in games before [66, 76] for handling interactions between physical objects and physiological needs, but not for complex psychosocial modeling, as discussed in this section.

Central to our approach is the generalized stimulus-response system shown in Fig. 2.9. This system operates by having objects create and listen for stimuli. By defining a general channel that allows only certain objects to interact with each other through a stimulus that is created and responded to, we have an easy to use mechanism for defining emergent interactions.

The object or event that begins the interaction by creating a stimulus is referred to as the Actor. We will refer to the object that detects and responds to the stimulus as the Reactor (which may then become an Actor through its response). The stimulus acts as a message that is broadcast from the Actor and is received by the Reactors in the area. Reactors listen for messages with particular stimulus types, since each object will react to different stimuli. The stimulus messages can also hold data about the

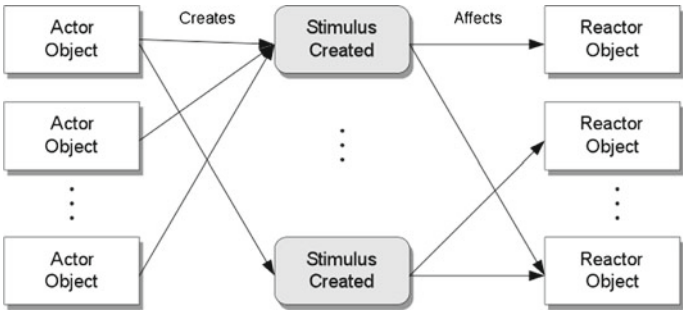


Fig. 2.9 A generalized stimulus-response system

attributes of the associated stimulus. These attributes can be mapped onto different behaviours; it is important to note that Reactors do not have to share the same response to a given stimulus.

For a bot to react in a believable fashion, they should have the ability to react to the actions of other bots, seeing bots or objects (i.e. seeing an enemy upon turning a street corner), as well as other events or occurrences in the world, such as natural disasters or accidents. All of these antecedents could be considered the Actors that cause some sort of stimuli. Since these events are to be affected and be reacted to in a psychosocial context, the stimuli created by these events would be psychosocial in nature, such as emotional stimuli (i.e. a witnessed attack may cause an emotional stimulus of fear) or a stimulus that causes a perceived change in situation (i.e. the same witnessed attack may cause the bot to feel the situation has become hostile). Psychosocial stimuli are created by Actors as messages and are broadcast to the social objects in the area, as done in Fig. 2.9. These messages hold data about the psychosocial stimulus, such as stimulus source, type, magnitude, propagation, and fall off.

Stimulus source refers to the origin of the stimulus. Stimulus type refers to various emotional and social stimuli, such as happiness, sadness, anger, hostile environment, and groupthink. Magnitude of the stimulus would likewise describe the magnitude of the psychosocial effect of the stimulus (i.e. was this a highly emotional event?). Propagation would describe whether the psychosocial stimulus only affects the people directly involved in the event (such as two people greeting each other), if it affects the people witnessing it at a distance as well (such as two people attacking each other). Other types of social stimuli can also be modeled using these stimulus messages, such as gossip and the spread of information. Social stimuli over long distances (such as gossip through a telephone) could be sent as a direct message from one person to another.

It is useful to note that a social fall off radius could serve as a way of simulating social phenomena. By defining the psychosocial stimulus to propagate by radius and defining a fall off rate, an entire group of people can be affected by the behaviour of a few. This may assist in modeling social behaviour such as riots or group-induced panics. While this may not be an accurate model, it can simulate reactive behaviour that appears to be life-like.

The Actors can be anything that broadcasts psychosocial stimuli, including bots, objects, and events. The psychosocial stimuli are broadcast to all other social objects in the nearby area. The social objects that are receiving these broadcasts are the interaction Reactors as described above. These Reactor objects can be any social object that can react, including bots or other social beings, such as groups. Note that non-responsive social objects (such as possessions) will not react to psychosocial stimuli in any way and therefore it would not be desirable to include these objects as Reactors. It may sometimes be desirable to have more general emergent gameplay, by having a stimulus-response system that processes both physical stimuli, and psychosocial stimuli simultaneously. In these cases non-socially-reactive objects may be treated as Reactors that are only listening for physical, not psychosocial stimuli.

Unlike some of the Reactors described above, social objects, in general, react to, or at least detect, psychosocial stimuli of all types. People tend to react to all emotions, any social situation or stressors they find themselves in, and participate in (or at least detect and make a decision in response to) social phenomena that are occurring around them. In cases where these reactions do not take place, it is because the person (or social object) has gone through some decision making process to decide how (or how not) to react. This process will require the detection and processing of the stimulus, regardless of its type. Therefore, social objects listen for all types of psychosocial stimuli. However, social objects will typically ignore events that are not relevant to them (i.e. a discussion between two strangers). Therefore social objects can listen for psychosocial stimuli with particular relevance types; some may only be relevant to individual social objects or groups of social objects, while others may be relevant to all.

By defining a psychosocial mechanism in the way described in this section, bots and other social objects can react to psychosocial stimuli in realistic as well as unexpected and novel ways to their environment. This can result in chain reactions as reactions to psychosocial stimuli create behaviours or events that can in turn cause more psychosocial stimuli that can be reacted to. This would achieve the goal of having emergent social behaviour in bots. By allowing the game player to participate in this mechanism through their actions in the game world, this enables meaningful player decision making and emergent gameplay.

Given the potential of this approach, it is also important to recognize its limitations. Bots equipped only with this stimulus-response system can only react to what is going on around them. They would come across as overly reactive or impulsive, with no long-term goals or planning [80]. In other words, they would not be very believable for very long. When something of import is going on, the bots could behave quite well, but in the absence of significant stimuli, the bots would be aimless and directionless. A solution to this problem comes in the form of proactive action, as discussed in the next section.

2.4.2.2 A Goal-Oriented Utility-Based Planning System

Given the limitations of a purely reactive stimulus-response system, as discussed above, a new approach was taken to introduce goal-oriented planning [80] to bot decision making processes. Doing so allows bots to be more proactive in their actions, carrying out actions to move themselves towards accomplishing their goals. This approach was then extended to support utility-driven psychosocial behaviour and appraisal theory [2, 56].

At the core of this system is the decision making process illustrated in Fig. 2.10. While more structured than the stimulus-response system of Sect. 2.4.2.1, this process is still an emergent one, with complex results arising from relatively simple interactions that occur during the process. The various elements of this process are discussed below in further detail. It is important to note that this system can either work on its

Fig. 2.10 A goal-oriented utility-based planning system

own, or together with the stimulus-response system of Sect. 2.4.2.1 to have reactions handled through that mechanism instead of through planning.

Event: A notification of activity in the game world, whether it is from other bots, the world itself, or simply the passage of time. When the planning system is connected to the stimulus-response system of Sect. 2.4.2.1, these events are Actors which could lead to the generation of stimuli upon appraisal.
Appraisal: The event is examined to see if it is of interest to the bot, and if so, does the event or its consequences have sufficient importance to the bot to warrant further consideration. Deadlines for action/reaction may also be set, depending on the event. When the planning system is connected to the stimulus-response system of Sect. 2.4.2.1, appraisal flags events requiring reaction to create stimuli for the stimulus-response system after coping has been completed.
Coping: The bot reflects on the event and updates its internal psychosocial and physiological state based on the event. This adjusts the bot's mood, emotional memory, and so on. When the planning system is connected to the stimulus-response system of Sect. 2.4.2.1, any events flagged by appraisal to create stimuli are sent to the stimuli-response system. The planning system continues to operate in parallel as goals and plans may also need to be adjusted in response to this event, in addition to any reactive actions generated by the stimulus-response system.

Goal Selection: Given the current state of the bot, its surroundings, its social context, and recent events, the bot determines the goals that it should be trying to meet. Goals might be immediate, short-term, medium-term, and long-term. Goals are largely selected from roles attached to the bot according to current relevant role importance, as discussed in Sect. 2.4.1.2 but could be derived from other sources as well.
Planning/Action Selection: Considering the goals in place, a plan is created to achieve the goals with the desired results and maximum utility, with acceptable side effects and consequences. If a plan already exists and is in process, it might be revised to reflect changes in goals, bot state, and so on. With the plan in mind, appropriate actions are selected to have the plan implemented. Note that if events indicated a reaction is required, as described above, the current plan may be temporarily bypassed or abandoned to carry out alternate actions. (This may be the case, for example, to ensure self-preservation.)
Output Events: Based on the actions selected, new events are generated and propagated into the game accordingly.

With these systems designed and in place, we can now examine how we can implement and provision these systems for use in creating believable bots.

2.5 Implementation and Results

Proof of concept of the framework of psychosocial models and systems presented in Sect. 2.4 has been completed through the creation of a series of prototype implementations, each of which capable of supporting believable psychosocial characters of varying degrees of complexity. Such characters would be suitable for story-driven role-playing or action-adventure games, or other types of games as discussed in Sect. 2.1. In this section, we discuss the creation of these prototypes and briefly highlight results obtained through their use.

2.5.1 First Generation Prototype

The first generation prototype was created as part of the work described in [4]. It focused primarily on the creation of the emergent stimulus-response system discussed in Sect. 2.4.2.1 based on a simple psychosocial model that was a precursor to the modeling approaches described in Sect. 2.4.

Microsoft Visual Studio was used to develop the prototype on the Microsoft Windows platform. To promote portability of the code, the prototype was developed in C++ using OpenGL to render a simple graphical representation of social objects, the social ties between them, and their moods. A limited character state was implemented including a categorical representation of emotion, a simple personality

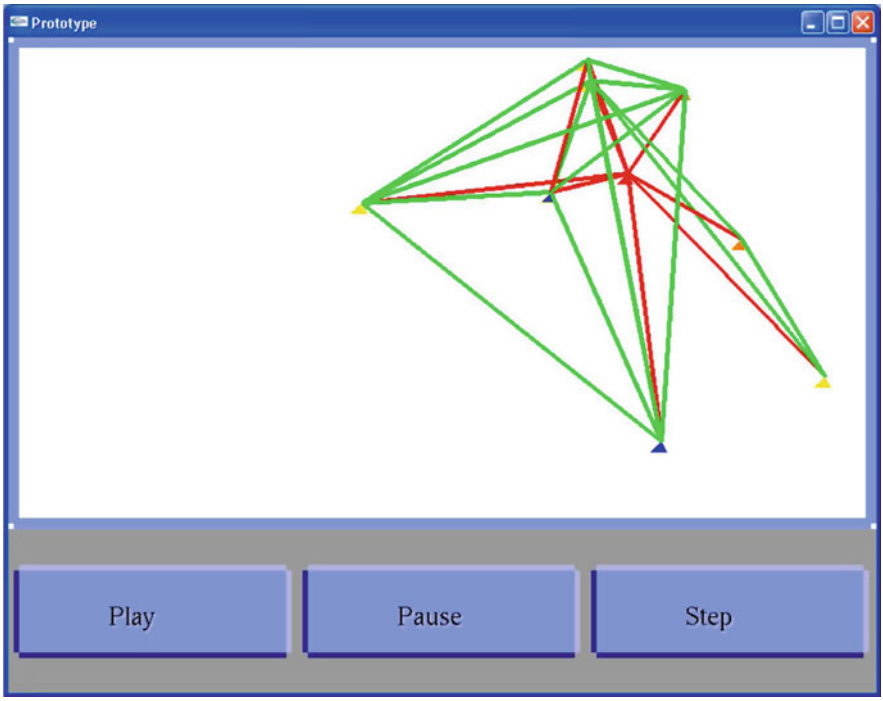


Fig. 2.11 Screenshot from first generation prototype

model, symmetric social ties, and a small behaviour/stimulus set. These elements provided sufficient proof of concept for initial validation and testing.

A screenshot from the first generation prototype system is given in Fig. 2.11. The system provided feedback on bots in the system and their relationships with one another through a set of colour-coded nodes and edges between them.

A series of experiments was conducted with this system. While it was found that bots would have believable reactions to one another in the system [4], the limitations of a purely reactive stimulus-response system were evident and it was clear that a more advanced system was necessary [80].

2.5.2 Second Generation Prototype

The second generation prototype was created as part of the work described in [80]. Building upon the first generation prototype system discussed in Sect. 2.5.1, this prototype was aimed at improving upon its predecessor in several ways. From a modeling perspective, it supported the multi-model integration discussed in Sect. 2.4.1.1 as well as a simple role-based social model that was a precursor to the model discussed in

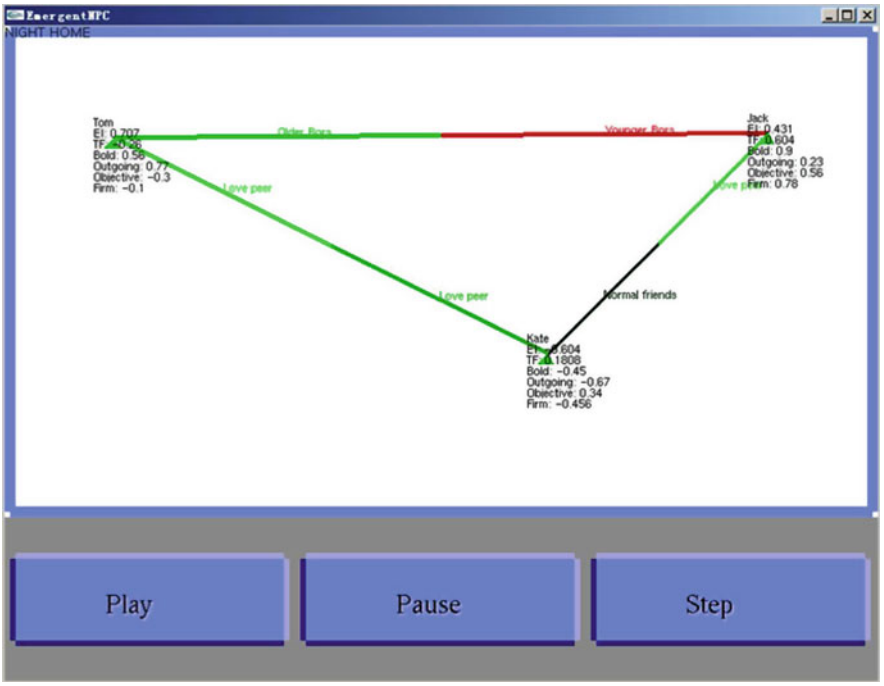


Fig. 2.12 Screenshot from second generation prototype

Sect. 2.4.1.2. This prototype also supported goal-oriented planning that was the first step towards what would become the goal-oriented utility-driven planning system from Sect. 2.4.2.2.

Like the first generation prototype, the second generation prototype system was developed for the Microsoft Windows platform, written in C++ using Microsoft Visual Studio, with OpenGL used for interface development. This prototype system also introduced a collection of management tools to help designers develop and work with psychosocial models. These tools were created for the same platform, using C# as a language instead, because of its rapid prototyping capabilities. Screenshots of these aspects of this prototype are provided in Figs. 2.12 and 2.13 respectively. Note that the interface in Fig. 2.12 has been augmented to provide additional feedback and data in comparison to its predecessor, while its overall functionality remained quite similar.

A series of experiments was also conducted with this system. Results were, in fact, improved over the previous system [80], and the system benefited from the more flexible approach to psychosocial modeling and the addition of planning. Bots were more flexible and more expressive in terms of their emotion and personality, and the use of goals and planning enabled bots to act more thoughtfully and intentionally, instead of in a simple, reactionary fashion. It was found, however, that the system

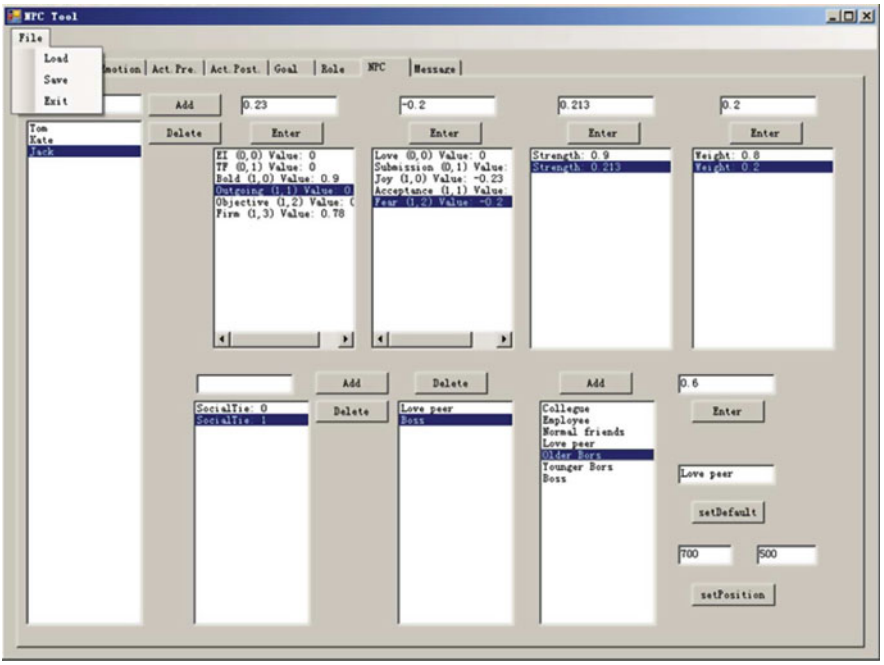


Fig. 2.13 Screenshot from second generation management tools

could be improved through an improved role model and a more robust planning system, which led to the creation of a third generation prototype.

2.5.3 Third Generation Prototype

The third generation prototype was created as part of the work described in [2, 56]. Improving on its predecessors, this prototype now supported a richer and more complete model of social roles, as discussed in Sect. 2.4.1.2, as well as the goal-oriented utility-driven planning system outlined in Sect. 2.4.2.2. This allows bots to make decisions and select courses of actions based on how well those actions satisfy their goals, taking into consideration the positive and negative consequences of doing so. (For example, an action might be very useful to satisfying a goal, but have very negative side-effects, making that action less desirable than one with less utility but no negative consequences.)

This prototype system was once again developed for the Microsoft Windows platform using C++ as part of Microsoft Visual Studio. This generation of the prototype system employed a logging system to report on the status of bots during testing

instead of a graphical interface, because there was now such a volume of information that the graphical approach grew too cluttered and difficult to use.

A series of experiments was also conducted using this prototype, to great success. By this stage in development, the prototype could re-enact segments of Shakespeare's Romeo and Juliet quite believably. Bots playing the characters from the play were given personalities and emotional states based on an analysis of the original play, and a social structure was put in place reflecting the familial struggles taking place. With these models in place, various scenarios were played out, producing similar results to the original play [2].

Consider, for example, one of the most pivotal moments in the play, Act 3, Scene 1, as the outcome of this scene ultimately leads to Romeo's exile and the suicide of the lovers. In the scene, Romeo and Mercutio are socializing when the sworn enemy Capulet cousin Tybalt enters. Mercutio and Tybalt fight, and Romeo is faced with a dilemma: should he fight the sworn enemy of his family or save his lover's cousin? In an attempt to save both lives, Romeo separates the two by putting himself between them. It is then that Tybalt slays Mercutio. Seeing the death of his best friend, Romeo is overcome with rage, and having lost all care of consequence, slays Tybalt and sets in motion the tragic events in latter parts of the play.

With appropriate psychosocial models in place to reflect the characters in the scene [2], we set the bots in motion using the prototype system and observed the unscripted results as recorded by the logs. The beginning of the scene can be viewed below in an excerpt of the log file's events. (Note that the actual logs are much more detailed, showing planning processes, utility calculations, consequence calculations, and so on at each step, but these details are only shown where necessary below for brevity.)

```
1) Event: Agent: Mercutio Action: talk Target: Romeo
   Utility: 0.89375
2) Event: Agent: Romeo Action: talk Target: Mercutio
   Utility: 0.7
** Enter Tybalt **
3) Event: Agent: Tybalt Action: Threaten_Yell Target:
   Mercutio Utility: 0.85125
```

As expected, Romeo and Mercutio converse until Tybalt enters the scene and threatens Mercutio, as the two are enemies according to their social roles and dislike for each other. During the appraisal here, Romeo adds an enemy role towards Tybalt in response to Tybalt's threat to his friend.

```
4) Event: Agent: Mercutio Action: Threaten_Yell
   Target: Tybalt Utility: 0.849304
5) Event: Agent: Romeo Action: talk Target: Mercutio
   Utility: 0.6915
6) Event: Agent: Tybalt Action: Threaten_Yell Target:
   Mercutio Utility: 0.850875
```

In the second log file excerpt, Mercutio angered by Tybalt's threat returns the favour that in turn causes Romeo to adjust his view towards Mercutio. Nonetheless,

Romeo continues to talk with Mercutio and Tybalt continues his verbal assault on Mercutio.

```
7) Event: Agent: Mercutio Action: Attack_Sword Target:
   Tybalt Utility: 0.731439
8) Event: Agent: Romeo Action: Defend_Separate Target:
   Tybalt Utility: 0.64784
9) Event: Agent: Tybalt Action: Attack_Sword Target:
   Mercutio Utility: 0.729146
*************** Mercutio Has Died ******************
```

Looking at event seven, we see that Mercutio has loosed his sword in response to Tybalt's threats. (As an interesting note, we had to lower Mercutio's sword skills in this scene. Otherwise, he was just as likely to kill Tybalt instead, which would lead to a very different, although still realistic, retelling of the scene.) We now have a pivotal moment for Romeo whose internal struggle is apparent with his planning process. Despite the tense situation, attacking either Mercutio or Tybalt is not an option. As we see below for the utility calculations for a strike against Tybalt, the consequence utility is zero, meaning that this action has very negative consequences. While Romeo views Tybalt as an enemy for his threats against his friend, he cannot bring himself to fight because of the consequences and his love for Juliet. So, even though there is equal utility to fighting and separating the two combatants, the poor consequences for fighting cause Romeo to opt for separating the combatants instead, as there are the consequences are better (positive) for doing so.

```
Action Selection: Attack_Hand Targets: Tybalt
                  Utility 0.64784
State: 0.64784
Consequence: 0
```

So, Romeo attempts to separate the two combatants, as this is the best option available. Tybalt, responding to Mercutio's threat, attacks and kills Mercutio. The result here is extreme anger in Romeo. Examining his state after the appraisal of Tybalt's killing blow, we find rage in Romeo. Below shows his logged anger emotion both before and after Mercutio's death.

```
Emotion: anger Value: 0.0387244 (Before Event 9)
Emotion: anger Value: 1 (After Event 9)
```

Being completely enraged, Romeo's planning reflects this state. His care of consequence drops to zero given the extreme emotions of the situation. (Using consequence values during planning is controlled by a care of consequence variable. In highly charged emotional states, people may care less about the consequences of their actions, and so this variable is tuned in bots according to certain emotional states. This allows them to care about the consequences of their actions or not, depending on their state, as Romeo does in this example.) As a result, attacking Tybalt becomes an option, because he is ignoring the negatively appraised consequences of such an action.

```
* Care of Consequence: 0
Target Watch Triggered: Tybalt
Action Selection: Attack_Hand Targets: Tybalt
                  Utility 0.74575
State: 0.62575
106
Consequence: 0.12
Target Watch Triggered: Tybalt
Action Selection: Attack_Sword Targets: Tybalt
                  Utility 1.02075
State: 0.75075
Consequence: 0.27
10) Event: Agent: Romeo Action: Attack_Sword Target:
    Tybalt Utility: 0.75075
11) Event: Agent: Tybalt Action: Attack_Sword Target:
    Romeo Utility: 0.689941
*************** Tybalt Has Died ******************
```

Therefore, Romeo in his fury slays Tybalt, thus ending the dispute and the scene. By examining this scene, Romeo's actions appear to be believable. Not only did we see restraint in his initial actions, action utilities, care of consequence calculations, and appraisal of events, but we also saw the raw power that emotion can have on the planning process. This resulted in the believable, unscripted re-enactment of Act 3 Scene 1 of Shakespeare's Romeo and Juliet.

Numerous other scenarios have been enacted using this system, also to good results. For brevity, these results have been omitted here. The reader is urged to consult [2] for further examples and details.

2.5.4 Fourth Generation Prototype

At present, we are currently working on the fourth generation prototype system. Functionally, the system will have the same psychosocial framework and capabilities as its predecessor. This prototype, however, is being developed using the Unreal Development Kit (UDK) [21]. In doing so, we can tap into a fully-functional 3D environment, and develop a system that can be readily deployed in modern video games.

In addition to the psychosocial framework, we are developing an environment in UDK, a house, for experimenting with various social scenarios. A screenshot of this environment is given in Fig. 2.14.

Fig. 2.14 Screenshot from fourth generation prototype (in development)

2.6 Performance Optimization

While we have made great strides towards the creation of more believable bots for modern video games, computationally, these bots are orders of magnitude more complex than traditional approaches to bot Artificial Intelligence. Consequently these bots introduce the potential for serious performance problems, especially when there are a great number of them inhabiting a game world [4]. This problem is only exacerbated by the computational needs of other game sub-systems, which together put an even larger strain on often limited and over-taxed system resources. As a result, if we are to truly take advantage of believable bots in modern video games, we must take into consideration issues of performance and scaling with Artificial Intelligence as it has been done with other aspects of games, such as graphics [56]. While others have examined similar problems in the past [79], this has not been done for bots with psychosocial behaviour, which presents unique and interesting challenges and opportunities.

This chapter examines a scalable approach to believable bots that utilizes several techniques to improve in-game performance, derived from our work in [56]. These include dynamic importance calculation, capability scaling or reduction, and pre-emptive, priority-based bot scheduling and dispatching. Through utilizing these techniques, we can ensure that scarce computational resources are allocated to bots where they are most needed, and that bots are using the decision-making processes best suited to their current state and importance to the game, while still maintaining believability.

2.6.1 Approach to Performance Optimization

We begin by recognizing that not every bot in a game has equal importance to the player and the game at any point in time. Some bots are the focus of attention, are engaged in significant activities, or are otherwise critical to what is unfolding in the game. Others are of less importance, and their activities will largely, if not completely, go unnoticed to the player. As the player moves through the game world and plays the game, the importance of the various bots changes, with some becoming more important and others less so.

Furthermore, we note that bots that are not visible to the player, or otherwise not important to the player, do not need the same richness, detail, and fidelity in their behaviour as those that are the focus of attention, or are otherwise important, to maintain believability. In those cases, a game can safely do less with those bots, in many cases significantly so, with no perceptible difference to the player's experience. For example, suppose we have two bots in a game that are hungry; one, Alice, is in the same room as the player, while the other, Bob, is in a different locale far across the game world, outside of the player's ability to sense. To maintain believability, Alice must recognize her hunger, formulate a plan to satisfy her hunger considering her current psychosocial and physiological state and surroundings, and then execute this plan. After all, the player is in the same room and is able to see and hear everything she does; the slightest out of place action could sacrifice believability and adversely affect the player's experience. On the other hand, Bob would not need to do anything except simply continue to live to maintain believability. In the player's absence, it is likely reasonable to believe that he could provide himself with the necessities of life, without actually needing to do anything about it. In fact, the game would not even need to track Bob's hunger level to maintain the same level of believability. (That said, Bob would still need to progress in his life in the absence of the player, as it is likewise unreasonable to believe that he would do absolutely nothing at all when the player is not around. How this should be done to maintain believability is discussed below.)

With this in mind, we can safely scale or reduce the capabilities of a bot according to their current visibility and importance to the player while still maintaining believability across all bots in the game, as shown in Fig. 2.15. In doing so, we can achieve tremendous performance savings in bots with reduced capabilities as they require significantly less processing and this processing is far less complicated than bots operating at full capacity. This, in turn, allows the game to support a much larger number of bots without requiring additional resources to be dedicated to Artificial Intelligence processing.

To support this approach, each bot in a game is a separately schedulable and executable entity, enabling computational resources to be allocated by a scheduling and dispatching subsystem on a bot-by-bot basis according to their importance. Furthermore, each bot has access to a range of decision making and processing algorithms, allowing their capabilities to be scaled or reduced based on their importance. Lastly, bot importance is calculated and updated regularly based on a variety of factors to

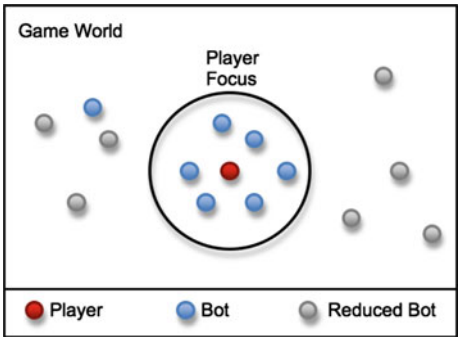


Fig. 2.15 Bots in the game world

ensure that scheduling, dispatching, and capability adjustment are all carried out using the best available information. Each of these activities is discussed in further detail in the sections that follow below.

2.6.1.1 Scheduling and Dispatching of Bots

To allocate computational resources to bots in a game, a scheduling and dispatching subsystem is added to the game. The goal of this subsystem is rather simple—choosing the most appropriate characters to run at each game tick or frame of game execution.

Producing an optimal schedule for each tick is itself a computationally expensive task, and so we take a simpler, more heuristic approach using a system of priorities assigned to each bot in the game. (Priorities are derived from the perceived importance of the bots, as discussed earlier in this section, and below in further detail.) With this in mind, the most appropriate bots to execute in any tick are simply those with the highest priorities. Resource starvation is averted in bots of low importance through an aging mechanism built into the calculation of priorities.

Each game tick, the x bots with highest priorities are selected to run, where x is a positive integer configurable and tuneable at run-time, based on a number of factors including the number of available processor cores, the workload being generated by other game subsystems, and so on. Each of the x selected bots is allocated one or more update cycles, depending on their relative importance. Each update cycle allows a bot to execute for a period of time. This execution can be pre-empted, and does not necessarily allow the bot to complete the task on which it was working. If the task is not completed when the bot's update cycle ends, the bot is paused and its priority is adjusted to reflect the work in progress.

Depending on the number of bots in the game, the priority system could be realized as either a single list sorted by priority, or a series of priority queues. While the mechanics of each approach is different, they can both deliver the same pre-emptive, priority-driven scheduling and dispatching of bots in a game.

2.6.1.2 Capability Scaling or Reduction

As discussed earlier, not every bot in a game needs the highest levels of richness, detail, and fidelity in their behaviour in order to be perceived by the player as believable. Indeed, some bots could function believably with greatly reduced capabilities, depending on the state of the player, the game, and the various bots within it.

When the goal-oriented utility-driven planning system from Sect. 2.4.2.2 is used, scaling a bot can be accomplished in several ways. Generally, these methods can be grouped into either data reductions or processing reductions.

The purpose of data reductions is to limit the amount of information that is used in the various stages of the decision making process used by the bot. When done properly, this can greatly collapse decision spaces into smaller problems, making them far more efficient and less costly to work with. With care, these reductions can be carried out with little to no perceivable change in bot believability. Data reductions can themselves take many forms.

One such form is a model or state reduction. Recall from Sect. 2.4.1 that every bot has a psychosocial model associated with them, defining their personalities, emotions, relationships, roles, values, and so on. A fully populated model could have a bot's state defined by dozens of traits, all of which need to be maintained, and all of which could affect decisions made by the bot. While rich, expressive, and useful during design to fully define a bot, this can be very expensive computationally to use at run-time, as these factors must be consulted during appraisal, coping, goal selection, and planning/action selection. A careful reduction of this design model to a few key traits can produce a run-time model that is orders of magnitude more efficient, and far less costly to use within a game, as depicted in Fig. 2.16.

Another form of data reduction is event reduction. In this case, the number of events used to trigger decision making or provide information during decision making is limited, by reducing the types of events of interest, the importance of the various events or their consequences, or the frequency of their reporting to the bot. This results in either fewer events reaching the bots, fewer events moving past the appraisal stage (as the events are now deemed irrelevant, unimportant or inconsequential), or simpler

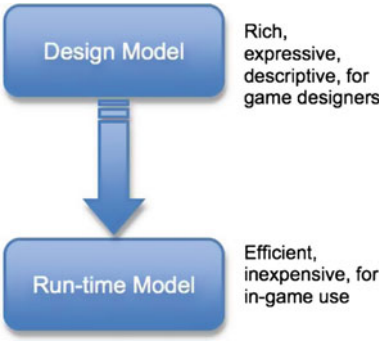


Fig. 2.16 Bot model reduction

decisions throughout the various stages of processing with fewer variables and factors to consider. All of these alternatives have the potential to improve performance substantially.

Processing reductions, on the other hand, generally involve altering a bot's decision making processes by changing algorithms or omitting aspects or complete stages of decision making to improve performance. Again, if done with care, these performance improvements can be achieved without sacrificing believability. Possible processing reductions include the following.

Use of Defaults: To accelerate the various stages of decision making, default strategies can be used instead of analyzing and developing them dynamically on demand. For coping, events could have default impacts on bot state, instead of determining this impact from the current state and other factors. For goal selection, bots could be assigned default goals to meet instead of developing them from scratch. For planning and action selection, default plans could be provided for each possible goal, complete with prescribed actions so that they do not need to be developed at run-time. While the defaults used will not do as well at reflecting the current state of bots or the game, the performance savings can be substantial even if this approach is used only for out of focus bots, or those that are unimportant.

Use of Randomization: Much like through the use of defaults, randomization can be used to replace various aspects of behaviour, although doing so likely makes sense only with unimportant bots in the game. It is likely also wise to constrain randomness to ensure that bots are still acting believably. This can be accomplished in a variety of ways, such as constraining randomness to choose from a pre-defined set of defaults, or by permitting some level of initial processing to develop sensible options that are chosen from randomly, instead of using a more expensive algorithm to make a more optimal choice.

Streamlining of Coping: Integrating the impact of events into a bot's internal state can be expensive, especially with a complex psychosocial model in use. To improve performance we can perform coping only in response to a limited set of critical events when bots are outside of the focus of the player. While this will result in bots whose moods and emotions change only in extreme circumstances, the player will be largely unaware of this.

Disabling of Goal Changing: Whenever a bot changes goals, any existing plan must be discarded and a new plan must be formulated, which can be quite expensive. To improve performance, bots can be prevented from modifying their goals while a plan is executing to avoid re-planning, unless very exceptional circumstances arise. While this will result in bots sticking with plans when they should likely be changed, this should not have too large an impact on their believability, provided that they are outside the focus of the player.

Increased Reliance on Reaction: When the stimulus-response system from Sect. 2.4.2.1 is used with the planning system, we can take advantage of this system to reduce load induced by the planning system in certain circumstances. For example, in a heated moment with a considerable amount of activity going on, such as a battle, the stimulus-response system can take over full control of a bot, while the

planning system is temporarily disabled. This will reduce processing load while still maintaining believability, as the stimulus-response system is sufficient to provide realistic bot reactions.

Automatic Achievement of Goals: As mentioned above, planning and action selection can be computationally expensive tasks. When a bot is outside the focus of the player, these tasks can be avoided entirely by simply allowing the character's goals to be achieved automatically. After all, if a goal is achievable by the bot, and the player is not present to witness the goal actually being achieved, planning and action selection and execution are not required for the player to believe what has happened. It is important to ensure, however, that the goal is likely to be achieved by the bot, and that the time required to meet the goal is properly taken into account; otherwise believability may be inadvertently sacrificed.

Disabling of All Processing: If a bot is relatively unimportant and is someone with whom the player has had no prior personal contact or knowledge thereof, it is possible to disable all, or nearly all, decision making in the bot. After all, the player would have little to no expectation of the bot's current mood, goals, or actions and so it is believable for the bot to be in their initial state when first encountered by the player. The player has no way of knowing that the bot was largely inactive up until when they first entered the focus of the player. Of course, this assumes a degree of randomness in initial bot settings, as a population of identical bots in this fashion would be unrealistic.

This strategy might also be applicable to important bots or bots that have been previously encountered, provided that they are out of the player's focus and will remain out of their focus until the next break in the game, such as a cut-scene, level transition, and so on. If important events are recorded, their effects on the bot, as well as the bot's goals, plans, and actions can all be simulated during the break, so that they are up-to-date when the player next encounters them. (This technique may also need to be applied to bots with no prior contact, as discussed above, as sufficiently important events should impact those bots and move them from their initial states accordingly. This, of course, depends on the nature of the events and the social networks that the bots are a part of, as these would also influence to flow of information to and from the bots in question.)

When we combine the various forms of data and processing reductions together, we have great flexibility in the amount of capability scaling or reduction available to a game. If taken too far, this can eventually impact the believability of the game, but if done with care in an intelligent fashion, we can achieve tremendous performance savings with little to no effect on the believability perceived by the player.

2.6.1.3 Character Importance and Priority Calculation

The process of scheduling and dispatching, as well as the process of capability scaling or reduction both use a measure of a bot's importance as a factor that ultimately affects both resource allocation and performance. Capability scaling or reduction

uses a measure of importance directly, adjusting the capabilities of a bot accordingly. Scheduling and dispatching use importance in the form of a priority in determining which bots are run in each game tick. Below, we examine how each measure is computed.

The importance of a bot is determined by a collection of factors, with one calculating the importance, i, of a character as:

$$i = (\alpha f + \beta d + \gamma r + \delta c)/4$$

where α, β, γ, and δ are weights between 0 and 1.0 to tune and balance the equation. The factor f is a measure of player focus on the bot, which takes into consideration the distance between the player and the bot, whether the bot is within range of the player's senses, and the strength of relationships between the player and the bot. The factor d is a designer-imposed measure of importance of the bot, usually with respect to the story of the game. The factor r is a measure of importance defined by the currently active roles of the bot in question, as some roles in the game are inherently more important than others. Lastly, the factor c is a measure of importance that comes from bot interactions. If a given bot is involved with other, more important bots, their own importance might require a boost to put the bots on more equal footing. (For example, if the two are fighting each other, an inherently more important bot could enjoy an unfair advantage over less important bots because the seemingly more important bot has more capabilities and better access to computational resources.) The factors f, d, r, and c all range between 0 and 1.0 and so after scaling, the importance of a bot also lies between 0 and 1.0.

With this in mind, the priority, pri, of a non player bot can be computed as follows:

$$pri = \varepsilon i + \zeta s - \eta rc + \theta p$$

where ε, ζ, η, and θ are also weights between 0 and 1.0 to tune and balance the equation. (Carefully setting of these weights can also result in various scheduling policies, such as fair, least-slack-time, and so on [56].) The factor i is the importance of the bot as defined above. The factor s is a starvation factor that increases at a certain rate for each game tick that the bot is not run; by doing this, even an unimportant bot's priority will eventually exceed the most important bot, allowing the unimportant bot to run and avoid starvation. The factor rc is a run counter used to ensure that a single bot is not over-allocated update cycles despite its importance. Lastly, the factor p is a progress measure that approaches 1.0 as the bot approaches completion of its task at hand, to allow scheduling to clear out near-complete tasks from bots. All of factors i, s, rc, and p are normalized to between 0 and 1.0.

If desired, we can add a fifth factor to priority calculations to reflect the amount of capability reduction being applied to a particular bot. Doing so may be reasonable since a bot with its capabilities reduced by data or processing reductions will require fewer computational resources and therefore can cope with its schedule being reduced as well. Ordinarily, this would be accomplished using the importance factor i, as a low importance would trigger both capability and schedule reduction simultaneously.

If importance and capability reduction were not so closely linked, a separate factor indicating reduction would then be necessary. (This can occur, for example, when there is a very large number of non player bots needing to be managed; in such a case, even the capabilities of fairly important bots would need reduction despite their importance in order to maintain game performance at an acceptable level.)

2.6.2 Implementation and Results with Performance Optimization

As a proof of concept, a scheduler and dispatcher module was added to the third generation prototype discussed in Sect. 2.5.3 to allocate computational resources to bots. This was based on a simple serial sort and search algorithm to determine the next bots to run based on priorities as described in the previous section. Capability scaling or reduction was implemented with multiple levels of reduction. A bot running with full capabilities uses the complete goal-based utility-driven planning system described in Sect. 2.4.2.2. The first level of reduction uses rudimentary partial planning in which planning is carried out over several update cycles, with actions selected from partial plans in earlier update cycles while the current cycle continues to refine the plan. The second level of reduction uses full appraisal, coping, and goal selection capabilities, but then uses default plans and actions associated with goals selected, instead of carrying out a full planning/action selection stage. Finally, the third level of reduction uses randomization to select a goal and select a plan and actions capable of achieving this goal. While this is not the most realistic of approaches, it can still be appropriate for non critical characters in the game.

To assess the operation and performance of this approach, detailed logs were collected during a series of experiments. All experimentation was executed on an Intel Core 2 Duo system with a clock speed of 2.0Ghz and 4.0GB of RAM. The 64-bit variant of Windows Vista was the operating system. This configuration provided more than enough power for the experimentation we conducted.

The prototype system was configured to use the psychosocial model described in Sect. 2.4.1, with bots having access to 4 roles, 5 goals, and 8 actions during processing. While a typical game would have more possibilities open to its characters, this configuration on its own was sufficient to demonstrate the effectiveness of the system. Time in the system was simulated so that 4 bots could run each game tick, there were 30 milliseconds between game ticks, and each action consumed one tick for execution. While actions would ordinarily take longer and have varied lengths in reality, this accelerated experiments and simplified analyses, as it was easier to confirm that factors such as importance were being properly handled by the system. Lastly, for simplicity and balancing, all weights used in calculating importance and priority were set to 1.0, except during starvation experiments. It is possible that better (or worse) results could be obtained through the fine-tuning of these weights.

2.6.2.1 Initial Experiments

Prior to more rigorous experimentation, initial testing was conducted to assess the basic operation of our prototype system. From this, we were able to verify:

- Equal fixed importance and priority resulted in an even distribution of resources to bots and equal opportunity for execution.
- Increased importance and priority translated into an increase in resource allocations to bots and a corresponding increase in execution time.
- Starvation of bots with low importance was effectively prevented by our approach to scheduling, and would be a serious issue if these measures were disabled or not provided in the first place.
- Bots with reduced capabilities required fewer resources to execute than bots with full capabilities intact.
- The prototype system could handle several bots of varying importance well, adjusting scheduling and capabilities accordingly without difficulty.

While these tests verified the correct operation of the prototype system, we still needed to assess the improved performance and scalability enabled by our approach. This is accomplished through experimentation outlined in the next section.

2.6.2.2 Stress Testing

To assess performance and scaling improvements, we used the prototype system to manage hundreds of characters simultaneously. In these experiments, we executed three scenarios with 100, 200, and 800 bots respectively. The bots used psychosocial models as described in Sect. 2.6.2, with traits initialized randomly. Non-violent actions and interactions were used to ensure that the bot population was not inadvertently thinned during experimentation. No human player was involved in these experiments, as this could introduce unwanted variations and anomalies in the results. Each scenario was itself run three times, once with all bots at full capability, once with all bots at the second level of reduction (as described in the previous section), and once with all bots at the third level of reduction. (In these experiments, locality was not used as a factor, to ensure that all bots stayed at a fixed level of reduction.)

Results from this experimentation are shown in Fig. 2.17, measuring the time to completion of 200 game ticks in seconds. With 100 bots executing, performance under full capacity suffered greatly. There was a slight improvement under the second reduction, but performance was still unacceptable. With the third reduction, performance improvements were substantial. As the number of bots increased, only the third reduction bots were able to complete. While their time to completion increased in a roughly linear fashion, performance was still improved dramatically through this reduction. (The linear scaling of third reduction bots is due to their simplicity.

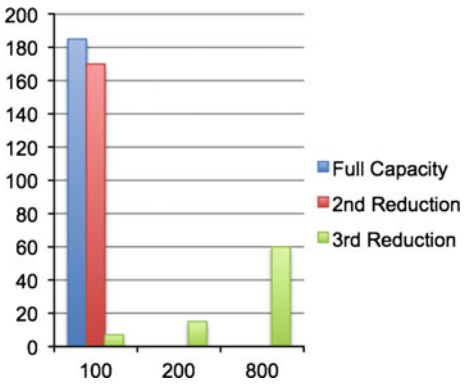


Fig. 2.17 Bot stress testing

Bots with more capabilities, especially those that are more socially active and conscientious enough to factor others into their decision making processes, would not enjoy this linear scaling.)

It is important to note that while full capacity and slightly reduced bots had performance issues, the system would never be expected to support this many at a time. Through dynamic adjustments to capabilities, only a few would run at these capability levels at a time, depending on the game, with the others reduced further. This experiment was to solely demonstrate performance improvements through our approach.

From these results, we can see great improvements in the performance delivered by our approach to scalable believable non player bots. We are able to deliver a collection of bots that can adapt to various computational requirements through proper scheduling and capability adjustment.

2.7 Concluding Remarks

This chapter has examined the creation of believable bots for video games through the use of psychosocial behaviour. By using bots with personality, emotion, social awareness and other psychosocial traits, we have shown how to create bots capable of creating engaging and immersive gameplay. Prototype implementations have proven our concepts and demonstrated some of the possibilities achievable through their use. Finally, we have discussed and tested various techniques for performance optimization to ensure that believable bots will meet the real-time requirements and resource constraints of modern video games.

References

1. Acton, G.: Great ideas in personality—theory and research (2006). http://www.personalityresearch.org. Accessed Jan 2011

2. Acton, G.: Playing the role: towards an action selection architecture for believable behaviour in non player characters and interactive agents. Masters thesis, Department of Computer Science, The University of Western Ontario (2009)
3. Alt, G., King, K.: A dynamic reputation system based on event knowledge. In: AI Game Programming Wisdom. Charles River Media, Hingham (2002)
4. Bailey, C., Katchabaw, M.: An emergent framework for realistic psychosocial behaviour in non player characters. In: Proceedings of FuturePlay 2008, Toronto, Canada (2008)
5. Baille, P.: The synthesis of emotions in artificial intelligences. Ph.D. Dissertation, University of Southern Queensland (2002)
6. Bates, J., Loyall, A., Reilly, W.: An Architecture for Action, Emotion, and Social Behaviour. Lecture Notes in Computer Science, vol. 830. Springer, Heidelberg (1994)
7. Beaumont, L.: Life's guidance system: traveling your path through life (2008). http://www.emotionalcompetency.com/guidance.htm. Accessed Jan 2011
8. Berger, L.: Scripting: overview and code generation. In: AI Game Programming Wisdom. Charles River Media, Hingham (2002)
9. Biddle, B., Thomas, E.: Role Theory: Concepts and Research. Wiley, New York (1966)
10. Brockington, M.: Building a reputation system: hatred, forgiveness, and surrender in never-winter nights. In: Massively Multiplayer Game Development. Charles River Media, Hingham (2003)
11. Byl, P.B.D.: Programming Believable Characters in Games. Charles River Media, Hingham (2004)
12. Cattell, R.: The Description and Measurement of Personality. Harcourt, Brace, and World, New York (1946)
13. Cesta, A., Miceli, M., Rizzo, P.: Robustness of the social attitude and system performance. In: Proceedings of the First International Conference on Multi-Agent Systems, Help Under Risky Conditions (1996)
14. Cole, S.: Modeling opinion flow in humans using Boids algorithm and social network analysis (2006). http://gamasutra.com/features/20060928/cole_01.shtml. Accessed Jan 2011
15. Crawford, C.: Chris Crawford on Interactive Storytelling. New Riders, Indianapolis (2005)
16. de Freitas, J.S., Imbert, R., Queiroz, J.: Modeling Emotion-Influenced Social Behavior for Intelligent Virtual Agents. Lecture Notes in Computer Science, vol. 4827. Springer, Heidelberg (2007)
17. Dias, J., Paiva, A.: Feeling and Reasoning: A Computational Model for Emotional Characters. Lecture Notes in Computer Science, vol. 3808. Springer, Heidelberg (2005)
18. Ekman, P., Friesen, W., Ellsworth, P.: Emotion in the Human Face: Guidelines for Research and an Integration of Findings. Pergamon Press, New York (1972)
19. Elliott, C.: The affective reasoner: a process model of emotions in a multi-agent system. Ph.D. Dissertation, Northwestern University Institute for the Learning Sciences (1992)
20. El-Nasr, M.: Modeling emotion dynamics in intelligent agents. Masters thesis, American University in Cairo (1998)
21. Epic Games. UDK—Unreal Development Kit (2010). http://www.udk.com. Accessed Jan 2011
22. Eysenck, H.: Biological Dimensions of Personality. In: Pervin, L.A. (ed.) Handbook of Personality: Theory and Research. Guilford, New York (1990)
23. Funder, D.C.: The Personality Puzzle. W. W. Norton & Company, New York (2001)
24. Funge, J.D.: Artificial Intelligence for Computer Games. A K Peters, Natick (2004)
25. Game Informer. Assassin's Creed. June 2006, Issue 158. Vol XVI: No 6 (2006)
26. Game Informer. Bully. September 2006, Issue 161, Vol XVI: No 9 (2006)
27. Gratch, J., Marsella, S.: A domain-independent framework for modeling emotion. Cogn. Syst. Res. **5**(4), 269–306 (2004)
28. Gratch, J., Marsella, S.: Evaluating a computational model of emotion. Auton. Agent. Multi-Agent Syst. **11**(1), 23–43 (2005)
29. Grond, G., Hanson, B.: Reputation system FAQ (1998). http://update.uo.com/repfaq. Accessed Jan 2011

30. Gruenwoldt, L., Katchabaw, M., Danton, S.: Achieving Realistic Reactions in Modern Video Games. In Worlds In Play. Peter Lang Press, New York (2007)
31. Guye-Vuilleme, A., Thalmann, D.: A high-level architecture for believable social agents. VR J. **5**, 95–106 (2001)
32. Hogg, L.M.J., Jennings, N.R.: Socially intelligent reasoning for autonomous agents. IEEE Trans. Syst. Man Cybern. Part A Syst. Hum. **31**(5), 381–393 (2001)
33. Imbert, R., de Antonio, A.: An Emotional Architecture for Virtual Characters. Lecture Notes in Computer Science, vol. 3805. Springer, Heidelberg (2005)
34. Ion Storm. Thief: Deadly Shadows. Eidos Interactive (2004)
35. Isbister, K.: Better Game Characters by Design. Morgan Kaufmann, San Francisco (2006)
36. Keirsey, D.: Please Understand Me II: Temperament, Character, Intelligence. Prometheus Nemesis, Del Mar, CA (1998)
37. Lawson, G.: Stop relying on cognitive science in game design—use social science (2003). http://www.gamasutra.com/php-bin/letter_display.php?letter_id=647. Accessed Jan 2011
38. LeBlanc, M.: Formal design tools: emergent complexity, emergent narrative. In: Proceedings of the 2000 Game Developers Conference (2000)
39. Livingstone, D.: Turing's test and believable AI in games. Comput. Entertain. (CIE) **4**(1), 6 (2006)
40. Loyall, A.: Believable agents: building interactive personalities. Ph.D. Dissertation, Stanford University (1997)
41. Mateas, M.: An Oz-centric Review of Interactive Drama and Believable Agents. Lecture Notes in Computer Science, vol. 1600. Springer, Heidelberg (1999)
42. McCrae, R. Costa, P.: Reinterpreting the Myers-Briggs type indicator from the perspective of the five-factor model of personality. J. Pers. **57**(1), 17–40 (1989)
43. Mehrabian, A.: Framework for a comprehensive description and measurement of emotional states. Genet. Soc. Gen. Psychol. Monogr. **121**(3), 339–361 (1995)
44. Myers, I.B., McCaulley, M., Quenk, N., Hammer, A.: MBTI Manual: A Guide to the Development and Use of the Myers Briggs Type Indicator, 3rd edn. Consulting Psychologists Press, Palo Alto (1998)
45. Mythic Entertainment. Reputation, karma and fame (2005). http://www.uoherald.com/node/167. Accessed Jan 2011
46. Ortony, A.: On Making Believable Emotional Agents Believable. Appeared in Emotions in Humans and Artifacts. MIT Press, Cambridge (2003)
47. Ortony, A., Clore, G., Collins, A.: The Cognitive Structure of Emotions. Cambridge University Press, Cambridge (1988)
48. Paanakker, F.: The Emotion Component: Giving Characters Emotions. Appeared in AI Game Programming Wisdom, vol. 4 (2008)
49. Panzarasa, P., Jennings, N., Norman, T.: Social mental shaping: modeling the impact of sociality on the mental states of autonomous agents. Comput. Intell. **17**(4), 738–782 (2001)
50. Parrott, W.: Emotions in Social Psychology: Essential Readings. Psychology Press, Philadelphia (2001)
51. Pfeifer, B.: Creating emergent gameplay with autonomous agents. In: Proceedings of the Game AI Workshop at AAAI-04 (2004)
52. Picard, R.W.: Affective Computing. Media Laboratory, Perceptual Computing TR 321, MIT Media Lab (1995)
53. Plutchik, R.: The nature of emotions. Nature **89**(4), 344 (2001)
54. Prendinger, H., Ishizuka, M.: Social role awareness in animated agents. In: International Conference on Autonomous Agents (2001)
55. Prendinger, H., Ishizuka, M.: Evolving social relationships with animate characters. In: Proceedings of the AISB-02 Symposium on Animating Expressive Characters for Social Interactions (2002)
56. Rankin, A., Acton, G., Katchabaw, M.: A scalable approach to believable non player characters in modern video games. In: Proceedings of GameOn 2010, Leicester, United Kingdom (2010)

57. Reiss, S.: Multifaceted nature of intrinsic motivation: the theory of 16 basic desires. Rev. Gen. Psychol. **8**(3), 179–193 (2004)
58. Reynolds, B.: How AI enables designers (2004). http://gamasutra.com/php-bin/news_index.php?story=11577. Accessed Jan 2011
59. Rizzo, P., Veloso, M., Miceli, M., Cesta, A.: Personality-driven social behavior in believable agents. In: Proceedings of the AAAI Fall Symposium on Socially Intelligent Agents (1997)
60. Romano, D., Sheppard, G., Hall, J., Miller, A., Ma, A.: BASIC: a believable adaptable socially intelligent character for social presence. In: Proceedings of the 8th Annual International Workshop on Presence, London, United Kingdom (2005)
61. Rousseau, D., Hayes-Roth, B.: Personality in synthetic agents. Technical report, Knowledge Systems Laboratory, Department of Computer Science, Stanford University, Stanford, CA (1996)
62. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, Englewood Cliffs (2003)
63. Sinclair, P.: The "Big Five" factors personality model (2011). http://www.odportal.com/personality/big-five.htm. Accessed Jan 2011
64. Sloman, A.: Beyond shallow models of emotion. Cognitive processing. Lengerich **2**(1), 177–198 (2001) (Pabst Science Publishers)
65. Smith, C., Ellsworth, P.: Attitudes and social cognition. In: Journal of Personality and Social Psychology, American Psychologists Association, Washington, vol. 48(4) (1985)
66. Smith, H., Smith, R.: Practical techniques for implementing emergent gameplay: would the real emergent gameplay please stand up? In: Proceedings of the 2004 Game Developers Conference (2004)
67. Straker, D. Big five factors (2010). http://changingminds.org/explanations/preferences/big_five.htm. Accessed Jan 2011
68. Sweetser, P.: An emergent approach to game design, development and play. School of Information Technology and Electrical Engineering, The University of Queensland (2006)
69. Sweetser, P.: Emergence in Games. Charles River Media, Hingham (2008)
70. Sweetser, P., Johnson, D., Sweetser, J., Wiles, J.: creating engaging artificial characters for games. In: Proceedings of the Second International Conference on Entertainment Computing. Carnegie Mellon University, Pittsburgh, Pennsylvania (2003)
71. Tomkins, S.: Affect theory. In: Scherer, K.R., Ekman, P. (eds.) Approaches to Emotion. Erlbaum, Hillsdale (1984)
72. Tomlinson, B., Blumberg, B.: Using emotional memories to form synthetic social relationships (2002). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.4475. Accessed Jan 2011
73. Tozour, P.: The Evolution of Game AI. In AI Game Programming Wisdom. Charles River Media, Hingham (2002)
74. Tupes, E., Cristal, R.: Recurrent personality factors based on trait ratings. Technical report ASD-TR-61-97, Lackland Air Force Base, TX: Personnel Laboratory, Air Force Systems Command (1961)
75. Velasquez, J.: From affect programs to higher cognitive emotions: an emotion-based control approach. In: Proceedings of the Workshop on Emotion-Based Agent Architectures, Seattle, Washington (1999)
76. Woodcock, S.: Game AI: the state of the industry (2000). http://www.gamasutra.com/view/feature/3570/game_ai_the_state_of_the_industry.php. Accessed Jan 2011
77. Wooldridge, M.J.: The logical modeling of computational multiagent systems. Ph.D. thesis, UMIST, Manchester (1992)
78. Wooton, B.: Designing for Emergence. AI Game Programming Wisdom, vol. 3. Charles River Media, Hingham (2006)
79. Wright, I., Marshall, J.: Egocentric AI processing for computer entertainment: a real-time process manager for games. In: Proceedings of the First International Conference on Intelligent Games and Simulation (GAME-ON 2000), London, United Kingdom (2000)

80. You J., Katchabaw, M.: A flexible multi-model approach to psychosocial integration in non player characters in modern video games. In: Proceedings of FuturePlay 2010, Vancouver, Canada (2010)
81. Zhou, C., Yu, X., Sun, J., Yan, X.: Affective Computation based NPC behaviours modeling. In: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology. IEEE Computer Society Washington, DC, USA (2006)

Chapter 3
Actor Bots

Maria Arinbjarnar and Daniel Kudenko

Abstract In many recent computer games there has been an increased use of Non-Player Characters as quest givers, tradesmen, members of the general public and as quest companions for the players. All these roles call for intelligent bots that can give the player an engaging experience by adjusting and adapting to the players' style of play. They need to behave as actors in a virtual improvisational theatre. They need to perform in a manner that the player finds believable and engaging so that the player becomes sufficiently immersed in the interactive drama and keeps coming back for more. The bot's intelligence needs to be both robust to respond to player actions and sufficiently complex to handle character traits, emotions and complex decision making that resembles that of a human player. A balance needs to be maintained between the bots' believability and their entertainment value for the player, as these two goals may be at odds in some situations.

3.1 Introduction

Modern technology offers us new and largely underutilized media in the form of vast virtual realities rich with graphics and sound. Large numbers of people log on every day to interact with others in online realities and some even run businesses selling virtual objects to other players. Additionally game worlds become ever richer in graphics and sound and players can spend months both in single player and online-mode; exploring, taking on quests and interacting with Non-Player Characters (NPCs). There are mainly three computer game genres that are of interest here;

M. Arinbjarnar (✉) · D. Kudenko
Department of Computer Science, University of York,
Deramore Lane, York YO10 5GH, UK
e-mail: maria@cs.york.ac.uk

D. Kudenko
e-mail: kudenko@cs.york.ac.uk

P. Hingston (ed.), *Believable Bots*, DOI: 10.1007/978-3-642-32323-2_3

First Person Shooters (FPSs), Adventure games also known as point and click games and Role-Playing Games (RPGs).

FPSs games are centred on using a gun or any projectile weapon and with it killing a large number of enemies that are bent on killing the player. The games are thus very action based and the player needs good coordination and reflexes to master the challenges of the games. There is usually a main story line that runs through the game, and there are multiple levels that the player needs to complete to progress in the game.

Adventure games are quite the opposite of FPSs in that there is very little action or speed required to complete the game. This is why they are frequently called "point and click" games because for the most part all the player does is to locate or combine items with a click of the mouse or talk to NPCs in the game through drop down dialogue boxes to progress the story. They are very puzzle oriented and the player frequently needs to solve several puzzles to complete the game. These games tend to be heavily story centred and are frequently based on a film or a novel.

Computer-Based Role-Playing Games (CB-RPGs) are a combination of FPS and adventure games and Pen and Paper RPGs (PP-RPGs). In CB-RPGs the player takes on a role in a rather rich virtual world and through a series of quests will save this world from impending doom or resolve some really important in-game issue. The player will gain experience through solving quests and killing monsters and increase in power (level up) as the game progresses. These quests and levelling up combine elements from FPSs and Adventure games. There is considerable action needed to kill the monsters encountered and the player also needs to pick up objects and solve puzzles. CB-RPGs have a rich story element; there is a main story that needs to be played through to complete the game and multiple short stories in the quests that the player takes on. In PP-RPGs a group of players are needed to play the game using pens and papers to keep track of their character actions and experience gain. There is also a need for a Game Master (GM) who tells the story of the game, plays the NPCs and in general takes care of managing the game-play.

At the dawn of CB-RPGs, there where high hopes for the extension of PP-RPGs into computer games. There were visions of vast worlds with endless possibilities and rich interaction with NPCs, possibly meeting your favourite heroes from your favourite fantasy series. The player would not be limited by the imagination and storytelling skills of the GM or by how successfully a good group of players could be brought together. The players could adopt any role they desired, shape the character to their likings and play through a virtual reality, receiving a unique play experience for each character that they play. Included in the unique play experience would be a clear sense of a story and purpose for the characters actions just like a quality GM is able to create in PP-RPGs.

The main motivation for this vision is that it can be very difficult to find a qualified GM that offers sessions in the worlds and playing-style of an individual player's likings. It can also be very difficult to find a good group that fits the player's playing-style. Essentially what players are seeking in the CB-RPG is a RPG tailored to their tastes, competence and play style.

Current computer games fall short of providing a complete RPG experience for a number of reasons, the most prominent being that the story-lines are pre-authored and do little to accommodate an independent role-playing style. There are also pre-authored stories for PP-RPGs but the GM is free to diverge from them when it suits the game and GMs can also make their own stories either in advance or develop them on the fly during game play. Frequently the CB-RPGs force the player to violate the characteristic behaviours of the role in order to complete quests and progress in the game's main story. Additionally the NPCs' responses are pre-authored making them very repetitive. The results are rich game worlds that lack replayability due to repetitive story-lines and limited NPC interactions.

This is where actor bots come in. Actor bots behave in a similar manner as improvisational actors, acting out a drama and engaging the player in a dynamically changing storyline. The actor bots have three primary goals; staying in character for believability, working with other actor bots in creating a structured drama, and interacting with players in an engaging manner.

The chapter is centred around what part actor bots play in the transition of RPGs from being pen and paper based to being computer based and so we start by giving a comprehensive description of PP-RPGs, what the player expects and specifically what player types there are. Understanding different player types is very important to realise how actor bots can enhance player experience. In Sect. 3.3 we discuss how drama is playing an increasing role in current computer games and how the Directed Emergent Drama engine (DED) is designed to facilitate the merge of drama and games. We describe the DED engine and specifically the actor bots which are the main part of the engine. In Sect. 3.4 we discuss Bayesian Networks and how they are the core of the actors' knowledge base and decision mechanism. We also discuss problems and limitations of Bayesian Networks and evaluate their performance. In Sect. 3.5 we review related work and in the last section we summarise the chapter.

3.2 Players' Expectations

To be able to understand the motivation for actor bots it is very important to realise what many current CB-RPGs are trying to accomplish. They are trying to realise a rich game world that allows for very free and vicarious game play akin to a good PP-RPG. This is why understanding PP-RPGs is vital to understanding the aims of actor bots.

PP-RPGs are played with a GM and players typically sitting around a table and keeping track of the games progress using pen and paper, see Fig. 3.1. Dungeons & Dragons is the original form of the game that was introduced by Gary Gygax in 1974, published by TSR [43].

The GM is the narrator or storyteller of the game and will play the roles of any NPC that the players encounter in the game, such as shopkeepers and quest-givers. Additionally the GM will settle any dispute and has the last word when interpreting rules.

Fig. 3.1 The basic setup for a PP-RPG

Fig. 3.2 The many sided dices for PP-RPGs

The players create characters for the game and play a specific role in the game such as a wizard or a fighter. Each character has a set of statistics that define both physical make-up (e.g. strength, constitution) and mental abilities (e.g. intelligence, wisdom). Each character will have a detailed inventory and keep a record of which weapons and armour are equipped. The players keep track of this using pen and paper during the game. The characters have an alignment towards good or evil and chaotic or lawful.

The aim of the game is for the GM to create a rich and engaging adventurous world where the players can vicariously experience adventures by playing the role of their characters, taking on quests and exploring the secrets of the game world. The only limitation of the PP-RPG is that of the players' and the GM's imagination [17].

There are usually both large and small quests that the players can take on in addition to playing through a main plot that may possibly exist in the game-world. They will need to fight monsters and solve puzzles in order to complete the quests. They will be able to buy and sell items from merchants and rest in pubs along the way. When deciding the physical and mental limitations of characters and monsters

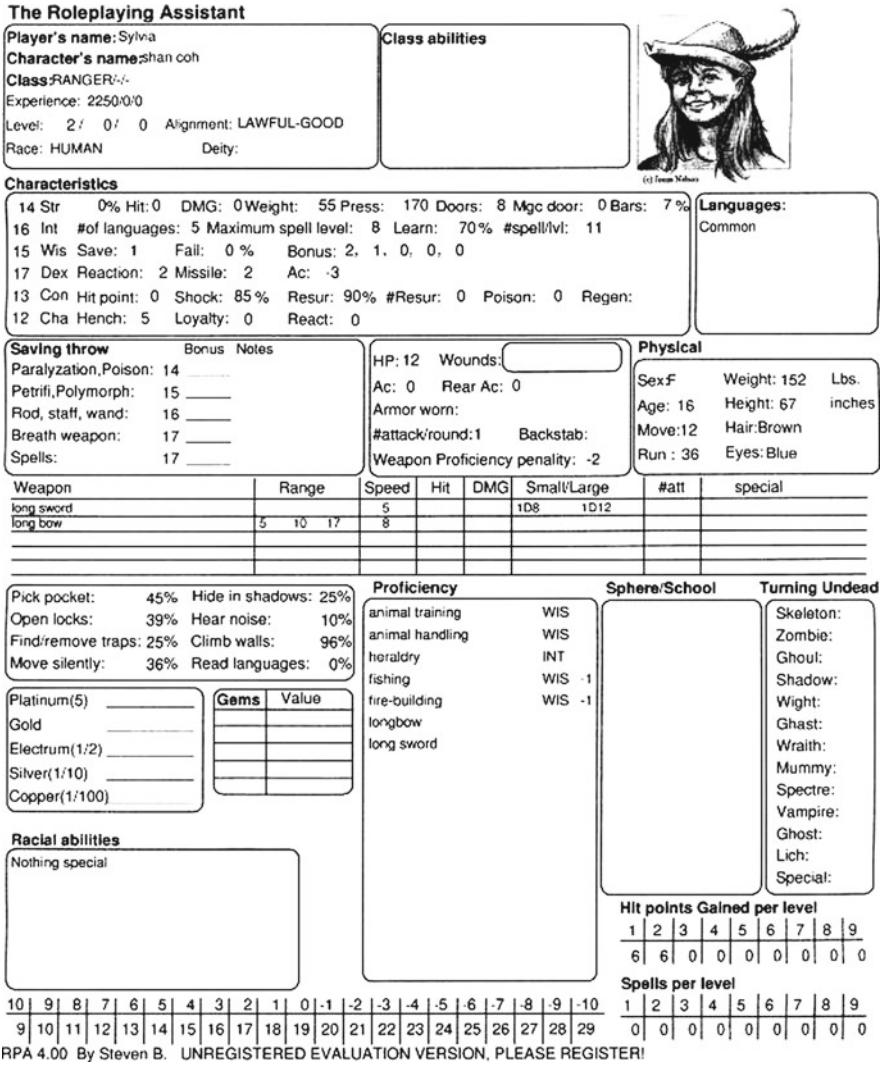


Fig. 3.3 A character sheet for PP-RPGs

during game play the GM and players are aided by many sided dices and character sheets, as shown in Figs. 3.2 and 3.3.

Initially the games where set in a medieval world with castles, dungeons, elfs, goblins, wizards etc., i.e. a typical J. R. R. Tolkien type of a world. Since the dawn of RPGs in 1974 they have been adapted to many genres including mysteries (Etherscope [13]), sci-fi (Star Trek [10]) and vampire-tales. RPGs can potentially be adapted to any genre with a bit of imagination.

Anyone who has played and enjoyed a good PP-RPG knows how rich the experience can be, to feel as if you are that character treading slowly down a wooden path not knowing what monster can spring up in front of you or whether a group of travellers that can be seen in the distance are friend or foe. This element of surprise is in current CB-RPGs but only the first time you play it. While PP-RPGs can be potentially played for years and the novelty of the experience is only limited by the players' imagination.

Players actions are limited by rules that attempt with considerable success to emulate the natural physical constraints of the world. As Laurel [27] explains, it *"is difficult to imagine life, even a fantasy life, in the absence of any constraints at all"*, for example the time spent on travelling is not limiting a player's freedom, it is deepening the players vicarious experience.

> "Computer games are inherently limited because they only give you a set number of options. In a game like this (PP-RPG*), what we can do is limited only by our minds." Sam Weiss, as quoted in [42] * added by authors.

3.2.1 Player Types

It is important to understand the different player types to realise player expectations and how actor bots can be of use. There are seven distinct player types that are described below, six of them are defined by Laws in [28]. Although it would be possible to designate other player types that grasp different aspects of the play-styles of players, it has not been done in a formalised way and Laws player types are describing the elements of play-styles that help us to identify what the Actor bots need to accomplish. Obviously the actor bots need to be catering to the needs or expectations of at least some of these player types or the exercise is futile. This is of greater importance than the game genre because the player types are more generic and applicable to all computer games. This does not mean that all computer games cater to every player type, quite the contrary. Moreover an individual can enjoy playing several different playing styles or a combination of two or more styles. It still remains that these player types are distinct and easily recognised. The interested reader can look at [48] for a more comprehensive discussion and user evaluation of the importance of player modelling.

As described by Laws there are six core player types; Power Gamer, Butt-Kicker, Tactician, Specialist, Method Actor, Storyteller and we add a seventh type that is frequently forgotten, the Casual Gamer:

1. **The Power Gamer**, loves to level up and acquire strong artefacts, magic and weapons. The current CB-RPGs cater well for this type of play it is mostly a matter of giving rewards with respect of the difficulty of the opponents fought and that is easy for a computer to calculate. This player is typically not very interested in the story but will appreciate companion bots that help him fight and to level up. If interacting with actor bots is embedded in the levelling up then this type of player will take care to play with the actor bots in order to level up

faster. When the Power Gamer has levelled up, the gamer will be searching for a more involved game and frequently joins a guild and or helps other players to level up. Clearly being able to participate in a deeper role appeals to this player type as currently when the main story is over or he has reached the highest level it is really game over for this player unless there is a good guild and other active players on-line.

2. **The Butt-Kicker**, tired after work or a stressful day, logs on as a strong character, for instance a fighter, and goes on a killing spree. Every First-Person Shooter (FPS) is designed for this playing-style. Still actor bots can come in as helpers and give a bit more depth to the game. That is what FPS games are heading towards. Most of their players want more depth to the game and they now have rather involved cut scenes that play out a kind of cinematic story for the player.
3. **The Tactician**, knows the rules, these players will plot and carefully calculate the movements of their characters, they like to be challenged. There are many strategy games on the market for this playing-style. As with the power gamer then this is mostly about calculations which computers are very good at. Many tacticians love to be in alliances and fight together with heroic comrades. A good alliance can be very hard to find.
4. **The Specialist**, usually has a favourite role and alignment that is always played, the player is constantly trying to improve on the role and really needs to be able to stay in character. This has little support in current CB-RPGs, although the players can pick roles and skills to advance. It needs more than that. A thief character would rarely venture out of cities where there are many opportunities for his trade in order to go on a goblin killing spree in a smelly dungeon. It does not really fit the role of a thief. Similarly a paladin will hardly care to sneak around somebody's dwellings breaking into locked containers to discover some vital information for the progression of a main story. Sadly this is repeatedly true for current computer games. Players find that they are unable to stay in-role if they want to complete the main story.
5. **The Method Actor**, is similar to the specialist except for not concentrating on a single role. The method actor tries to really enter the psyche of the character and to play the character in as a convincing way as possible. This has no more support than the specialist in current CB-RPGs. There is in fact no allowance for choosing specific characteristic except for charisma, intelligence, wisdom and alignment. There is no allowance for defining a caring, attentive cleric that becomes impulsive when angry, is slow to anger and bears grudges for a long time. Although players can of course imagine this when they play computer games then it does not have any actual affect on the game play as it does in PP-RPGs.
6. **The Storyteller**, this player enjoys a good story and will get quickly bored by needing to spend a lot of time levelling up as is required in CB-RPGs.
7. **The Casual Gamer**, this player is in for the ride and is frequently forgotten because of being passive. It is difficult to say whether one game or another specifically caters to their expectations. Still this is an important player type because it is one of the fastest growing type. The Casual Gamer frequently becomes a serious gamer when the player finds a good game. Moreover the Casual Gamer

is an essential fill in on-line games to show large numbers and to advertise the game by word of mouth. This is why a successful game should try to appeal to Casual Gamers as well as the serious gamers described above. Quite obviously, actor bots that engage players will help greatly in facilitating large numbers of Casual Gamers.

> "Games give you a chance to excel, and if you're playing in good company you don't even mind if you lose because you had the enjoyment of the company during the course of the game." Gary Gygax, as quoted in [43]

What is mostly lacking in current CB-RPGs is to offer this companionship and the sense of a vicarious experienced based around the players personal play stile. Specifically for the Specialist, Method Actor and Storyteller but also for the other player types. This is where actor bots can play a crucial role in engaging the player and involving the player in a drama that is tailored to the player type and the players actions in the drama.

3.3 Drama and Games

If we compare the role of a GM to that of a CB-RPG we see that the CB-RPG is not really offering the same RPG experience. A CB-RPG has a pre-made storyworld which has only limited adaptability to players interactions. Many CB-RPG such as Fallout 3 [30] and Neverwinter Nights [6] try to incorporate such adaptability into their games.

Neverwinter Nights [6] uses alignment of the character to determine the available paths through the game. Each player creates a character and determines the initial alignment. The players get a list of options to use to interact with NPCs in the game; some of these options are for good alignment and some for evil. When players choose evil actions then their alignment is slightly shifted to evil and if they choose a good action the alignment is shifted towards good. Quests that are accepted and completed are also used to adjust players' alignment. Good deeds adjust the players' alignment to good and evil deeds towards evil. The alignment affects the player's progress in the game by for instance hindering players playing druids in getting full access to their powers and the druid community if the player has not been careful to keep the character alignment as good. It will also affect what sub quests are available especially when it comes to quests that are for the advancement of a specific role and need a specific alignment. Still it does not change the main story or the main story plot, the player will need to finish specific main quests in order to advance and complete the main story. Nothing the player does will affect the pre-authored plots of the game.

Fallout 3 [30] has brought this idea even further, players actions greatly affect the characters reputation and opportunities. The availability of quests is directly linked to players actions; how much they explore the world, what they say in conversation, and what skills they pick when levelling up. Still the same can be said for Fallout 3 as with Neverwinter Nights: nothing the player does will change the pre-authored

plots or the main storyline. The player is always playing along one branch of a multi-linear story that essentially follows the same fundamental plot-points each time. This means that players will not get a novel story experience when replaying the game, which makes it very repetitive and lacking replayability.

Another issue with these games is that players save and reload the game in order to get the plot development that they prefer. If they say something to an NPC that has negative results, they can simply reload an earlier saving and try again. Equally if they fail in a battle or a quest, they can simply reload an earlier save. This means that not only is the game trying to adapt to their play style but the players are also adjusting the outcome to their personal preference. This is a very natural behaviour on behalf of the player and there is nothing wrong with it, but it needs to be accounted for. Furthermore some players will systematically try every action and explore every area and put their acquired information on the web where other players can benefit from it if they choose.

This is not the case in PP-RPGs there is no save-reload function the story emergence depends on player actions and the GMs skill in providing an engaging and rich environment. The core difference is that PP-RBGs are interactive dramas while CB-RPGs are incorporating a narrative into a game. The latter has a number of problems that Juul relates well in [23]. Essentially you can't have narration and interactivity at the same time any more than you can have panorama and drama at the same time. The two are a contradiction. To narrate means to recount [37], when a player is interacting with a gameworld then it is at odds to try to tell the player a story at the same time. On the other hand the player can easily participate in a drama which at its core is a "play".

> "Pen-and-paper role-playing is live theatre and computer games are television." Gary Gygax, as quoted in [42]

As we have discussed, drama is clearly an integral part of RPGs despite the fact that some player types do not pay much attention to it such as the "Butt-kicker" and "Power gamer". Perhaps surprisingly drama is also becoming more and more integrated into First-Person Shooter (FPS) games.

FPS games are not very replayable in their basic single player form. The reason for this is that once you have played through it then when you play it again there are no surprises, no suspense. You know where the bosses are and how to deal with them; there are no significant changes between each game play and no new challenges. Multiplayer FPS are on the other hand highly replayable because then the player is constantly up against a new challenge. In recent years single player FPS games have evolved towards having the opponents, the Non-Player Characters (NPCs), more intelligent to provide a greater challenge for the player. They will simulate the normal behaviour of people in similar situations, including going to sleep, eating and in general going about their daily lives. One of the most recent and greatly popular games in these series is Assassin's Creed II [49] where the player is an assassin that is assigned a number of assassination jobs. In Assassin's creed as in many other recent FPS there is a much greater emphasis on the story and character development than before. The player gets an in-depth character description of the

assassin and why he is in this job. The story progresses gradually through the game and is delivered in seamless cut scenes. The player is playing the protagonist and as the story progresses the protagonist faces many moral dilemmas and challenging revelations that dramatically change his perspectives and shape his character. This is very typically of other recent games such as Fallout 3 [30] and Heavy Rain [9]. Heavy Rain carries this character development even further and allows the player to make decisions on what the character does based on the characters emotions and thoughts, forcing the player to make challenging moral decisions. There is a very clear trend towards making the NPCs more intelligent and more autonomous in order to give the player a more fulfilling game experience. A good example of this is S.T.A.L.K.E.R. [50], there the NPCs actively simulate the behaviours of soldiers or mercenaries that try their best to kill the player in heavily polluted Chernobyl. Their behaviour is very believable and they provide challenging game play, they do not simply continuously walk the same circle. In this respect there is a clear disparity between the increased playability of FPS which has every potential of providing rich replayability, as becomes clear when these games offer multiplayer mode, and the still pre scripted unchanging storyline that the player is forced to follow. Creating more human like opponents is the topic of Chaps. 6 and 7, we concern ourselves with enriching the drama element.

The popularity of titles like Heavy Rain and Assassins creed clearly demonstrate that there is a market for games that provide richer story worlds and character development. This calls for an even greater emphasis on character development and drama that is not so pre-scripted that it is counterproductive to replayability. There needs to be a merge between drama and game-play. The drama should emerge around the player, influenced by the player actions and still provide a structured story experience. One way to accomplish this is to make the NPCs even more autonomous so that they can enact a drama for the player based on abstract goals rather than pre scripted story-lines. The NPCs need to be believable in their performance; this means that they should not violate the expectations of the audience. For example they need to stay in character such that a calm, gentile character remains calm and gentile unless extremely provoked etc. The NPC behaviour should be plausible in order to suspend disbelief. Effectively the actor bots should demonstrate human-like behaviour which calls for large knowledge bases and reasoning under uncertainty.

Bayesian Networks are specifically suitable for this task because they are causal and they provide a means of calculating action utilities based on probabilities in a highly efficient manner. Bayesian Networks are an effective way of incorporating psychological elements such as traits and emotions which are very important for believability [5, 7, 14, 19, 26, 47].

3.3.1 Directed Emergent Drama

In order for the drama to become an integral part of game-play and support it rather than counter it we have designed an architecture called Directed Emergent Drama

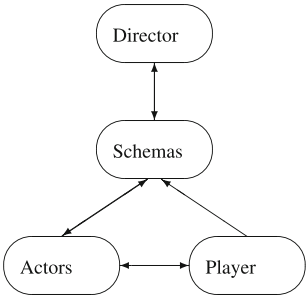


Fig. 3.4 The DED architecture [1]

(DED) [1, 3]. As seen in Fig. 3.4 it consists of a director, schemas, actors and a player. The player only ever interacts with schemas and actors and never the director. The actors receive all their information from the player and from schemas, they do not interact directly with the director.

Rather than having predefined options like current CB-RPGs do, DED engine facilitates the player to act freely within the constraints of the virtual world. Actions for interacting with the characters are supplied by the schemas and are the same as those that the actors receive. These actions are multiple for each schema, players should not feel forced or railroaded in any way.

For example in a mystery drama players will frequently find themselves in an interrogation schema which provides all the core elements for the actors and players to participate in the schema. This includes a knowledgebase for the actors to understand what an interrogation is and what will be expected of their roles in that context. Additionally the actors receive any action (physical and speech) that is appropriate for that context, for instance asking about motive or means of the suspect. The actions received are generic and the actors and players need to fill it with knowledge from their own knowledge base. Generic action: *Ask about motive* may become: *Was Fred blackmailing you?*

3.3.2 Director

The director overlooks the emergence of the drama and uses schemas to direct the drama by giving the actors appropriate schemas to play out, in this way the director is the GM in regards to directing the emergence of an engaging drama.

The drama can not move between acts until the objectives of the acts have been adequately satisfied. As an example, If a drama goal for Act I is to introduce characters, the drama will not move from Act I to Act II until characters' key traits have been exposed. If a character is to be intelligent, playful and curious then the character needs to have played out actions that are intelligent for a value above a given threshold $\mathscr{T}$, and the same for playfulness and curiosity.

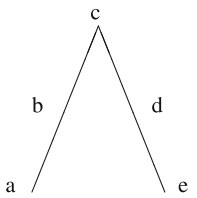


Fig. 3.5 Freytag's Pyramid or "dramatic arc" [12]

Skilfully winning a chess game or making 3–4 correct estimates about, for instance, the age of old furniture or showing good arithmetic skills would be sufficient. The algorithm summarises the percentiles to see if it has reached the threshold $\mathscr{T}$. The director's role is to give the actors an opportunity to show their characteristics by choosing schemas that would be a good fit.

The director uses known structures as an aid in picking a suitable schema to develop an interesting drama. Freytag's Pyramid or "dramatic arc", [12], as shown in Fig. 3.5 is very useful. This referred to as a 'dramatic arc' [27] in this chapter. The dramatic arc outlines the rise and fall that can be found in an engaging narrative. The story will start with an inciting incident, which aims to capture the audience interest (a). Followed by a steadily climbing suspense, as the plot thickens, in order to further captivate the audience (b). Until the story reaches its climax (c). Followed by the resolution as the plot uncurls and the player learns the full truth (d) after which the drama may reach closure (e).

The director is not planning every detailed move of the actors, but rather the overall dramatic structure which makes his planning of acts and episodes a much more tractable problem. As with PP-RPGs it is the players and Actor bots that drive the story through their actions rather than the Director or GM.

3.3.3 Schemas

Each schema has a finite set of roles that are annotated as being essential or non-essential. It is only necessary to fill the essential roles to successfully execute a schema; the non-essential roles add variety and increase flexibility. Each role is annotated with a finite set of characteristics that it supports. The characteristics also have a numerical value attached to them, this represents to what degree the display of this characteristic is supported by the role.

The director uses the set of characteristics to match the roles to actors, deploying the schemas that best compliment the various characteristics of the characters. The director is not in a good position to make decisions about direct interactions with the player, because the director would need to be constantly aware of everything that takes place—including the internal state of every character in the

drama. This would quite rapidly escalate into an intractable computation problem for the director.

3.3.4 Actors

Actors play characters in the drama, their main task is to decide on an appropriate action if any. There are three primary conditions for acting; *response to stimuli, response to internal process, response to a request to act made by other actors.*

3.3.4.1 Response to Stimuli

When an actor or a player acts, it will inform each of the actors that share the same scene of the action taken. When an actor is informed of an action then a separate internal thread first checks whether the action is directed at the actor and whether the action is detailed in any of the drama-schemas that the actor is currently in. If neither applies then the action is not relevant to the actor and the actor will not attempt a response, the actor will add any gained knowledge from the action to its knowledge base, (for instance information regarding the fulfilment of schema goals and info on the current location of other actors).

If the action is relevant to this actor then the actor will start a separate thread that evaluates an optimal response.

The algorithm is as follows:

For actor a_1 and actor a_2.
$-a_2$ Sends out a notification to all actors on the scene that it has taken an action.
$-a_1$ Reads the notification and realises that it was a question addressed to it.
$-a_1$ Computes a set of optimal responses to the question with respect to its knowledge base, character, situation, emotions and its current goals.
$-a_1$ adds the set of optimal responses to an array of other applicable actions with added weight to indicate priority.
$-a_1$ then evaluates what the set of optimal actions is and picks one to execute.

As seen in the algorithm, the actor needs to first add his set of optimal responses to an array containing other applicable actions because the actor may be preoccupied with something else. For example the actor may be talking on the telephone and not ready to answer.

3.3.4.2 Response to Internal Process

The actor will also act due to an internal process that initiates an action. This is to engage the player if the player has been inactive or if there are unfulfilled drama goals that will progress the drama but the player has not initiated actions that would lead to them. All actors have both drama related goals and character related goals. The drama related goals are of the form of aiding the drama progress as expected. For instance, actors have the goal of revealing relevant clues to the user. The actors also have character related goals such as showing specific characteristics or hiding their lack of alibi in order to portray a believable character.

3.3.4.3 Response to Request

During the unfolding of the drama each actor is responsible for both acting a convincing role and to fulfilling drama-specific schema goals [2]. For instance, in an interrogation where two characters John and David are present and David has a very strong false alibi then the actor bot playing David can make a request of the actor playing John that he has John reveal the flaw in the alibi as David would hardly sabotage his own cover directly. John might then say "How could you have caught the 08:15 train when I saw you at 08:20 in the garden". This does not require advanced logistics. The Bayesian networks facilitates this type of sentence generation so long as it is domain specific and the Bayesian networks facilitates such domain specific querying due to the fact that the BN is causally structured. The actor playing John searches for any knowledge in the specific domain that will increase the utility of the opportunity variable.

The algorithm is as follows:

Where David's actor is a_1 and John's actor is a_2.
$-a_1$ recognises that he needs to reveal that his alibi is faulty.
$-a_1$ recognises that he can't reveal that directly to the user in a believable way.
$-a_1$ sends a request to a_2 in the form of drama-goal: reveal opportunity of a_1.
$-a_2$ queries his knowledge base for actions that will satisfy the goal.
$-a_2$ adds any action that satisfies the goal in the request to an array of possible actions with added weight to indicate priority.
$-a_2$ then evaluates what the set of optimal actions is and picks one to execute.

As seen in the algorithm, a_2 will not necessarily act on the request. a_2 will first query its knowledge base for relevant actions and after adding them, with suitable weights, to the set of other relevant actions, a_2 will pick an action from the whole set

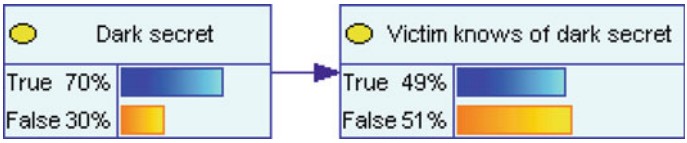


Fig. 3.6 A simple Bayesian net

of possible actions based on a_2's characteristics and other goals. This means that a_2 may or may not aid the requesting actor depending on context and a_2's priorities. This is necessary as it may not fit either a_2's knowledge base or a_2's current interaction with the user to comply with a_1's request.

3.4 Bayesian Networks

Bayesian Networks (BNs) are very good to reason about uncertainty such as in human behaviour because they provide the means to determine the probability of the various possible causal outcomes. The variables in a BN can represent anything ranging from system mechanics (e.g. whether there is fuel in a car or how fast the car can accelerate) to the intricacies of human traits and emotions. BNs offer the possibility of mapping the transiency of emotions and the resulting transient decisions that humans make. There has been a growing interest within Psychology and Philosophy as well as in Computer Science on using BNs to model human thinking [14, 26, 47]. We use causality extensively in our daily lives from deciding how to best manage to get the kids to school on time, to trouble shooting why the car will not start, and also when deciding on actions when highly influenced by emotions like anger or fear.

There are obviously many other AI techniques that have been used to try to emulate or simulate human like thinking, e.g. planning, search, neural networks etc. No AI method, to date, has been entirely successful. In fact all of them still have a long way to go to reach any sort of consistent believability for any length of time with minimum of complexity in the agents' environment. Since we are specifically making believable actor bots for drama based computer games, we are not really interested in fully emulating human like thinking but rather to give an appearance of human like reasoning. We discuss other approaches to achieve similar results in Sect. 3.5.

A BN is a directed acyclic graph made up of variables and arrows between the variables that indicate inference [22]. In Fig. 3.6 a simple Bayesian net is shown with two variables {*dark secret, victim knows of dark secret*}. These variables are found in the characters' knowledge bases and the former indicates whether a character has a dark secret or not and the latter indicates whether the victim knew of it, which could indicate the possibility that the victim was blackmailing the character which is a motive for murder. Each variable has two or more states that are mutually exclusive,

Table 3.1 The definition of the *victim knows of dark secret*

Victim knows of dark secret	Variable	
Dark secret	True	False
True	0.7	0
False	0.3	1

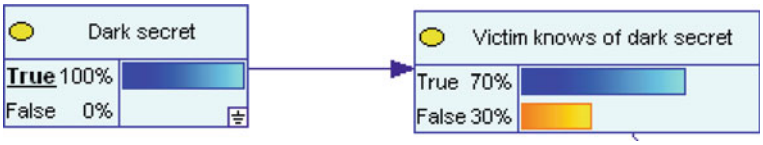


Fig. 3.7 Dark secret instantiated to true

in Fig. 3.6 both variables have the states {True, False}. The directed arrow indicates a causality between *dark secret* and *victim knows of dark secret*. The victim cannot know of any dark secret unless there is one.

The percentages shown in Fig. 3.6 are inferred values from calculating the network. There are underlying values that are defined in the network before calculating it. First we look at the *dark secret* variable which has no parent variable. Its defined values are simply {True = 0.7, False = 0.3}. The *victim knows of dark secret* variable is slightly more complex because it has *dark secret* as a parent, each state of the parent needs to be assigned a value, see Table 3.1

The percentage values that are assigned to each state are the inferred values from the net. For example, the value of {True} of the *victim knows of dark secret* variable will increase if the value of {True} of the *dark secret* variable increases. If *dark secret* state {True} is set to equal one (instantiated) then the inferred values change as can be seen in Fig. 3.7.

3.4.1 The BN Architecture

The variables in the actor bots BNs represent knowledge items such as hair colour, shoe size, relationship status, scenes and objects in the virtual world and the causal connections between these. Each state of a variable can be used to make a speech action. For example, in Fig. 3.8, the variable shoe size with states in the range 36–45 can be used by the actor bot to say that Kenneth's shoe size is 42. For example, the actor bot has been given evidence about basic attributes such as shoe size. This means that the actor bot has evidence for state 42 of the shoe size variable, see Fig. 3.8.

The actor bot can use this to form the speech "Kenneth's shoe size is 42" by mapping the state of the variable to the corresponding authored sentence, e.g. "{actor bot} shoe size is {value}" = Kenneth's shoe size is 42. If the actor bot is not certain, as in Fig. 3.8, the sentence can be Kenneth's shoe size *could be* 42. We format sentences

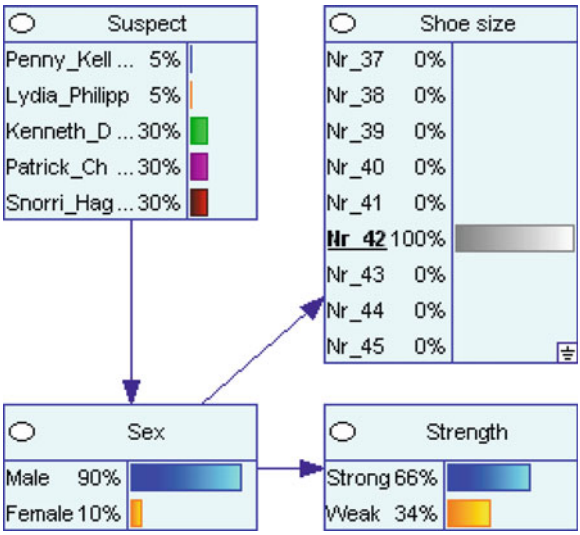


Fig. 3.8 A small example to demonstrate sentence creation

in this way by authoring them in XML and attaching tags that indicate what the input and goal variable is. In this example the input variable is shoe size and the goal is suspect.

The example in Fig. 3.8 is just a small piece of the whole net. We use Object Oriented BNs (OOBNs) [24] to create an actor bot's network from a script file containing descriptions of around 150 variables. These variables describe the bare essentials of a mystery plot such as the motive, means, opportunity and character features. From these variables a much larger network is created to represent the actor bot's beliefs of itself and of other actor bots in the drama. The actor bot also has beliefs of what each of the other actor bot's beliefs are of each other. The resulting BN has over 3,000 variables.

3.4.2 Complexity

Three thousand variables is not considered to be a very big network to implement a virtual actor bot. We expect that an average actor bot in an interactive drama could easily have a few hundreds of thousand variables. However, even this small BN takes several seconds to update with basic BN reasoning algorithms, which clearly is not sufficiently fast for real-time applications. Not only is it not fast enough but it also leaves no room for all the other processes, such as graphics and animation that are highly demanding on resources in any high graphics computer game.

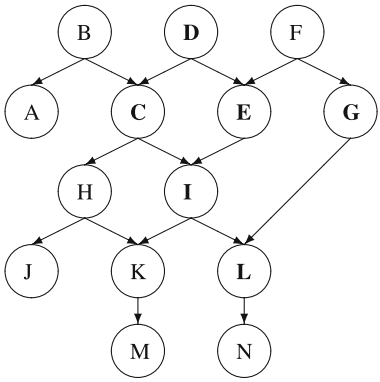


Fig. 3.9 A small Bayesian net to explain relevance reasoning

3.4.3 Relevance Reasoning

Relevance Reasoning is a well-known technique for BNs, that uses d-separation to determine which variables need to be updated to compute inference on one or more target variables given evidence [29]. D-separation holds when two distinct variables A and B in a network are not affected by each other receiving evidence. In other words, A can be given evidence without it affecting the inference of B and vice versa [22]. Relevance Reasoning identifies the variables that need to be updated based on which variables receive evidence and which variables become affected and are needed to correctly compute the inference of the target variables. This method greatly speeds up the updating of the network in the general case. The worst case is if all the variables in the network are affected by the evidence and are needed to compute the target variable. This means that in the worst case it is NP-hard.

For example if we look at the small BN in Fig. 3.9. If the input variable I is **C** and the target variable T is **L** then only variables C, D, E, G, I and L need to be updated to get correct values in **L**.

3.4.4 The Process

When an actor bot needs to decide on a speech action, the actor bot will take the following steps:

1. Find applicable sentences
2. Evaluate each sentence
3. Choose an optimal sentence

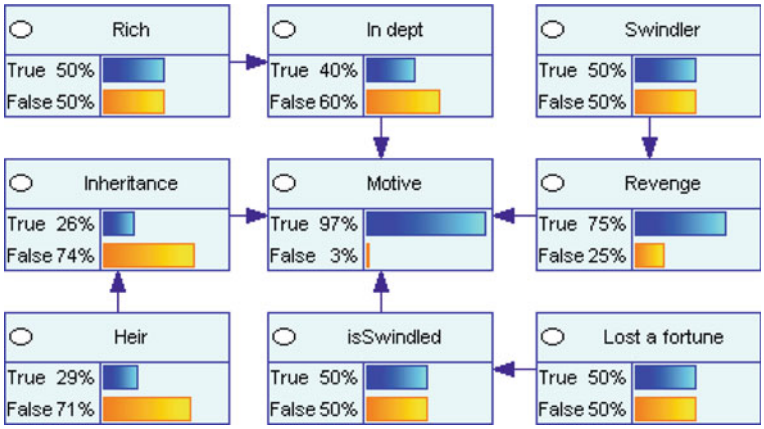


Fig. 3.10 A small example to demonstrate context

3.4.4.1 Applicable Sentences

First the actor bot finds a set of applicable sentences to evaluate, by querying the BN for a set of sentences that satisfy the goal and are contextual to the input. The input is any speech act that is causally connected to what was last said in the conversation. For example in Fig. 3.10 then all the variables are contextual to the $Motive$ variable. If any of their states is given value, it will affect the belief of the $Motive$ variable. For example if the $Lost\ a\ fortune$ variable is instantiated to true, it will positively effect the belief of the $IsSwindled$ variable, increasing the probability that the suspect was being swindled by the victim.

The actor bots first decides which actions are applicable. This the actor bot does by means of relevance reasoning as described above. T (target) is the goal of the actor bot and I (input) is for instance a speech action that the actor bot is evaluating to realise a goal. The actor bot uses T and I to extract the corresponding object O. O is extracted from a fully updated BN with the values that the variables in O had when in the BN. Because the BN is fully updated then the values in each variable are valid and up to date. The input values are then given as evidence to I and the values are updated for all the variables in O. The values of T are then read and their distance (difference between the intended values of the goal and the inferred values of the goal variable) from the goal is stored. Doing this repeatedly for all the actions that the actor bot is evaluating, results in a set of applicable actions. Using the net in Fig. 3.9 as an example, if the C variable is I and L is T then we extract (C, D, E, G, I, L) as O. C then receives evidence and only the variables in O are updated. The result is the values in the L variable.

For example, see Fig. 3.10, if we want a sentence that says that *motive* is true $= 100\,\%$ and if we put evidence in the *Lost a fortune* variable, true $= 100\,\%$. The sentence could be: "Kenneth could have a motive because he lost a fortune". The *motive* variable would then have true $= 98\,\%$ which makes the difference between

the target 100 % and the value 98 % be 2 which is stored with the possible sentence in an array of applicable actions. We choose only those actions that reduce the gap between the target value and the initial value.

3.4.4.2 Evaluate Sentences

When we have a set of applicable actions, that is actions that we have just extracted because they contribute towards the goal as discussed above. It now remains to filter the applicable actions for those actions that satisfy the character traits and mood. This can be accomplished with several existing personality models. For instance the personality model created by Ball and Breese for Microsoft [5]. The applicable sentences should be qualified in some way by what type of personality they cater to and how they will be influenced by emotion. We mark each possible action with a range of traits and emotions that the action would be characteristic of. This means that if the actor bot is angry then the actor bot will be more likely to choose actions that indicate anger such as accusing someone. The actor bot's traits such as intelligence or arrogance will also make the actor bot more likely to choose a sentence that demonstrates this. For instance an intelligent actor bot will explain why a suspect has a motive.

Using BNs each sentence is evaluated and those that best match the characters traits and emotions are entered into a set of optimal action. From the set of optimal actions one action is chosen at random and executed.

3.4.5 Tests and Results

Tests have been run in three ways, first is a test of computation speed, e.g. are the Bayesian algorithms sufficiently efficient to reach a decision within acceptable time limits, see Sect. 3.4.5.1. The second test is in the Second Life virtual reality, see Sect. 3.4.5.2. The third test is a web based game, see Sect. 3.4.5.3.

3.4.5.1 Computation

To test our method we generated a Bayesian network from a script that encompasses an actor bot's belief about itself, its belief about other actor bots and its beliefs about other actor bots' beliefs about other actor bots. This network contains 3348 variables, one decision variable and one value variable, 3,638 arcs, 10,606 states and 58,610 parameters.

The tests were run on a laptop with Windows XP, Intel(R) Core(TM)2 Duo CPU 2.59 GHz, 3.00 GB RAM.

We tested two set-ups, one without Relevance Reasoning and one with Relevance Reasoning, see Table 3.2. When there is no Relevance reasoning applied it took the

Table 3.2 Results

Queries (%)	Without RR in seconds	With RR in seconds
50	10–30 s	<1
30	30 s–3 min	<1
20	>3 min	<1

actor bot on average 10–30 s to decide on an action and more than 20 % of the decisions took over 3 min.

When we use relevance reasoning it never takes more than a second for the actor bot to complete the whole process and decide on a sentence.

3.4.5.2 Second Life

We need to take into account that the actor bots need to be animated in a virtual reality which needs a lot of resources and thus the AI computation may not take up all the time and resources available. We tested our implementation in the Second Life virtual reality [25] to see if the actor bots could perform at an acceptable speed when taking into account all the overhead of the virtual reality and animation that comes with it.

Second Life needs 25–33 % of the CPU time just for running the virtual reality display and no interaction is taking place. This means that the AI algorithm has reduced resources to manage the computations in.

The actor bots' responses should be a fluent reaction to user interaction. This means that response by the AI algorithms should not be so slow that the user becomes impatient and stops paying attention to the game.

In our implementation we needed to use a 5 s buffer to slow the actor bots down so that the user could keep up with what the actor bots were saying. Video clips showing a few of the test runs and explaining what is taking place can be found at http://youtu.be/HyVuHDSO1MI,http://youtu.be/dfFo8BNUap0, http://youtu.be/pfxz5RbmnFE.

3.4.5.3 Web Game

To test whether the output of the engine are coherent sentences that can be used in drama based game we created a web demo prototype mystery game. As the game starts the player is told that a murder has been committed and he enters the murder scene where he can examine the body and determine the cause of death. Each initiated game has a randomly generated cause of death and the BN is used to generate clues on the body to be found by the user that are logically consistent with the cause of death. For example there will be marks around the neck if the

Fig. 3.11 Screen shot of examining the body in the web game, note that what can be examined is also tailored by the BN depending on cause of death

victim was strangled and there may be strange odour if the victim was poisoned etc., see Fig. 3.11.

The player can then explore other rooms and interrogate suspects on their possible motives and attempt to determine which motive, if any, each suspect has. An example dialogue with two suspects shows how they are clearly different personalities, see excerpt below and in Fig. 3.12

Dialogue (Miss Jane Marple and Mr Bryan Eastley):

1. *Miss Jane Marple* - Did you lose a fortune?
2. *Mr Bryan Eastley* - what. Why? I have all my money
3. *Miss Jane Marple* - Did you have a reason to kill Harold?
4. *Mr Bryan Eastley* - Me? no..no reason!

Dialogue (Miss Jane Marple and Mr Cedric Crackenthorpe):

1. *Miss Jane Marple* - Did you lose a fortune?
2. *Mr Cedric Crackenthorpe* - No, I have all my money
3. *Miss Jane Marple* - Did you have a reason to kill Harold?
4. *Mr Cedric Crackenthorpe* - I assure you, I'm no murderer!

The game clearly demonstrated that the actor bots could act in character, e.g. show individual character traits and respond in a coherent manner to the questions asked by the player. More than 90 % of those who have played the game identify different character traits between the actor bots and find that the actors respond as they would expect murder mystery suspects in general to respond. The game is accessible online at http://www.weenrich.com.

The game is still quite simple but will be extended in the next few months to allow for more thorough testing with more detailed user evaluations.

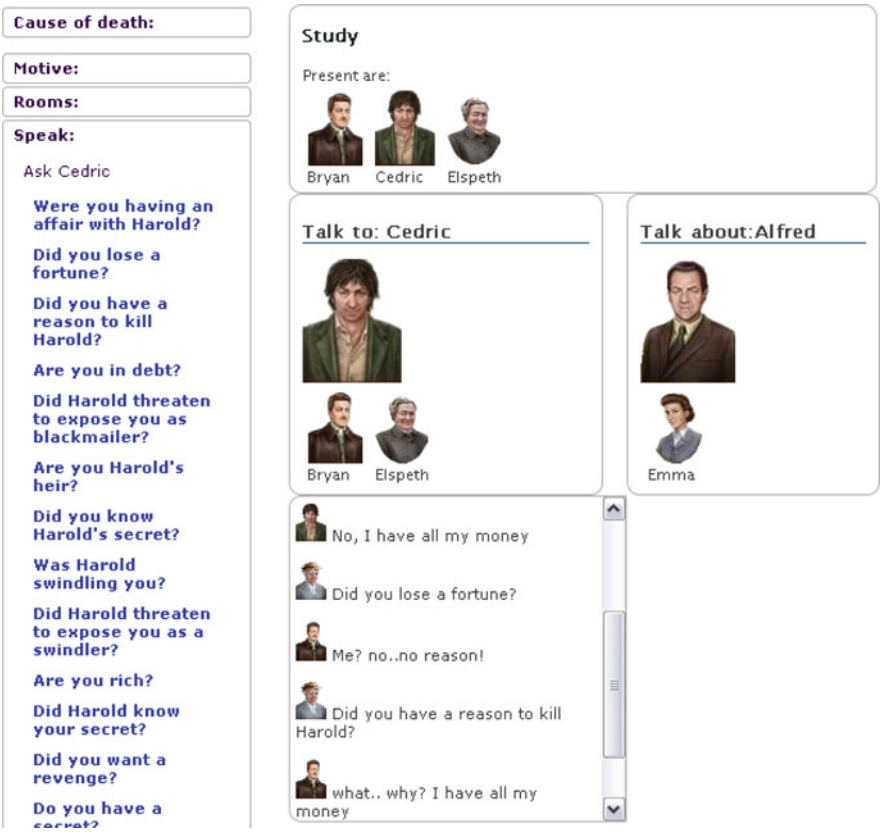


Fig. 3.12 A screen shot of a dialogue with suspects in the web game

3.4.6 Complications

The worst case scenario is that in order to update the target variables we need to update the whole network which is NP-hard. We reduce the chance of this by always having only a single target variable. Each target variable that is used in a query will need a certain portion of the BN to be updated given the input, the greater the number of target variables obviously the greater portion of the network needs to be updated and in the worst case the whole network would need to be updated. Instead of querying for many target variables simultaneously we query the net for each target variable and its inputs which allows us to reduce the size of the BN that needs to be updated. It is much more efficient to run multiple fast queries than a single large query that needs to update a large part of the BN.

Another complication is that if the output of the extracted subnet is needed in the big BN then the whole BN needs to be updated which means that our solution is not applicable for all instances, we still have not solved the problem of efficiently updating the whole network which will be necessary to some extent when the actor bots gain new information. Still the problem that we are looking at is to make the actor bot decide on a speech act and for this the actor bot does not need to update the whole BN.

3.5 Related Work

Impressive work in this area has been carried out at Southern California's Institute for Creative Technologies (ICT) and it is very well discussed in Chap. 4.

The most prominent and successful effort to take on complex social interactions is Façade [34]. Façade is an interactive drama where the player is asked to mitigate in a marital dispute between the players' friends. Short action sequences called beats control the sequence of events and these beats are explicitly pre-authored, with all actions within the beat being fully defined, and the actions of all roles being assigned to allow for multi-actor bot coordination [33]. Higher level goals for the characters are in these beats but the characters do retain autonomy in achievement of base-level goals and in performing actions such as facial expressions or personality moves [33]. The Façade characters are thus not fully autonomous bots, they are not acting as such but rather following directions from the drama manager and giving their actions a bit of personal character for believability.

FAtiMA (FearNot! affective mind architecture) is a character based emergent drama system [4, 38]. The drama emerges around character actions. The test base is FearNot! which is an anti bullying educational game to guide children in how to cope with bullying situations. The character agents are autonomous in interacting with the user by following a set of emotional reaction rules. These emotional reaction rules have pre and post condition and have appraisal values such as: desirability, desirability-for-other, and praiseworthiness [4, 21]. Using STRIPS-based partial-order continuous planner the characters evaluate the probability of success and importance of actions. Actions that are expected to generate the strongest emotional reaction are chosen.

FearNot! is the only system that has had proper user testing by an empirical study on 345 children, 172 male (49.9 %) and 173 female (50.1 %) between the ages of 8 and 11 [18]. The results showed that the children were able to empathise with the characters.

The Interactive Drama Architecture (IDA) [31] uses a director and a set characters that enact a fully structured story that is authored by a human author. The director in IDA gives direct commands to the characters, and the characters are semi-autonomous. They will act while they have no instructions from the director, for example drinking. They receive explicit directions from the director that have priority, for example 'perform dialogue #131 with John in the library and then run

away to another room' [32]. This is very different from DED. In DED the agents do not get such explicit directions and they can decide to not comply with suggestions from the director by simply not choosing actions in schemas because they are not optimal for the drama development.

The story consists of plot points in a partially ordered graph using STRIPS with pre- and post-conditions. There is some variation in ordering of plot points, such as where a certain scene can take place. Still the main story is fixed and the user is modelled to enable the drama manager to subtly guide them through the storyworld using actions such as; deniers, causers, creations, shifters, hints.

Similar to this is the Mimesis [41, 51, 52] system which was created as part of the work of the Liquid Narrative Group. In Mimesis an attempt is made to give the user the illusion of complete freedom. The system evaluates each user actions and decides whether the user's action can be "accommodated" or must be "intervened" with. If the action can be accommodated a new plan needs to be constructed to achieve the story's goal. If accommodation is not possible the system must intervene with actions similar to IDA. Rather than pre-planning the DED system uses the BNs to evaluate in response to user action how the situation can best be developed further into a structured drama experience. This means that there is no call for these director actions because no plan is violated.

The BARDS system uses a Heuristic Search Planner (HSP) with RTA* to plan for emotional development in the characters, rather than for actions [39, 40]. The group use an ontology created by Gustave Flaubert as the basis for the planner. Flaubert's novel, *Madame Bovary* [11], provides the test scenario. The user using natural language makes comments which may cause characters to react emotionally and this will influence the story, for instance madam Bovary may feel guilt when she is flirting with her friend and is reminded of her children. The effect will vary depending on the characters' feelings and situation. This is a novel approach in that the user is the audience that can influence how the story unfolds. Similar to watching a movie and when the heroine is about to enter a murder infested room the audience can say: "No stop you will be killed!" and the heroine can choose to obey or not depending on her feelings. They have shown that planning can be used for this type of emotional evaluation sufficiently efficient to run dramas spanning a few minutes. The architecture is in all main areas significantly different from DED.

The virtual storyteller [44, 45] is a story telling system that creates stories from character agents interactions with an emphasize on emergent improvisational theatre. The stories are told by a narrative agent. There is no interaction between the character agents and the player; the player can only interact with the story teller.

Actor bots are specifically designed for RPGs. This is contrary to other related research for video games which aims to make bots for FPS, adventure games and strategy games because they do not involve complex interactions between player and NPCs [46].

The aims for Icarus [8], an automated FPS combat player, are the basic components that are needed for bots in games such as FPS if they are to be believable human. These include a hierarchical organization of knowledge with multiple levels of abstraction that provides a rich, human-like vocabulary and strategies. The bots use goal-directed

and reactive behaviours and should support incremental learning and be able to use it later on in a believable fashion. Finally they should have the sufficient knowledge base to support complex reasoning. Icarus has a complex memory structure to support human like decision making based on well established cognitive theories. It does not accommodate emotion or characteristics though and its main goal is not interaction with the user in form of acting out an engaging drama like the actor bots.

This is the case with other similar bots for FPS and adventure games. Their aims are to play against the player rather than to be a companion or entertainment for the player as in Chaps. 7 and 8.

Imitating or learning from human players, as in Chap. 6 and 12, has seen some success in action based games such as in 3D boxing simulation game [35] and in FPS [16, 20, 53]. In the 3D boxing case based imitation showed a clustering of behaviour around that which the human player performed. There was a clear difference in the play of a bot imitating an experienced or weak player [16]. Use a Bayesian based probability function in a goal oriented bot. The bot imitates that actions of a human player and forms goal oriented strategies based on its observations [20]. Use pre-set probability distribution tables that the bot utilises with a Bayesian function. The bot can then learn new behaviours and playing styles by imitating other players. Finally [53] use actor bot modelling to learn good combat strategies for RPGs. All of these are successful in imitating combat actions of human players but do not contribute towards a more complex interactions as those handled by the actor bots.

3.6 Summary and Future Work

We have seen that current computer games are very good at computing character statistics and there are many quality games for power gamers and tacticians. Currently the game market has not been able to meet the demands of method actors and specialist or players that simply want a good flexible story. The reason is that current games tend to railroad players, forcing them to follow pre-authored stories.

It is the lack of a truly vicarious adventure that the Directed Emergent Drama (DED) engine aims to fill by facilitating the emergence of a structured drama from player and actors' interactions. In this chapter we shared how the DED engine facilitates similar experiences as provided by Pen and Paper Role-Playing Games, due to parallels in the fundamental approach to interactive drama.

Increasing basic role-playing concepts in Computer Based Role-Playing Games is very likely to increase player satisfaction by appealing to a larger audience and by increasing the degree of engagement and immersion. This remains to be tested with empirical user studies.

In this chapter, we presented real-time applicable algorithms based on relevance reasoning that are both efficient and scalable. Using Bayesian Networks (BNs) is a viable choice to represent the knowledge base and the characteristics of an intelligent agent. Since these agents are likely to need vast BNs, it is necessary for them to be able to decide on an action without updating a BN with possibly hundreds of thousands

of variables. This we do by extracting the relevant subnet from the main BN and only updating the values in the subnet. Using d-separation and context we greatly reduce the size of the network that needs to be updated to compute inference of a target variable each time, effectively removing the exponential growth of BNs. Care needs to be taken to preserve d-separation where needed and to realise that the result needs to be propagated to the network when its influence on BN as a whole is needed.

We still have not worked out a way to update the main BN with information gained by the agent. This is a prominent problem that needs to be looked at. There are some possible solutions including hierarchical BNs and updating the BN in stages in separate threads using d-separation or clustering to divide it into ordered subnets that can be updated independently and the resulting values propagated to the neighbouring subnets.

In order to fully demonstrate scalability, the system needs to be applied to a larger domain. In particular, more content should be added so that the resulting BNs are both larger and more interconnected. Additionally adding greater complexity in the form of more traits and emotions will be good to test the bot's decision making. We expect that for a reasonably large drama an actor bot could have a knowledge base consisting of tens or hundreds of thousands of variables and it clearly remains to be solved how such a large BN would be authored by a non-programmer.

As future work it will be interesting to include psychosocial behaviours and deeper personality and emotional reaction such as described in Chap. 2. The framework defined there would go very well with a Bayesian Network.

Additionally it would be constructive to explore how Recursive Modelling Method (RMM), as described in [15], can aid in giving the agents a more complete theory of mind without needing to go into full game theory and the subsequent search for Nash Equilibrium [36].

The engine is equally able to operate on physical actions as on speech actions and in our future experiments we will add this.

References

1. Arinbjarnar, M., Kudenko, D.: Schemas in directed emergent drama. In: Proceedings of the 1st Joint International Conference on Interactive Digital Storytelling (ICIDS08), Erfurt (2008)
2. Arinbjarnar, M., Kudenko, D.: Duality of actor and character goals in virtual drama. In: Proceedings of the 9th International Conference on Intelligent Virtual Agents, Amsterdam (2009)
3. Arinbjarnar, M., Kudenko, D.: Bayesian networks: real-time applicable decision mechanisms for intelligent agents in interactive drama. In: Computational Intelligence and Games (CIG), Copenhagen, Aug 2010
4. Aylett, V., Dias, J., Paiva, A.: An affectively driven planner for synthetic characters. In: Proceedings of International Conference on Automated Planning and Scheduling (ICAPS06), UK (2006)
5. Ball, G., Breese, J.: Relating personality and behavior: posture and gestures. In: Paiva, A. (ed.) Affective Interactions, pp. 196–203. Springer, Berlin (2000)
6. Bioware. Newerwinter nights. http://nwn.bioware.com/ (2002)

7. Bohus, D., Horvitz, E.: Dialog in the open world: platform and applications. In: Proceedings of the 2009 international conference on Multimodal interfaces (ICMI-MLMI '09), pp. 31–38. ACM, New York (2009)
8. Choi, D., Konik, T., Nejati, N., Park, C., Langley, P.: A believable agent for first-person shooter games. In: Proceedings of the Third Annual Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 71–73. AAAI Press, Stanford (2007)
9. Sony Computer Entertainment. Heavy Rain, http://www.heavyrainps3.com/ (2009)
10. FASA. Star trek: The role playing game (1982)
11. Flaubert, G.: Madame Bovary. France, Paris (1856)
12. Freytag, G.: Technique of the Drama. Benjamin Blom, New York (1863)
13. Goodman Games. Etherscope, http://www.goodman-games.com/etherscope.html (2006)
14. Glymour, C., Danks, D.: Reasons as causes in bayesian epistemology. J. Philos. **104**(9), 464–474 (2007)
15. Gmytrasiewicz, P.J., Durfee, E.H.: A rigorous, operational formalization of recursive modeling. In: International Conference on Multiagent Systems, pp. 125–132 (1995)
16. Gorman, B., Thurau, C., Bauckhage, C., Humphrys, M.: Believability testing and bayesian imitation in interactive computer games. In: From Animals to Animats 9, vol. 4095, pp. 655–666. Springer, Berlin (2006)
17. Gygax, G.: Dungeon Module B2: The Keep on the Borderlands—Introductory Module for Character Levels 1–3. Berkeley Top Line Distributing, Berkeley (1980)
18. Hall, L., Woods, S., Aylett, R., Newall, L., Paiva, A.: Achieving empathic engagement through affective interaction with synthetic characters. In: Proceedings of the International Conference on Affective Computing and Intelligent Interfaces, pp. 731–738. Springer, Berlin (2005) (LNCS 3784)
19. Horvitz, E., Paek, T.: A computational architecture for conversation. In: Proceedings of the Seventh International Conference on User Modeling (UM'99), pp. 201–210. Springer, New York (1999)
20. Le Hy, R., Arrigoni, A., Bessiere, P., Lebeltel, O: Teaching bayesian behaviours to video game characters. Robotics and Autonomous Systems (Elsevier) Special issue: Robot Learning from Demonstration, **47**, 177–185 (2004)
21. Dias, J., Paiva, A.: Feeling and Reasoning: A Computational Model for Emotional Characters. In: 12th Portuguese Conference on Artificial Intelligence (EPIA 2005), pp. 127–140. Springer, Berlin (2005)
22. Jensen, F. V.: Bayesian Networks and Decision Graphs. Springer, Berlin (2001)
23. Juul, J.: Games telling stories? Int. J. Comput. Game Res. **1**(1) (2001)
24. Koller, D., Pfeffer, A.: Object-oriented bayesian networks. In: Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, pp. 302–313. San Francisco (1997)
25. Linden Lab. Second Life, http://secondlife.com/, June (2008)
26. Lagnado, D.A., Waldmann, M.R., Hagmayer, Y., Sloman, S.A.: Beyond covariation: cues to causal structure. In: Gopnik, A., Schultz, L. (eds.) Causal Learning: Psychology, Philosophy, and Computation, 1st edn. Oxford University Press, USA, (2007)
27. Laurel, B.: Computers as Theater. Addison-Wesley Publishing Company, Boston (1993)
28. Laws, R.D.: Robin's Laws of Good Game Mastering. Steve Jackson Games, Austin (2002)
29. Lin, Y., Druzdzel, M.: Computational advantages of relevance reasoning in bayesian belief networks. In: Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI-97), pp. 342–35. Morgan Kaufmann, San Francisco (1997)
30. Bethesda Softworks LLC. Fallout 3. http://fallout.bethsoft.com/index.html (2002)
31. Magerko, B.: Story representation and interactive drama. In: Proceedings of the Artificial Intelligence and Interactive Digital Entertainment conference (AIIDE) (2005)
32. Magerko, B.: Player modeling in the interactive drama architecture. Ph.D. thesis, The Department of Computer Science and Engineering, University of Michigan (2006)
33. Mateas, M.: Interactive drama, art, and artificial intelligence. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh. Technical Report CMU-CS-02-206 (2002)

34. Mateas, M., Stern, A.: Build it to understand it: ludology meets narratoloy in game design space. In: Proceedings of the Digital Interactive Games Research Association Conference, Vancouver. Included in the Selected Papers volume, June 2005
35. Mozgovoy, M., Umarov, I.: Building a believable agent for a 3D boxing simulation game. In: 2010 Second International Conference on Computer Research and Development, pp. 46–50. May 2010
36. Nash, J.: Non-Cooperative Games. Ann. Math. **54**(2), 286–295 (1951)
37. Oxford, English: and Dictionary. Oxford University Press, Oxford English Dictionary (2007)
38. Paiva, A., Dias, J., Sobral, D., Aylett, R., Sobreperez, P., Woods, S., Zoll, C., Hall, L.: Caring for agents and agents that care: building emphatic relations with synthetic agents. In: Proceedings of the Autonomous Agents and Multi-agent Systems (AAMAS), ACM Press, UK (2004)
39. Pizzi, D., Cavazza, M.: Affective storytelling based on characters' feelings. In: Proceedings of the AAAI Fall Symposium on Intelligent Narrative Technologies, Arlington, Nov 2007
40. Pizzi, D., Charles, F., Lugrin, J., Cavazza, M.: Interactive storytelling with literary feelings, In: Proceedings of the Second International Conference on Affective Computing and Intelligent Interaction (ACII), Lisbon, Sept 2007
41. Riedl, M.O., Saretto, C., Young, R.M.: Managing interaction between users and agents in a multi-agent storytelling environment. In: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2003)
42. Schiesel, S.: Dungeon masters in cyberspace. The New York Times (2006)
43. Schiesel, S.: Gary Gygax, game pioneer, dies at 69. The New York Times, 157(54240) (2008)
44. Swartjes, I., Vromen, J.: Emergent story generation: lessons from improvisational theater. In: Proceedings of the AAAI Fall Symposium on Intelligent Narrative Technologies, Arlington, Nov 2007
45. Swartjes, I., Vromen, J., Bloom, N.: Narrative inspiration: using case based problem solving to support emergent story generation. In: Proceedings of the International Joint Workshop on Computational Creativity, Goldsmiths, University of London, June 2007
46. Tencé, F., Buche, C., De Loor, P., Marc, O.: The challenge of believability in video games: definitions, agents models and imitation learning. CoRR, abs/1009.0451 (2010)
47. Tenenbaum, J.B., Griffiths, T.L.: Structure learning in human causal induction. Adv. Neural Inf. Process. Syst. **13**, 59–65 (2001)
48. Thue, D., Bulitko, V., Spetch, M.: Passage: a demonstration of player modelling in interactive storytelling. In: The Fourth Conference on Artificial Intelligence and Interactive Digital Entertainment, AAAI Press, Palo Alto (2008)
49. Ubisoft. Assassin's Creed II, http://assassinscreed.uk.ubi.com/assassins-creed-2/ (2009)
50. GSC Game World. S.T.A.L.K.E.R., http://www.stalker-game.com/ (2009)
51. Young, R.M., Riedl, M.: Towards an architecture for intelligent control of narrative in interactive virtual worlds. In: Proceedings of the International Conference on Intelligent User Interfaces, Jan 2003
52. Young, R.M., Riedl, M.O., Branly, M., Jhala, A., Martin, R.J., Saretto, C.J.: An architecture for integrating plan-based behavior generation with interactive game environments. J. Game Dev. Vol. 1 (2004)
53. Zhao, R., Szafron, D.: Learning character behaviors using agent modeling in games. In: Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-09), pp. 179–185, Oct 2009

Chapter 4
Embodied Conversational Agent Avatars in Virtual Worlds: Making Today's Immersive Environments More Responsive to Participants

Jacquelyn Ford Morie, Eric Chance, Kip Haynes and Dinesh Rajpurohit

Abstract Intelligent agents in the form of avatars in networked virtual worlds (VWs) are a new form of embodied conversational agent (ECA). They are still a topic of active research, but promise soon to rival the sophistication of virtual human agents developed on stand-alone platforms over the last decade. Such agents in today's VWs grew out of two lines of historical research: Virtual Reality and Artificial Intelligence. Their merger forms the basis for today's persistent 3D worlds occupied by intelligent characters serving a wide range of purposes. We believe ECA avatars will help to enable VWs to achieve a higher level of meaningful interaction by providing increased engagement and responsiveness within environments where people will interact with and even develop relationships with them.

4.1 Introduction

Embodied conversational agents (ECAs) refer to intelligent programs that are delivered through an embodied persona, typically a two- or three-dimensional graphic entity that has the ability to converse via text or speech with a human interactor, either on the web or via standalone code running on a computer [6]. Among Artificially Intelligent (AI)-driven entities, ECAs not only have conversational interactions with humans, they are often able to remember information about those interactions from session to session [2]. They incorporate techniques from Natural Language Processing and include a corpus of knowledge that can cover a single topic or span multiple ones [3, 11]. Some ECAs, often called Virtual Humans (VH), can include specialized reasoning and even emotions [12, 23]. The newest form of ECAs are those deployed within persistent virtual worlds, where they are embodied as 3D avatars,

J. F. Morie (✉) · E. Chance · K. Haynes · D. Rajpurohit
Institute for Creative Technologies, University of Southern California,
Playa Vista, CA 90094 , USA
e-mail: morie@ict.usc.edu

P. Hingston (ed.), *Believable Bots*, DOI: 10.1007/978-3-642-32323-2_4,

the same representations used by human participants. Since these virtual world AI entities share the same constraints and stylistic designs of avatar embodiment and interaction as the users of the virtual world, we have elected to call them embodied conversational agent avatars (ECAAs). Unlike their more well-known ECA counterparts, ECAAs must be able to adapt to an environment that changes over time, and recognize a wide range of users embodied by different forms of avatars. They must remain persistent in a world that is theoretically never turned off, yet be able to restart automatically should that world reboot for some reason.

Virtual human research has progressed rapidly over the last 15 years [1, 14, 30, 32, 33]. Yet, Embodied Conversational Agent Avatars are still in their infancy, with the first ones being implemented only around four years ago [15]. As the articles in this book show, making better and more functional bots is a prominent research topic. This chapter focuses primarily on intelligent agent avatars in virtual worlds, and especially the work being done at the University of Southern California's Institute for Creative Technologies (ICT), which has a large research effort in both VHs and ECAAs. ECAAs can be considered sophisticated conversational bots that look like other inhabitants of the world and interact meaningfully with humans within that space. Because of this, they can contribute to more robust scenarios in virtual worlds, covering a wide range of topics, from training to health.

Today's agent avatars in virtual worlds are the result of a merger of 3D virtual reality environments with interactive artificially intelligent agents. These two technologies started as separate lines of research, and in the last decade have come together to mutual advantage.

Virtual reality (VR) technology provides digitally fabricated spaces where we can experience that which we cannot in real life, whether we are barred from it though distance, temporal displacement or its imaginary nature. VR relies on building, through the use of computer graphics and specialized viewing systems, complete environments that our disembodied selves can traverse as if we were really, truly there [25]. The task of early VR researchers was to find ways to convince humans of the believability of these digital spaces built only with computer graphics tools, and no physical materials. Much of the research focused on how to bring the viewer inside that intangible world. Researchers designed displays that shut out signals from actual physical reality and replaced these with malleable and controllable computer graphics [13]. Zeros and ones became digital sirens that fooled our minds by providing experiences that stimulated our neural circuits in ways very similar to actual reality. Unlike actual reality, however, it was a boundless expandable frontier, limited only by the creator's imagination.

The other area of research was artificial intelligence (AI), which focused on making machines truly intelligent. Rather than creating spaces we could inhabit, the early AI community sought to capture the internal mechanisms of human thinking processes within the confines of a computer system [28]. The overarching concept here was to understand how the brain worked, and to then make a machine appear smart in a way that mimicked basic human intelligence. One trajectory of this research was to develop programs that could exhibit intelligence and interact with humans

in a conversational manner. Early efforts concentrated on text-based queries and responses, with a human asking a question and the machine answering as if it was a perceptive, thinking entity. Weizenbaum's early program, Eliza, very nearly did the trick—more than one person was convinced his interactions were with a real person rather than a computer program [39]. But it was a thin disguise and these early so-called "chat bots" began to evolve into more sophisticated systems through dedicated hard work and ongoing advances.

Each of these technologies—VR and AI—struck a chord with the general public, which began to believe that computers could do almost anything conceivable. Escape the real world? Download your brain into a computer? No problem! This led to unrealistic expectations and researchers simply could not keep pace with the hype generated by public excitement fed by the overactive imagination of the press, science fiction books and even film. A period of disillusionment set in [26]. However, researchers entering the second decade of the 21st Century have moved beyond these issues and are forging ahead on the paths early visionaries trod only in their dreams.

Going beyond Eliza's model of a disembodied conversation with computer programs masquerading as a Rogerian psychotherapist, a key group of people realized that conversing with a visible character would enhance the interaction between human and machine [8]. In the 1990s, these intelligent virtual characters began to take on graphical forms that could visually depict human-like actions and responses during conversational interactions. Unlike the more sophisticated depictions of computer-generated humans that were part of movies (for example, the film Final Fantasy in 2001) where each frame of a character's existence took hours to create, these AI virtual humans had to run in real time to support their interactive nature. This task was difficult given the capabilities of the computers of that time. Therefore, real time depictions were necessarily less about realism and more about behavioural believability.

Computer games were also quickly advancing during this time, and game makers adopted techniques from many domains, including VR and AI. As a real time interactive medium driven by a new generation of demanding players, these games pushed the envelope of realtime graphics while also incorporating some basic forms of intelligence into their systems. Most of these AI resources were allocated to the behaviours of non-player characters, including rudimentary player interaction and simple pathing algorithms [31]. However, a few AI characters had "starring" roles. The SIMS, for example, while not a goal-driven game, stands out as a prime example of characters acting with complex human-like behaviours via scripted rule-based AI, decision trees and neural networks [22]. These characters were given basic intelligence, beliefs and goals commensurate with the needs of the game system. People developed strong associations with these virtual "humans" [4].

Another example of a game AI in a "starring" role is "The Creature" from the 2001 strategy game Black and White (developed by Lionhead Studios). It was widely acknowledged as the best example of character game AI for its time. With intelligence designed by Richard Evans, The Creature was programmed to do your godly bidding as your representative on a planet. The Creature learned from the way you interacted with, rewarded and punished it. It used an AI architecture called BDI, which provided

much of the same functionality used in the SIMS. Black and White became an immensely popular game, due in part to its sophisticated character AI [9].

By the first decade of the 21st Century, VR was supplanted by games in the popular imagination because these were much easier to access and did not need any specialized display equipment. Some games used multiple players in connected, persistent, and easily accessible graphic worlds, such as World of Warcraft and Final Fantasy, and were therefore known as networked games. Parallel to these, open-ended virtual worlds emerged, which depended more on social interactions than quests or game mechanics. Though the first implementations happened a decade earlier virtual worlds have become popular recently, with hundreds of existing worlds and millions of users (see current statistics from the marketing firm KZero Worldswide at http://www.kzero.co.uk/universe.php). Unlike VR environments, which are typically set up in a laboratory or clinician's office where one must be physically present, these virtual spaces need no special equipment to experience, and can be accessed from anywhere, on any standard computer connected to the Internet. These worlds are always running, and therefore they grow, change, and evolve as people use them, with the participants themselves often customizing the world for their own purposes. Placing ECAAs within them involves special challenges such as keeping them running in a persistent world and giving them functionality to interact with a wide and constantly changing variety of users.

4.2 Advancing Intelligent Virtual Human Research

4.2.1 Chatterbots

As noted previously, the earliest virtual agents were non-graphical conversational characters comprising computer programs that could typically "understand" text typed by a human. The program achieved this by matching key words to a database of responses. Such interactions were often limited in scope and lacked the richness of normal in-person communication. The common term given these autonomous software programs was chatterbots, chat bots or "bots" for short [24].

4.2.2 Embodied Conversational Agents

The next advancement was to depict these interactive agents in some visual form. These "embodied conversational agents," or ECAs, did more than make the agent a person—they set the stage for understanding that more complex requirements were needed to support the believability of an embodied agent. For example, a personality of sorts could be achieved by writing clever interactive dialog, but the visual depictions also needed to reflect this in some way. Each stage of development revealed a new understanding of the cues we take for granted in face-to-face communication.

Much more work was needed in combining AI and character depictions to make a visual agent appear convincing.

Researcher Justine Cassell describes ECAs as "multimodal interfaces" that implement as many of the usual elements humans use for communication as possible. These can include speech, gestures and bodily movements, facial animation, and more complex aspects such as subtle body language and responses with embedded emotional overtones [7].

Research in ECAs started in earnest in the late 1990s. Several investigators and institutions took the lead in advancing the state of the art. In addition to Dr. Cassell and her colleagues (then at MIT) advanced work was being done by Joe Bates's team at Carnegie Mellon University, and at the Information Sciences Institute (ISI), part of the University of Southern California. The work at ISI brought the virtual character Steve to life, which served as a pedagogical agent who could interactively teach you about the operation of a ship's control panel (see Fig. 4.1).

Steve was aware of whether or not you were paying attention to his training, and would urge you back on task via a synthesized voice. Steve was one of the early forms of a pedagogical agent that actually possessed a 3D animated body (albeit without legs!) and this opened up new avenues of engagement with pupils using virtual training environments [14].

4.2.3 Modeling Human Communication

The global research question was this: How many of the aspects of face-to-face human interaction can be simulated via ECAs, and what disciplines might inform

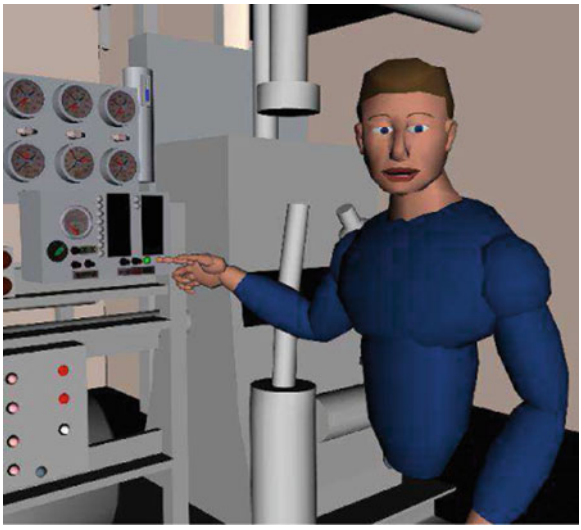


Fig. 4.1 Steve in the machine room of the ship

the implementation of these affordances? Dr. Cassell advocated the study of human communication as a preliminary, necessary step to creating believable intelligent embodied agents. She identified the following basic functional requirements:

- The ability to recognize and respond to verbal and non-verbal input.
- The ability to generate verbal and non-verbal output.
- The ability to deal with conversational functions such as turn taking, feedback, and repair mechanisms.
- The ability to give signals that indicate the state of the conversation, as well as to contribute new propositions to the discourse [5].

In the 2001 AAAI Symposium, ISI researcher Jeff Rickel described the development of autonomous agents that can take on a variety of human roles as "a daunting task" that demanded integration of several core technologies, some in place and some requiring development. All, however, were focused on creating a common representation and exposition of task knowledge [30]. In 2000, several members of the virtual human research group at ISI moved to the newly formed USC Institute for Creative Technologies (ICT), which was tasked with building better and more immersive simulations for military training. A key aspect of this effort, the group realized, was to populate the 3D training environments with believable characters playing various roles to enhance their immersive and compelling aspects. This focus expanded upon the Steve project already developed at ISI, and enabled new collaborative areas of investigation, taking the research into increasingly advanced levels of sophistication.

4.2.4 Multidisciplinary Approach

From its inception, the ICT built virtual humans as a multidisciplinary endeavour, and therefore the characters not only began to interact more naturally with their real world counterparts, they did so within more complex virtual environments. According to the head of the virtual human research team, Dr. William Swartout, the goal was to create "virtual humans that support rich interactions with people [to] pave the way toward a new generation of interactive systems for entertainment and experiential learning" [14].

This required integrating several core technologies as building blocks that worked together to form a more multifaceted complex agent. No mere chat bots, the research centred around creating agents that were visual, gestural, aware of their surroundings, and even able to exhibit emotions that could be changed on the fly. Over a decade of research and development at the ICT resulted in smart pedagogical agents that could help people develop skills in task analysis, negotiation, decision-making, and complex leadership skills. They were given sophisticated appraisal systems and embedded goals. They featured complex emotional modeling and could get angry, or refuse to go along with your plans. They integrated task representation and reasoning along with natural language dialogue. They appeared as convincing visual representations, with realistic behaviours including subtle movements like body language,

idle behaviours, facial expressions, gaze models and the like. These virtual humans knew about their environment and the other persons within it (both real and virtual). They were programmed with the basic rules of face-to-face spoken dialogue. They could show nonverbal behaviours that people exhibit when they have established rapport. They could understand both text and spoken word, and even deal with off-topic remarks. In short, these virtual intelligent agents combined a broader range of capabilities than any other work being done at that time [14, 20, 34] (see Fig. 4.2).

ICT's virtual human architecture includes a number of components listed below that support the agents' intelligent behaviours [21]. The simplest question-answer agents use the first three components; more complex agents can use all the components listed.

- Speech Recognition: parses the trainee's speech and produces a string of text as output.
- Natural Language Understanding: analyzes the word string produced by speech recognition and forms an internal semantic representation.
- Non-verbal Behaviour Generation: takes the response output string and applies a set of rules to select gestures, postures and gazes for the virtual characters.
- Intelligent Agent: reasons about plans and generates actions. Simple question and answer agents use the NPC Editor, whereas complex agents are created using other cognitive architectures. The agents contain task models, a dialogue manager and a model of emotions.
- SmartBody: an in-house animation control system that uses Behavioural Markup Language to perform advanced control over the characters in the virtual environment by synchronizing speech output with gestures and other non-verbal behaviour [37].

Fig. 4.2 Typical ICT virtual human agents from 2007

Fig. 4.3 The SimCoach character, Bill Ford, from 2010

- Real Time Graphics: a range of commercially available game engines are used for building the custom environments and as the foundation for real time rendering. As of this writing, Gamebryo and Unity are the most widely used engines at the ICT.

ICT's newest virtual human project, SimCoach, provides three versions of a web-based intelligent coach to encourage people in need of help to take a first, anonymous step towards finding that help. It will assist veterans in locating medical advice from a variety of online and staffed resources. Three coach figures have been designed to appeal to military personnel: a retired Vietnam veteran (Fig. 4.3), a female aviator and an active duty soldier. Special attention is being given to the facial expressions and non-verbal body movements to provide a high level of behavioural believability. The SimCoach can answer typed queries in real time, while the movements are procedurally triggered via SmartBody markups to the dialog. As the first ICT virtual human to be deployed via the Internet, SimCoach takes interactive ECAs to the next level [17].

4.3 Convergence: Virtual Agent Avatars in Real-Time Virtual Worlds

As discussed above, the most advanced virtual humans or intelligent agent applications have been achieved in custom-built environments designed for a specific purpose, providing a bounded framework to contain the extent of knowledge and interaction the character must provide. This is true even for the SimCoach. However, when the environment is more open, the domain more permeable, or the world in which the agent exists subject to ongoing change, it becomes more difficult to create intelligent characters that believably populate those spaces. This is the challenge faced when operating ECAAs within virtual worlds.

4.3.1 Challenges of Virtual Worlds

Networked, persistent VWs are not perfect. A person can enter a virtual space one time and find a group of interesting people interacting, chatting and doing things together. Another time, the space might be devoid of people, leaving one to wonder what to do. Some spaces themselves can be confusing with little clues regarding the purpose of their construction. Having to self-discover what to do in a space can lead to frustration. The task of discerning how to interact with the virtual world's affordances, control one's avatar, and navigate the space is often overwhelming to a first time user. The luckiest initiates learn from their friends, who take them "in hand" and see that they are mentored through all the basics. The unlucky ones may join the ranks of a rather large drop-out statistic. In 2007 Linden Lab, the company responsible for Second Life (SL), one of the most popular VWs today, reported that their drop out rate for new users—those who logged in once but never again—was a rather shocking 90 % [35].

4.3.2 ECAAs as Solutions

Adding ECAAs to virtual worlds seems like one obvious solution to these issues because they have the same embodiment and interaction potential as real users. Such agents can serve as helpers, information resources, orientation aids and virtual tour guides. In addition, ECAAs may be employed indefinitely to maintain spaces created for a specific purpose, whereas employing live support staff for the same task may be untenable. This approach makes a great deal of sense given the world-wide base of VW users and the expansive nature of their spaces.

ECAAs can serve educational purposes as well. In fact, any of the purposes for which virtual humans or intelligent agents have been created can be duplicated within the virtual world with embodied agent avatars. However, in 2007, worlds like Second Life made surprisingly little use of any form of agents, or their simpler cousins, chat bots. They were not part of the offerings of Linden Lab—the company that created Second Life—whose focus was on building a platform that encouraged user-generated content such as buildings, clothes, furniture and the like—merchandise that could be used primarily for commerce.

The first SL avatar-based bots were used to model clothing in virtual shops. One was instructed not to talk to those models; they were just there to show how the clothes would look on a 3D figure. So they were useful, but not intelligent and certainly not conversational. Other practical uses for bots were to monitor visitors within a space, using the built in aspects of an avatar to gather information and the like. Less sanctioned uses included using them to increase the visitor traffic count and make your area appear to be more popular than it actually was [36].

4.3.3 Using Virtual Worlds as ECAA Research Platforms

The ICT recognized a great opportunity for expanding its expertise in creating virtual humans to the virtual world domain. VW space seemed like an ideal platform for importing some or all of ICT's virtual human technology. Not only would the virtual world provide the underlying graphics and physics engine for little or low cost (until now we had used game platforms such as Gamebryo and Unreal Tournament), avatar agents could be designed with much less overhead (no building the 3D characters or having to animate them), allowing more focus on their intelligence and conversational attributes.

The virtual world, especially Second Life, also offered intrinsic benefits of greater availability, affordability (and free to users), an in-world scripting language for data gathering and other peripherals, persistent usage, and not having to bring participants into a research lab for interaction. It also provided a rather large pool of virtual world residents who could potentially interact with agents we might deploy. Even though reports of its demise have been rumoured, SL continues to be a very stable platform with tens of thousands of users at any given time [29].

With these thoughts in mind, in 2008 we set about adapting some of the technology behind ICT's virtual humans to create ECAAs within Second Life. This was made possible, in part, by leveraging an open source software library then known as libsecondlife (now called libOpenMetaverse). This software enables communication with the servers that control the virtual world of Second Life.

We had already built a large Iraqi Village in SL for a previous project, but the village seemed quite empty and dull when no one was around. We chose this location for a proof of concept exercise and filled it with simple avatar-based bots acting as villagers, who went about their daily activities in a scripted fashion. For example, a mother and her son would shop at the various stores in the market place, conversing with several bot shopkeepers (see Fig. 4.4). An old man would walk through his

Fig. 4.4 Village bots going about their daily business

village, have tea served to him by a waitress bot, and then go to the mosque to pray. Child bots played in the schoolyard, and the shopkeepers even visited with each other. These were not interactive or intelligent agent avatars, just avatars scripted to perform as actors, but they did give the village a liveliness that was not often found in a Second Life environment.

4.3.4 ECAAs for Instruction and Training

In late 2008 a new project at the ICT was funded to build a virtual healing centre for U.S. Military veterans in Second Life, on a space called Chicoma Island. This space, consisting of four "sims" (a.k.a. simulators, or standardized large regions of virtual space in SL), would offer in-world activities that could help reduce stress, build resilience, and serve as a supplement to standard medical care. One of the activities added to the virtual space was a labyrinth, as these are often associated with contemplation and relaxation. We realized that people might not know how to walk a labyrinth, so the first agent avatar we created was a simple scripted character that sat near its entrance, and could be summoned to provide very brief non-denominational instructions (see Fig. 4.5). This character was a small step up from the village bot actors, in that it did interact, but only when summoned.

We moved on to a more elaborate implementation for our next Second Life agent. To prevent the widely acknowledged confusion that can occur when a human-driven avatar comes to a new place, we implemented a guide for our area. As soon as a new visitor was detected, this guide, an agent avatar named "Info Clarity," would

Fig. 4.5 The Labyrinth guide on Chicoma Island

appear near the visitor and welcome them to Chicoma Island via a text chat message. Info could answer any questions typed to him about the purpose of the island and what one could do there. The agent avatar could also take the visitor on a walking tour of the space or teleport them to specific destinations when requested. The challenge of building this navigational agent helped answer many questions regarding ECAAs.

The first question we needed to answer was if the ICT framework for ECAs could be connected to avatar representations in Second Life. We were able to solve this problem by implementing the connection to the NPC Editor (a statistical classifier that could parse the text and match it to appropriate answers) using OpenMetaverse, a set of reverse-engineered protocols allowing a bot to mimic the interaction of a client program with the SL server. In other words, our ECAA appeared to be an avatar like any human-driven avatar, logging into the sim and accessing all the standard avatar features as well as the AI capabilities we were providing. Soon we had a guide agent whose domain included all parts of Chicoma Island.

Other questions were answered as the bot stayed logged in and interacted with users in the world. We determined that the bot could stay running and stable for an extended period of time, it could handle more than one person asking questions, and it could respond to people who were not in proximity to it by communicating over Instant Messaging, rather than local chat. When touring a person around the four sims of the island (each being served by a separate CPU), we solved the problem of handling the disruptions and navigational discontinuities caused by crossing sim boundaries. We analyzed conversational logs between ECAA guides and visitors, and improved the range of topics and questions that could be addressed. As was our standard practice, we also added responses to off-topic questions designed to bring the visitor back on track.

Shortly after we started this project, our work came to the attention of a training arm of the DoD. They were building areas within two virtual worlds, and wanted to populate them with intelligent agents for various purposes. This project—Virtual Intelligent Guides for Online Realms, or VIGOR—resulted in a number of interesting instances of agent avatar technology in virtual world space.

The first ECAA we created for VIGOR was to play the role of a knowledgeable African diplomat in a virtual information centre in Active Worlds. Active Worlds was a much older VW platform, with fewer tools available to access the internal workings of the system, but we produced a fairly simple conversational agent for our sponsors that could answer a range of questions about his African country (see Fig. 4.6).

We were also tasked with creating a guide for a public Army-oriented space they were setting up in Second Life. Building on the ideas present in the Chicoma Island guide, we created a sophisticated navigational and informational agent to tour people around the Army space, answer their questions and give them Army-themed virtual gifts. This guide went beyond the island guide in several ways. Its navigational functions included being able to guide groups, and know when people were keeping up with him or not. It could not only answer questions, it could handle both local and messaged chat inquiries and even correct for spelling mistakes. If this guide

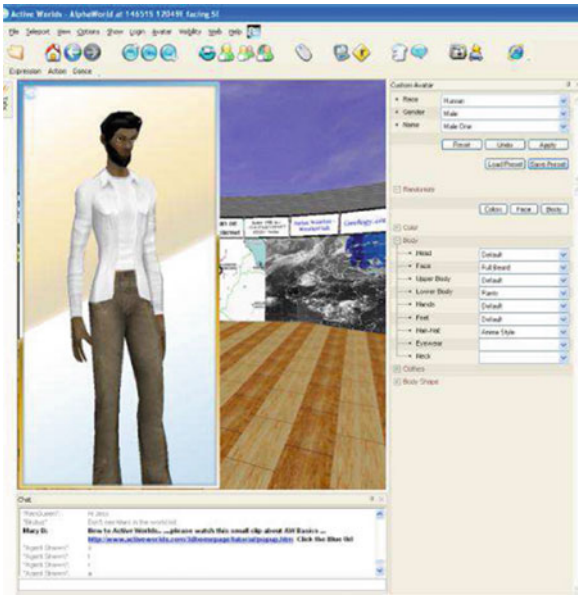


Fig. 4.6 Sudanese official in active worlds

ECAA did not know the answer to a specific question, it could send a message or an email to a live person who could send back an answer, which the bot would dutifully relay [19].

The next task was to implement an embodied agent avatar that could tell people how to make a parachute jump in the virtual Army world. What made this request challenging was the specification to make him a "crusty old sergeant" who would bark out orders and get annoyed if you weren't doing things fast enough. We had only made agents in SL that used text chat, and typing is not an efficient way to convey "crustiness." Therefore we decided to give this one a recorded voice, with which he could speak to the participants. Standing at the entrance to a rustic jump shack, he would greet visitors saying: "So you wanna jump off my mountain, troop?" He'd then say: "Well ya better get on one of those parachutes back there, before someone puts you in a body bag!" motioning to a shelf of parachutes in the jump shack. The visitor could type certain responses. For instance, if he or she said "No" to the jumpmaster's original question, then the jumpmaster agent would simply wait for the next visitor. If the person took too long to get their chute on, he'd offer the exact steps to do it, and yell at you impatiently if you still took your time (see Fig. 4.7).

The jumpmaster agent underscored some of the challenges that AI researchers discovered when their characters became more human-like, and these challenges were exacerbated within the Second Life platform. While we had successfully created a jumpmaster that yelled at you, SL did not offer tools that allowed us to make facial

Fig. 4.7 The Jumpmaster bot in Second Life

expressions to visually support the behaviour indicated by the vocal track. Second Life does offer a rudimentary lip-synching for when avatars are using a microphone and voice functions in the world, but it is not very sophisticated. It works moderately well for ordinary conversation, but not for voices that are highly modulated, as in singing or yelling. However, it is possible to access approximately a dozen key frames of default facial expressions through the native scripting language, LSL. With no way to access any control points on the agent avatar's face, we instead used a script to rapidly trigger and stop these available key frames in custom sequences to produce the illusion that the intelligent virtual agent is speaking the phrases being heard. It was moderately successful, and an interesting surprise to those Second Life citizens who encountered our less than polite jumpmaster [10].

The next challenge for the VIGOR Project was to develop an agent that could give what is called a "staff ride," which is typically a visit to a battleground or location of interest with someone relaying the events that took place on various parts of the terrain. Other environmental conditions such as weather can also be used as part of the analysis of the events that transpired. Staff rides are valuable training mechanisms, and today are most often conveyed through PowerPoint presentations in a classroom rather than by a visit to an actual site of interest [31]. Our staff ride guide was to tell the story of an incident during the Iraqi war at a checkpoint along the road to the Baghdad Airport. In this situation a car taking an Italian journalist to the airport failed to stop at a temporary checkpoint, and was fired upon, resulting in casualties. The geography of that area was built in Active Worlds, and an ECAA was developed with knowledge of the event. The tour started in an information centre where the initial details of the incident were conveyed to a group of human-driven avatars. The virtual staff ride guide then led the group to that checkpoint area, showing the terrain from several vantage points such as from an overpass and from the soldiers' location. Unlike staff rides where some of the details of the area might have changed, the Active Worlds environment could maintain the location of key

items that were there during the original incident, such as the temporary barriers and even the journalist's car, for better understanding of how they played into the events.

The training, guide and informational ECAAs we created in Second Life were fairly successful, and served as excellent proofs of concept. Back on Chicoma Island we decided to make an intelligent agent avatar that was a more prominent part of the activities we were building. The Warriors' Journey Story Tower was one such activity, where a veteran could go to see and hear a story about a classic warrior figure from history such as the Cheyenne Dog Soldier, or a Samurai Warrior. The stories, shown through a series of narrated panels along the spiral path of a tower structure, were designed to highlight guiding principles common to all soldiers, such as defending one's homeland, fighting with honour, protecting your comrades and returning from injury. We realized that four narrated image panels could only begin to tell the full story of these heroes, and that a conversational agent in the form of the warrior whose story was being told would be an ideal means to provide context, history and additional personal information about the character.

After seeing and hearing the story along the spiral path, a visitor to the tower reaches the topmost room, where the embodied conversational agent (whose avatar is designed to appear historically accurate) is situated, surrounded by a backdrop that represents where and when he lived. This ECAA finishes the story with additional narration, and gestures towards elements of the space around him as he does so. When he is done, text appears on the screen telling the visitor they can now talk to the warrior and ask him anything they want to know about his time, his life and the battles he has fought. The character has about 300 responses covering questions that could be asked about these areas of interest, as well as responses that address off topic comments. As veterans visit the Story Tower experience, we use the logs of their conversations to add to the warrior's knowledge base. At the present time, a visitor can choose from two classic Warrior's Journeys, each with their own set of background narrative and conversational responses [27].

This activity uses the power of narrative to help reinforce a more positive attitude and a connection to the long history of soldiers dedicated to protecting their people. We are taking the next step with this work and making it possible for a veteran who has experienced these stories to author not only their own story but also their own story-telling agent for the Story Tower. Authoring a conversational agent with full embodiment and a knowledge base is not a simple task: there is no Virtual Human Toolkit available to help one through the process (although this is an active area of research at the ICT). This effort is just starting and promises to be a major advancement for ECAAs in Second Life.

The implementations discussed thus far are all individual examples of agent avatars that converse one-on-one with a human being. As part of another task in VIGOR, we wanted to see if we could make an entire scenario dependent on the interactions of many agents, including a virtual teammate that collaborates on all tasks and investigations. We chose to use the Iraqi Village described earlier as a stage for a checkpoint situation that would involve military personnel and a range

of villagers, including children and a pregnant woman. While this was a proof of concept project for us, and not designed as an actual training exercise, it still included aspects a soldier might encounter during a checkpoint duty. These included having to stop, detain and search a person if they looked or acted suspicious, having to interrogate villagers when looking for a suspect, and dealing with cultural issues, like having to search a pregnant woman. A soldier ECAA acts as your partner at the checkpoint. A group of villagers are coming down the road and it is your job to make sure the people going in the village actually belong or have legitimate business there. There are two main scenarios: The first is where the crowd creates a diversion that enables a child to sneak past the gate with either medical supplies for his grandfather or bomb-making materials (the scenarios randomly selects one situation).

The second scenario involves a woman who appears to be wearing bulgy clothing and says she is going to visit an aunt in the village. These scenarios have randomized story outcomes, such as the woman either being secretly pregnant or simply trying to smuggle contraband. These divergent outcomes require that many of the characters be able to adjust their knowledge base to account for the scenario at hand. A scripted Heads Up Display (HUD) provides the actions and tools needed to maintain the checkpoint: the ability to stop an avatar, wand them, detain them and search their belongings (see Fig. 4.8). This HUD works for both the user and his virtual teammate. Each scenario may also require more information to be gained by walking through the village and asking questions of the local inhabitants. Here there are street vendors, shoppers, teachers and children who, as ECAAs, can offer information if asked. All these agents can be questioned (or pursued if they decide to flee) either by the user or the virtual teammate, who is also an ECAA [18].

Fig. 4.8 Some of the many bots that are part of the checkpoint exercise in Second Life

4.4 Creating Meaning with ECAAs

While we have done extensive work in creating useful ECAAs in virtual worlds, there is much more to be done. For example these intelligent characters need to be able to communicate better, especially in the area of gestures, behaviours, and natural body movements. However, this is also true even for the human-driven avatars in virtual worlds. Current communication methods, as provided by the makers of virtual worlds, require a person to choose gestures or animations from a limited list, which is far from intuitive. One needs to consider which gesture might match what one wishes to convey, and then actually trigger the gesture to have it played in the world. All this takes planning and time [38].

ECAAs also would be much more useful if they could understand a person's voice, as this has become a preferred means of communication over texting in most virtual worlds. While ICT's traditional virtual humans understand and respond to voice, implementing this functionality in a virtual world remains a challenge. In most virtual worlds people communicate via a voice over IP solution using a variety of consumer microphones, and this is not adequate to properly train a voice recognition system. Using voice would also require more sophisticated control of an avatar's facial movements. However, much of the complexity that exists in ICT's advanced virtual human models is not available in the VW avatars that form the basis of the ECAAs. Ultimately, we would like to integrate these virtual world agents with our more advanced virtual human tools, such as SmartBody, the ICT behavioural markup system in development that is integrated with the intelligence of the virtual human system to procedurally generate appropriate movements synchronized with spoken audio and other control channels [37].

While we have now made several ECAAs that serve a variety of purposes, the main goal driving our research is to have these agents help provide meaning for one's interactions in the virtual world. When we first began working on the veterans' healing centre we consulted with an existing Second Life veterans' group that had set up its own space where its members could feel comfortable and welcome. They had built a hospitality centre with resources ranging from GI Bill information to legal aid, a chapel, separate buildings for each of the services, a nightclub and more. The focal point of the space, however, was a recreation of the Tomb of the Unknown Soldier, a highly meaningful symbolic memorial for them. When we first approached them, they were rather wary, and reticent to share information or join our activities. However, as they began to learn more about our work, especially the intelligent agent avatars we had created, one of the members came to us with a request: could we make the ceremonial honour guard for the virtual Tomb of the Unknown Soldier? It had bothered them that their tomb stood unguarded, unlike its physical counterpart at Arlington Cemetery that is watched over 24 h a day, seven days a week. We, of course, said we would do this thinking that, since the guard does not actually have to talk to anyone, it would be easy. The challenge, however, was to make sure the honour guard conformed in its actions to the precise ritualistic patrol the guard maintains during his watch. This walk, 21 steps with specific turnings and time specifications,

was a very slow procedure, in accordance with the solemnity of its purpose. It was not simple to make a walk that proscribed and slow in Second Life because it offers only two fixed speeds of movement. We devised a method that would animate an invisible object along the walkway very slowly, and then attached the Honour Guard bot to that while overriding his default animations and replacing them with customized ones that could be synchronized with the movement of the object. In this way, the actions were precise and at the correct pace.

One big challenge was to ensure that the tomb guard agent stayed online all the time, which was more difficult than we thought it would be because the persistence of the virtual world is sometimes volatile. The overall world is indeed constantly on and available, but it happens that a particular sim, or island, may suddenly go offline for an unspecified amount of time or be restarted periodically by Linden Lab to install updates. An island's owner might also decide to intermittently reboot the sim to improve interaction and reduce lag, a common problem. We set up scripts to monitor the agent's status, and to relog it into the world if its sim went offline. However, if it tried to log in before that sim came back online, it would be automatically routed to some nearby sim. The monitor would then indicate it was back online, but we would soon hear from the veterans group that it was not in the right place [16].

The creation of the Honour Guard for the veterans gained a trusting relationship for us with the group. It also serves as an excellent example of why such agents should be encouraged in Second Life. While some people might find ways to use agents or bots in objectionable ways, the potential of these agents for positive interaction outweighs arguments for their exclusion.

4.5 Conclusion

We foresee a very active future for ECAAs deployed in virtual worlds. While they are not yet as sophisticated as many of their virtual human counterparts in the research arena, they can still enrich virtual spaces, serve as much needed guides and docents, teach valuable lessons, provide symbolic presences, and fill a myriad of uses—only some of which have yet to be imagined. The mercurial architectures of persistent, networked virtual worlds must be continually assessed to both determine what these agents can potentially achieve, and to plan for spaces that can support their newest uses. The advancement of virtual world ECAAs will only be achieved through diligent study and exploration of these worlds because it is the limitations of current virtual worlds that restrict the agents' abilities to exhibit intelligence, emotions, and learning capacity similar to that of the more mature virtual human technologies. Virtual worlds are here to stay, and will be an increasingly active part of how we interact with one another in the future. We encourage more work in this field and look forward to the many changes such efforts will bring.

Acknowledgments Some of the projects described herein have been sponsored by the U.S. Army Research, Development, and Engineering Command (RDECOM). Statements and opinions

expressed do not necessarily reflect the position or the policy of the United States Government, and no official endorsement should be inferred.
We would like to thank the many talented members of all the interdisciplinary teams that comprise the Virtual Human research group at the ICT for their dedicated work and contributions that enabled the adaptation of virtual human technology to virtual worlds. More about their work can be found at www.ict.usc.edu.

References

1. Becker-Asano, C., Wachsmuth, I.: Affective computing with primary and secondary emotions in a virtual human. Auton. Agents Multi-Agent Syst. **20**, 32–49 (2010)
2. Bickmore, T., Cassell, J.: Social dialogue with embodied conversational agents. In: van Kuppevelt, J., Dybkjaer, L., Bernsen, N.O. (eds.) Advances in Natural Multimodal Dialogue Systems, Speech and Language Technology, Chap. 2, vol. 30, pp. 23–54. Springer, Berlin (2005)
3. Bos, J., Oka, T.: Building spoken dialogue systems for believable characters. In: Proceedings of the Seventh Workshop on the Semantics and Pragmatics of Dialogue (DIABRUCK), (2003)
4. Brooks, D.: Oversimulated suburbia. The New York Times Magazine, Nov 2002
5. Cassell, J.: Embodied conversational agents: representation and intelligence in user interfaces. AI Mag. **22**(4), 67–84 (2001)
6. Cassell, J., Bickmore, T.W., Campbell, L., Vilhjálmsson, H.H., Yan, H.: More than just a pretty face: conversational protocols and the affordances of embodiment. Knowl.-Based Syst. **14**(1–2), 55–64 (2001)
7. Cassell, J., Vilhjálmsson, H.H.: Fully embodied conversational avatars: making communicative behaviors autonomous. Auton. Agents Multi-Agent Syst. **2**(1), 45–64 (1999)
8. Catrambone, R., Stasko, J., Xiao, J.: ECA as user interface paradigm. pp. 239–267. Kluwer Academic Publishers, Norwell, USA (2004)
9. AJ Champandard: Top 10 most influential AI games, 2007. http://aigamedev.com/open/highlights/top-ai-games/
10. Chance, E., Morie, J.F.: Method for custom facial animation and lip-sync in an unsupported environment, Second Life™. In: Proceedings of the 9th International Conference on Intelligent Virtual Agents, IVA, pp. 556–557. (2009)
11. Churcher, G.E., Atwell, E.S., Souter, C.: Dialogue management systems: a survey and overview. Technical Report, University of Leeds 1997
12. Egges, A., Kshirsagar, S., Magnenat-Thalmann, N.: Generic personality and emotion simulation for conversational agents. J. Vis. Comput. Animation **15**(1), 1–13 (2004)
13. Ellis, S.R.: What are virtual environments? IEEE Comput. Graph. Appl. **14**, 17–22 (Jan 1994)
14. Gratch, J., Rickel, J., Andre, E., Cassell, J., Petajan, E., Badler, N.I.: Creating interactive virtual humans: some assembly required. IEEE Intell. Syst. **17**(4), 54–63 (2002)
15. Hill, M.: Second Life is frontier for AI research: intelligence tests use virtual world controllable environment, 2008. http://www.msnbc.msn.com/id/24668099/ns/technology_and_science-innovation/t/second-life-frontier-ai-research/
16. ICT: Coming Home project description, 2010. http://projects.ict.usc.edu/force/cominghome/
17. ICT: Simcoach project description, 2010. http://ict.usc.edu/projects/simcoach/
18. Jan, D., Chance, E., Rajpurohit, D., DeVault, D., Leuski, A., Morie, J., Traum, D.: Checkpoint exercise: training with virtual actors in virtual worlds. In: Kopp, S., Marsella, S., Thòrisson, K. (eds.) Hannes Vilhjálmsson, Intelligent Virtual Agents, Lecture Notes in Computer Science, vol. 6895, pp. 453–454. Springer, Berlin (2011)
19. Jan, D., Roque, A., Leuski, A., Morie, J., Traum, D.: A virtual tour guide for virtual worlds. In: Ruttkay, Z., Kipp, M., Nijholt, A., Vilhjálmsson, H. (eds) Intelligent Virtual Agents, Lecture Notes in Computer Science, vol. 5773, pp. 372–378. Springer, Berlin (2009)

20. Hill, R.W. Jr., Gratch, J., Marsella, S., Rickel, J., Swartout, W.R., Traum, D.R.: Virtual humans in the mission rehearsal exercise system. KI **17**(4), 5 (2003)
21. Kenny, P.G., Hartholt, A., Gratch, J., Traum, D.R., Marsella, S., Swartout, W.R.: The more the merrier: multi-party negotiation with virtual humans. In: AAAI, pp. 1970–1971 (2007)
22. Laird, J.E.: Using a computer game to develop advanced AI. IEEE Comput. **34**(7), 70–75 (2001)
23. Martin, J.-C., Abrilian, S., Devillers, L., Lamolle, M., Mancini, M., Pelachaud, C.: Levels of representation in the annotation of emotion for the specification of expressivity in ECAs. In: IVA, pp. 405–417 (2005)
24. Mauldin, M.L.: ChatterBots, TinyMUDs, and the Turing Test: entering the Loebner prize competition. In: AAAI, pp. 16–21 (1994)
25. McLellan, H.: Virtual realities. In: Handbook of Research for Educational Communications and Technology. Macmillan, New York (1996)
26. Mims, C.: Whatever happened to virtual reality? MIT Technology Review, 2010. http://www.technologyreview.com/blog/mimssbits/25917/
27. Morie, J.F., Haynes, E., Chance, E.: Warrior's Journey: a path to healing through narrative exploration. In: Proceedings of the International Journal on Disability and Human Development (2011)
28. Nilsson, N.J.: Eye on the prize. AI Mag. **16**(2), 9–17 (1995)
29. Papp, R.: Virtual worlds and social networking: reaching the millennials. J. Technol. Res. 1–15 (2010)
30. Rickel, J., Marsella, S., Gratch, J., Hill, R., Traum, D.R., Swartout, W.R.: Toward a new generation of virtual humans for interactive experiences. IEEE Intell. Syst. **17**(4), 32–38 (2002)
31. Robertson, W.G.: The Staff Ride. United States Army Center of Military History, 1987. http://www.history.army.mil/html/books/070/70-21/index.html
32. Swartout, W.R.: Virtual humans. In: Twenty-First National Conference on Artificial Intelligence (AAAI-06), (2006)
33. Swartout, W.R.: Lessons learned from virtual humans. AI Mag. **31**(1), 9–20 (2010)
34. Swartout, W.R., Gratch, Jonathan., Hill, R.W. Jr., Hovy, E.H., Marsella, S., Rickel, J., Traum, D.R.: Toward virtual humans. AI Mag. **27**(2), 96–108 (2006)
35. Nino, T.: Peering inside Second Life user retention, Dec. 2007. http://massively.joystiq.com/2007/12/23/peering-inside-second-lifes-user-retention/
36. Nino, T.: Second Life traffic gaming: a chat with a bot-operator, and dire portents for lucky chairs, June 2009. http://massively.joystiq.com/2009/06/03/second-life-traffic-gaming-a-chat-with-a-bot-operator-and-dire/
37. Thiebaux, M., Marsella, S., Marshall, A.N., Kallmann, M.: Smartbody: behavior realization for embodied conversational agents. In: AAMAS (1), pp. 151–158 (2008)
38. Verhulsdonck, G., Morie, J.: Virtual chironomia: developing standards for non-verbal communication in virtual worlds. J. Virtual Worlds Res. **2**(3) (2009)
39. Weizenbaum, J.: Eliza—a computer program for the study of natural language communication between man and machine. Commun. ACM **9**, 36–45 (1966)

Chapter 5
Human-Like Combat Behaviour via Multiobjective Neuroevolution

Jacob Schrum, Igor V. Karpov and Risto Miikkulainen

Abstract Although evolution has proven to be a powerful search method for discovering effective behaviour for sequential decision-making problems, it seems unlikely that evolving for raw performance could result in behaviour that is distinctly human-like. This chapter demonstrates how human-like behaviour can be evolved by restricting a bot's actions in a way consistent with human limitations and predilections. This approach evolves good behaviour, but assures that it is consistent with how humans behave. The approach is demonstrated in the UT^2 bot for the commercial first-person shooter videogame Unreal Tournament 2004. UT^2's human-like qualities allowed it to take second place in BotPrize 2010, a competition to develop human-like bots for Unreal Tournament 2004. This chapter analyzes UT^2, explains how it achieved its current level of humanness, and discusses insights gained from the competition results that should lead to improved human-like bot performance in future competitions and in videogames in general.

5.1 Introduction

Simulated evolution has proven to be a powerful policy-search method for solving challenging reinforcement learning problems [5, 10, 16, 18, 21, 26]. However, evolutionary methods are also notorious for taking advantage of any trick available to achieve high fitness: any loopholes present in the domain simulation software are sure to be exploited. A similar problem arises in the context of evolving human-like

J. Schrum (✉) · I. V. Karpov · R. Miikkulainen
University of Texas at Austin, Austin, TX 78712, USA
e-mail: schrum2@cs.utexas.edu

I. V. Karpov
e-mail: ikarpov@cs.utexas.edu

R. Miikkulainen
e-mail: risto@cs.utexas.edu

P. Hingston (ed.), *Believable Bots*, DOI: 10.1007/978-3-642-32323-2_5,

behaviour for videogames. Because humans are skilled at videogames, it is reasonable to evolve bots for performance in order to get human-like behaviour. However, evolution may exploit domain tricks for the sake of performance, which results in bots behaving in a non-human-like manner.

However, if the senses and actions available to the bot are constrained such that they both simulate the restrictions humans deal with, and make common human actions easy to carry out, then it is possible to achieve human-like behaviour by evolving for good performance, even when good performance is defined in terms of multiple conflicting objectives. This maxim is demonstrated by the UTˆ2 bot, which placed second in BotPrize 2010, a competition to develop human-like bots for the commercial First-Person Shooter (FPS) videogame Unreal Tournament 2004 (UT2004).

This chapter describes the UTˆ2 bot, with emphasis on its combat behaviour, the policy for which was determined by a neural network whose weights and topology were evolved using Evolutionary Multiobjective Optimization (EMO). The UTˆ2 bot is further discussed in Chap. 6 of this book, which describes how UTˆ2 makes use of human trace data to navigate when pathfinding fails. The two techniques are complimentary, and can be used separately or together, as was done in UTˆ2.

Understanding how UTˆ2 exhibits human-like behaviour requires an understanding of the role of bots in the FPS genre (Sect. 5.2). The particulars of UT2004 and BotPrize are discussed in Sects. 5.2.1 and 5.2.2 respectively. Given this context, UTˆ2 can be discussed in detail (Sect. 5.3) with emphasis on its combat behaviour (Sect. 5.3.2). The combat behaviour was learned using neuroevolution and evolutionary multiobjective optimization, which are discussed in Sects. 5.4.1 and 5.4.2 respectively. How these methods were used to produce the final combat behaviour for UTˆ2 is discussed in Sect. 5.4.3. After fully describing the bot, it is evaluated in Sect. 5.5. This evaluation leads to discussion and ideas for future work in Sect. 5.6. Then Sect. 5.7 concludes the chapter.

5.2 Bots in First-Person Shooters

FPS games display the game world to the player through the first-person perspective of the agent controlled in the game. Early games pitted players against simplistic computer controlled opponents. Since the available weapons, ammo, health, and general capabilities of players differed so much from that of the computer opponents, it mattered little if the enemies behaved in a human-like manner.

However, FPS games eventually began incorporating multiplayer modes that allowed players to compete against other humans over a network connection. A free-for-all competition between several human competitors is called a Deathmatch. In this style of play, all players are on equal footing with regards to weapons, health and abilities. From the advent of human multiplayer combat, it was only a small step to FPS games entirely based around the concept of multiplayer-style play.

5.2.1 Unreal Tournament 2004

The original Unreal Tournament (1999) was the first FPS to fully embrace the multiplayer style of gameplay. Although the game had a single-player mode, this mode consisted exclusively of a series of matches against bots played with the exact same rules used in multiplayer mode against humans. Thus arose the need for convincingly human bots in FPS games.

UT2004 is the second sequel to the original Unreal Tournament, and continues the trend of focusing on multiplayer-style play against humans. In addition to Deathmatch mode, all Unreal Tournament games feature several additional types of team play, such as Team Deathmatch and Capture the Flag, but since these modes of play are not yet part of BotPrize, they will not be discussed further in this chapter.

In a Deathmatch, players spawn at random spawn points with only the most basic weapons. They then run around the level, accruing more powerful weapons and other useful items in order to help them kill each other in combat. An event where one player kills another is called a frag, and is worth one point. After dying, players immediately respawn at a new, randomly chosen spawn point with full health, but only rudimentary weapons, as at the start of the match. If a player kills himself or herself, for example by jumping in a pit or by firing a rocket at a nearby wall, the penalty is the loss of one point, which can result in a negative score. The goal of a Deathmatch is to either get the most points within a preset time limit, or be the first to attain a preset number of points.

Because this chapter deals primarily with bot combat behaviour, the specific weapons available in UT2004 will be reviewed in detail. Each weapon has both primary and alternate firing modes which are often very different from each other. Sometimes the alternate firing mode does not fire at all, but instead activates some special ability of the weapon. Several weapons also have a charging attack, which requires holding down the fire button to charge up a projectile whose properties depend on how long the weapon is charged before being released. Each weapon is explained in detail so that later descriptions (Sect. 5.3.2.3) of how the bot handles each weapon will be understood:

- Shield Gun: A last resort weapon whose ammo recharges automatically. Players spawn with this weapon.
  - Primary: Charges weapon until the player is close enough to touch an opponent, at which point the weapon automatically discharges to deal an amount of damage proportional (within bounds) to how long the weapon was charged.
  - Alternate: Creates a defensive shield in front of the player that deflects projectiles while the fire button is held.
- Assault Rifle: A weak but rapid firing gun that all players spawn with.
  - Primary: Automatic fire that is rapid but weak.

  - Alternate: Charges a grenade that is launched in an arc on release. The grenade bounces off of level geometry but explodes on impact with players. Powerful, but difficult to aim.

- Shock Rifle: Weapon with both a fast, focused attack and a slower attack that explodes to affect a large area on impact.
  - Primary: Immediately hits target in the crosshairs and knocks players back on impact, which can disorient them. However, the delay between subsequent shots is significant.
  - Alternate: Fires a large, slow moving orb that explodes on impact. There is also a special combo attack that creates a larger, more powerful explosion if the primary fire mode is used to shoot the orb out of the air. Bots can only perform this "shock combo" by chance because they cannot determine the locations of their own projectiles.

- Bio-Rifle: Weapon whose projectiles fire in an arc and linger on the ground, where they explode on impact with any player that comes into contact with them. Note that the bots in BotPrize have no way of seeing these potential traps.
  - Primary: Rapidly fires small explosive green blobs.
  - Alternate: Charges the weapon in preparation for firing a large blob that deals an amount of damage proportional (within bounds) to the duration of the charge. If the shot misses, then the large blob explodes into a batch of small blobs upon hitting the ground.

- Minigun: A rapid fire machine gun.
  - Primary: High rate of fire, but slightly inaccurate, and therefore best suited to close quarters combat.
  - Alternate: Slower rate of fire, but is more accurate and fires shots that deal more damage.

- Flak Cannon: Versatile weapon whose primary firing mode is effective at close range and whose alternate firing mode works well at medium range.
  - Primary: Several small shards of flak are launched in a wide spread, each doing little damage, but dealing a great deal of damage together.
  - Alternate: Launches a flak grenade in an arc. Damaging flak is spread in all directions on impact.

- Rocket Launcher: Fires slow but powerful explosive projectiles.
  - Primary: Immediately fires a single rocket.
  - Alternate: Charges up to three rockets to be fired simultaneously. When released, however many rockets are currently loaded will be fired. The default firing pattern is a wide spread that becomes wider as the rockets get farther away. However, pressing the primary fire button while still charging causes the rockets to shoot in a tighter, forward moving spiral.

- Sniper Rifle: Very accurate and powerful, but slow firing weapon.
  - Primary: Fires a single shot that instantly hits whatever is in the crosshairs.
  - Alternate: For humans, alternate fire activates the sniper scope. Holding down the alternate fire button zooms in to allow the player a better view of what is in the distance at the cost of not being able to see nearby surroundings. While zoomed in, the player can use primary fire to shoot. However, bots are unable to use this feature because they do not see the world the way humans do.
- Lightning Gun: Functionally the same as the Sniper Rifle, except that the bolt of lightning fired by this gun can be seen by humans, making it easier to trace an attack back to its source. Bots cannot see these lightning bolts.
  - Primary: Fires a single bolt of lightning that instantly hits its target.
  - Alternate: Switches to a sniper scope, as with the Sniper Rifle.

This list shows that UT2004 provides viable weapons for any combat situation. Certain weapons are only useful within certain ranges, though when under attack players may be forced to improvise with the weapons and ammo currently available to them. Given a choice of what weapon to use in combat, there are several salient features that can be used to choose an appropriate weapon. Perhaps more importantly, these features dictate how the weapon is used once it has been chosen.

- Rate of Fire: Rapid firing weapons work best in hectic, mid-range combat scenarios when players are actively dodging, whereas slow-firing weapons tend to be better at longer range, in which case the shooter can take time to make each shot. The latter statement is especially true of the sniping weapons.
- Projectile Speed: Some weapon shots instantly hit any target in the crosshairs, while others take time to reach their destinations. Humans using weapons with slower projectiles tend to compensate for the slowness by anticipating where their opponents will be in the next few seconds.
- Firing Trajectory: The alternate firing modes of both the Flak Cannon and the Assault Rifle launch projectiles in curved arcs that tend towards the ground. Both firing modes of the Bio-Rifle also fire in an arc. When using these weapons, players must account for gravity, which usually means aiming higher than one would aim with a straight firing weapon.
- Splash Damage: Weapons with an explosive component deal "splash" damage. Splash damage is particularly useful against players that dodge well, and are therefore hard to hit, since near misses will also damage them. However, splash damage weapons are also dangerous since they can damage the shooter as well. For this reason, splash damage weapons are not preferred in close quarters combat. When fired, it makes sense to aim at an opponent's feet, since the explosion from hitting the ground may damage the opponent even when the shot misses.

These weapon features are all relevant in defining the combat behaviour of UTˆ2. However, UTˆ2 was designed not only to perform well in UT2004, but in the modified version used in the 2010 BotPrize competition, which is described next.

5.2.2 BotPrize 2010

The original 2008 BotPrize competition [12] was billed as a "Turing Test for Bots" in which, as in a traditional Turing Test [23], each judge attempted to distinguish between a computer controlled bot and a human confederate in a three-player match. Many changes to this scheme were introduced in the 2010 competition [13]. The most important is the inclusion of a judging gun, which replaces a weapon not mentioned in Sect. 5.2.1: the Link Gun. All human players and bots spawn with the judging gun, which has infinite ammo. Both the primary and alternate fire modes of the gun look and sound the same to all observers, but these two modes are different in that one is meant to be fired at bots and the other is meant to be fired at humans. If a bot is shot using the primary firing mode, then the bot instantly dies and the shooter gains 10 points. Similarly, if a human-controlled agent is shot using the alternate firing mode, then the human-controlled agent instantly dies and the shooter gains 10 points. In contrast, if either firing mode is used against an agent that is the opposite of the intended type, then the shooter instantly dies, and loses 10 points. In any case, a player is allowed to judge any other player only once; subsequent attempts to judge the same player will have no effect.

The judging gun not only changes how judging is done, but completely changes the game from a pure Deathmatch to a judging game. Since the bots are being tested in this new judging game, they also have access to the judging gun, which adds the challenge of deciding if and when a bot should use the judging gun. Unfortunately, humans can now benefit from pretending to be bots. Such "distortion effects" are discussed in Chap. 9.

Because all players have the judging gun, there is no longer a division between human judges and human confederates. Furthermore, matches are no longer limited to three players. Several bots and a roughly equal number of humans play simultaneously. All human players are judges, but they are ultimately competing for the highest score. Of course, judging correctly is a good way to get a high score, since correct judgments are worth 10 points each.

Other than the judging gun, all weapons function as usual, except that all damage dealt is only 40 % of normal, in order to give humans ample chance to observe opponents before one of them dies. The levels used were three publicly available maps designed by members of the UT2004 community: `DM-DG-Colosseum` (Colosseum), `DM-IceHenge` (IceHenge), and `DM-GoatswoodPlay` (Goatswood). Each match lasted 15 min between the five competing bots, one to two native UT2004 bots, and six to seven humans. There were a total of 12 matches conducted during three separate one-hour sessions.

All of this information, along with the maps and the game modification which implemented the competition rules, were available to the entrants before the competition. UT^2 was designed to compete within the parameters of this competition.

5.3 The UT^2 Bot

The UT^2 bot was developed at the **U**niversity of **T**exas at Austin for use in the game **U**nreal **T**ournament 2004, hence the exponent of 2 after `UT` in the name. Specifically, the bot was designed for BotPrize 2010 using Pogamut 3 [9], a platform for writing Java code to control UT2004 bots via a customized version of the Gamebots message protocol [1]. This section outlines the overall architecture of the UT^2 bot, and then focuses on the bot's battle controller.

5.3.1 Architecture

The architecture controlling UT^2 is a behaviour-based approach similar to both the POSH interface [4], which is integrated into Pogamut 3, and behaviour trees [14], which were introduced in the commercial videogame Halo 2. The bot has a list of behaviour modules, each with its own triggers. On every time step the bot iterates through the list, checking triggers for each module until one of them evaluates to `true`. The module associated with the chosen trigger takes control of the bot for the current time step. Each module can potentially have its own set of internal triggers that further subdivide the range of available behavioural modes.

The specific bot architecture is shown in Fig. 5.1. The highest priority action is getting *UNSTUCK*. Several triggers detect if the bot is stuck, but if any of them fire, it means the bot's ability to navigate has failed, and emergency action is needed to return to a state where the bot can function as normal. UT^2's method for getting unstuck is based on human trace data, and is explained in full detail in Chap. 6.

The next highest priority action is picking up weapons that have been dropped by killed opponents (*PICKUP DROPPED WEAPON*). Whenever an opponent dies, the weapon the opponent was using, along with whatever ammo it had, becomes available for pickup for a short time before disappearing. Humans tend to pick up these weapons immediately when they are dropped provided they are close enough, so it was decided that a human-like bot should do the same.

The next highest priority module is *GET IMPORTANT ITEM*. Some items are highly desirable, either in absolute terms or in certain contexts, and should be pursued even if it means running away from combat. One such item is the Keg o' Health, which gives a player 100 health points, exceeding the normal limit of 100. The Double Damage powerup is always desirable as well. It makes a player's weapons deal twice the normal amount of damage for a period of 30 s. Items that are circumstantially important are health items when the bot is low on health, and weapons/ammo when the bot can only use the basic starting weapons. The *GET IMPORTANT ITEM* module makes the bot focus on and pursue any important item that is visible and close enough to obtain in a relatively short amount of time.

The next module is the *JUDGE* module, which uses the battle controller, the primary focus of this chapter. The bot remembers all opponents that it has judged so

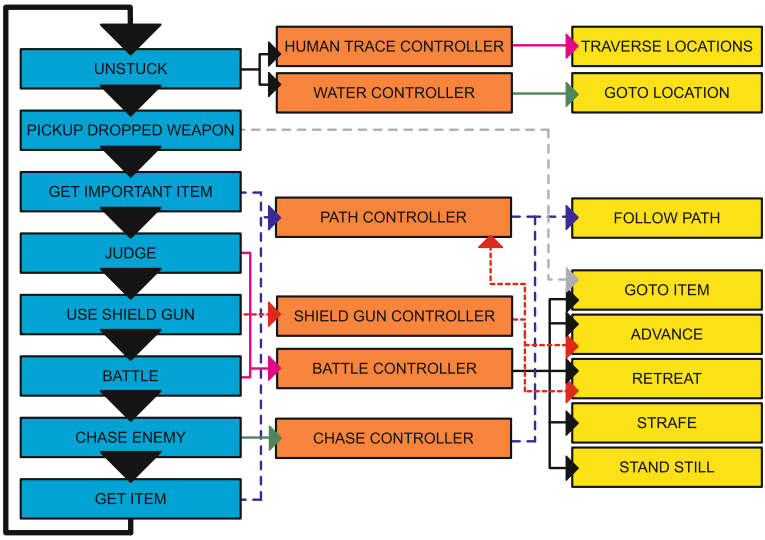


Fig. 5.1 Agent architecture for the UTˆ2 bot. The *left* column shows the behaviour modules in priority order from *top* to *bottom*. The *middle* column shows the individual controllers used by each module. Notice that the battle controller is used by both the *JUDGE* and *BATTLE* modules. The *right* column shows the individual actions available to each controller. This architecture can also be thought of as a POSH plan or a two-level behaviour tree. This behaviour-based architecture modularized bot behaviours, making the overall behaviour easier to understand, and making programming and troubleshooting easier

that it will not attempt to judge anyone twice. The bot's decision to judge is based on how much interaction it has had with a given opponent, and how much time is remaining in the match. Judging is more likely if the bot has interacted a lot with an opponent, and if there is little time remaining in the match. Once the decision to judge has been made, the actual decision is based on knowledge of previous judgments and an assumption (only approximately true) that the number of bots in a match equals the number of humans. Whenever the bot judges an opponent, it knows the identity of the opponent after the judgment, regardless of the outcome. This knowledge is used to determine the probability that any remaining player is a human or a bot, which is in turn used to make a random but informed decision about how to judge an opponent. As for how the bot behaves while making its judgment, this is determined by the battle controller, which is described below in Sect. 5.3.2.

The *JUDGE* module is high in the priority list because every player has infinite ammo for the judging gun, which makes its use a viable option at all times. However, if the bot chooses not to judge, but has no other ranged weapons available, it will resort to the *USE SHIELD GUN* module. Proper Shield Gun usage separates the good players from the experts, but typical players avoid using it because the ranged weapons are so much easier to use in comparison. Most players only resort to the

Shield Gun when there is no other option. Because the Shield Gun is so different from the other weapons in the game, it has its own scripted controller.

Without ranged weapons, a player is more vulnerable. Human players will typically seek out better weapons rather than risk fighting with just the Shield Gun. Therefore, the controller's design is based on the idea that a human's primary concern when using the Shield Gun is getting a better weapon. The bot is programmed to approach the nearest relevant item while facing the nearest opponent and using the shield mode of the gun to defend itself. Relevant items consist of ammo for weapons that the bot has, and weapons that the bot does not have (picking up a previously possessed weapon provides no extra ammo for that weapon). However, the bot only pursues such a relevant item if it is closer than the nearest enemy. If the enemy is very close, then the bot will rush in with the attack mode of the Shield Gun. If the enemy is closer than a relevant item, but not close enough to do a Shield Gun rush, then the bot will simply try to put as much distance between itself and the opponent as possible while using the shield mode to defend itself.

The Shield Gun is only used if ranged weapons are unavailable. For ranged weapons, the *BATTLE* module takes over. The bot avoids combat if its health is very low, or if it is very far away from visible opponents, but otherwise it equips whatever available weapon is best for the given circumstances and uses the battle controller to drive its behaviour, as described in Sect. 5.3.2.

A static lookup table indexed by distance from the opponent determines the best weapon in each situation. Distance from the opponent is partitioned into close (less than 100 UT units), medium (100–2000 UT units) and long range (greater than 2000 UT units). In general, sniping weapons and splash damage weapons are favoured at long range. At medium range, the Flak Cannon is favoured, followed by rapid fire weapons and splash damage weapons. At close range, splash damage weapons and sniping weapons have lowest priority, and rapid fire weapons are favoured. This table was tuned based on experience in UT2004, as well as trial and error.

If the bot is otherwise ready for battle, but sees no enemy, it checks its memory of where it last saw an opponent, and uses the *CHASE ENEMY* module. The bot runs to the last location that it remembers seeing an enemy in hopes of reacquiring its target and reengaging in combat, though it will break off the chase for any opponent that it sees. If the bot reaches the last known location of an enemy and still sees no opponents, it gives up the search. The bot also gives up the search after too long a period passes without encountering an enemy.

Given nothing better to do, the bot will simply head towards the nearest desirable item (*GET ITEM*), where desirability is based on current equipment and vital statistics. Basically, the bot will pursue weapons it does not have, ammo for weapons it does have, and health and armour if it has less than the full allowance.

Note that although the battle controller is primarily responsible for all combat actions, the bot is still capable of firing at opponents while using any of the non-combat-oriented modules above. In particular, the bot does not stop shooting if it needs to get unstuck, and it will fire on enemies it sees while attempting to pick up an important item. This sort of behaviour is common among human players, and was deemed an essential component of a human-like bot.

Still, the majority of UTˆ2's combat behaviour can be attributed to the battle controller. Furthermore, most interactions between the bot and other players occurs during combat, so the bot's capacity to appear human depends very much on the battle controller, which is described next.

5.3.2 Battle Controller

The battle controller is used by the combat and judging modules. It controls the bot using an artificial neural network. Artificial neural networks mimic some of the information processing capabilities of organic brains, but at their most basic level they can be thought of as universal function approximators between $\mathbb{R}^N$ and $\mathbb{R}^M$ for arbitrary integers N and M [11]. Some network architectures also have an internal recurrent state which influences network output [11], thus making the network behaviour a function of all previous inputs rather than just the current input.

This section describes the input sensors of the battle controller's neural network, which is evolved (Sect. 5.4.1), followed by a discussion of the network's outputs, including how these outputs are interpreted and filtered to produce behaviour that is both effective and human-like.

5.3.2.1 Network Inputs

The network used by UTˆ2 processes inputs based on the bot's sensors every time step the battle controller is in use. The network produces several outputs for each set of inputs, and the outputs are used to produce an action for UTˆ2. The numerical inputs to UTˆ2's neural network are:

- Ten Pie Slice Enemy Sensors: These sensors are identical to those used by van Hoorn et al. [25] to evolve combat behaviour for a UT2004 bot. From an overhead perspective, the space around the bot is divided into slices, with the slices near the front of the bot narrower (and therefore more precise) than the slices near the rear of the bot (Fig. 5.2). For each sensor, the value of the input is higher if an enemy sensed within that slice is closer. Given multiple enemies in one slice, the distance of the closest enemy defines the sensor value.
- Twenty-two Ray-Tracing Level Geometry Sensors: Gamebots provides a way to define periodically updated ray traces, each of which senses the distance to the first piece of level geometry that the ray trace intersects. The bot is surrounded by twelve such ray traces which are parallel to level ground. These twelve sensors are identical to the wall sensors used in [25]. However, because BotPrize levels have complicated 3D geometry, additional ray traces were added which radiated out at 45° angles both above and below the bot to sense unusual ground and ceiling geometry. There were six ceiling traces and four ground traces. Traces at each level were spread evenly around the bot (Fig. 5.3). However, it was discovered after the competition that the BotPrize version of Gamebots actually disabled all

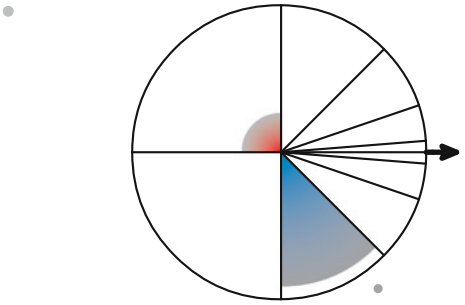


Fig. 5.2 Pie Slice Enemy Sensors. There are more slices of smaller size near the front (*right*) so the bot can better distinguish locations in front of it. The *dots* represent enemies, and *filled portions* of the pie slices show the relative activations for opponents at different distances. Activation increases as opponents get nearer, which is why the opponent in the *upper left* causes the corresponding pie slice to be filled less than the slice for the nearer opponent on the *lower right* side of the figure. Adapted from [25]

Fig. 5.3 Ray-Tracing Level Geometry Sensors. Gamebots has a debugging option for viewing ray traces on a bot. The figure shows all 22 ray trace sensors around the bot, with the contrast heightened to improve visibility. Some of the rays aiming upward are brighter because they are not colliding with any level geometry. These sensors provide the bot with information about the structure of its immediate environment, which helps it reason about how best to dodge enemy attacks. Though disabled in BotPrize 2010, these sensors should help the bot be more aware of its surroundings in future competitions

ray traces, meaning all these network sensors returned a value of 0. This problem is being fixed for future competitions, which will give future versions of the bot better awareness of their surroundings.

- One Crosshair Sensor: There is an additional ray trace projecting straight in front of the bot which can sense agents. If this ray trace hits an agent, then this sensor is 1.0; it is 0.0 otherwise. As with the ray traces for level geometry, this sensor

only returned a value of 0 during the competition because the BotPrize version of Gamebots did not support ray traces.
- One Damage Sensor: 1.0 if the bot is currently being damaged, 0.0 otherwise.
- One Movement Sensor: 1.0 if the bot is currently moving, 0.0 otherwise.
- One Shooting Sensor: 1.0 if the bot is currently shooting, 0.0 otherwise.
- One Damage Inflicting Sensor: 1.0 if the bot is currently inflicting damage, 0.0 otherwise.
- One Ledge Sensor: 1.0 if the bot is on a ledge, 0.0 otherwise (potentially helps the bot avoid falling off of cliffs).
- One Enemy Shooting Sensor: 1.0 if the currently targeted enemy is shooting, 0.0 otherwise. UTˆ2 usually targets whichever enemy is closest, but if another enemy is damaging the bot, then the threatening enemy will be targeted. Also, if the bot has already invested time damaging a particular enemy, it continues targeting that enemy unless it gets very far away, while another enemy gets much closer.
- Eight Current Weapon Sensors: For two of these sensors, 1.0/0.0 values represent yes/no answers to the following questions: Is it a sniping weapon? Does either fire mode deal splash damage? The remaining sensors report the rates of fire of both firing modes, the start-up times for firing with both modes, and the damage dealt by both modes. However, it was discovered after evolving the bot that the values Pogamut 3 returns for the damage of some weapons is incorrectly set to zero. Furthermore, alternate fire damage values for the Rocket Launcher and the Bio-Rifle are equal to the primary damage values, which does not indicate the high damage potential that these modes actually have. However, evolution seems to have been robust enough to account for these deficiencies.
- Six Nearest Item Sensors: 1.0/0.0 values represent yes/no answers regarding properties of the closest item to the bot: Is it visible? Is it health? Is it armour? Is it a shield? Is it a weapon? Is it a Double Damage powerup?
- Four Nearest Health Item Sensors: Scaled relative distances to the nearest health giving item along the x, y and z axes, as well as the scaled direct distance.

Though some of the inputs used by UTˆ2 were based on sensors used in other work, some sensors were provided simply because there was a chance they would be useful. Though this particular set of inputs proved sufficient to generate good combat behaviour for the 2010 competition, the task of trying to find an ideal set of inputs with which to evolve is future work.

5.3.2.2 Network Outputs

The outputs of the network were chosen to assure that in battle the bot would choose among actions similar to those commonly used by humans. When evolving neural networks (as described below in Sect. 5.4.1) to control agents, it is common for both the inputs and the outputs to be ego-centric (cf. [20, 25]). The inputs listed above are ego-centric, but the outputs are defined both in terms of the UTˆ2 bot and the opponent that it is currently targeting. This approach works because the battle

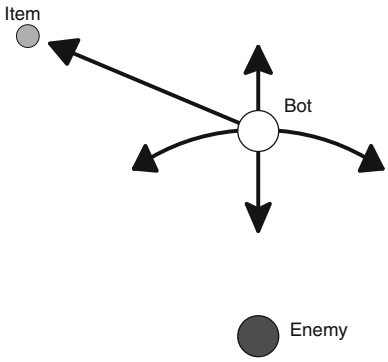


Fig. 5.4 Opponent-relative movement actions. During combat the bot has six available movement actions depicted by the *arrows* in the figure (*STAND STILL* is not shown). These actions are defined with respect to the opponent the bot is currently targeting. Forcing the bot to always focus on an opponent makes it seem interested in the opponent, and therefore more human-like

controller is only used when there is an opponent to fight, and it makes sense because human opponents pay attention to the opponents they face. Focusing on opponents is both good strategy and typical human behaviour.

Specifically, the network has eight outputs: five compete to define the type of opponent-relative movement action taken by the bot, and three determine whether the bot shoots, which firing mode to use, and whether or not to jump. The five available movement actions are *ADVANCE* towards opponent, *RETREAT* from opponent, *STRAFE* left around opponent, *STRAFE* right around opponent, *GOTO ITEM* which is nearest, and *STAND STILL* (Fig. 5.4). The *GOTO ITEM* action is the only non-opponent-relative movement action. The movement action performed by the bot is the action whose network output has the highest activation.

While executing all actions, the bot looks at the targeted opponent. It is important that the bot seems interested in the human opponents it fights. The bot can also fire its weapon at the targeted opponent during any movement action. If the shooting output of the network is in the upper half of the output range, the bot shoots. The mode of fire depends on whether the fire mode output is in the lower, for primary fire, or upper, for alternate fire, half of the range. If the jumping output is in the upper half of the range, then the bot jumps while performing its movement action.

This scheme is enough to evolve effective combat behaviour in UT2004, but because the objective is to evolve human-like behaviour, some additional restrictions are required to filter and adjust certain actions.

5.3.2.3 Action Filtering

In terms of movement, the bot will not move towards items that are not desirable (as defined with respect to the *GET ITEM* module from Sect. 5.3.1). Also, when using

a sniping weapon or the dangerously explosive Rocket Launcher, the bot will not *ADVANCE* towards enemies to which it is already close enough. In these cases, the action with the next highest activation is considered until a suitable action is found. The bot is also not allowed to jump if it is performing the *STAND STILL* action, since jumping in place generally looks very bot-like.

In terms of weapon usage, one of the clearest signs that an opponent is a bot is superhuman accuracy, particularly with single-shot, instant-hit weapons. Therefore, to make the accuracy of the bot more human-like when using such weapons, the bot is actually commanded to fire at a point equal to the location of the target plus some random noise. The maximum potential magnitude of the noise depends on both the distance between the bot and the opponent, and the relative velocities of the two agents. To account for a human's difficulty in aiming at targets that are far away, greater distances between the bot and the opponent result in greater random noise potential. To account for human difficulty in hitting moving targets, the magnitude of the noise added along the x, y and z directions is also proportional to the differences in velocity between the bot and the opponent along each of these axes. Therefore, if both the bot and the opponent are moving in the same direction at the same speed, then they are both standing still relative to each other, and no noise is added. However, such perfect synchronicity is unlikely, and in most cases the faster either agent moves, particularly when moving in different directions, the greater the noise will be and the harder it will be to aim with an instant-hit weapon.

The standard setup also needs to be more human-like regarding how automatic weapons are used. Humans generally fire these weapons in continuous bursts as long as they can keep roughly on target. Because one network needs to handle proper control of all weapon types, automatic weapon use can become choppy and intermittent, which only makes sense with single-shot weapons. Therefore, whenever the bot initiates fire with an automatic weapon, it will remain firing as long as its target is available, regardless of whether the network commands it to shoot or not.

Weapons that need to be charged are similar to automatic weapons in that the network controller is likely to release the fire button while charging. The fix for the problem is similar, except that releasing the charge needs to happen while still facing the opponent. In order to make the bot effectively use charged weapons, a random check is used for as long as the bot is charging the weapon: for every time step after starting to charge a weapon, the chance of releasing the charge and firing is 25 %. Because the triple rocket attack of the Rocket Launcher takes longer to charge, and is a very useful attack, this percentage is reduced to 15 % for this weapon.

Other important features of the Rocket Launcher are that its projectiles are explosive, and take extra time to reach their target. The secondary fire of the Shock Rifle shares these features. Humans adjust to the slowness by firing at locations where they believe their target will be by the time the projectile hits. Therefore, when using the Rocket Launcher or the alternate fire of the Shock Rifle, UTˆ2 adjusts its target along the direction of enemy movement with a small amount of random noise whose maximum magnitude along each axis is proportional to the corresponding components of the target's velocity along each axis. In other words, the bot will always aim

slightly ahead of its target along the target's direction of movement. Additionally, in order to take advantage of splash damage from explosions that hit the ground near opponents, the bot will further adjust its target down by a small amount whenever its current position is higher than that of the opponent.

Weapons that lob projectiles in an arc are also problematic. Gamebots uses a one-size-fits-all firing command that does not work well for lobbing projectiles. The behaviour of the default fire command is neither human-like nor particularly accurate. The default behaviour often results in projectiles lobbed over the heads of opponents. To compensate for this problem, the target for all lobbing projectiles is adjusted to be a point slightly in front of the opponent along the line between the bot and the opponent. Random noise is used to determine exactly how much to adjust the aim.

Finally, weapons primarily intended for close to middle range are prevented from firing when the bot is too far away from its target. The decisions over which weapons to restrict and to what ranges were made with the help of volunteer human players.

Some of these modifications could effectively be added to the bot after the controlling network is evolved, but one of the main ideas of this chapter is that having these constraints and filters in place before evolution takes place requires evolution to find policies that perform well within the context of these constraints. For example, reducing the accuracy of the Sniper Rifle when moving at high speeds makes the bot more likely to evolve to stand still when using it, which is what humans do.

However, creating a network for the battle controller requires a method for evolving neural networks, which is the topic of the next section.

5.4 Evolution

Evolutionary Algorithms (EAs) are inspired by Darwin's Theory of Evolution by Natural Selection [6]. Though there are many different types of EAs, they are all population-based search methods. They depend on mutation operators to modify existing solution representations in order to search the space of available solutions. Selection is applied to favour the better solutions for inclusion in the next generation.

Many EAs also involve some form of crossover, which takes two existing solutions and recombines them to form a new solution, sharing traits of each parent. Though crossover is generally considered to be advantageous, there is some evidence [8] that crossover is unnecessary in evolutionary search, and in some circumstances detrimental, since the crossover operation often creates individuals that are highly dissimilar from either parent despite being derived from both of them. Simple mutation, on the other hand, always results in an individual that is a slight variation from its "parent" genotype. In fact, the Evolution Strategy (ES) paradigm relies exclusively on mutation [2]. Based on these arguments and preliminary work evolving with and without crossover, the decision was made to not use crossover in the evolution of UT^2. Further details about what methods were used to evolve UT^2's combat behaviour are given next.

5.4.1 Neuroevolution

Neuroevolution is the application of an EA to artificial neural networks. UT^2's combat behaviour was learned via constructive neuroevolution, meaning that the networks start with minimal structure and only become more complex as a result of mutations across several generations. The initial population of networks consists of individuals with no hidden layers, i.e. only input and output nodes. Furthermore, these networks are sparsely connected in a style similar to Feature Selective NeuroEvolution of Augmenting Topologies (FS-NEAT [27]). Initializing the networks in this way allows them to easily ignore any inputs that are not, or at least not *yet*, useful. Given the large number of inputs available to UT^2, it is important to be able to ignore certain inputs early in evolution, when establishing a baseline policy is more important than refining the policy.

Three mutation operators were used to change network behaviour. The weight mutation perturbs the weights of existing network connections, the link mutation adds new (potentially recurrent) connections between existing nodes, and the node mutation splices new nodes along existing connections. Recurrent connections transmit signals that are not processed by the network until the following time step, which makes them particularly useful in partially observable domains. In the context of reinforcement learning problems [22], such as UT2004, an environment is partially observable if the current observed state cannot be distinguished from other observed states without memory of past states. Recurrent connections help in these situations because they encode and transmit memory of past states. These mutation operators are similar to those used in NEAT [21].

This section explained the representation that was used to evolve policies for UT^2. The next section explains the algorithm controlling how the space of policies was searched.

5.4.2 Evolutionary Multiobjective Optimization

In multiobjective optimization, two or more conflicting objectives are optimized simultaneously. A multiobjective approach is important for domains like UT2004, which involve many conflicting objectives: kill opponents, conserve ammo, avoid damage, etc. Important concepts in dealing with multiple objectives are Pareto dominance and optimality. The following definitions assume a maximization problem. Objectives that are to be minimized can simply have their values multiplied by -1.

Definition 5.1 (Pareto Dominance) Vector $\mathbf{v} = (v_1, \ldots, v_n)$ dominates $\mathbf{u} = (u_1, \ldots, u_n)$ if and only if the following conditions hold:

1. $\forall i \in \{1, \ldots, n\} : v_i \geq u_i$, and
2. $\exists i \in \{1, \ldots, n\} : v_i > u_i$.

The expression $\mathbf{v} \succ \mathbf{u}$ denotes that $\mathbf{v}$ dominates $\mathbf{u}$.

Definition 5.2 (Pareto Optimality) A set of points $\mathscr{A} \subseteq \mathscr{F}$ is Pareto optimal if and only if it contains all points such that $\forall \mathbf{x} \in \mathscr{A}$: $\neg\exists \mathbf{y} \in \mathscr{F}$ such that $\mathbf{y} \succ \mathbf{x}$. The points in $\mathscr{A}$ are non-dominated, and make up the non-dominated Pareto front of $\mathscr{F}$.

The above definitions indicate that one solution is better than (i.e. dominates) another solution if it is strictly better in at least one objective and no worse in the others. The best solutions are not dominated by any other solutions, and make up the Pareto front of the search space. Therefore, solving a multiobjective optimization problem involves approximating the Pareto front as best as possible, which is exactly what EMO methods do. In particular, the EMO method used in this work is the Non-Dominated Sorting Genetic Algorithm II (NSGA-II [7]).

NSGA-II uses a $(\mu + \lambda)$ selection strategy. In this paradigm, a parent population of size μ is evaluated, and then used to produce a child population of size λ. Selection is performed on the combined parent and child population to give rise to a new parent population of size μ. NSGA-II uses $\mu = \lambda$.

NSGA-II sorts the population into non-dominated layers in terms of each individual's fitness scores. For a given population, the first non-dominated layer is simply the Pareto front of that population (usually not the same as the true Pareto front of the search space). If this first layer is removed, then the second layer is the Pareto front of the remaining population. By removing layers and recalculating the Pareto front, the whole population can be sorted. Individuals in layers dominated by fewer other layers are considered more desirable by evolution.

Elitist selection favours these individuals for inclusion in the next parent generation. However, a cutoff is often reached such that the non-dominated layer under consideration holds more individuals than there are remaining slots in the next parent population. These slots are filled by selecting individuals from the current layer based on a metric called *crowding distance*.

The crowding distance for a point p in objective space is the average distance between all pairs of points on either side of p along each objective. Points having an objective score that is the maximum or minimum for the particular objective are considered to have a crowding distance of infinity. For other points, the crowding distance tends to be bigger the more isolated the point is. NSGA-II favours solutions with high crowding distance during selection, because the more isolated points in objective space are filling a niche in the trade-off surface with less competition.

By combining the notions of non-dominance and crowding distance, a total ordering of the population arises by which individuals in different layers are sorted based on the dominance criteria, and individuals in the same layer are sorted based on crowding distance. The resulting comparison operator for this total ordering is also used by NSGA-II: the way that a new child population is derived from a parent population is via binary tournament selection based on this comparison operator.

Applying NSGA-II to a problem results in a population containing a close approximation to the true Pareto front (an approximation set) with individuals spread out evenly across the trade-off surface between objectives. The details of how this process was carried out in UT2004, as well as an explanation of how one network was selected from the resulting Pareto front, are covered in the next section.

5.4.3 Evolution of UT^2

How can the above techniques be used to generate a network for UT^2's battle controller? In order to evolve bots for UT2004, fitness objectives need to be designed to favour good behaviour, opponents against which the bots can evolve need to be chosen, and maps within which the Deathmatches will occur are needed. After evolving a population in this manner, the results of evolution need to be examined in order to pick an appropriate network to serve as the brain for UT^2's battle controller.

5.4.3.1 Fitness Objectives

Some of the objectives used to evolve UT^2 were the same as those used in [25] (Damage Dealt, Accuracy, Damage Received), though additional objectives were added to discourage collisions with level geometry and other agents, since such collisions are characteristic of bot-like behaviour.

- Damage Dealt: This objective measures both kills and damage dealt by the bot. It is possible to kill an opponent without being responsible for depleting all of its hit points, but the kill is still attributed to whoever delivered the final hit. Therefore, for each of the bot's kills, this fitness measure rewards it with an extra 100 fitness, since 100 is the starting health of all agents. Additionally, the bot keeps track of how much damage it has dealt so far to each opponent. These amounts are reset to zero when the corresponding opponent dies. At the end of the match, whichever value is highest is added to the fitness score. Thus this fitness measure rewards kills, as well as additional damage that comes short of a successful kill.
- Accuracy: This objective measures the accuracy of the bot in hitting opponents. In Sect. 5.3.2.3 some restrictions on UT^2's accuracy were described. These restrictions can be overcome if the bot chooses to stand still or otherwise move such that it can aim better. The exact measure used is the number of hits divided by the amount of ammo used. This measure works well for most weapons, but it has become clear since the competition that it does not make sense for some weapons. For example, each shard fired by the Flak Cannon registers as a separate hit, and the secondary fire of some weapons consumes more than one unit of ammunition even though they may only register a single hit when successful.
- Damage Received: This objective needs to be minimized. Each time the bot dies, it counts as 100 damage received. However many hit points fewer than 100 the bot has at the end of a match are added to this amount.
- Level Collisions: A level collision registers whenever the bot bumps into some aspect of level geometry, usually a wall. Because these collisions look awkward, and can lead to the bot getting stuck, the goal of looking human requires the bot to minimize collisions of this type.
- Agent Collisions: Bumping into other agents in the world can also look awkward and should be avoided, so this is another objective to be minimized.

Though each of the above objectives measures an aspect of performance that is important in a skilled bot, it is not necessarily the case that this is the best set to evolve with in order to discover quality Deathmatch behaviour. In particular, it would probably have been better to evolve with fewer objectives, since NSGA-II's performance is known to degrade with increased numbers of objectives. In future work, it would make sense to combine the collision objectives into a single objective, or perhaps simply drop them both. The accuracy objective is also problematic, as described above, and will need to be fixed before use in future competitions.

One meta-objective was also used in order to help evolution effectively explore the range of possible behaviours. Behavioural diversity [17, 18] was used to encourage different types of behaviours to assure that evolution did not get stuck in local optima. The objective is a generalized form of behavioural diversity [18] that uses a different set of randomized input vectors per generation to generate a behaviour vector for each individual in the population. A behaviour vector is the concatenation of all output vectors derived from processing each of the randomized input vectors through an individual's neural network. The behavioural diversity objective is to maximize the average distance of an individual's behaviour vector in Euclidean space from all other behaviour vectors in the population, thus favouring diverse network/agent behaviour.

5.4.3.2 Agents

Given these objectives, decisions still need to be made regarding the scenario in which the bot will evolve. Because BotPrize involves competing simultaneously against multiple opponents, it was decided that the bot should evolve in a similar scenario. Evolving against native UT2004 bots would have been ideal, but because Pogamut 3 was fairly new at the time UT^2 was being developed, an easy way to do this was not yet available (support has since been added). Therefore, the opponents for the evolving bots were instances of the Hunter bot, a simple but effective scripted bot that is provided with the Pogamut 3 platform. Specifically, during evolution one bot participated in a Deathmatch against five Hunter bots per evaluation.

In order to evolve a battle controller that would eventually be used by UT^2 in the competition, a slightly modified version of UT^2's architecture was used. The architecture presented earlier (Sect. 5.3.1) was changed in two ways. First, the *JUDGE* module was disabled (the Hunters could not judge either), since in a scenario consisting entirely of bots there is no sense in judging. Secondly, the *HUMAN TRACE CONTROLLER* of the *UNSTUCK* module was replaced with a simple hand-coded controller for getting unstuck. This `SimpleUnstuckController` tries to move away from any obstacle that it collides with, and resorts to one of several random movement actions if it is stuck for some other reason. The `SimpleUnstuckController` was used because during evolution the version based on human traces was not yet fully developed. Both the `SimpleUnstuck Controller` and the *HUMAN TRACE CONTROLLER*, which was actually used by UT^2 in the competition, are described in full detail in Chap. 6.

5.4.3.3 Maps/Levels

When evolving UT^2's battle controller, the exact level used and the length of evaluation depended on the current generation. The sequence of levels used was meant to increase in size and challenge, so that early generations would have a chance to learn basic behaviours before having to deal with more complicated situations. Evaluations in earlier levels were shorter than in later levels, both because more time is required to find enemies in larger levels, and because it is not worthwhile to evaluate networks for a long time in early generations, since much time would likely be wasted on bad solutions. The exact level/duration sequence was `DM-TrainingDay`/100 s, `DM-Corrugation`/200 s, `DM-DG-Colosseum`/300 s, `DM-GoatswoodPlay`/400 s, and then `DM-IceHenge`/500 s.

Though this sequence served well for the purpose of evolving combat behaviour, competition experience has indicated that a better sequence is likely possible. For example, although Goatswood and IceHenge have challenging water hazards, Colosseum is difficult for the bot to deal with because it is easy to get lost and stuck in the columns. Furthermore, lessons learned by bots in earlier levels may have been forgotten in order to better specialize in later levels. Though the cost in evaluation time would be high, in future work it might be better to have bots face Deathmatches in multiple levels per evaluation.

Twenty generations were spent on `DM-TrainingDay`, and ten generations on each subsequent level, for a total of 60 generations. This is a very small number of generations, and better performance could likely have been achieved given more time. However, evaluation time is a major bottleneck in UT2004, so a lesser number of generations was used. For the same reason, the population size was only 20, which due to NSGA-II's $(\mu + \lambda)$ selection strategy meant that each generation involved selection upon a population of size 40. This number is fairly small for evolutionary computation, but good results were obtained despite this practical restriction.

5.4.3.4 Results of Evolution

Figure 5.5 compares values of the hypervolume indicator [29] for both the starting and final parent populations in each of the levels of BotPrize. The hypervolume indicator measures the hypervolume of the region dominated by all points in a given approximation to a Pareto front. The hypervolume indicator is special in that it is a Pareto-compliant metric [28], meaning that an approximation set that completely dominates another approximation set is guaranteed to have a higher hypervolume.

Other Pareto-compliant metrics are the multiplicative (I_ε^1) and additive ($I_{\varepsilon+}^1$) unary epsilon indicators [15]. Both indicators are defined with respect to a reference set R. The multiplicative indicator I_ε^1 measures how much each objective for each solution in a set would have to be multiplied (divided for minimization) by such that each solution in R would be dominated by or equal to a point in the resulting set. The additive indicator $I_{\varepsilon+}^1$ measures how much would have to be added (subtracted for minimization) to each objective in each solution such that each point in R would

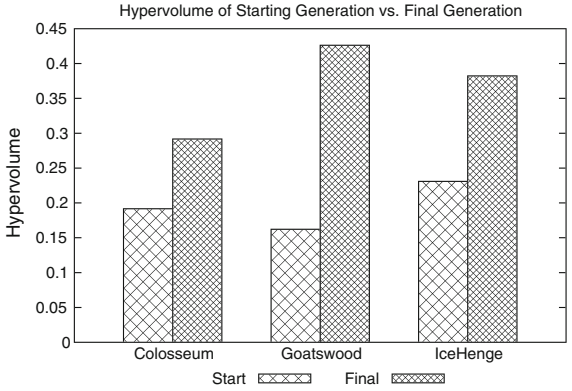


Fig. 5.5 Hypervolume of first generation versus final generation. This figure shows the gains made by evolution in each of the three levels used in BotPrize. In order to get accurate scores, each of the 20 members of both the start and final parent generations was evaluated in each level for 500 s against five Hunter bots ten times each. The objective scores for each individual were the averages of scores attained in each objective across the ten trials. Pareto fronts of the resulting scores were calculated, and the scores for each objective were normalized according to the maximum magnitude scores for each objective in the given map (hypervolumes between maps are not comparable). These normalized fronts were used to calculate hypervolume. In each level, the hypervolume in the final generation is greater than in the start generation, showing that the population evolved to dominate a larger region of objective space across generations

be dominated by or equal to a point in the modified set. For both indicators, smaller values are better because they indicate that a smaller adjustment is needed to dominate the reference set.

The scores from the start and end generations were compared using these unary epsilon indicators with a separate reference set for each level defined as the super Pareto front (Pareto front of several Pareto fronts) of the fronts from the start and end generations, as suggested in [15]. The results are shown in Fig. 5.6.

The evolved population has better hypervolume and epsilon values, but it is actually not the case that the approximation sets from the final generation are strictly better than those from the starting generation, although the sets from the final generation do tend to contain points that completely dominate points in the first generation sets. The lack of complete domination is likely caused by use of such small populations and so many objectives, which in combination make it hard for the population to cover all trade-offs.

5.4.3.5 Network Selection

There are still many trade-offs to consider, however, and one network had to be selected from all those available to compete in BotPrize 2010. In order to get a bot that performed well, the population was first filtered based on the highest Deathmatch

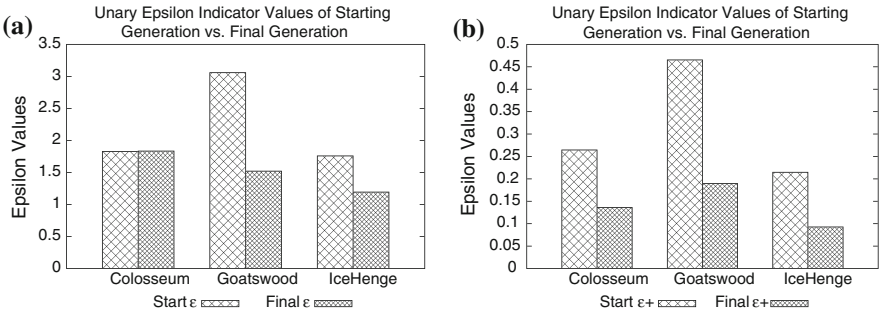


Fig. 5.6 Epsilon indicator values of starting generation versus final generation. The normalized Pareto fronts used to compute the hypervolumes for the first and final generations in each level were used to compute unary epsilon indicator values with respect to reference sets, which were the super Pareto fronts of the two approximation sets under consideration in each level. With the exception of the I_{ε}^{1} values for Colosseum, all epsilon values indicate that the solutions in the final generation are better than those in the first generation. **a** Multiplicative indicator I_{ε}^{1}; **b** additive indicator $I_{\varepsilon+}^{1}$

scores across all levels, in a manner similar to [25]. This process resulted in a set of three high-scoring networks. Each of these three networks attained a high score by being aggressive, which was considered a human trait. These bots also tended to die more as a result of their aggressiveness (more on this in Sect. 5.5.1).

The final decision of which network to use in BotPrize was made by the authors along with the help of two human volunteers. In Deathmatches between four humans (two of which were involved in programming the bot), the three candidate bots, and one native UT2004 bot, the humanness ratings (number of human judgements divided by total judgements) across multiple matches were used to single out the most human-like bot of the available candidates. This bot became UT^2 in the 2010 BotPrize competition.

5.5 Evaluation

Having fully described how UT^2 was developed, it is now time to evaluate UT^2 to see how well it performs in UT2004. UT^2's performance is analyzed both in terms of its ability to achieve high fitness scores, and in terms of how human-like the judges in BotPrize 2010 considered it to be.

5.5.1 Evaluation of Objective Scores

The ultimate goal of UT^2 is to look as human-like as possible, but the route to accomplishing this goal was evolving for good objective performance. This section deals with the quality of UT^2's performance with respect to the objectives used in

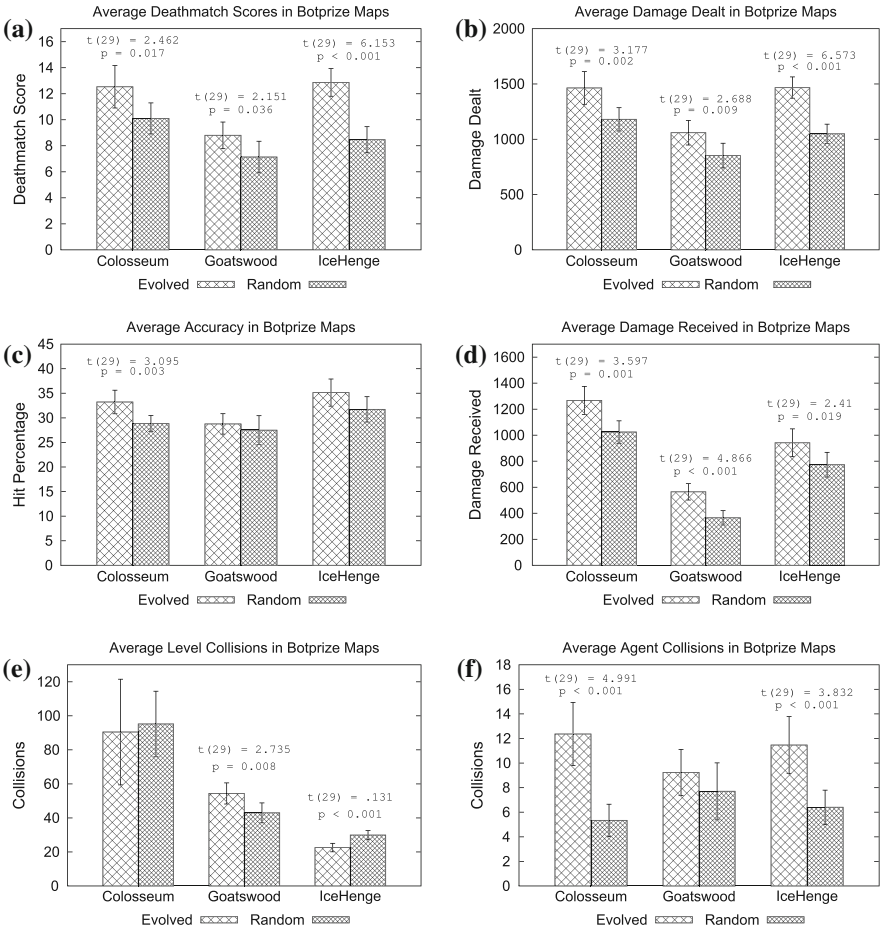


Fig. 5.7 Each figure compares the average performance of the evolved network from the competition bot to that of the bot using random action selection in the battle controller. Performance in each of the three levels used in BotPrize 2010 is shown. Bot performance is measured in competition with five Hunter bots in 500 s matches. Averages are across 30 trials, and 95 % confidence intervals are shown. For each score and level, *t*-tests were done to compare the evolved network to random action selection. When the difference is significant, the resulting *t* and *p* values for the test are shown above the *bars*. Though the evolved network is not significantly better than a random controller in some objectives, its performance in these objectives can be attributed to focusing on high-scoring networks when the evolved network was chosen from those available on the trade-off surface. **a** Average deathmatch score; **b** average damage dealt; **c** average accuracy; **d** average damage received; **e** average level collisions; **f** average agent collisions

evolution as well as Deathmatch score, which played an important role in deciding which network from the Pareto front to use in the competition.

Figure 5.7 compares the performance of UT^2 using the evolved network chosen for BotPrize 2010 with the same bot using a randomized action selector for the

battle controller. Specifically, randomized vectors are treated like output vectors from a network in order to determine combat behaviour. In these evaluations, both versions of the bot had access to the complete human-trace-based *UNSTUCK* module (Chap. 6) used in the final competition. Evaluations were performed for 500 s in each competition level against five Hunter bots.

The results show that purposely picking a network based on Deathmatch score and aggressiveness has resulted in an ability to deal significantly more damage, and therefore get significantly better scores, than a random battle controller. Accuracy was generally better too, though only significantly so in Colosseum.

However, favouring aggressive, score-increasing behaviour has resulted in significantly more damage received in all levels. This result highlights the importance of a multiobjective approach in helping to find the best trade-off between objectives. It makes sense that aggressively pursuing enemies and actively engaging in combat will result in both more frags earned, as well as more deaths experienced.

In terms of collisions with level geometry, differences between the evolved network and the random bot were inconsistent across levels. In general, this behaviour seems more level-dependent than bot-dependent: level collisions are more common in Goatswood than in IceHenge and more common in Colosseum than in either of the other levels. Since the design of the bot depended primarily on ray traces to detect surrounding obstacles, and these ray traces were unavailable in the competition (Sect. 5.3.2.1), it is not surprising that collision behaviour does not seem strongly affected by evolution. Though battle style clearly affects collision frequency, this objective did not play as important a role in final network selection as the others did; this decision may have been a mistake (Sect. 5.5.2).

In terms of collisions with enemy agents, the evolved network is worse than the random controller. Once again, this behaviour is a result of favouring aggressive combat behaviour. Many collisions occur because the bot is chasing the opponent. An aggressive player is more likely to be near its opponents, and therefore also likely to bump into them more often. Furthermore, these results are based on battle against Hunter bots, which mindlessly rush at their opponents in a way unlike humans. The behaviour of the Hunter bots made agent collisions even more likely. Since in all cases the actual number of collisions is fairly small (averages below 13), it was assumed that the number of collisions would drop to an insignificant level when fighting human opponents.

Given the priorities across objectives, the evolved network has succeeded in performing well in the Deathmatch domain. To what degree this good empirical performance translated into human-like performance in the competition is discussed next.

5.5.2 Evaluation of Human-Like Performance

The results from BotPrize 2010 are in Table 5.1. UT^2 placed second among entrants, though the humanness rating of the native UT2004 bots is also shown (native bots have the advantage of being written in UT2004's `UnrealScript`, which has some

Table 5.1 BotPrize 2010 Results (UT^2 highlighted)

Bot	Humanness (%)
Native UT2004 bot	35.3982
`Conscious-Robots`	31.8182
`UT^2`	27.2727
`ICE-2010`	23.3333
`Discordia`	17.7778
`w00t`	9.3023

Humanness equals the number of human judgments divided by the total judgments, all multiplied by 100. UT^2 beat three entries to get second place

advantages over Gamebots in terms of sensing, latency and action execution). The lowest humanness rating for a human was 35.4839 %.

UT^2 did not win, but it did beat three other entrees, losing only to `Conscious-Robots`. No competitor has yet won the grand prize, requiring a humanness rating of at least 50 %. In fact, of the seven human judges, only two had humanness ratings over 50 %. However, this result could be in part due to the fact that the new judging format actually encourages humans to act like bots in order to trick opponents into losing points for bad judgments. This strategy is one of the distortion effects mentioned in Chap. 9.

In any case, the humans were still clearly more human than the bots, although compared to previous competitions the gap is narrowing. Based on the many demo files made during the competition, some analysis of bot behaviours related to specific judgments is possible. Demos and logs of the competition are available online.[1] Each one allows a viewer to see exactly what any given judge saw during specific matches of the competition.

Most actions taken by UT^2 seem fairly human, or are at least difficult to distinguish from human actions. This statement is based on the fact that most humans interacted with UT^2 on several distinct occasions before making any sort of judgment. However, such extensive interactions make it hard to discern what aspect of the bot's behaviour influenced the judgment. Despite this difficulty, certain behaviours were noticed that judges tended to associate, though not always correctly, with either bots or humans.

Most judges assumed it was human to stand still for long periods with sniping weapons while being oblivious to nearby surroundings. Although UT^2 would sometimes stand still to get a better shot when firing, these pauses were usually brief moments between dodging actions. This behaviour is one case where UT^2's behaviour was more effective in combat, but less human-like. However, lack of this behaviour did not seem to cause UT^2 to be judged as a bot; it simply meant that UT^2 missed some chances to be judged as human.

One problematic behaviour exhibited by UT^2 is actually a bug that is not consistent with the description of the battle controller in Sect. 5.3.2. In order to appear

[1] http://www.botprize.org/2010.html

Table 5.2 Number of judgments of each type against UT^2 and the resulting humanness, divided by map

Map	Human	Bot	Humanness (%)
Colosseum	2	6	25.00
Goatswood	2	9	18.18
IceHenge	5	9	35.71

The bot was least human in Goatswood, which is unfortunate because five sessions were played in Goatswood, whereas four were played in IceHenge and three in Colosseum

attentive, the bot is supposed to look at the targeted opponent during all combat actions. However, this was actually not the case for the *GOTO ITEM* action: the bot would look at, and sometimes even shoot in the direction of, the item towards which it was moving instead of the opponent it was fighting. This bug caused UT^2 to be judged as a bot on several occasions.

Other issues seem to be more level specific. Table 5.2 breaks down judgments against UT^2 by level, and shows that the bot fared best in IceHenge and worst in Goatswood. The effect of the level on the humanness rating of the bot is closely tied to its ability to navigate within that level. Judgements of UT^2 that seem to have been based on its navigational abilities are discussed in Chap. 6.

The current level also affected the bot's combat behaviour. UT^2 likely fared well in IceHenge because most areas are wide open with few obstacles. Also, the fact that the last ten generations of evolution were spent in IceHenge probably made the bot's behaviour better tailored to this level than others. In contrast to IceHenge, Goatswood is mostly comprised of narrow corridors and has several waist-high obstacles over which the players must jump. A few of the bot judgments that UT^2 received in Goatswood seem to be the result of the bot unnecessarily colliding with walls, however briefly, in the midst of dodging during combat. The judges presumably expected humans to be more aware of their surroundings so as to avoid such contact. These judgments indicate that the Level Collisions objective should have been considered more important when deciding which network to use in BotPrize.

Humans also expected other humans to be aware of the judging aspect of the competition. Some judges would purposely miss with the judging gun in combat to see if they could elicit human reactions from their opponents. It is impossible to know what individual judges expected in these situations, but completely ignoring the judging gun and attacking as normal seems to have been considered bot-like. UT^2 was labelled a bot at least once for such behaviour.

Humans also expected humans to use the judging gun. There are some occasions where UT^2 killed a human with a correct judgment, and was in turn immediately judged as a human by the judge the next time the two met. There are other occasions where the exact reason UT^2 was judged as a human was unclear due to the large number of interactions preceding the judgment, but in most cases where a judge saw UT^2 several times before judging it as human, at least one of the things the judge witnessed was UT^2 using the judging gun.

However, despite the role that judging behaviour may have played in earning human judgments for UT^2, it is not necessarily true that judging behaviour is vital to the competition. Neither the winner, `Conscious-Robots` (Chap. 7), nor the more human native UT2004 bot did any judging at all. It seems that bots can get away with not judging and still look human due to the fact that most interactions are brief and spaced out across the match. In other words, there are many chances to use the judging gun out of sight of any given opponent, so no human would necessarily expect to see every opponent use the judging gun.

Other judgments against UT^2 are harder to interpret. Sometimes a judge saw UT^2 many times in a match, and eventually judged the bot as a human near the very end. Such judgments likely indicate that over the course of several interactions the bot did nothing overtly bot-like, and the most sensible course of action given little remaining time was to judge the bot as human.

In a few cases, UT^2 was quickly judged based on very little interaction. It is not clear from the demo replays what criteria the judges were using in these cases. It is possible in some cases that judges are able to discern the identity of an opponent simply by subtle movement patterns within mere seconds, but it is also possible that some judgments are the result of errors, such as mistaken identity or weapon misfire. Throughout all of the demo files there are many instances of snap judgments, both correct and incorrect.

Still, regardless of the reason behind such judgments, they must be accounted for in order to succeed at BotPrize. Ideas on how to do this, as well as some general ideas about how a bot can appear more human, are the topic of the next section.

5.6 Discussion and Future Work

Most of UT^2's behaviour seems to be passably human. Many judges were unable to come to a conclusion about the bot's humanness, even after three or more interactions. However, UT^2 would look more human if it both performed certain actions that most humans are certain a human would do, and if it avoided the few very bot-like actions that crept into its behaviour.

Fixing the bug caused by the *GOTO ITEM* action is simple. The availability of working ray-traces in future competitions should also help the bot avoid bumping into obstacles as often. There are also ideas for improving navigation with the use of human traces, discussed in Chap. 6. With regards to how the combat behaviour was evolved, there is room for improvement.

Obvious steps to improve the performance of the bot would be to evolve with a larger population for more generations, but there are also ways in which the basic evolutionary setup can be improved. These improvements are discussed below.

5.6.1 Opponent Interactions

One issue regards the opponents against which bots evolve. For practical reasons, these opponents are themselves bots. The Hunter bot was used to evolve UTˆ2, though the native bots would probably make better opponents. However, it would likely be even better to evolve against many different types of bots. The justification for this approach is that each of the BotPrize participants had a different play style and skill level. An evolved bot should be accustomed to the possibility that different opponents behave differently, and more importantly, a bot evolved against varied opponents is more likely to learn behaviours to deal with different types of players. In retrospect, evolving against the Hunter only may have resulted in the evolution of a one-size-fits-all behaviour that is mostly effective, but perhaps too predictable and/or bot-like. Humans are very good at adapting and improvising. Having learned how to respond to a wide array of opponent strategies should at least give the impression that the bot is improvising.

An important concept when considering how agents interact is "attention": humans pay attention to the agents they interact with, and generally continue to do so until some note-worthy event shifts that attention. The opponent-relative movement commands of UTˆ2 assure that it pays attention to whichever opponent it is fighting, but when multiple opponents are present, UTˆ2 picks one of them to pay attention to according to a scripted routine (see under "Enemy Shooting Sensor" in Sect. 5.3.2.1). It might be more human-like to use a cognitive approach to this attention problem as done by the winning bot `Conscious-Robots` (Chap. 7).

Linked to the issue of interaction is the idea of mimicry. Mimicry is important because it establishes an agent's ability to comprehend what another agent is doing, and utilize that knowledge for its own gain. Mimicry can involve copying what an opponent is doing at the moment, or it can mean that agents mirror each other's behaviour, such that one is always countering the other to maintain equilibrium. An example of the first type would be jumping or dodging in a similar manner to an opponent. An example of the second type would be maintaining distance during combat, such that the bot moves forward when its opponent moves backward and vice versa. In either case, such behaviour would make a bot look more human when fighting a human judge, since the bot would be acting the way the human judge acts.

One potential way to make such mimicry evolvable is to have opponent-relative input sensors in addition to opponent-relative actions. Rather than simple awareness of where an opponent is, the bot could sense whether the opponent was advancing, retreating, strafing, jumping, etc. If the bot can sense when an opponent is performing an action that it can also perform, learning to act the same way via a neural network would be quite easy. Of course, such behaviour would only be favoured by evolution if it also improved fitness, but given that humans favour such strategies it is believed that mimicking behaviour will indeed lead to increased fitness.

Mimicry could be more directly encouraged by evolution if some measure of mimicry were used as a fitness function. However, rewarding mimicry directly could result in evolved bots that behave in non-human ways when interacting with human

judges that behave stupidly as a ruse to gauge humanness, or with other bots, which are of course bot-like.

5.6.2 Scope of Evolved Policy

A limitation on how UTˆ2's behaviour was evolved is that only the battle controller was evolving within a bot that had many other components. While this approach assured that its combat behaviour would make sense in the context of its other behaviours, the approach is perhaps too inefficient. It may be better to evolve the combat behaviour separately, at least initially, within a specialized combat scenario where the majority of the bot's other modules are disabled. For example, the bot could be given infinite ammo in a small level, thus making navigation and weapon collection irrelevant, and freeing evolution to focus on how the bot behaves in combat.

Such an evolved battle controller could be integrated with other evolved subcontrollers to build up a hierarchical controller as was done by van Hoorn et al. [25], whose approach was directly based on that of Togelius [24]. The method used in these works involves evolving the components of a subsumption architecture [3] within several separate subtasks leading up to the full task. Learning good behaviour for many small tasks in an incremental way is easier for most machine learning methods, but requires a knowledgeable human to construct the training and control hierarchies that define the agent's final behaviour.

Though such approaches may make learning easier, for any evolved subcomponent, one must keep in mind that evolution will favour increased fitness, potentially at the cost of human behaviour. In order for an evolved controller to act like a human, it must be both constrained as humans are and allowed to easily carry out actions that are common and easy for humans to carry out. However, the work of determining the proper constraints is task specific, and requires some knowledge both of how humans perform in the task, and of how a bot is likely to cheat at the task.

5.6.3 Weapon Usage

Evolving the battle controller in isolation would make it easier to control what weapons the bot has. It would then be possible to evolve a specialized controller for each weapon, or at least each class of weapons. Humans expect other humans to have varied behaviour across weapons. Sniping weapons are an obvious example. Although UTˆ2 knew some information about its current weapon, this information may not have been enough to serve as a basis for different combat styles, especially since some of the information was faulty (Sect. 5.3.2.1). Simply having separate controllers for each weapon would assure the bot's behaviour matched the weapon. However, such an approach would take even more time to evolve, and would be brittle with respect to new weapons, even if they were similar to existing ones.

An alternative option is to evolve multi-modal networks [19]. The networks have distinct output modes for different situations, which seems well-suited to having different behaviours for different types of weapons.

Knowing which weapon to use in a given situation is also an important aspect of gameplay in UT2004. In past BotPrize competitions, the University of Texas at Austin's entry (named `U Texas`) learned weapon preferences automatically [12]. The bot would learn estimates of the expected damage and accuracy of each weapon in each of the three ranges used by UT^2's static weapon lookup rules.

The original intent was to integrate weapon preference learning into UT^2, but this learning method was afflicted with the same problems that make the Accuracy objective (Sect. 5.4.3) problematic. Basically, the Gamebots protocol registers each source of damage separately, which makes gauging the accuracy of weapons that fire multiple projectiles at once difficult. Furthermore, accuracy is not as important for weapons that have higher rates of fire. Because of these difficulties, it was decided that UT^2 would use static weapon preferences instead. However, including weapon preference learning is still a good idea, which future versions of UT^2 will likely include, once a way to work around the limitations of Gamebots is found.

5.7 Conclusion

Evolving neural networks to provide the combat behaviour for the UT^2 bot in UT2004 helped it earn second place in the 2010 BotPrize competition. The key to evolving human-like behaviour, despite evolving for raw performance, is to restrict the actions available to the bot to common human actions, and to filter the overall bot behaviour such that the bot is restricted in ways that humans are. Evolving the bot to perform well in the context of human limitations naturally results in human-like performance. The UT^2 bot focused on the most obvious of such limitations, but much more is possible. By further tailoring bot actuators and sensors in this manner it should be possible to evolve more human-like bots for UT2004 and other domains in the future.

Acknowledgments The authors would like to thank Niels van Hoorn for the use of his source code in getting started evolving bots in UT2004. They would also like to thank Christopher Tanguay and Peter Djeu for volunteering to critique and evaluate versions of UT^2. This research was supported in part by the NSF under grants DBI-0939454 and IIS-0915038 and by the Texas Higher Education Coordinating Board grant 003658-0036-2007.

References

1. Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S.: Gamebots: a 3D virtual world test-bed for multi-agent research. In: Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS (2001)

2. Bäck, T., Hoffmeister, F., Schwefel, H.P.: A survey of evolution strategies. In: Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 2–9 (1991)
3. Brooks, R.A.: A robust layered control system for a mobile robot. IEEE J. Robot. Autom. **2**(10), 14–23 (1986)
4. Bryson, J.J.: Intelligence by design: principles of modularity and coordination for engineering complex adaptive agents. Ph.D. thesis, Massachusetts Institute of Technology (2001)
5. Butz, M., Lonneker, T.: Optimized sensory-motor couplings plus strategy extensions for the TORCS car racing challenge. In: Computational Intelligence and Games, pp. 317–324 (2009)
6. Darwin, C.: On the Origin of Species by Means of Natural Selection or the Preservation of Favored Races in the Struggle for Life. Murray, London (1859)
7. Deb, K., Agrawal, S., Pratab, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. PPSN VI, pp. 849–858 (2000)
8. Fogel, D.B., Atmar, J.W.: Comparing genetic operators with gaussian mutationsin simulated evolutionary processes using linear systems. Biol. Cybern. **63**(2), 111–114 (1990)
9. Gemrot, J., Kadlec, R., Bida, M., Burkert, O., Pibil, R., Havlicek, J., Zemcak, L., Simlovic, J., Vansa, R., Stolba, M., Plch, T.C.B.: Pogamut 3 can assist developers in building AI (not only) for their videogame agents. Agents for Games and Simulations, LNCS, vol. 5920. Springer, Heidelberg (2009)
10. Gomez, F., Miikkulainen, R.: Active guidance for a finless rocket using neuroevolution. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 2084–2095. Morgan Kaufmann, San Francisco (2003). http://nn.cs.utexas.edu/keyword?gomez:gecco03
11. Haykin, S.: Neural Networks, A Comprehensive Foundation. Prentice Hall, Upper Saddle River (1999)
12. Hingston, P.: A Turing Test for computer game bots. IEEE Trans. Comput. Intell. AI Games **1**(3), 169–186 (2009)
13. Hingston, P.: A new design for a Turing Test for bots. In: Computational Intelligence and Games (2010)
14. Isla, D.: Managing complexity in the Halo 2 AI system. In: Proceedings of the Game Developers Conference, San Francisco, CA (2005). http://www.gamasutra.com/gdc2005/features/20050311/isla-01.shtml
15. Knowles, J., Thiele, L., Zitzler, E.: A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers. TIK Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich (2006)
16. Kohl, N., Miikkulainen, R.: Evolving neural networks for strategic decision-making problems. Neural Networks **22**, 326–337 (Special issue on Goal-Directed Neural Systems) (2009)
17. Mouret, J.B., Doncieux, S.: Using behavioral exploration objectives to solve deceptive problems in neuro-evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 627–634. ACM (2009). doi:10.1145/1569901.1569988
18. Schrum, J., Miikkulainen, R.: Evolving agent behavior in multiobjective domains using fitness-based shaping. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 439–446. Portland, Oregon (2010). http://nn.cs.utexas.edu/?schrum:gecco10
19. Schrum, J., Miikkulainen, R.: Evolving multi-modal behavior in NPCs. In: Computational Intelligence and Games, pp. 325–332 (2009). http://nn.cs.utexas.edu/?schrum:cig09
20. Stanley, K.O., Bryant, B.D., Karpov, I., Miikkulainen, R.: Real-time evolution of neural networks in the NERO video game. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence (2006). http://nn.cs.utexas.edu/keyword?stanley:aaai06
21. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**, 99–127 (2002). http://nn.cs.utexas.edu/keyword?stanley:ec02
22. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998). http://citeseer.ist.psu.edu/sutton98reinforcement.html
23. Turing, A.M.: Computing machinery and intelligence. Mind **59**(236), 433–460 (1950)
24. Togelius, J.: Evolution of a subsumption architecture neurocontroller. J. Intell. Fuzzy Syst. **15**, 15–20 (2004)

25. van Hoorn, N., Togelius, J., Schmidhuber, J.: Hierarchical controller learning in a first-person shooter. In: Computational Intelligence and Games, pp. 294–301 (2009)
26. Waibel, M., Keller, L., Floreano, D.: Genetic team composition and level of selection in the evolution of multi-agent systems. Evol. Comput. **13**(3), 648–660 (2009)
27. Whiteson, S., Stone, P., Stanley, K.O., Miikkulainen, R., Kohl, N.: Automatic feature selection in neuroevolution. In: Proceedings of the Genetic and Evolutionary Computation Conference (2005). http://nn.cs.utexas.edu/keyword?whiteson:gecco05
28. Zitzler, E., Brockhoff, D., Thiele, L.: The hypervolume indicator revisited: on the design of pareto-compliant indicators via weighted integration. In: Conference on Evolutionary Multi-Criterion Optimization (EMO 2007), vol. 4403, pp. 862–876 (2007)
29. Zitzler, E., Thiele, L.: Multiobjective optimization using evolutionary algorithms—a comparative case study. In: Conference on Parallel Problem Solving from Nature, pp. 292–304 (1998)

Chapter 6
Believable Bot Navigation via Playback of Human Traces

Igor V. Karpov, Jacob Schrum and Risto Miikkulainen

Abstract Imitation is a powerful and pervasive primitive underlying examples of intelligent behaviour in nature. Can we use it as a tool to help build artificial agents that behave like humans do? This question is studied in the context of the BotPrize competition, a Turing-like test where computer game bots compete by attempting to fool human judges into thinking they are just another human player. One problem faced by such bots is that of human-like navigation within the virtual world. This chapter describes the Human Trace Controller, a component of the UT^2 bot which took second place in the BotPrize 2010 competition. The controller uses a database of recorded human games in order to quickly retrieve and play back relevant segments of human navigation behaviour. Empirical evidence suggests that the method of direct imitation allows the bot to effectively solve several navigation problems while moving in a human-like fashion.

6.1 Introduction

Building robots that act human requires solutions to many challenging problems, ranging from engineering to vision and natural language understanding. Imitation is a powerful and pervasive primitive in animals and humans, with recently discovered neurophysiological correlates [1]. Children observing adult behaviour are able to mimic and reuse it rationally even before they can talk [2]. As robotics platforms

I. V. Karpov (✉) · J. Schrum · R. Miikkulainen
The University of Texas at Austin, Austin, TX 78712, USA
e-mail: ikarpov@cs.utexas.edu

J. Schrum
e-mail: schrum2@cs.utexas.edu

R. Miikkulainen
e-mail: risto@cs.utexas.edu

P. Hingston (ed.), *Believable Bots*, DOI: 10.1007/978-3-642-32323-2_6,

continue to develop, it is becoming increasingly possible to use similar techniques in human-robot interaction [3].

How can we use imitation when building agents with the explicit goal of human-like behaviour in mind? We study this question in the setting of the BotPrize 2010 competition, which explicitly rewards agents for exhibiting believable human-like behaviour [4].

One problem faced by such bots is that of human-like navigation within the virtual world. Due to problems in level design and the interface used by the bots when acting in the environment, bots can get stuck on level geometry or fail to appear human when following the built in navigation graph.

As a way to address such challenges, this chapter introduces the *Human Trace Controller* (HTC) , a component of the UTˆ2 bot inspired by the idea of direct imitation of human behaviour. The controller draws upon a previously collected database of recorded human games, which is indexed and stored for efficient retrieval. The controller works by quickly retrieving relevant traces of human behaviour, translating them into the action space of the bot, and executing the resulting actions. This approach proves to be an effective way to recover from navigation artefacts in a human-like fashion.

This chapter is organized as follows. Related work is discussed in Sect. 6.2. The necessary background including a description of the BotPrize competition and domain used in it is discussed in Sect. 6.3. The main focus of this chapter, the Human Trace Controller, is described in detail in Sect. 6.4. Section 6.5 presents the results of qualitative and comparative evaluations of the controller. Sections 6.6 and 6.7 discuss future work and conclusions.

6.2 Related Work

An active and growing body work uses games as a domain to study Artificial Intelligence [5–8].

The use of human player data recorded from games in order to create realistic game characters is a promising direction of research because it can be applied both to games and to the wider field of autonomous agent behaviour. This approach is closely related to the concept of Imitation Learning or Learning from Demonstration, especially when expanded to generalize to unseen data [9–11].

Imitation of human traces has previously been used to synthesize and detect movement primitives in games [12], however this approach has not been evaluated in the framework of a human-like bot competition. The use of trajectory libraries was introduced for developing autonomous agent control policies and for transfer learning [13, 14]. Hladky et al., developed predictive models of player behaviour learned from large databases of human gameplay and used them for multi-agent opponent modeling [15]. Human game data is also collected in an attempt to design non-player characters capable of using natural language [16]. Imitation learning using supervised models of human drivers was used in order to train agent drivers in the

TORCS racing simulator [17]. In robotics, imitation learning approaches have been shown effective as well, for example in task learning and programming robosoccer players in simulation [18]. Statistical analysis of player trajectories was used in order to detect game bots in the Quake first person shooter game [19]. Sukthankar et al. use similar techniques in order to assign teams and recognize team player behaviour in multiagent settings [20]. Most recently, a competitor team, `ICE`, is using an interface for creating custom recordings of human behaviour in the BotPrize competition [21].

While human behaviour traces and learning from demonstration techniques are finding increasing use in both games and robotics applications, the BotPrize competition offers a unique opportunity to test such methods in creating human-like behaviour directly. The challenge of combining imitation and demonstration methods with other types of policy design methods remains to be met.

6.3 Background

This section describes the domain of Unreal Tournament, the BotPrize Competition, and some of the challenges to developing human-like game bot behaviour posed by the software interface used when developing the bot.

6.3.1 Unreal Tournament 2004

Unreal Tournament 2004 is a commercial sequel in a series of first person shooter computer games developed by Epic Games and Digital Extremes [22]. After it was published in 2004, the game received Multiplayer Game of the Year awards from IGN, Gamespy and Computer Gaming World.

The Unreal 2004 game engine consists of a server which runs the game simulation including 3D collision detection, physics, player score, statistics, inventories and events. Importantly, the Unreal 2004 game engine includes an embedded scripting system that uses Unreal Script, an interpreted programming language that provides an API to the game engine. This scripting interface is used by higher-level wrappers such as `GameBots2004` and `Pogamut` to allow external programs to control bot players and receive information about their state [23, 24]. Unreal Script also allows the recording of detailed game traces from games played by humans and bots.

Players connect to the server using Unreal Tournament clients (either locally or via a network using TCP and UDP protocols). Clients provide 3D graphics and audio rendering of the view of the Unreal level from the perspective of the player, and allow the player to control their character via keyboard and mouse commands. Commands are customizable, but basic keyboard controls include movement forward, back, left and right, jumping up, crouching, and selecting a weapon from the player's inventory, while the mouse allows the player to turn, aim, and fire primary and secondary weapons. Each player has some amount of health and armour and an

array of weapons from one of the predefined weapon types as well as some amount of ammunition for each.

Multiplayer games can include up to sixteen players per game simultaneously. These players can include both humans and bots. Normally players can choose from a number of *native bots* within Unreal—these are implemented as internal subroutines within the engine and can have a different skill level depending on a numerical parameter. Both bots and human players can choose from a wide variety of avatars or *skins* which represent them during the game. These are all humanoid in appearance and roughly similar in size—the engine uses the same underlying character animations during movement, using custom animations only in special situations such as taunts.

The goal of the normal Death Match mode is to be the first to eliminate a predetermined number of opponents by hitting them with weapon fire and reducing their health and armour to zero. Players who are killed quickly re-spawn at one of several predefined spawn locations on the level map. Games can optionally have a time limit, at the end of which the player with the highest number of frags (kills) wins.

6.3.2 The BotPrize 2010 Competition

The BotPrize competition aims to create a Turing-like interactive test for bot behaviour [4, 25]. During the 2010 competition, five teams competed during the course of three one-hour rounds consisting of three 20-minute games each. Each game took place on one of the three large levels pre-selected by the competition organizer (and known to the participants beforehand). These levels (*Goatswood*, *IceHenge* and *Colosseum*), were created by the Unreal Tournament player community after the release of the original game and are large relative to the character avatar and avatar motion speed, allowing a large number of participants to interact frequently but without too much interference from other players. The levels also contain complex 3D geometry and visual and physical obstacles such as trees, walls, columns, fences, bridges and water. Equal number of bot participants, human judges and native bots participate in each game. All the participants are anonymized by using a random in-game name and a random in-game skin. The rules of the game are modified from a regular death match as follows:

- One of the default weapons in the player arsenal is a "Judging gun" that allows players (both humans and bots) to record on-the-fly decisions about whether or not they think their opponent is a human.
- Damage by normal weapons is reduced to 40 % to allow players more time to interact.

Hitting a player with the primary fire of the judging gun for the first time designates the player as a human. Hitting a player with the secondary fire of the judging gun for the first time designates the player as a bot. If the designation is correct (the player

correctly guesses whether the opponent is a human) the player gets 10 frags and the opponent is instantly killed. If the designation is incorrect (the player fails to guess whether the opponent is a human) the player gets —10 frags and dies instantly. The judgement can only be made once per opponent—judging the same opponent twice has no effect.

The goal of the human players is to get the highest score—thus it is in their interest to guess whether or not their opponents are bots correctly. The goal of the bot players is to appear human, thus ideally their policy should be indistinguishable from what is expected from a human player in a given situation.

Additional details about the competition are available in previous publications by the organizer, Philip Hingston [4, 25], and in Chap. 5 of this book.

6.3.3 The GameBots2004 and Pogamut Interfaces

The `GameBots2004` interface allows a program to control a bot within Unreal Tournament using a network socket connection [23]. Synchronous and asynchronous messages are exchanged between the agent and the `GameBots2004` BotConnection running on the Unreal Tournament server, allowing the agent to receive updates about the game and the bot state and to send commands controlling its motion and its actions.

The `Pogamut` framework is a Java library that uses `GameBots2004` to provide the developer with a convenient API for accessing game and agent state and writing custom behaviours for Unreal Tournament 2004 bots [24]. In particular, the framework takes care of the details of updating agent's memory (state) information, wraps `GameBots2004` messaging protocol in a Java API, and provides a class hierarchy for representing useful data structures for building bot behaviour, including navigation graphs, inventory items, sensors, actions, and so on.

6.3.4 Navigation in Unreal Tournament 2004

The interfaces to the Unreal Tournament game engine support two main styles of navigation for bots. In one of these, the bot specifies the location (or locations) in its immediate vicinity where it wishes to move, and the game engine executes this motion, calculating the appropriate animations and adjusting the bot's location according to reachability and physical constrains. In the second mode of navigation, a *navigation graph* provided by the creators of the level maps can be used in order plan longer-term routes that take the bot from one location on the level to another.

In general, a navigation graph for a level consists of a relatively small (on the order of several hundred) number of named vertices, or *navpoints*, distributed throughout the level and connected by a network of *reachability edges* (Fig. 6.1). If two adjacent navpoints are connected by an edge, the engine should be able to successfully move the avatar between them.

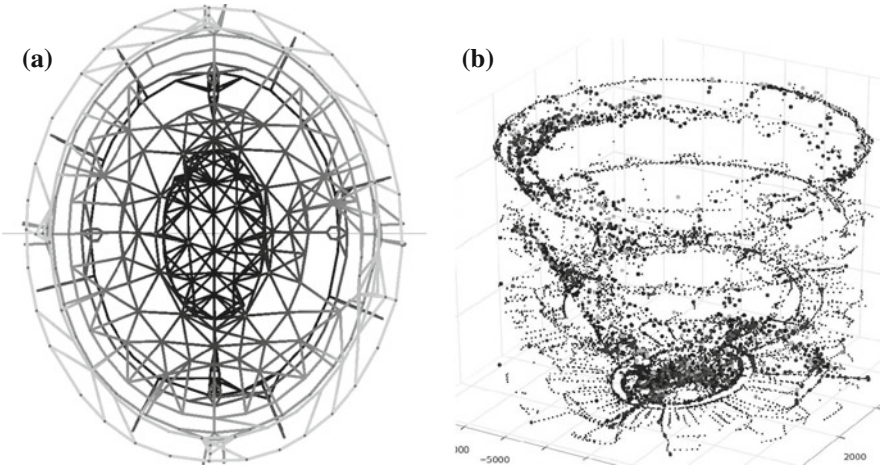


Fig. 6.1 A side-by-side comparison of the two-dimensional projections of the navigation graph (a) used by native Unreal bots and of three-dimensional view of the position and event samples of human behaviour (b) used by the Human Trace Controller and on one of the competition levels. **a** Navigation graph for Colosseum. **b** Human traces for Colosseum

Bots can use standard A* pathfinding such as the Floyd-Warshall method [26] to get from one location on the map to another. A* and Floyd-Warshall are pre-packaged in the Pogamut framework, and they work well, but they are different (and seemingly not quite as good as) what is built into UT2004 and used by the native bots. Part of the reason for this difference may be that path following sometimes involves well-timed jumps. The navigation and jumping capability is dramatically improved in the Pogamut 3.1 release, however, the BotPrize 2010 and the evaluations presented in this chapter were done using the competition version.

Even with the best of interfaces, error-free navigation is not usually available for domains with high complexity. This is certainly the case with mobile robot navigation where sensor and motor errors, physical slips and other inaccuracies can combine to both stochastic and systemic errors over time. Even in simulated environments such as Unreal Tournament, the well-timed execution of new destination commands along the path could be adversely affected by the additional network latency or computational overhead of the interfaces.

Because navigation primitives used by the bot are not error-free, it often gets "stuck", where the actions that it chooses to execute as part of the path following (or combat) behaviour do not cause any progress because an obstacle is in the way. This can be caused by navigation command execution errors, by collisions with other players, by imprecisions in the navigation graph, or by simulated physics interactions with weapon fire. Humans do not play in this manner, and judges often exploit this behaviour to make negative decisions . The UTˆ2 bot uses the *Human Trace Controller* (HTC) in order to quickly detect and recover from such navigation problems.

6.4 The Human Trace Controller

The UT^2 bot is implemented using a method similar to behaviour trees. The overall architecture of the bot is described in detail in Chap. 5. In the BotPrize 2010 competition, the bot uses the HTC in order to improve its navigation behaviour: when the bot gets stuck while moving along a path or during combat (Sect. 6.4.3), it executes actions selected by the controller.

The HTC uses a database of previously recorded human games to execute navigation behaviour similar to that of human players. This section describes how the data is recorded (Sect. 6.4.1) and indexed (Sect. 6.4.2) to select what actions to execute (Sect. 6.4.4).

6.4.1 Recording Human Games

Player volunteers were selected from graduate and undergraduate students at UT Austin. All players had considerable previous experience with video games in general and first person shooter games in particular. Familiarity with the Unreal Tournament 2004 game varied between the players.

An early version of the competition mod that allowed the judging gun to be used was instrumented with the ability to log each player's pose and event information into a text file, as follows. The standard BotPrize mod was decompiled into its original UnrealScript using the standard tools that come with the Unreal Tournament 2004. The script was then modified to write out a detailed log of the events and commands used by the players into a text file. The text file was then processed in order to extract traces of human behaviour, and these traces were stored in a SQLite database [27]. The frequency of the samples in the human database thus roughly coincided with the average logic cycle of the Pogamut/GameBots configuration, around ten times a second.

Two types of data points were recorded in the database for each human player: pose data and event data. The player's *pose* includes the current position, orientation, velocity and acceleration of the player. This data was recorded every logic cycle, or around ten times a second. The player's *events* were recorded as they happened together with their time stamp, and included actions taken by the player or various kinds of interactions with other players or the environment. Example event types include picking up inventory items, switching weapons, firing weapons, taking damage, jumping, falling of edges, and so on. Taken together, all the pose and event samples for a particular player in a particular game form a *sequence*, and are stored in such a way as to allow the controller to recover both preceding and succeeding event and pose samples from any given pose or event.

Games were conducted on the three levels selected for the competition (Fig. 6.1). Knowledge of the levels in this competition allowed the possibility of using a method that does not generalize to previously unseen levels.

Three types of games were recorded: standard games, judging games, and synthetic coverage games. Standard games were recorded in order to capture what human players do when playing a standard first person shooter death match variant of Unreal Tournament 2004. The judging games were recorded in order to overcome the potential differences in behaviour that manifest themselves when human players are also judging. Finally, because the competition levels were relatively large and because some problematic parts of the maps were visited infrequently by human players in the other data sets, synthetic data sets were collected where human players intentionally spent time navigating around areas with low data coverage.

In the first type of recorded games, two human volunteers were separated so that they could not see or hear each other outside of the game. They were joined by an equal number of native Unreal bot players. The rules were standard death match rules (first player to get 15 frags wins) and the human data was recorded and used as part of the database of human behaviour.

In the second type of recorded game, two human volunteers unfamiliar with the details of the bot were separated in two different rooms such that they could not see or hear each other outside the game. One of the authors (alternating) and two instances of an early version of the bot constituted the other participants of the games. The volunteers were asked to be vocal about what their thought process is and how their judgements were made. Recorded human traces of all participants in these games were stored in the database and used as part of the dataset for the Human Trace Controller.

In the third type of game recording, the authors participated in games designed specifically to fill in the areas of the dataset where the performance of the Human Trace Controller was found to be weaker due to lack of human data. These games involved alternating normal combat with movement localized to areas of the map where most of the stuck events were seen during testing, such as under the bridge in *Goatswood* or among the columns in the *Colosseum*.

The results presented in this chapter were obtained using a relatively modest number of human traces (Table 6.1), however, even for this dataset an efficient storage scheme had to be developed to support timely retrieval and playback. The next section discusses three data indexing schemes that were used over the course of development with properties that make it possible to scale this method to much larger data sets.

Table 6.1 Size of the recorded human game dataset used in the competition

Level	Unique players	Events	Pose samples
Colosseum	6	4,318	40,474
GoatswoodPlay	7	6,085	40,961
IceHenge	4	8,927	29,736
Total	17	19,330	1,11,171

The total dataset represents about ten hours of play time

6.4.2 Indexing Databases of Human Behaviour

In order to be able to quickly retrieve the relevant human traces, an efficient indexing scheme of the data is needed. In particular, it is necessary to quickly find all segments of the recorded games database that pass within the vicinity of the bot's current location. Throughout the course of development of the UT^2 bot, several such schemes were tested. Two of the most effective indexing schemes are described below.

6.4.2.1 Octree Based Indexing

An *octree* is a data structure routinely used in computer graphics and vision to index spatial information [28]. It is constructed by finding the geometric middle of a set of points, and subdividing the points into eight subsets defined by the three axis-aligned hyperplanes passing through the point. The process continues recursively until a termination condition, such as depth, smallest leaf dimension, or smallest number of points per leaf, is reached.

In order to index the pose data in the database of human game traces, an octree was constructed over the set of all points with the termination condition defined to be the first of (a) reaching the smallest leaf radius (set to twice the average distance moved per logic cycle), or (b) reaching the minimum number of points allowed in a leaf (set to 20).

Each point in the pose database was then labeled with an octree leaf. Given the bot's location, it is possible to traverse the octree index structure and find the smallest octree node that encloses it, and quickly retrieve the set of points within this octree node. Only part of the octree is stored in memory, while the rest is backed by SQL queries which dynamically retrieve the points needed. The Java/SQLite implementation allows the entire database to be stored in memory if it is small or to scale to other storage if it exceeds available memory.

6.4.2.2 Navigation Graph Based Indexing

As described in Sect. 6.3.4, levels in Unreal Tournament and many other similar games come with *navigation graphs*, which connect a number of named vertices distributed throughout the level with reachability edges.

It turns out that the nodes of the navigation graph can be used as labels for the points in the trace database, allowing the controller to quickly retrieve those points which are closest to a particular node. While this indexing process does take some time, it can be done efficiently by forming a KD-tree [29] over the navigation graph and iterating over the points in the database, performing a nearest neighbour search over the (relatively few) vertices in the navigation graph.

Once the nearest navpoints to the bot's location is found, the database yields the set of points in the pose database that are closest to the navpoint (belong to the navpoint's

Voronoi cell [30, 31]). Like the octree implementation above, the database-backed data structure can scale to a very large number of points because the tradeoff between memory and speed is adjustable.

Both of these schemes yield a subset of the pose database which is considered during *playback* (Sect. 6.4.4), however, the navigation graph scheme relies more on the availability of good level meta-data, while the octree-based indexing relies more on the quality of the human games database.

6.4.3 Detecting When the Bot is Stuck

Several different heuristics were employed in order to quickly determine when the bot is stuck. These were formulated after observing the behaviour of several earlier versions of the bot and characterizing the times when it was stuck.

- **SAME_NAV**—the bot keeps track of the number of logic cycles it finds itself next to the same navigation point as the previous cycle. The SAME_NAV condition is triggered when this number increases past a threshold.
- **STILL**—the bot keeps track of the number of logic cycles it finds itself within a short distance of the previous position. The STILL condition is triggered when this number increases past a threshold.
- **COLLIDING**—the Unreal Tournament game engine and the `GameBots2004` API report when the bot is colliding with level geometry, and this information is incorporated into the bot's senses by the Pogamut framework. The COLLIDING condition is set to true whenever the corresponding sense is true.
- **BUMPING**—the Unreal Tournament game engine and the `GameBots2004` API report when the bot is bumping into movable objects such as other players and this information is incorporated into the bot's senses by the Pogamut framework. The BUMPING condition is set to true whenever the corresponding sense is true.
- **OFF_GRID**—this condition is triggered when the bot finds its distance from the nearest navpoint reach a threshold.

If one of these conditions is set to true, the bot is considered stuck and the Human Trace Controller is executed as the controller for that logic cycle.

6.4.4 Retrieval and Playback of Human Behaviour Traces

When the bot finds itself stuck, if calls upon the Human Trace Controller in order to get unstuck. The controller keeps track of when it was last called, and either follows a previous path or creates a new one.

Several conditions have to be met in order for the bot to follow an existing path.

These are:

- The path was started recently enough.
- The bot has not strayed from the selected path. The "current" point along the path is within a set distance of the bot's current position.
- The path is not interrupted or terminated by recorded events such as falls, death, or large gaps between sampled positions.

If one or more of these conditions is not met, the Human Trace Controller attempts to start a new path. Otherwise, it continues along the previously started one.

The bot selects a set of relevant points from the pose database using one of the indexing techniques described in Sect. 6.4.2. Once the points are selected, one of them is picked as the starting point. Two methods for doing so that were tested during the development of the UTˆ2 bot—the random point selection and the nearest point selection. Based on the results of these tests, the competition version of the bot picks the nearest point unless this point is picked twice in a row, in which case a random point is selected.

Selecting a point from the subset specifies the sequence the agent will follow. The agent can then estimate the parameters of the movement action by using an estimate of the logic cycle length and the time stamps of the pose records.

In order to continue along the path, the agent uses the time passed since the starting point was selected in order to pick the next point from the database. The time delay between the pose samples along the stored paths in the database and the logic frames executed by the agent are both variable and different from each other. In order to address this problem, the agent interpolates between two database points in order to get an estimate for where on the path it should move to. The controller keeps an estimate (arithmetic mean) of the logic frame cycle length from its experience in order to do so.

Executing the planned motions outside of the navigation mesh poses its own challenges. If a point is not specified far enough in advance as the target of motion, the bot will appear to stall after every logic cycle. If a point too far ahead is specified, however, the bot will appear to change directions suddenly. The navigation interface provided by the GameBots API includes several different kinds of location-based movement primitives. Because the logic frame rate can be variable and latency can influence when an action actually reaches the engine and begins to execute, the `MoveAlong` primitive is used. This primitive takes two positions, $\mathbf{p_1}$ and $\mathbf{p_2}$, in order to make the movement appear smooth in the face of unpredictable latency. In effect, the `MoveAlong` action *schedules* the motion to $\mathbf{p_1}$ and then to $\mathbf{p_2}$, making the assumption that the next `MoveAlong` command will arrive when the agent is somewhere between p_1 and p_2. If the actions are interpolated carefully as part of a continuous path such as a trace of a human game, the resulting path appears smooth and purposeful.

The recorded and indexed human data, the firing conditions, and the retrieval and playback of the human trajectories together constitute the entirety of the Human Trace Controller component of the UTˆ2 bot. The next section discusses how this component was evaluated.

6.5 Results and Discussion

The performance of the human trace module was evaluated in several ways. First, it was used in the competition version of the UT^2 bot, which placed second out of five competing systems in the BotPrize 2010 competition (Sect. 6.5.1). The results of this evaluation were limited to high-level performance metrics based on human judgements and to qualitative insights extracted after the competitions based on video recordings of agent performance. The second type of evaluation was performed after the competition and was aimed at empirically comparing how the different variants of the unstuck controller contributed to the bot's ability to get unstuck and to the overall performance (Sect. 6.5.2).

6.5.1 BotPrize 2010 Competition Results

A summary of the overall bot humanness rating results is given in Table 6.2. In addition to these results, the detailed records of the judgements as well as game demo files from the games were made available after the competition.[1] This section summarizes the qualitative evaluation of the Human Trace Controller part of the bot based on these records.

The game records are provided in the Unreal demo format, which allows playback of the entire game from the perspective of a free camera (spectator mode) or from the perspective of any human player (follow mode). Several qualitative observations can be made based on reviewing these records. These observations are described below.

6.5.1.1 Data Sparseness

One problem that the bot frequently encountered on one of the levels, Colosseum, was getting stuck in the narrow hallways radiating outward from the centre of the level. Because the navigation graph does not extend into these areas, the bot will often bump into a wall if it tries to run to a node or an item after finding itself there.

Table 6.2 BotPrize 2010 results, including the average humanness rating of the native bots. The humanness rating is the percentage of judgements of the bot (by humans) that identified it as a human player

Bot	Humanness
Native UT2004 Bot	35.3982
Conscious-Robots	31.8182
UT^2	27.2727
ICE-2010	23.3333
Discordia	17.7778
w00t	9.3023

[1] http://www.botprize.org/2010.html

The Human Trace Controller can solve this problem, but only when a record exists for the particular area where the bot is stuck. Observations of the competition records from the UT^2 bot confirm this, because the first version of the bot's database used on the Colosseum level resulted in the bot being often stuck in the columns area, and this situation improved dramatically with the addition of traces specifically in the problem areas.

However, as noted below, one future direction for the controller is to use machine learning techniques in conjunction with egocentric sensors to generalize between environments that look similar. The hallways are a great example of where such an approach would be particularly useful.

6.5.1.2 Correspondence Problem

One fundamental problem faced by all designers of human-like behaviour is the *correspondence problem*, or the difference between the actions and observations available to humans and those available to artificial agents [11]. This problem can include differences in decision frequency, the kind and amount of information expressed in the sensors, differences in body morphology or capability and so on.

This problem sometimes leads to the bot's inability to reproduce a human reaction either because its observations are insufficient or because its actions are limiting.

For example, in the BotPrize competition, the human players control the bot via keystrokes and mouse movements that are processed by the game engine at a very high frequency. They receive information about the environment from a two-dimensional rendering of the three-dimensional world, which includes rich information such as texture, colour, shadow, effects of explosions, sounds, and so on. Further, they can rely on the powerful ability of the human mind to interpret and synthesize these events into higher-level stimuli, using highly parallel and complex "wet-ware", the workings of which we are only beginning to understand.

Bots, in contrast, send commands controlling their player about ten times a second, and the command repertoire does not include direct equivalents to keystroke and mouse based control of a human player. The observations of a bot include a lot of information about the environment that the human is not given immediately (such as reachability grids, navigation graphs, exact locations of items and event notifications) but also exclude important features, such as an effective way to react to sound, the ability to see the nature and extent of different environments such as lava or water, and an effective ability to deal with level geometry.

As a concrete example of this problem, in the *Goatswood* level, the bot would sometimes get stuck next to short obstacles in its path, and trying to follow a human trace would not lead to a successful navigation because in order to do so the bot would have needed to send a (well timed) jump command in addition to a `MoveAlong` action. Humans, in contrast, were able to "push through" such obstacles because such small jumps are built in to the pawn behaviour.

6.5.1.3 Environmental Features

Another class of problems has to do with special features of the environment. For example, both the *Goatswood* and the IceHenge level contain areas with water flowing through them. These special areas change the way the bot responds to commands, making movement slower in some directions and faster in others. This in turn causes human path execution to fail, because the bot tries to execute the same actions in two very different environments, as it does not operate in the same action space as the human did.

Because the water hazards are particularly difficult for the human trace controller to navigate, during the competition the UT^2 bot used a specially designed *Water Controller* when it was stuck in water areas. The controller uses a *goto* primitive along with additional hand-crafted navigation nodes in order to get out of the water as quickly and smoothly as possible. Additionally, proximity to important items located on the side of the water hazard in IceHenge can cause the bot to go to that location instead.

While this partial solution improved the performance of the bot on levels with water hazards, it does not result in a particularly human-like behaviour. For example, people occasionally used water as cover to sneak up on opponents or to escape pursuit.

Taken together, these observations provide important insights into the kinds of problems that need to be addressed when designing human-like behaviour. In Sect. 6.6, we discuss some future work that may help address these issues.

6.5.2 Comparative Evaluation

In order to evaluate the contribution of the Human Trace Controller to the overall UT^2 bot, comparisons were made between three versions of the bot, where the only difference was the controller used to get the bot unstuck. The following three controllers were used in the comparison:

- The *Null Controller* simply ignores the stuck condition and continues to whatever action would fire in the bot otherwise.
- The *Scripted Unstuck Controller* is a scripted controller designed to get the bot unstuck during evolution (Sect. 6.5.2.1).
- The *Human Trace Controller* as used during the BotPrize 2010 competition.

In all three versions the overall bot is modified from the competition version by removing level-specific special cases and the `WaterController` in order to make sure that only one controller is used to get unstuck during the comparison.

6.5.2.1 The Scripted Unstuck Controller Baseline

The Scripted Unstuck Controller is a scripted controller designed to get the bot unstuck reliably and quickly. Because it is relatively simple and does not rely on changing external human databases, the Scripted Unstuck Controller was used whenever the bot got stuck while evolving the battle controller for the UTˆ2 bot (see Chap. 5). It also provides a strong benchmark for comparisons with the performance of the Human Trace Controller.

In our experiments, the controller picks one of the following actions depending on the current state of the bot:

- If the bot is currently at location $\mathbf{x_1}$ and detected a collision at $\mathbf{x_2}$, the controller requests a move of $5 \cdot (\mathbf{x_2} - \mathbf{x_1})$.
- If the bot is currently at location $\mathbf{x_1}$ and detects a bump at location $\mathbf{x_2}$, the controller requests a move of $5 \cdot (\mathbf{x_2} - \mathbf{x_1})$.
- Otherwise,
  - with probability 0.5, the controller performs a `DodgeShotAction`, which results in a single jump in a random direction.
  - with probability 0.25, the controller requests the bot to run forward continuously until another command is selected.
  - with probability 0.25, the controller requests the bot to go to the nearest item.

6.5.2.2 Comparison Results

The results of these comparisons for the three levels are given in Figs. 6.2, 6.3 and 6.4. The three bars represent UTˆ2 bot using no unstuck controller (*NONE*), the Human Retrace Controller (*HTC*), and the scripted controller (*SCRIPTED*). Each bar is an average of thirty runs, where each run represents ten minutes of game time in a game with two Hunter bots and the modified UTˆ2 bot. Standard error is indicated by the error bar.

Overall, both the Scripted Unstuck Controller and the human trace controller perform similarly in terms of the number of cycles stuck. However, the Human Trace Controller still has two advantages: it generates qualitatively smoother paths when executed in isolation with random restarts, and the average length of a stuck segment for the Human Trace Controller is shorter than that of the scripted controller. Because the evaluations are very noisy, further evaluations are needed in order to determine statistical significance of these findings. However, the need for an unstuck controller is significant in that both controllers outperform the Null Controller.

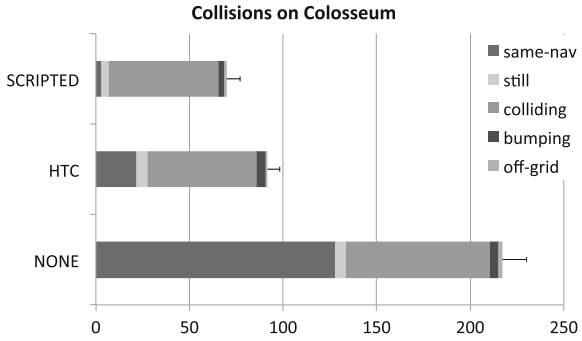


Fig. 6.2 Number of logic cycles stuck by condition on the *Colosseum* level. Averages of thirty ten-minute runs and standard error are shown

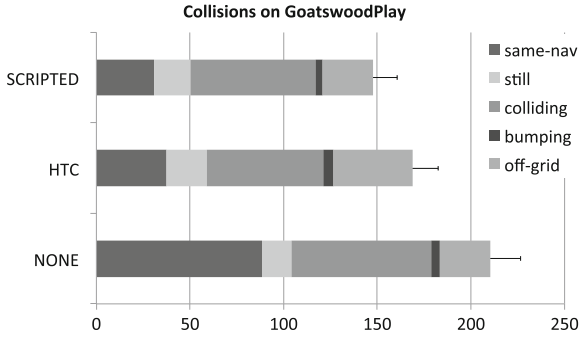


Fig. 6.3 Number of logic cycles stuck by condition on the *Goatswood* level. Averages of thirty ten-minute runs and standard error are shown

6.5.2.3 Post-competition Improvements

After the 2010 BotPrize competition, the UTˆ2 bot was modified to improve several aspects of using human traces. First, the scripted unstuck controller was integrated with the Human Trace Controller to allow the scripted controller to take over when traces are unavailable or failing to replay correctly. Second, the HTC was also used by an additional top-level UTˆ2 controller that allowed the bot to explore the level in a human-like fashion in the absence of any other goals. Third, the database was filtered to include only smooth segments, not interrupted by jumps or other artefacts. Finally, HTC playback was modified to ensure that positive path progress was made by keeping an estimate of bot speed and selecting points along the path by distance traveled according to this estimate.

The results of these comparisons are shown in Fig. 6.5. The amount of time the bot spends stuck decreases both when using Human Traces to get unstuck and when using them to explore the environment. Additionally, the human trace replay used

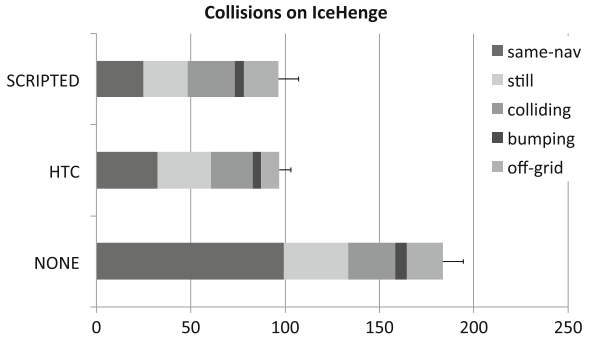


Fig. 6.4 Number of logic cycles stuck by condition on the *IceHenge* level. Averages of thirty ten-minute runs and standard error are shown

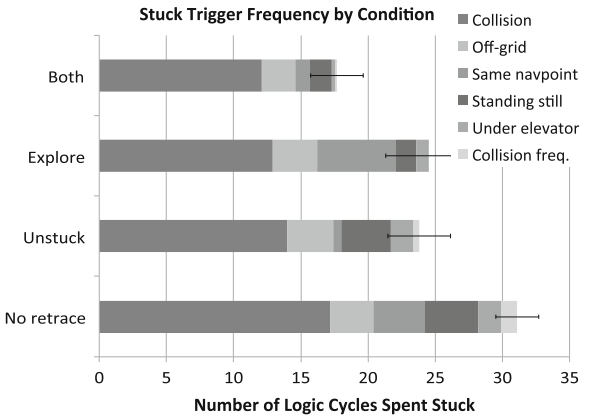


Fig. 6.5 A comparison of several versions of the 2011 version of the UTˆ2 bot. Average unstuck counts are shown when the bot is using no human traces (*No traces*), only to get unstuck (*Unstuck*), only to explore the level (*Explore*), or both to get unstuck and to explore (*Both*). Average over 10 runs with standard error bars are shown for the Osiris2 level

for exploration allows the bot to avoid traveling along the navigation graph and looks smooth and human-like when observing.

6.6 Discussion and Future Work

The Human Trace Controller presented in this chapter is a simple way to utilize recorded human behaviour to improve parts of the agent's navigation policy. However, the technique is much more generally applicable and can be extended to further

improve the navigation system, to generalize to previously unseen environments, and to support higher-level decision making such as opponent modeling.

One further area where human trace data can be useful is to improve other components of the navigation subsystem. Because the levels for Unreal Tournament 2004 are designed by hand, some areas of the levels are missing navpoints or edges in places where they would be quite useful to bots. In such places, it is possible to use the human data to induce a more complete version of the navpoint graph. Such a graph could then be used without modification by graph navigation and path planning modules.

The Human Trace Controller as described in this chapter suffers from one major weakness—it does not generalize to new environments (or even to unseen parts of existing environments). Enabling the bot to generalize to previously unseen environments would require (1) recasting of the problem in an egocentric state space and (2) selecting an appropriate machine learning technique to support better generalization. Some work has already been done towards achieving (1)-the combat module of the UT^2 bot uses a set of ego-centric and opponent-centric sensors and actions which could be reused when building a human player model. In order to achieve (2), the indexing techniques described in Sect. 6.4.2 can be naturally replaced with an instance-based machine learning algorithm such as a decision tree or a nearest-neighbour algorithm. This instance-based method can be compared and contrasted with other machine learning methods capable of compressing the data such as neural networks or probabilistic techniques. Whatever solution is used, one property that would be useful to retain from the current implementation is the ability to select what parts of what kinds of environments should be added to the database for maximum gain in the model's accuracy.

In order to address the correspondence problem, or the difference between the human and the bot's observation and action spaces, and to support the use of machine learning for effective generalization based on previously seen human behaviour, a translation scheme between human and bot observations and actions needs to be developed. This could be done automatically by learning the bot actions and parameters that most accurately recreate small sections of human behaviour, and using those actions as primitives when replaying new traces or outputs of a learned human behaviour model.

In addition to using a model of human behaviour to mimic it as part of the agent's policy, it is possible to use this information in other ways. For example, it may be possible to use the human behaviour database to design a "humanness fitness function" for evolving human-like control policies. Such a function can be used for example as part of the multi-objective neuroevolution technique discussed in Chap. 5 to compare the behaviour generated by a candidate bot with that available in the human database. As another promising future use of human trace data, the bot can use its model of human behaviour to predict and reacquire a human opponent it is chasing if it loses track of him or her. Such behaviour can be seen as purposeful and cognitively complex, and thus very human.

6.7 Conclusion

The Human Trace Controller component of the UTˆ2 bot takes a step towards building human-like behaviour in a complex virtual environment by directly replaying segments of recorded human behaviour.

Evaluation of the resulting controller as part of the BotPrize competition and via comparative experiments suggests that the replay of human traces is an effective way to address the navigation problems faced by the UTˆ2 bot. Additionally, human traces can be used for exploration of the level in the absence of other goals. The resulting behaviour appears smooth and human-like on observation, while also allowing the bot to navigate the environment with a minimal number of failures.

Finally, the work demonstrates the feasibility of using large databases of human behaviour to support online decision making. The approach can scale and improve with experience gained naturally from domain experts; it is applicable to the explicit goal of building human-like behaviour; and it supports imitation, a primitive that is ubiquitous in examples of intelligent behaviour in nature.

Acknowledgments The authors would like to thank Philip Hingston for organizing the BotPrize competitions and 2K Australia for sponsoring it. The authors would also like to thank students in the Freshman Research Initiative's Computational Intelligence in Game Design stream and members of the Neural Networks Research Group at the University of Texas and to Christopher Tanguay and Peter Djeu for participating in recordings of human game traces and for volunteering to critique and evaluate versions of UTˆ2. This research was supported in part by the NSF under grants DBI-0939454 and IIS-0915038 and by the Texas Higher Education Coordinating Board grant 003658-0036-2007.

References

1. Rizzolatti, G., Fogassi, L., Gallese, V.: Neurophysiological mechanisms underlying the understanding and imitation of action. Nat. Rev. Neurosci. **2**(9), 661–670 (2001)
2. Gergely, G., Bekkering, H., Király, I.: Developmental psychology: rational imitation in preverbal infants. Nature **415**(6873), 755 (2002)
3. Nicolescu, M., Matarić, M.J.: Task learning through imitation and human-robot interaction. In: Dautenhahn, K., Nehaniv, C. (eds.) Models and Mechanisms of Imitation and Social Learning in Robots, Humans and Animals: Behavioural, Social and Communicative Dimensions, Cambridge University Press, Cambridge (2005)
4. Hingston, P.: A Turing Test for computer game bots. IEEE Trans. Comput. Intell. AI Games **1**(3), 169–186 (2009)
5. Laird, J.E., van Lent, M.: Interactive computer games: human-level AI's killer application. AI Mag. **22**(2), 15–25 (2001)
6. Aha, D.W., Molineaux, M.: Integrating learning in interactive gaming simulators. In: Proceedings of the AAAI'04 Workshop on Challenges of Game AI, AAAI Press (2004)
7. Bowling, M., Fürnkranz, J., Graepel, T., Musick, R.: Machine learning and games. Mach. Learn. **63**, 211–215 (2006)
8. Molineaux, M., Aha, D.W.: TIELT: a testbed for gaming environments. In: Proceedings of the Twentieth National Conference on Artificial Intelligence (Intelligent Systems Demonstrations), AAAI Press (2005)

9. Schaal, S.: Learning from demonstration. In: Advances in Neural Information and Processing Systems, pp. 1040–1046. Citeseer (1997)
10. Atkeson, C., Schaal, S.: Robot Learning from Demonstration. In: Proceedings of the Fourteenth International Conference on Machine Learning, ICML'97, pp. 12–20. Citeseer (1997)
11. Argall, B.D., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Robot. Auton. Syst. **57**(5), 469–483 (2009)
12. Thurau, C., Bauckhage, C., Sagerer, G.: Synthesizing movements for computer game characters. Lect. Notes Comput. Sci. **3175**, 179–186 (2004)
13. Stolle, M., Atkeson, C.G.: Policies Based on Trajectory Libraries. In: Proceedings of the International Conference on Robotics and Automation (ICRA 2006) (2006)
14. Stolle, M., Tappeiner, H., Chestnutt, J., Atkeson, C.G.: Transfer of policies based on trajectory libraries. In: Proceedings of the International Conference on Intelligent Robots and Systems (2007)
15. Hladky, S., Bulitko, V.: An evaluation of models for predicting opponent positions in first-person shooter video games. In: Proceedings of the IEEE 2008 Symposium on Computational Intelligence and Games (CIG'08). Perth, Australia (2008)
16. Orkin, J.D., Roy, D.: Automatic learning and generation of social behavior from collective human gameplay. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'09), vol. 1, pp. 385–392 (2009)
17. Cardamone, L., Loiacono, D., Lanzi, P.L.: Learning drivers for TORCS through imitation using supervised methods. In: Proceedings of the IEEE 2009 Symposium on Computational Intelligence and Games (CIG'09) (2009)
18. Aler, R., Valls, J.M., Camacho, D., Lopez, A.: Programming robosoccer agents by modeling human behavior. Expert Syst. Appl. **36**(2), Part 1, 1850–1859 (2009)
19. Pao, H.-K., Chen, K.-T., Chang, H.-C.: Game bot detection via avatar trajectory analysis. IEEE Trans. Comput. Intell. AI Games **2**(3), 162–175 (2010)
20. Sukthankar, G., Sycara, K.: Simultaneous team assignment and behavior recognition from spatio-temporal agent traces. In: Proceedings of Twenty-First National Conference on Artificial Intelligence (AAAI-06) (2006)
21. Murakami, S., Sato, T., Kojima, A., Hirono, D., Kusumoto, N., Thawonmas, R.: Outline of ICE-CEC 2011 and its mechanism for learning FPS tactics. In: Extended Abstract for the Human-like Bot Workshop at the IEEE Congress on Evolutionary Computation (CEC 2011) (2011)
22. Epic Games, Inc. and Digital Extremes, Inc.: Unreal Tournament 2004. Atari, Inc., Mar 2004
23. Kaminka, G.A., Veloso, M.M., Schaffer, S., Sollitto, C., Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S.: GameBots: a flexible test bed for multiagent team research. Commun. ACM **45**(1), 43–45 (2002)
24. Gemrot, J., Kadlec, R., Bída, M., Burkert, O., Píbil, R., Havlíček, J., Zemčák, L., Šimlovič, J., Vansa, R., Štolba, M. et al.: Pogamut 3 can assist developers in building AI (not only) for their videogame agents. In: Agents for Games and Simulations, pp. 1–15 (2009)
25. Hingston, P.: A new design for a Turing Test for bots. In: IEEE Transactions on Computational Intelligence and AI in Games (2010)
26. Floyd, R.: Algorithm 97: shortest path. Commun. ACM **5**(6), 345 (1962)
27. "SQLite". http://www.sqlite.org/
28. Jackins, C., Tanimoto, S.: Oct-trees and their use in representing three-dimensional objects. Comput. Graph. Image Process. **14**(3), 249–270 (1980)
29. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18**(9), 509–517 (1975)
30. Dirichlet, G.L.: Über die Reduktion der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen. J. für die Reine und Angewandte Mathematik **40**, 209–227 (1850)
31. Voronoi, G.: Nouvelles applications des paramètres continus à la théorie des formes quadratiques. J. für die Reine und Angewandte Mathematik **133**, 97–178 (1907)

Chapter 7
A Machine Consciousness Approach to the Design of Human-Like Bots

Raúl Arrabales, Jorge Muñoz, Agapito Ledezma, German Gutierrez and Araceli Sanchis

Abstract This chapter introduces Machine Consciousness as a new research field applied to the development of human-like behaviour of non-player characters (NPCs) in video games. Key aspects, advantages, and challenges of this young research area are discussed using the cognitive architecture CERA-CRANIUM as an illustrative example of an autonomous control system inspired by cognitive theories of human consciousness. Additionally, other cognitive architectures used in video games are also analyzed. The bot codenamed CC-Bot2, winner of the 2K BotPrize 2010 competition and based on the CERA-CRANIUM cognitive architecture, is also described in this chapter. Specifically, the particular way in which CC-Bot2 processes the sensory-motor information and generates sequences of adaptive human-like actions is discussed. We also analyze the main differences between CC-Bot2 and other bots, focusing on the key features that allowed CC-Bot2 to win first place in the competition. Finally, we conclude by describing the main lines of work for future CC-Bot implementations and pointing out major conclusions about the application of Machine Consciousness to the design of believable bots.

R. Arrabales (✉) · J. Muñoz · G. Gutierrez · A. Ledezma · A. Sanchis
Universidad Carlos III de Madrid, Avda. de la Universidad 30, 28911 Leganés, Madrid, Spain
e-mail: rarrabal@inf.uc3m.es

J. Muñoz
e-mail: jmfuente@inf.uc3m.es

G. Gutierrez
e-mail: ggutierr@inf.uc3m.es

A. Ledezma
e-mail: ledezma@inf.uc3m.es

A. Sanchis
e-mail: masm@inf.uc3m.es

P. Hingston (ed.), *Believable Bots*, DOI: 10.1007/978-3-642-32323-2_7,

7.1 Machine Consciousness and Believability in Game Bots

Machine Consciousness is a young and noticeably multidisciplinary field of research that aims at understanding, simulating and even replicating human consciousness in machines [12]. From the broad range of approaches currently being explored in this area, here we will focus on the specific research lines centred on the use of artificial cognitive architectures, i.e. control systems modeled after theories of human cognition.

The development of believable video game bots, truly indistinguishable from human players, remains an open problem both for science and for the game industry. Taking a Cognitive Machine Consciousness approach to address this problem implies the use of existing psychological models of human cognition as the main inspiration for the design of more engaging characters. While typical Artificial Intelligence (AI) techniques are usually inspired by partial aspects of cognition, such as planning or learning, the Machine Consciousness approach that we propose aims at effectively integrating key cognitive functions into a single autonomous control architecture.

Classical game character implementations, like *Pac-Man* ghosts or *Space Invaders* alien spacecraft, used simple scripted behaviour. However, as high-end video games have evolved into more complex virtual environments, they require the generation of much more realistic behaviours in their NPCs. Pre-programmed behaviours are valid to some extent in certain scenarios, but in realistic games consumers expect to find artificial opponents or collaborators behaving like humans. However, players usually consider disappointing the behaviour of artificial characters [25].

The generation of behaviour in humans involves several interrelated cognitive mechanisms, such as attention, emotions, agent modeling, set shifting, learning, etc. Artificial cognitive architectures like CERA-CRANIUM aim at integrating all these aspects together into effective and adaptive control systems. However, the design of such architectures is not straightforward. We argue that the Cognitive Machine Consciousness approach might contribute to tackle complexity and provide a useful framework for the design of more appealing video game bots. This hypothesis is being put to test with the CC-Bot implementation series based on the CERA-CRANIUM architecture. The current implementation of CERA-CRANIUM is designed to incorporate all the cognitive functions mentioned above; however, at this stage only the attention mechanism has been completely implemented. Although the architecture is in an early stage of development, latest experiments have given good results at generating human-like behaviours. Recently, CC-Bot2 implementation won the first place in the third edition of the 2K BotPrize competition.

The 2K BotPrize competition is an adaptation of the Turing Test for first person shooter video games [16]. Specifically, the game used in this contest is *UT2004* [10] (*Unreal Tournament 2004*), a first person shooter set in a fictional future with futuristic weapons. The objective of the game is to kill as many opponents as possible without being killed by the other players. Bots connect to the game server by means

of an application library named *Pogamut 3* [13], which uses the interface *GameBots 2004*[1] for UT2004.

As illustrated by BotPrize competition results current artificial characters can be intelligent in some sense, but they cannot match the behaviour produced by a human player. To date, playing with other humans is generally more realistic and engaging than playing with the most advanced bots.

In this work we put forward the hypothesis that using cognitive architectures inspired by models of consciousness might be an effective approach to deal with the complexity of this problem. Additionally, we argue that the field of Machine Consciousness research can also benefit from the experimentation with such architectures in the domain of video games.

In the remainder of this chapter we focus on the use of cognitive architectures for the design of computer game bots. Firstly, we analyze the cognition in virtual agents, specifically game bots, and how the cognitive architectures can be used to control the virtual agents. Then, we briefly review related work, discussing the main features of relevant examples of cognitive architectures used in different games. The next section describes the design of our bot CC-Bot2, including a thorough explanation of the main mechanisms implemented so far in the CERA-CRANIUM cognitive architecture. Finally, we analyze the operation of CC-Bot2 during the third edition of the 2K BotPrize competition and draw some conclusions from our experience in this domain.

7.2 Cognition in Video Games

Cognition is usually associated with some key aspects observed in biological organisms, like embodiment and situatedness. Similarly, physical embodiment and real world situatedness have also been nominated as key factors in the production of consciousness. Consequently, computer game bots expected to replicate or simulate human cognition should also cover these features. This possibility has been questioned because bots do not have real physical bodies, nor do they interact with a physical world [27]. In contrast, we argue that software agents, like NPCs, can also be embodied and situated. This position has also been advocated by other authors in relation with software agents designed after cognitive models of consciousness [11].

Embodiment means to have a body equipped with sensors and actuators that enable structural coupling with the surrounding of the agent. Therefore, a video game agent is embodied in that sense (or "virtually embodied" [14]), as it retrieves exteroceptive and proprioceptive sensing data through its software sensors, and can also perform actions using its software actuators (Fig. 7.1).

Situatedness refers to the causal interaction with the world. In the case of a computer game bot, interaction with the simulated world affects the game environment, which in turns affects the agent. Furthermore, as human players can interact with

[1] http://gamebots.sourceforge.net/

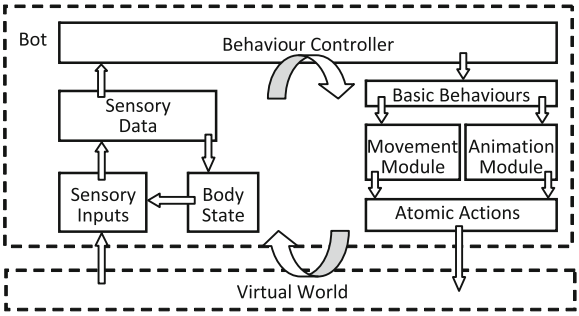


Fig. 7.1 Video game bot generic control architecture. The higher level of control (Behaviour Controller) can enclose a cognitive control architecture that takes the processed sensory data as input and continuously generates a selection of basic behaviours

the bot, even the real world is causally affected by bot decisions (analogously, bot sensory input is causally affected by the actions performed by the human player in the real world).

In short, designing intelligent control systems for video game bots is essentially equivalent to the task of designing intelligent control systems for physical autonomous robots. In this chapter, we focus specifically on multiplayer online games like FPSs. Common features of these games are a virtual world where the action takes place, local and remote access through computer networks, and the possibility to play either with or against other humans and/or bots.

Although the design of the cognitive architectures for a physical robot and for a computer bot are in essence the same process, there exist important differences:

- Robots, unlike bots, possess a physical body.
- Robots, unlike bots, interact directly with the physical world.
- Robots, unlike bots, usually have to deal with much more noise and uncertainty.

We see these features as benefits rather than problems for the application of the Cognitive Machine Consciousness approach in the domain of video games. In fact, the typical characteristics of computer games and other virtual environments are ideal for focusing on high level control, avoiding the usual problems of physical machinery. This research approach has also been adopted from the point of view of Artificial General Intelligence (AGI) in simulated environments like Second Life [14, 15].

We think that one of the main reasons why current AI-based bots are not able to perform as humans is because their control systems do not have enough "cognitive power". In other words, they are not successful at integrating the whole set of cognitive skills that are usually found in humans (see Chap. 8 for a comprehensive list of cognitive skills).

In general, what is missing is an effective combination of cognitive capabilities like attention, learning, or Theory of Mind (the ability to attribute mental states to oneself and others [30]). While limited computational models of these skills could be

individually implemented using existing AI techniques, the integration of all these functional components calls for the design of a flexible cognitive architecture.

Before going into more details about cognitive architectures it is important to analyze what is the typical software architecture of video game bots and where exactly the cognitive skills components should be located. Most implementations of AI bots are based on techniques like Finite State Machines (FSMs), Artificial Neural Networks, Bayesian Programming, Behaviour Trees, etc. Additionally, different software architectures are used as a means to integrate several major components: character animation, low-level movement, teamwork coordination, combat skills, and action selection [29].

In this work we focus on the high level control of bots, setting aside aspects related with low-level control issues. Examples of the latter include character's body animation, inverse kinematics, and the actual execution of basic actions like jumping or firing. Usually, a high level layer, where the goals of the bot are defined, is implemented on top of other control subsystems (Fig. 7.1 depicts a general bot architecture). The generic higher control layer represents the top level of control. The final behaviour of the bot is determined by the Behaviour Controller. While the Movement Module determines how the bot can move effectively from one point to another, the Behaviour Controller is in charge of deciding where to move. As a general rule, higher levels send abstract commands to lower levels, which are responsible for the concrete sequence of actions.

As bots are designed to achieve mission specific goals, a repertory of preprogrammed behaviours can be defined (see Basic Behaviours module in Fig. 7.1). In this case, it is the responsibility of higher control levels (the Behaviour Controller in this case) to decide what basic behaviour to activate at any given time. A number of approaches have been proposed about the design of Behaviour Controllers. When approaches inspired by psychological models are adopted, this higher control layer contains the cognitive functions and is usually referred to as the cognitive control layer.

In the context of this work, when we refer to the cognitive architecture that controls the bot, we mean a system embedded in the general control architecture acting as Behaviour Controller. Therefore, two levels of architectures can be distinguished: the general bot control architecture (as sketched in Fig. 7.1), and the cognitive control architecture (the main topic of the remainder of this chapter), which is enclosed within the higher layer of the former.

State of the art multiplayer video games provide a rich environment where both humans and artificial bots can interact. In addition, unlike real world tasks using physical agents, video games provide an environment where physical sensory-motor noise and uncertainty do not represent an extra challenge. Consequently, these games have become a suitable and convenient platform for the research in artificial cognitive systems. Although the issue of reproducing human-like behaviour in synthetic characters has been approached by many authors (see Sect. 7.3), most of the research efforts neglect cognitive models of consciousness as a possible source of inspiration for new developments.

We believe the reason why consciousness has been largely ignored is the poor understanding of this phenomenon within the field of classical AI (in contrast to Strong AI or Artificial General Intelligence). The growing interest in the scientific study of consciousness during the last two decades has motivated new research lines in artificial cognitive systems. Specifically, the emergence of the Machine Consciousness research community and associated computational models and implementations (see [8, 17] for instance) calls for the application of these new models to the domain of computer games.

7.3 Related Work

A number of significant cognitive architectures have been developed over the last decades [28], but they have not been extensively used in video games. Most of them were created to be used in autonomous agents, like physical robots, and also in software agents, but the application in the domain of game bots has not been mainstream. It is only during the last years that the idea of testing cognitive architectures in video games has become more popular. Games have become very complex, reaching great accuracy in the simulation of real worlds. Additionally, as discussed in the former section, they provide the researchers with some advantages in respect to physical robots.

There exist several cognitive architectures based on different theories of the mind (a brief summary of some of them can be found in [22] and [28]). However, few of them have been used in video games. Some remarkable examples are: *Soar* [19], *ACT-R* [1] and *Icarus* [21]. Interestingly, none of them is a cognitive architecture based on a model of consciousness such as the one presented in this chapter (CERA-CRANIUM).

7.3.1 Soar

The Soar architecture was first developed in 1983, and the first publication was in 1987 [18]. It is a goal oriented architecture where a set of rules represents the long term knowledge. These rules, called operators, determine the decisions the architecture takes. They are selected and applied based on the current state of the environment and goals of the agent. They can be simple operators which modify the internal state of the agent or execute actions that affect the environment, or they can be more complex operators that describe abstract activities. These complex operators are composed of other operators, indeed when a complex operator is selected and executed it is decomposed in its simpler operators, and these are also decomposed recursively until only simple operators remain, which the agent is able to execute.

The basic cycle of execution in Soar is composed of three stages. First, the architecture proposes operators based on the contents of the working memory, that is, all

the operators with their pre-conditions satisfied are selected. Second, the selected operators are evaluated and the best one is selected. Finally, this selected operator is executed, its rules are fired changing the working memory and the commands to the motor system are sent. Sometimes the knowledge to select an operator in the second step is insufficient. When this happens the architecture creates a new goal in the working memory that will be taken into account in the next cycle to select the next operator. This process can lead to a cycle where a goal is decomposed into sub-goals. The goals are removed from the working memory when the sensors of the agent report that the goal is resolved or when the goal that generates the subgoal is resolved. From the functional point of view, this working memory mechanism could be seen as equivalent to that of CERA-CRANIUM architecture.

While current implementation of CERA-CRANIUM lacks any complex learning mechanisms, Soar includes two learning mechanisms: reinforcement learning and *chunking*. Reinforcement learning is used to adjust numerical values of the operators that affect how they are evaluated before selecting and executing one. Chunking is the way the architecture learns new production rules. These new rules contain the process through the resolution of a subgoal, leading to one or more unexpected results.

One application of the Soar architectures in games was the Quakebot [20]. It was designed to be a human-like expert player for the game *Quake II* in the deathmatch game mode. The bot does not have complete access to the game state, but it has the same sensory information as a human player. Quake is a FPS video game older than UT2004 but with a very similar gameplay.

Another application of Soar in a game was a bot for SORTS [31], an open source RTS (Real Time Strategy) game. The RTS games differs from the FPS shooter games in that they generate more perceptual information and they need more abstract representations of the environment in order to make human-like decisions. Because of this, SORTS includes some FSMs to control some basic behaviours and a perceptual system in charge of filtering all the perceptual information. The perceptual system includes two human abilities: grouping and attention. The former is able to see and treat similar close objects as a unitary whole. The latter is used to filter all the perceptual information that is not related to a specific place of the map. In CERA-CRANIUM we have implemented similar functionalities using contexts as flexible tools for grouping, and an attention mechanism based on the activation of contexts.

Soar has not only been used to control bots in games, but has also been used as the engine of a story-based game [24] called *Haunt II*. In the game Soar was in charge of creating the plot for the interactive story, creating the artificial characters needed and communicating the events of the story to those characters. The human player moves through the game while Soar constructs a story based on a pre-written script. In that case, instead of creating human-like bots with Soar, the architecture is used to create the challenges and conflicts that match with the plot and keeps a coherent story experience.

7.3.2 ACT-R

ACT-R is a revision of the ACT architecture, which was first developed in 1973 [26]. The architecture was created with the objective of modeling human behaviour. In contrast to the layered design of CERA-CRANIUM, ACT-R is divided into modules, each one in charge of processing a different type of information as visual stimuli, motor skills, reasoning, etc. Each module has a buffer that contains the short term memory in form of declarative structures, named "chunks". The modules can operate in parallel and they use the buffers to pass information from one module to another.

Like in Soar, ACT-R is also a goal-oriented architecture and it has a production system. On each cycle the architecture determines which productions match the chunks in the short-term memory, and determines the expected profit of the production rule based on the activation of its chunks, executing the rules with higher profit. Unlike Soar, the latest versions of ACT-R are less goal-focused. They allow the production system to match to any source of information instead of only the current goals in the short-term memory. The sources of information can be the data in the declarative memory, the objects in the focus of attention of the perceptual modules and the internal state of the action modules.

Learning occurs in ACT-R similarly as in Soar. A mechanism of reinforcement learning is used to modify the base activation of the production rules, increased when they are used and decreased otherwise. Another mechanism in ACT-R is used to learn new production rules by analyzing the dependencies of fired rules and combining them into a new production rule.

ACT-R has several applications in autonomous agents like robots and some board games to model human behaviour, but we are only aware of one application in a real-time videogame [7]. The architecture has been used to model synthetic opponents in a customization of the *Unreal Tournament* game. The objective of this work was not to create an amusing game but to create realistic bots to model soldiers for military operations on urban terrain. The game is in fact a simulation with two teams of two soldiers each where the attackers have to clear a building while the defenders have to avoid this action. Both teams have different behaviours, the attackers only employ military strategies while the defenders use a more diversified set of behaviours.

7.3.3 Icarus

Icarus is a more recent architecture, it was first developed in 1991 [23]. It is composed by two types of hierarchical knowledge sets: concepts and skills. The former are related with the state of the environment and the agent. The sensors of the agent generate the perceptual information. Then the system checks out whether the percepts match or not with the primitive concepts, and when they match the concepts are added to the short-term memory as beliefs. The beliefs are checked out again in order to match more complex concepts and the process is repeated until there are no more

matches. The latter knowledge set, the skills, describes how the goals are decomposed into a set of ordered subgoals, and how these subgoals are also decomposed in other subgoals or basic actions.

The Icarus architecture is goal-oriented. In an execution cycle, after the architecture finishes checking the perceptual information, a goal is selected from the intentions of the agent and the best skill which satisfies that goal is fired. Then the goal is decomposed into subgoals and atomic actions, which are finally executed to interact with the environment. When the selected skill leads to a dead path where a relevant top goal cannot be satisfied the problem solving module is activated. This module goes backward in the actions and tries to satisfy the goals and subgoals with other skills. When the problem solving finds a solution it is stored as a new skill. This is how the learning process is performed in this architecture.

A bot for the videogame *Urban Combat*, a FPS game, was developed using Icarus cognitive architecture [9]. In that work the authors focus their interests in creating a bot able to find and defuse *Improvised Explosive Devices* (IED) at various locations in a map. The bot showed an interesting human-like behaviour in order to search and find objects in maps, first by exploring unknown maps and using later that information to reach the objectives faster.

In the following section we describe CERA-CRANIUM architecture, which significantly differs from the cognitive architectures reviewed above. While these typical cognitive architectures could be regarded as goal-oriented production systems, CERA-CRANIUM is better described as a blackboard-based system where behaviour emerges out of the dynamics of collaboration and competition of multiple specialized processors.

7.4 CC-Bot2 Design

The bot developed by *Conscious-Robots* team for the *2K BotPrize* 2010[2] contest, also known as *CC-Bot2*, is based on the *CERA-CRANIUM* cognitive architecture [2, 3]. CERA-CRANIUM is inspired by a computational model of consciousness and it is intended to control different autonomous agents, like physical robots or virtual characters. The main inspiration of the current version of CERA-CRANIUM is the Global Workspace Theory (GWT) [4, 5].

7.4.1 The Global Workspace Theory

A workspace is a kind of shared short-term memory space where different specialized processors can collaborate and compete with each other dynamically (see Fig. 7.2). According to the GWT, a large number of unconscious specialized processors provide

[2] http://botprize.org/

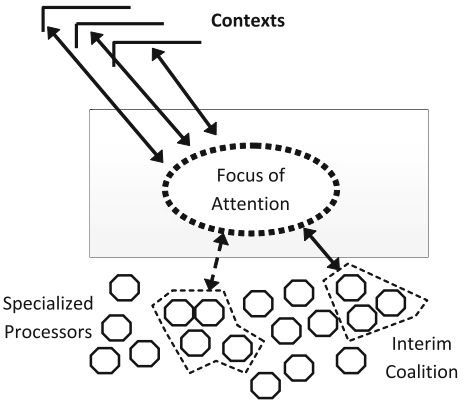


Fig. 7.2 Global workspace model of access consciousness. Interim coalitions of specialized processors compete for access to the focus of conscious attention. The application of contexts modulates the activation of processors, thus shaping the contents being formed in the working memory

data to a global workspace in which coherent information patterns are selected. Baars describes this workspace as a "theater", where the spotlight represents the focus of conscious attention [4]. In this model, all possible conscious contents are located in the scene (or working memory). The information obtained under the spotlight is distributed globally throughout the theater to all processors (the "audience"). This is the mechanism by which conscious experience activates unconscious contexts. Behind scenes, contextual processors shape the events taking place in the spotlight and help to interpret future conscious events. Recent evidence obtained using brain imaging techniques seems to support that real neuronal processes associated with consciousness follow similar mechanisms [6].

7.4.2 CERA-CRANIUM Cognitive Architecture

While classical control architectures focus exclusively on selecting which should be the next action to be executed (given current and past sensory information), CERA-CRANIUM includes an additional key process: it integrates a set of cognitive mechanisms in order to determine which should be the specific content of conscious perception. The agent will use this information to select the next action to be executed. In short, the generation of behaviour in CERA-CRANIUM is based on a competitive selection process that establishes, at any given time, a limited set of percepts considered the conscious content of the agent.

The architecture CERA-CRANIUM consists of two main components: CERA and CRANIUM. The former is a control architecture of four layers and the latter is a platform able to run and manage a large amount of parallel specialized processors.

7.4.2.1 CERA

CERA is a four layers cognitive architecture which uses CRANIUM as a tool to run and manage the specialized processors which process the information (see Fig. 7.3). As in other control architectures, the top layers manage more abstract meaning while low layers are closer to the simulated sensors and effectors of the bot. Current CERA implementation layers are: sensory-motor services layer, physical layer, mission-specific layer, and core layer.

The lowest layer of this architecture is the sensory-motor services layer. This layer is an interface between the architecture and the sensors and actuators of the bot. Basically, this layer includes the required services to retrieve sensory data readings and to send motor commands to the actuators. In the case of CC-Bot2, this layer is an adaptation of the Pogamut 3 functions to the CERA-CRANIUM architecture.

The next layer is the physical layer, which encloses the low-level representations of the sensory data and effectors commands. Typically, this layer includes processors that create more abstract representations out of the sensory information and processors that decompose high-level commands into atomic actions, which can be sent directly to the sensory-motor services layer.

A mission-specific layer produces and manages elaborated sensory-motor content related to agent's assigned missions. The information from the physical layer is combined into more meaningful contents related to the goals of the agent and processed to execute the corresponding mission behaviours.

The top layer is the core layer, where the mechanisms associated with the cognitive functions are regulated. The cognitive control modules enclosed in this layer are in charge of regulating the way the lower layers work, that is, how the processors interact with each other and what it is the most relevant content to be processed at any given time. In particular, CC-Bot2 included in this layer only an attentional mechanism, that prioritizes those contents closer to the current active sensory-motor context of the bot.

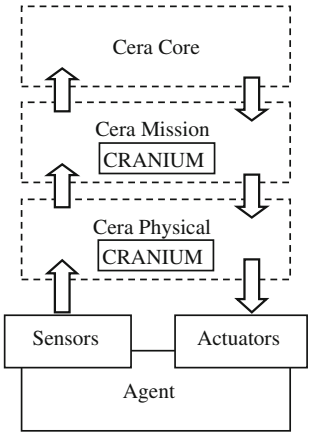


Fig. 7.3 Overview of CERA-CRANIUM architecture. The physical and mission layers use CRANIUM in order to manage short-term memory and associated specialized processors

7.4.2.2 CRANIUM

CRANIUM is an implementation of a global workspace. On one hand, CRANIUM is in charge of managing and dispatching the execution of specialized processors; on the other hand, it has a shared short-term memory space—the CRANIUM workspace—where the processors can interchange information. Each of the processors is designed to perform a specific function on certain types of data (some examples of these will be shown in Sect. 7.4.3, where CC-Bot2 is described).

CRANIUM includes a mechanism to select which processors are assigned more priority for execution. Although in principle all processors are treated as equals and all of them are supposed to run asynchronously and concurrently, limitations on computational resources have to be taken into account. Consequently, there is an implicit competition for activation amongst all the processors. The level of priority attained by a processor does not only affect its available execution time, but the specific information it might generate and submit to the workspace. This mechanism can be seen as a winner-takes-all algorithm, where the most activated signals are the ones most likely processed. From this point of view, CRANIUM is a particular implementation of a "pandemonium", where daemons—or specialized processors—compete with each other for activation. The activation of each processor is calculated based on a heuristic estimation of how much it can contribute to the global solution currently sought in the workspace. The concrete parameters used for this estimation are established by the CERA core layer. As a general rule, CRANIUM workspace operation is constantly modulated by commands sent from the CERA core layer.

The architecture CERA-CRANIUM is not composed of one CRANIUM workspace but two. There is one in the physical layer and other one located in the mission layer (see Fig. 7.3). Both workspaces are connected with each other through CERA sensory-motor information flows and share selected contents generated by the processors, specifically those with the highest activation level. So the outcome of the workspaces can be seen as a filter where only the signals that contribute more to the global solution pass from one layer to the other.

The workspace in the physical layer contains all the processors in charge of processing all data coming from the sensor services and also all processors which decompose the actions into basic commands, atomic actions, for the effectors. The workspace located in the mission layer is populated with higher-level specialized processors, which process the most activated information generated in the physical layer and the signals produced in the workspace itself. All this processed information generated by the specialized processors are integrated into data packages called single percepts, complex percepts, and mission percepts, depending on the sort of information they contain. The single percepts are those which include atomic information, usually sensory signals, while complex percepts include more elaborated and multimodal information. Finally, mission percepts contain information related with goals (and they are only produced in the mission layer).

There are two flows of information in CERA-CRANIUM. One is the bottom-up flow, where the information from the sensors, the percepts, are processed and combined in order to obtain abstract representations of the environment and the

state of the agent. The bottom-up flow can be referred as to the perception flow (see Fig. 7.4). The top-down information flow is concerned with the generation of adaptive behaviours oriented to achieve the agent's goals. Complex behaviours are decomposed into simpler actions and, then, in atomic commands that can be sent to the sensory-motor layer (see Fig. 7.5). The top-down flow could be considered, to some extent, to be equivalent to behaviour trees, in the sense that behaviours are associated to given contexts or scopes. However, the way CERA-CRANIUM selects the next action is quite different, as it depends on the level of activation of the action. As mentioned above, this activation is set by a heuristic function that estimates the contribution to the global solution, which in CC-Bot means how close the action is related to the current active sensory-motor context of the agent. A sensory-motor context defines a specific region in the multi-dimensional sensory-motor space, i.e. specific values within the range of the possible parameters for each sensory and motor modality. If the sensory input was limited to a three-dimensional space, a sensory context could be defined by specifying three parameters (x, y, and z). However, in the case of CC-Bot additional sensory dimensions, like relative time, are considered.

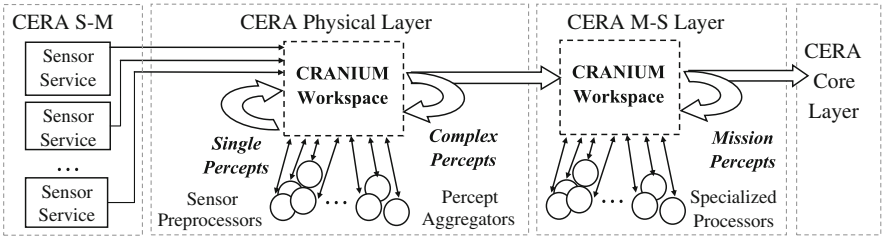


Fig. 7.4 CERA-CRANIUM bottom-up flow: perception. The sensor services generate the single percepts with the state of the world. These percepts are processed by the physical layer and combined in order to generate more complex percepts. The percepts with the most activation flow to the mission layer and are used to create the mission percepts. No percepts flow from the mission to the physical layer

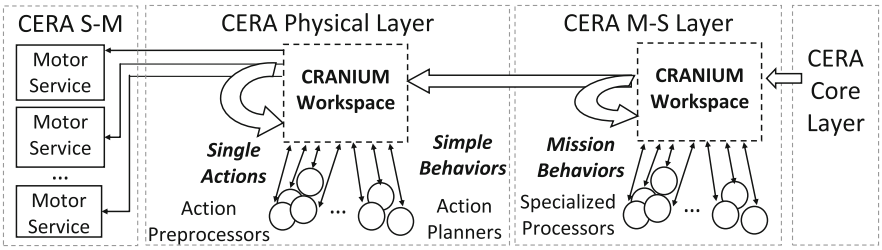


Fig. 7.5 CERA-CRANIUM top-down flow: behaviour generation. The mission layer generates the mission behaviours with the perceptual information. The decision of which behaviours are selected to be executed is modulated by the core layer. These mission behaviours are decomposed into simple behaviours and the ones with the greater activation flow to the physical layer. Physical layer decomposes simple behaviours into single actions and these ones are executed by the motor services

7.4.3 CC-Bot2 Processors

As a detailed description of all processors implemented in CC-Bot2 cannot be included in this chapter due to space limitations, we will describe some illustrative specialized processors. Specifically, we describe the processors related with shooting and moving behaviours. Table 7.1 shows the shooting processors, including the name of the processor, the CERA layer where it is included, and a brief description of its processing task.

The perceptual information required to shoot an enemy flows through the processors in the same order they are shown in Table 7.1. CRANIUM does not impose a specific order in the execution of processors. However, the particular processing chain described here is the result of the restrictions on data types consumed and produced by each processor. The first three processors are set in the physical layer and they are in charge of creating a more abstract perception about the enemies. Then, there are two processors in the mission layer that decide whom to shoot and whom not to. The last two processors are the ones that send the proper commands to the actuators in order to shoot a particular player.

The first two processors are in charge of managing the information about the players in the field of view of the bot. The processor *PlayerNoveltyDetector* processes the information provided by the sensors regarding new player appearances in the field of view. It also updates this information while the player is still in the field of view. *PlayerDisappearDetector* signals when a player completely disappears from the field of view, it uses a timer for each player, and when no notification about a particular player arrives for some time it triggers the creation of a new percept informing of the disappearance of that player.

The *AttackDetector* is a processor that creates a new percept when the health of the player is decreasing and it is caused by enemy fire. It is like a self-state sensor that informs the other processors when the player is being damaged. This information

Table 7.1 Processors involved in the generation of the shooting behaviour

Name	Layer	Description
PlayerNoveltyDetector	Physical	Detects when a new enemy appears in the field of view
PlayerDisappearDetector	Physical	Detects when an enemy disappears from the field of view
AttackDetector	Physical	Detects when the bot is being damaged
EnemyDetector	Mission	Detects all the possible enemies that are shooting the agent when it is being damaged
SelectEnemyToShoot	Mission	Selects an enemy amongst all players within the field of view to shoot to
ShootEnemy	Physical	Selects the point to shoot to in order to hit the selected enemy
ShootTo	Physical	Fires the corresponding actions to aim and shoot to a given point

is used along with the percepts shared by *PlayerNoveltyDetector* and *PlayerDisappearDetector* processors in the physical workspace, and by the *EnemyDetector* processor in the mission layer workspace. It reads the percepts from the workspace and generates a new percept indicating the probability that a specific player is shooting to the bot. Therefore, if nobody is within the field of view of the bot, the processor will create a percept indicating that someone is shooting the bot, but it does not know who that player is or where it is located. If there is an enemy in the bot's field of view and it is looking at the bot and shooting, then it is likely shooting at bot and a percept is created notifying that belief. Indeed, a percept indicating that a player is shooting at the bot is created per each enemy that is looking to the bot and shooting at the same time. Additionally, when no one is looking at the bot, but there are players firing their weapons they are also considered because they could have shot the bot some time ago.

Using all the percepts from the processors explained above, the *SelectEnemyToShoot* processor decides which enemy to attack. If there is an enemy shooting to the bot, it will be the first option, but if no one is shooting, then the closer player will be chosen. This processor is located in the mission layer and when an enemy is selected an action indicating to shoot that enemy is sent to the workspace. This action is in turn interpreted by the *ShootEnemy* processor in the physical layer. Using the information about the movement of the enemy and the current weapon the processor calculates the point where it has to aim to hit the enemy. This information is used to create a new action representing where the bot has to shoot. This action is decomposed into atomic actions by the *ShootTo* processor. First the processor sends a command to the effectors to aim at the target point, and when it is looking at that point it sends a command to fire the weapon. It also sends a command to stop shooting when it is not looking to the right point to shoot. In order to increase the believability of the bot we included imperfect accuracy when aiming. Instead of selecting the exact point where the target is, we add some noise in the angle of vision. Adding this noise the bot get good accuracy in closer targets but not so good for targets located far away, increasing the error proportionally to the distance to the target point.

The latter is a simple example of how the bot shoots an enemy. The shooting behaviour has been explained like a sequence of events that takes place within the architecture, but it is important to notice that all these events run in parallel. It is also important to remark that a given type of percept can be generated by more than one processor. In fact, two percepts of the same type but containing contradictory information can be generated in the same workspace, and both can be processed. The final behaviour shown by the bot depends on the activation level of all of these percepts and the actions generated. The most activated percepts and actions are the ones that are selected to pass from one layer to the next one, and the ones that are selected to be executed by the bot. The activation value of the percepts and actions is set by the cognitive functions of the bot. In the case of CC-Bot, the only cognitive function fully implemented was the attention mechanism. It modifies the behaviour of the bot depending on the current active sensory-motor context and also selects the next active context based on acquired sensory information.

In the case that a player starts shooting the bot, initially it will respond to the attack with the shooting action and by setting the current active sensory-motor context to the relative location of the enemy. If a new opponent appears later on, also shooting at the bot, the corresponding specialized processor will generate two actions, one to shoot each one of the enemies that are attacking. As the bot cannot shoot both bots at the same time, it will shoot the bot where the current context is set because the related percepts and actions would have a higher activation value. In the case that the first enemy stops shooting and the other keeps attacking, CC-Bot could keep shooting the first enemy while the context is set closer to that enemy. In other control architectures the normal response to such stimulus would be different, having the bot to start shooting the second bot immediately. Although, CC-Bot's response to the attack is not logical, it is a typical response by humans. When our attention is focused on a point, we tend to miss information about the rest of the environment (at least during a transient period of time).

The shooting behaviour and movement behaviour are probably the most important behaviours in order to make the bot behave human-like. Table 7.2 shows some processors related with the movement behaviours, all of them belong to the physical layer.

The movement behaviour is split in different simple behaviours, some of them with more priority than others. The most low level priority behaviour is the *RandomNavigation* processor. This processor make the bot wander in the map following the navigation points. The *RandomNavigation* knows which one is the current navigation point, then selects the next possible navigation points and selects one among the next visible navigations points. If no one is visible then other navigation point is selected. Once the next target point is selected the processor triggers a movement action towards that point. It also triggers a low-priority action of looking at the next navigation point. Some noise is added to the target point in order to make the movement more real. When the bot is close to the target point, the *RandomNavigation* processor generates a percept reporting it and the *RandomNavigation* processor selects the next point. The way the next point is selected had to be tweaked, the graphs of some maps include cycles without an exit. In order to avoid getting stuck in these cycles the processor detects when the bot has revisited some navigations

Table 7.2 Some of the processors involved in the generation of the movement behaviour

Name	Layer	Description
NotMoving	Physical	Detects when the bot is not moving during some time
StuckDetector	Physical	Detects when the bot is stuck in a given place
RandomMove	Physical	Moves randomly trying to unstuck the bot
RandomNavigation	Physical	Random movement among the navigation points. A wander behaviour
MoveCloserPosition	Physical	Fires a percept when the bot is close to a target position
RandomJumpGenerator	Physical	Performs a jump action randomly
AttackingMovement	Physical	Realizes a random movement when is in attacking mode

points for a while and starts to select other navigation points of the graph. This random navigation movement along the navigation points is only triggered when the *NoMoving* processor detects that the bot is still at a given spot.

In order to avoid obstacles when moving, we developed an obstacle detector based on raycasting. When an obstacle was detected in the direction of the movement another processor generates a jump action to avoid it. Unfortunately the raycasting did not work properly in the first stage of the 2K BotPrize and we had to deactivate it. Instead of this behaviour we used the *RandomJumpGenerator* processor in order to avoid persistent obstacles. This processor decided to randomly generate a jump action. This helped the bot to get unstuck and we think it might also have added some believability to the bot by jumping sometimes when it was not really needed.

The *AttackingMovement* processor is a specific movement behaviour that is issued when the bot is attacking an enemy or being attacked. This processor has higher priority than the *RandomNavigation* processor, therefore when both are active only the actions generated by *AttackingMovement* are executed. The processor performs very simple evasive movements executed during half a second.

Sometimes the movement behaviours lead the bot to get stuck, usually where the map does not have enough navigation points. In these cases the *StuckDetector* processor detects this condition and the *RandomMove* processor starts generating random movements until the bot gets out of the area where it was stuck. This is the highest priority movement behaviour given that it does not make sense to perform any other movement when the bot is stuck.

Apart from the processors described above, a total of 31 processors were implemented in CC-Bot2. Most of them were related with the movement of the bot and the shooting behaviour. These are the behaviours we identified as the most important to create a believable bot. Therefore, our efforts were focused in creating a human-like movement around the map using the map navigation points provided by the game, and a realistic shooting behaviour that selected the most appropriate enemy to shoot with an imperfect aim.

7.4.4 CC-Bot2 in the 2010 BotPrize Competition

The 2010 version of the 2K BotPrize included a total of five finalist teams from different countries around the world that competed against each other to design the most human-like bot (see Chaps. 1, 5, 6 for more information about other bots). The competition took part in three different days, in each session the bot competed against other humans and bots in four different maps on a deathmatch game, making a total of 12 rounds of 15 min each. The human players were able to judge the rest of the competitors as humans or bots, but only once per map. Detailed rules for the 2010 BotPrize competition are discussed in Chaps. 5 and 6. The results of the humanness ratio of the bots are shown in Fig. 7.6.

CC-Bot2 achieved the best results with a 31.81 % of humanness. This result was obtained based on 44 votes of the judges. This fact gives more reliability to the

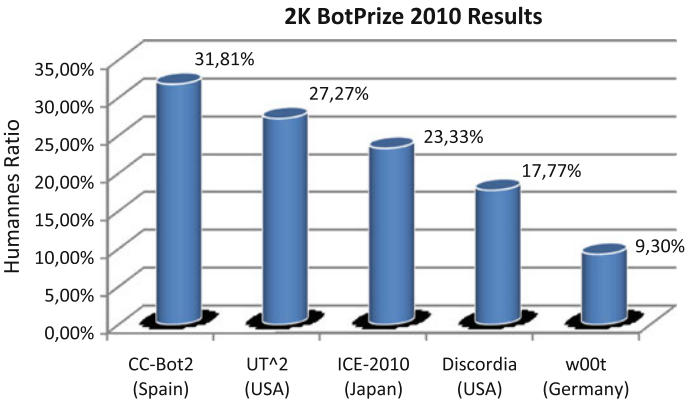


Fig. 7.6 2K BotPrize 2010 results. The graphic shows the percentage of humanness for each bot

humanness results of CC-Bot2. A discussion of why judges found CC-Bot more human can be found in Chap. 8.

7.5 Conclusions

In this chapter we have discussed a specific example—CERA-CRANIUM—of the application of the Machine Consciousness approach to the design of believable bots. Although the implementation of CERA-CRANIUM is in a very early stage, in which only the attention mechanism is fully implemented, the results obtained so far are promising and encourage us to keep on working in the same direction. In the following, we review the main advantages and disadvantages of using such an approach and we outline the main research lines that we plan to follow in the future.

7.5.1 Advantages and Disadvantages of Using Machine Consciousness Techniques for Video Games

Machine Consciousness seems to be a new promising approach in order to design believable bots, not only in games but also in the realm of robotics. A cognitive architecture, like CERA-CRANIUM, could provide the necessary mechanisms to develop cognitive features like attention or emotions, which we argue to be necessary in order to create complex human-like behaviours. Depending on the point of view, this can be seen as a previous stage before progressing onto the creation of human-like robots, but from the standpoint of the video games industry we have to adopt the technology with caution. When we are creating human-like behaviours with a cognitive architecture which implements some cognitive features, the activation of the programmed behaviours in the architecture starts to be more unpredictable,

and more dependable on the cognitive skills implemented instead of the behaviours programmed. Externally, the unpredictability could be seen like a hallmark of human-like behaviour, but for game developers, who are not sure how the bot will behave in unknown circumstances, this could be a problem if it behaves oddly and causes a failure in the game. On the other hand, a human-like agent could solve unpredicted situations where another agent would fail.

The attention cognitive skill is a powerful mechanism to filter and discard irrelevant information. Instead of spending time processing information which is not relevant for the current context of the agent, this information is discarded at some point and it never becomes part of the conscious content of the agent. For a video game, the attention capability reduces the required CPU load. Indeed, if there is an overload in the CPU at some point due to other tasks, the attention mechanism would discard more information and keep only the most relevant for the agent. In contrast to the current algorithms designed to create behaviours that tend to process all available information they are programmed to, the proposed attention mechanism is flexible enough to deal with special cases not considered in the development, thus preventing delays in the response of the agent and guaranteeing a good game experience.

Although attention was the only major cognitive skill used by CC-Bot2, other cognitive functions like the episodic memory or a model of emotions could increase the believability of the bot (see Chap. 8). The episodic memory is not only useful to remember what the agent was doing and resume the actions it was previously performing, but it is also useful to recall past experiences and learn not repeat the same mistakes when something similar happens again. In fact, it should be combined with learning skills in order to improve the behaviour of the agent.

A model of emotions can be considered another cognitive function that could increase the believability of a bot. Although all the agents in a game could be programmed in the same way with the same behaviours and other cognitive skills, the emotion model can be used to modulate the activation of behaviours in the agent and to make it behave differently. For example, a bot could fear being damaged and show a behaviour more likely to hide from other players, while another bot could have more courage and try to attack opponents all the time. In both cases, the behaviours would be the same but it would be the emotional model which decides what specific behaviours are more often used. These sorts of mechanisms could be considered the root for the design of models of personality.

The main disadvantage of the application of Machine Consciousness to video games is the complexity involved both in the design and implementations and the unpredictable behaviours that can emerge out of these techniques. The behaviours generated using cognitive architectures inspired by consciousness are not a programmed sequence of actions where the programmer can predict what is going to happen. The way that the final behaviour emerges from the programmed actions depends on the cognitive functions and how they modulate the activation signal of the actions. This increases the complexity associated to the understanding of the behaviour of the bot. Debugging this code can be very confusing for AI game developers without a background on Machine Consciousness.

Regarding performance, current CC-Bot2 code seems to scale fairly good in an experimental environment, where a dedicated server has been used to effectively control up to 10 bots simultaneously. However, a production environment would likely require a much more efficient code.

7.5.2 Final Discussion and Future Work

Although CC-Bot2 is a prototype to test the possibilities of Machine Consciousness in the field of video games, the results in the 2K BotPrize 2010 show that such an approach can achieve as good results as other AI techniques in the domain of human-like NPC behaviour for FPS videogames. The advantage, and main difference, of Machine Consciousness in comparison with other techniques is that it is an emerging research field which is expected to grow and progress during the next years. In relation to this, we have proposed a specific roadmap based on *ConsScale* (Chap. 8), a framework inspired by consciousness and conceived to measure the level of cognitive development of an agent.

Future versions of CC-Bot2 are planned to integrate more cognitive functions such as a model of emotions, a mechanism for episodic memory, advanced planning, better explicit reasoning, reinforcement learning, etc. Given the great effort required to implement all these features, we plan to follow the roadmap proposed in *ConsScale* in order to prioritize the design and implementation of those functionalities that are considered the basis of a cognitive system.

Acknowledgments We are indebted to the reviewers and the editor of the book for their helpful comments and critique.

References

1. Anderson, J., Bothell, D., Byrne, M., Douglass, S., Lebiere, C., Qin, Y., et al.: An integrated theory of the mind. Psychol. Rev. **111**(4), 1036–1060 (2004)
2. Arrabales, R., Ledezma, A., Sanchis, A.: CERA-CRANIUM: a test bed for machine consciousness research. In: International Workshop on Machine Consciousness. Towards a Science of Consciousness 2009, p. 105. Hong Kong (2009)
3. Arrabales, R., Ledezma, A., Sanchis, A.: Towards conscious-like behavior in computer game characters. In: IEEE Symposium on Computational Intelligence and Games, pp. 217–224. IEEE Press, Milano, Italy (2009)
4. Baars, B.: A Cognitive Theory of Consciousness. Cambridge University Press, New York (1988)
5. Baars, B.: Global workspace theory of consciousness: toward a cognitive neuroscience of human experience. Prog. Brain Res. **150**, 45–53 (2005)
6. Baars, B.J.: The conscious access hypothesis: origins and recent evidence. Trends Cogn. Sci. **6**, 47–52 (2002)
7. Best, B., Lebiere, C., Scarpinatto, C.: Modeling synthetic opponents in MOUT training simulations using the ACT-R cognitive architecture. In: Proceedings of the 11th Conference on Computer Generated Forces and Behavior Representation (2002)

8. Chella, A., Manzotti, R.: Artificial Consciousness. Imprint Academic, UK (2007)
9. Choi, D., Konik, T., Nejati, N., Park, C., Langley, P.: A believable agent for first-person shooter games. In: Proceedings of the third annual artificial intelligence and interactive digital entertainment conference, pp. 71–73 (2007)
10. Epic Games, Inc.: Unreal Torunament 2004. http://www.unrealtournament2003.com/. Accessed Oct 2010
11. Franklin, S.: A Consciousness Based Architecture for a Functioning Mind. Visions of Mind. IDEA Group Inc, Hershey (2005)
12. Gamez, D.: Progress in machine consciousness. Conscious. Cogn. **17**(3), 887–910 (2008)
13. Gemrot, J., Kadlec, R., Bída, M., Burkert, O., Píbil, R., Havlíček, J., Zemčák, L., Šimlovič, J., Vansa, R., Štolba, M., Plch, T., Brom, C.: Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. Agents for Games and Simulations, pp. 1–15 (2009)
14. Goertzel, B.: Achieving advanced machine consciousness through integrative, virtually embodied artificial general intelligence. In: Haikonen, P.O.A. (ed.) Proceeding of the Nokia Workshop on Machine Consciousness, pp. 19–21. Helsinki (2008)
15. Goertzel, B., Pennachin, C., Geisweiller, N., Looks, M., Senna, A., Silva, W., Heljakka, A., Lopes, C.: An integrative methodology for teaching embodied non-linguistic agents, applied to virtual animals in Second Life. In: Artificial General Intelligence 2008, Proceedings of the First AGI Conference, pp. 161–175 (2008)
16. Hingston, P.: A Turing Test for computer game bots. IEEE Trans. Comput. Intell. AI Games **1**(3), 169–186 (2009)
17. Holland, O.: Machine Consciousness. Imprint Academic, UK (2003)
18. Laird, J.: Soar: an architecture for general intelligence. Technical Report, DTIC Document (1987)
19. Laird, J.: Extending the Soar cognitive architecture. In: Proceedings of the First AGI Conference, pp. 224–235. IOS Press (2008)
20. Laird, J., van Lent, M.: Developing an artificial intelligence engine. In: Proceedings of the Game Developers Conference, pp. 577–588 (1999)
21. Langley, P., Cummings, K., Shapiro, D.: Hierarchical skills and cognitive architectures. In: Proceedings of the twenty-sixth annual conference of the cognitive science society, pp. 779–784. Citeseer (2004)
22. Langley, P., Laird, J., Rogers, S.: Cognitive architectures: Research issues and challenges. Cogn. Syst Res. **10**(2), 141–160 (2009)
23. Langley, P., McKusick, K., Allen, J., Iba, W., Thompson, K.: A design for the Icarus architecture. ACM SIGART Bull. **2**(4), 104–109 (1991)
24. Magerko, B., Laird, J., Assanie, M., Kerfoot, A., Stokes, D.: AI characters and directors for interactive computer games. Ann Arbor **1001**(48), 109–2110 (2004)
25. Nareyek, A.: AI in computer games. Queue **1**, 58–65 (2004). http://doi.acm.org/10.1145/971564.971593.
26. Polson, P.: Human associative memory. Am. J. Psychol. **88**(1), 131–140 (1975)
27. Prem, E.: Epistemological aspects of embodied artificial intelligence. Cybern. Syst. **28**(6), 3 (1997)
28. Samsonovich, A.V.: Toward a unified catalog of implemented cognitive architectures. In: Samsonovich, A.V., Jóhannsdóttir, K.R., Chella, A., Goertzel, B. (eds.) Biologically Inspired Cognitive Architectures 2010—Proceedings of the First Annual Meeting of the BICA Society, pp. 195–244. IOS Press (2011)
29. Tozour, P.: First-Person Shooter AI Architecture, Chap. 8, pp. 387–396. AI Game Programming Wisdom. Charles River Media, Hingham (2002)
30. Vygotsky, L.S.: Mind in Society: The Development of Higher Psychological Processes. Harvard University Press, Cambridge (1980)
31. Wintermute, S., Xu, J., Laird, J.: SORTS: a human-level approach to real-time strategy AI. Ann Arbor **1001**(48), 109–2121 (2007)

Chapter 8
ConsScale FPS: Cognitive Integration for Improved Believability in Computer Game Bots

Raúl Arrabales, Agapito Ledezma and Araceli Sanchis

Abstract *ConsScale* is a framework for the characterization and measurement of the level of cognitive development in artificial agents. The scale is inspired by an evolutionary perspective of the development of consciousness in biological organisms. In this context, consciousness is considered as an evolutionary advantage that provides the individual with a highly adaptive super-function, i.e. effective synergistic integration of a number of cognitive functions. *ConsScale* FPS is a specific instantiation of the scale aimed at describing the hierarchical structure of the cognitive skills required by a bot in the domain of a first-person shooter (FPS) video game. Using *ConsScale* FPS as reference framework, this work proposes specific strategies for the design of more believable bots. The concept of cognitive dependency is introduced and cognitive abilities implemented in state of the art bots are analyzed from the point of view of effective cognitive integration. Finally, in light of the cognitive development of current bots, particular improvements are suggested towards the design of more human-like bots.

8.1 Introduction

ConsScale is a comprehensive framework for measuring, comparing, and characterizing the level of cognitive development of artificial agents [6]. The scale is inspired by the development of consciousness in biological creatures and can be used both to analyze the cognitive development of an existing agent and also to guide the design

R. Arrabales (✉) · A. Ledezma · A. Sanchis
Carlos III University of Madrid, Avda. Universidad 30, 28911 Leganés, Madrid, Spain
e-mail: rarrabal@inf.uc3m.es

A. Ledezma
e-mail: ledezma@inf.uc3m.es

A. Sanchis
e-mail: masm@inf.uc3m.es

P. Hingston (ed.), *Believable Bots*, DOI: 10.1007/978-3-642-32323-2_8,

process of new artificial agents [4]. This chapter is mostly concerned with the latter issue, approaching the design question from the perspective of believable computer game characters.

In the context of this work a cognitive agent (or a cognitive computer game bot) is an artificial system designed to operate based on a set of typically human processes such as perception, attention or learning. As defined by Neisser [11],

> Cognition refers to all processes by which the sensory input is transformed, reduced, elaborated, stored, recovered, and used. It is concerned with these processes even when they operate in the absence of relevant stimulation, as in images and hallucinations.

The problem of believability can be seen from two perspectives: on one hand, believability has to be assessed by an observer; on the other hand, game bots have to be designed in such a way they appear believable to humans. Obviously, the design process has to be driven by the feedback about believability provided by the observer. However, it is not straightforward to obtain the required feedback and to effectively use it to improve the design of bots. Taking initially a very simplistic approach, believability could be considered as a binary property, i.e. a bot could be simply considered either believable or not believable. This is actually consistent with the framework proposed by Turing six decades ago [12], and it is indeed a compelling approach when it comes to empirically proving that an artificial agent flawlessly generates human-like behaviour. Consequently, adapting the Turing Test to the domain of video games, as in the BotPrize contest [9], seems to be a perfectly valid approach to detect human-like bots. In other words, testing protocols based on the Turing Test are good if the goal is to detect human-like bots. However, they are not that appropriate tools at development time, when successive versions of bots are expected to be more and more believable as their design is being improved.

The root problem of the Turing Test is that it is based on a binary output: the agent is either indistinguishable from a human or not. This is an important drawback from the point of view of the Artificial Intelligence engineer. A much more specific feedback is required in order to effectively guide computer bots design towards more human-like behaviour. Even in adapted versions of the test, such as the BotPrize, the humanness ratio is obtained from the combination of binary decisions made by human judges. The task assigned to the judges is to tell humans and bots apart, but they are not asked to rate the level of humanness of bots. One of the reasons why believability is not typically assessed using a non-binary measurement is the lack of suitable reference frameworks. *ConsScale* has been designed to address this issue in the broader domain of artificial agents. We argue that *ConsScale* can be used as a human-like behaviour scale because it evaluates the presence of behaviours correlated with higher cognitive functions present in human beings. Furthermore, *ConsScale* provides a comprehensive ordered list of levels associated to the cognitive development of consciousness. Lower levels represent basic and primitive behaviours, while higher levels correspond to more sophisticated and human-like behaviours. This framework permits a practical evaluation of the cognitive power of an agent, considering the existing cognitive gap between full human-like behaviour and its current behavioural pattern.

In the following we describe the *ConsScale* generic framework and *ConsScale* FPS, an instantiation of *ConsScale* to the specific domain of first-person shooter video game bots, and we review the associated assessment tools. A number of state of the art FPS bots that participated in the BotPrize 2010 contest are evaluated using *ConsScale* FPS, and the results are confronted with judge's assessments. Additionally, human observers' evaluation process is analyzed from the point of view of *ConsScale* and the believability criteria they use are compared with those of *ConsScale*. Finally, relevant conclusions are discussed and next steps for the design of more believable bots are suggested, proposing a specific roadmap based on *ConsScale* FPS.

8.2 *ConsScale*

The design of believable bots is a complex problem that calls for the planning of a detailed roadmap. Generating human-like complex behaviour is not the kind of task that can be addressed using a single technique and fully solved at the first attempt. Having a framework for the evaluation and characterization of the cognitive capabilities of an artificial agent is a key aspect for a successful approach. *ConsScale* is a pragmatic tool intended to define such a framework. This scale evaluates the functional aspects of an artificial agent using architectural and behavioural criteria. The main goal of the application of *ConsScale* is to characterize how cognitive functions are integrated and how inter-function synergies contribute to the generation of conscious-like behaviours. Using *ConsScale* the cognitive power of an artificial agent is characterized using a hierarchical list of levels of cognitive development.

8.2.1 ConsScale Levels of Cognitive Development

Although *ConsScale* levels are defined using both architectural and functional criteria, in this chapter we will focus primarily on the functional or cognitive capabilities. Architectural criteria are covered in detail elsewhere [6]. From the total 13 levels defined in *ConsScale* (from level −1 to level 11, including level 0), here we will cover only the most common 9 levels. Table 8.1 summarizes the cognitive skills (CS) required in these levels. Note that agents can qualify as level *n* only if all lower levels are also satisfied. In other words, higher levels subsume lower ones.

From the point of view of behaviour each level defines a set of generic cognitive skills that must be satisfied. Therefore, in order to apply the scale to the problem of bot believability, these CS need to be grounded to behavioural tests that can be evaluated during game play. This instantiation process is described below.

Table 8.1 *ConsScale* level 2–10

L_i (level name)	Cognitive skills ($CS_{i,j}$)[a]
2 (*reactive*)	$CS_{2,1}$: Fixed reactive responses ('reflexes')
3 (*adaptive*)	$CS_{3,1}$: Autonomous acquisition of new adaptive reactive responses
	$CS_{3,2}$: Usage of proprioceptive sensing for embodied adaptive responses
	$CS_{3,3}$: Selection of relevant sensory information
	$CS_{3,4}$: Selection of relevant motor information
	$CS_{3,5}$: Selection of relevant memory information
	$CS_{3,6}$: Evaluation (positive or negative) of selected objects or events
	$CS_{3,7}$: Selection of what needs to be stored in memory
4 (*attentional*)	$CS_{4,1}$: Trial and error learning. Re-evaluation of selected objects or events
	$CS_{4,2}$: Directed behaviour toward specific targets like following or escape
	$CS_{4,3}$: Evaluation of the performance in the achievement of a single goal
	$CS_{4,4}$: Basic planning capability: calculation of next n sequential actions
	$CS_{4,5}$: Depictive representations of percepts
5 (*executive*)	$CS_{5,1}$: Ability to move back and forth between multiple tasks
	$CS_{5,2}$: Seeking of multiple goals
	$CS_{5,3}$: Evaluation of the performance in the achievement of multiple goals
	$CS_{5,4}$: Autonomous reinforcement learning (emotional learning)
	$CS_{5,5}$: Advanced planning capability considering all active goals
	$CS_{5,6}$: Ability to generate selected mental content with grounded meaning
6 (*emotional*)	$CS_{6,1}$: Self-status assessment (background emotions)
	$CS_{6,2}$: Background emotions cause effects in agent's body
	$CS_{6,3}$: Representation of the effect of emotions in organism (feelings)
	$CS_{6,4}$: Ability to hold a precise and updated map of body schema
	$CS_{6,5}$: Abstract learning (learned lessons generalization)
	$CS_{6,6}$: Ability to represent a flow of integrated percepts including self-status

(continued)

Table 8.1 (continued)

L_i (level name)	Cognitive skills ($CS_{i,j}$)[a]
7 (*self-conscious*)	$CS_{7,1}$: Representation of the relation between self and perception
	$CS_{7,2}$: Representation of the relation between self and action
	$CS_{7,3}$: Representation of the relation between self and feelings
	$CS_{7,4}$: Self-recognition capability
	$CS_{7,5}$: Advance planning including the self as an actor in the plans
	$CS_{7,6}$: Use of imaginary states in planning
	$CS_{7,7}$: Learning of tool usage
	$CS_{7,8}$: Ability to represent and self-report mental content (continuous inner flow of percepts)
8 (*empathic*)	$CS_{8,1}$: Ability to model others as subjective selves
	$CS_{8,2}$: Learning by imitation of a counterpart
	$CS_{8,3}$: Ability to collaborate with others in the pursuit of a common goal
	$CS_{8,4}$: Social planning (planning with socially aware plans)
	$CS_{8,5}$: Ability to make new tools
	$CS_{8,6}$: Inner imagery is enriched with mental content related to the model of others
9 (*social*)	$CS_{9,1}$: Ability to develop Machiavellian strategies like lying and cunning
	$CS_{9,2}$: Social learning (learning of new Machiavellian strategies)
	$CS_{9,3}$: Advanced communication skills (accurate report of mental content)
	$CS_{9,4}$: Groups are able to develop a culture
	$CS_{9,5}$: Ability to modify and adapt the environment to agent's needs
10 (*human-like*)	$CS_{10,1}$: Accurate verbal report. Advanced linguistic capabilities
	$CS_{10,2}$: Ability to pass the Turing Test
	$CS_{10,3}$: Groups are able to develop a civilization and advance culture and technology

[a]As per the definition of *ConsScale* version 3.0

8.2.2 *Characterizing the Global Degree of Cognitive Development of an Agent*

Using the proposed list of cognitive skills (see Table 8.1), an artificial agent can be analyzed to determine which functions from the list are implemented. However, a practical characterization of the global cognitive power of the agent calls for the combination of the results obtained for each level. *ConsScale* provides two related integrative tools in order to represent, at a glance, the degree of cognitive development of an agent. One is a quantitative score, the CQS, the *ConsScale* Quantitative Score [4], and the other is based on a graphical cognitive profile representation, the GCP, Graphical Cognitive Profile [2].

8.2.2.1 *ConsScale* Quantitative Score

The CQS is an assessment tool intended to provide a numerical value as an indication of the degree of cognitive development of an agent. The CQS can be calculated in three steps:

L_i (degree of compliance of level i) provides a measure of the compliance with level i (ranging from 0.0 to 1.0). This measure uses an exponential curve to represent the existing synergy between cognitive functions located at the same level (see Eq. 8.1).

$$L_i = \begin{cases} 0 & \text{if } ncsf \text{ is } 0 \\ \frac{(ncsf+(J-J_i))^3}{10^3} & \text{otherwise} \end{cases} \tag{8.1}$$

The parameter *ncsf* (number of cognitive skills fulfilled) is the number of cognitive skills ($CS_{i,j}$) that the agent entirely fulfils. J is the maximum number of cognitive skills considered for any level, and J_i is the total number of cognitive skills defined for level i.

CLS (Cumulative Level Score) combines all L_i measures into one single aggregated value. This score follows a logarithmic progression which prevents the final score to be distorted by the combined effect of large scores in higher levels and poor scores in lower levels (see Eq. 8.2).

$$CLS = \sum_{i=2}^{11} \left(\frac{L_i}{i-1} \right)^2 \tag{8.2}$$

Finally, the CQS provides a single value, ranging from 0 to 1000, which indicates the synergy produced by the integration of cognitive skills across all levels. CQS is designed as an exponential curve (see Eq. 8.3) favoring those agents that follow the developmental path implicit in *ConsScale* level ordering (see Fig. 8.1).

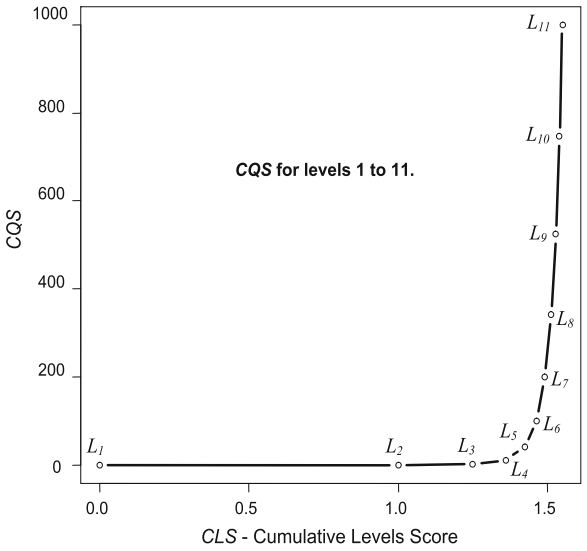


Fig. 8.1 Possible CQS values as a function of CLS. X-axis represents the CLS value and Y-axis represents the resulting CQS

$$CQS = \frac{e^{CLS^5/K} + a}{10} \tag{8.3}$$

K and a are constants (0.97062765 and -1 respectively) specifically defined to normalize the score between 0 and 1000, i.e. the minimum score 0 corresponds to *ConsScale* levels -1, 0, and 1; and the maximum level of cognitive development is represented by a CQS of 1000 (*ConsScale* level 11).

Using the former equations the value of CQS could be estimated for a particular agent by analyzing the potential presence of cognitive skills.[1] However, for an accurate evaluation the scale has to be instantiated to a particular problem domain, and the cognitive skills need to be translated into specific behavioural profiles.

8.2.2.2 *ConsScale* Cognitive Profiling

Although it is useful to have a single quantitative score, like the CQS, for a quick global evaluation, there is an associated drawback: CQS does not convey much information about the specific cognitive strengths and weaknesses of an agent. Bearing this issue in mind, we have proposed the complementary use of a graphical representation of the *ConsScale* cognitive profile of an agent.

L_i scores have to be considered in order to represent the cognitive profile of an agent (see Eq. 8.1). Both CLS and CQS are one-dimensional parameters, which are

[1] A CQS calculator is available online at http://www.consscale.com.

calculated as a function of the vector of L_i values corresponding to all *ConsScale* levels. The information about the degree of development of each level resides in these L_i parameters. Therefore, the proposed cognitive profile representation is based on these values that directly describe the achievement of an agent in terms of the definition of each *ConsScale* level.

Given that the first three levels of *ConsScale* (level −1 or *disembodied*, level 0 or *isolated*, and level 1 or *decontrolled*) are just used as theoretical references that do not usually exist in practice, they have been excluded from the graphical representation of cognitive profiles.

Regarding the specific form of the graphical representations, several types of charts can be used.[2] Although we have used radar charts in former works [2], in this chapter we will use horizontal bar charts, as they provide an ideal layout for the effective comparison of different profiles (see Fig. 8.2 for basic descriptions of the proposed graphical representation).

Current artificial cognitive architectures usually have good scores only in the lower section of the charts [5]. Considering the challenge of generating human-like behaviour, we think that cognitive profiles of believable agents should have good L_i scores in the whole bottom half of the cognitive profile chart. Nevertheless, robust believable agents would require complying with all levels up to 9 or 10 (at least in the context they are tested) in order to be completely indistinguishable from humans. In the following, we focus on the specific problem domain of FPS game bots and discuss in detail the required cognitive profiles for believability.

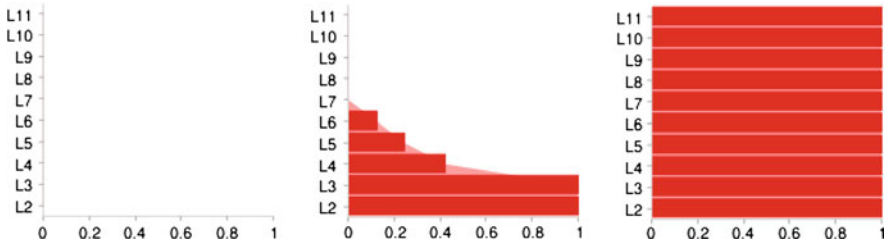


Fig. 8.2 The figure shows three *horizontal bar* charts representing different *ConsScale* cognitive profiles. Each *bar* represents the degree of accomplishment in one *ConsScale* level. Bars corresponding to lower levels are located in the *bottom* and higher levels are represented in the *top* of the chart. Possible values of each L_i bar range from 0.0 (no $CS_{i,j}$ is fulfilled) to 1.0 (all $CS_{i,j}$ are fulfilled). The chart in the *left* represents an empty *ConsScale* profile, where all L_i are 0. The corresponding system also has a CQS of 0. The chart in the *middle* represents a level 3 (*adaptive*) agent. Although this agent shows features from higher levels it can only be considered as level 3. However, its CQS (3.08) is higher than that of a pure level 3 agent (which is 2.22). The chart in the *right* represents a level 11 (*super-conscious*) agent. In this case, as all levels have an assigned L_i value of 1, the corresponding CQS value is 1000 (the maximum CQS value)

[2] The online application available at http://www.consscale.com allows the user to select and generate a number of different chart types.

8.2.3 Cognitive Functions Hierarchy and Integration

There appear to be a contradiction between *ConsScale* CQS measure and the way human observers assess believability. In fact, there also seems to be an inconsistency between the concept of human-like behaviour in *ConsScale* (a level 10 agent) and the idea of humanness of BotPrize judges (see Fig. 8.8). CQS reaches large values only when cognitive functions are implemented on top of completely fulfilled levels. Analogously, an agent can only be rated as level 10 if all lower levels are also completely fulfilled. The main idea behind this approach is that there exists a functional dependency between some cognitive functions. Therefore, *ConsScale* aims at representing these dependencies and establishing a hierarchical model that can be used both for assessing cognitive development and for guiding the design of more human-like agents.

The concept of cognitive dependency is the basis of the order established in *ConsScale* levels [5]. Actually, the set of all cognitive skills can be considered a partially ordered set (poset) in which the skills from one level depend on other skills from the immediate lower level (see Fig. 8.3). For instance, $CS_{7,4}$ depends on $CS_{6,4}$. In other words, self-recognition capability ($CS_{7,4}$) requires the ability to hold a precise and updated map of body schema ($CS_{6,4}$). These cognitive dependencies are of especial application in real world physical agents, where lower sensorimotor skills have to be acquired in order to effectively cope with more complex cognitive tasks. However, in the domain of video games, such dependencies might be less significant in some cases, as higher cognitive skills do not really depend on real world sensorimotor interactions. Nevertheless, we think these dependencies are still present and must be taken into account from the perspective of the bot designer. An interesting example in this context is Theory of Mind (the ability to reason about other's beliefs, desires and intentions). According to *ConsScale* an agent is able to fully develop the Theory of Mind cognitive skill if and only it is able to reason about itself in the first place. Therefore, the "I know you know" cognitively depends on the "I know".

From the point of view of the human observer who assesses the degree of humanness of a bot, dependencies might not be overtly perceived, as discussed above. But that does not mean that they are not present. We believe that they should be taken into account at design/development time because that is the appropriate way to design and implement the cognitive skills human observers are expecting to see.

8.3 *ConsScale* FPS

The generic definition of the levels of cognitive development included in *ConsScale* is virtually applicable to any kind of agent. However, an instantiation process is required in order to practically assess the cognitive profile of a particular implementation. The

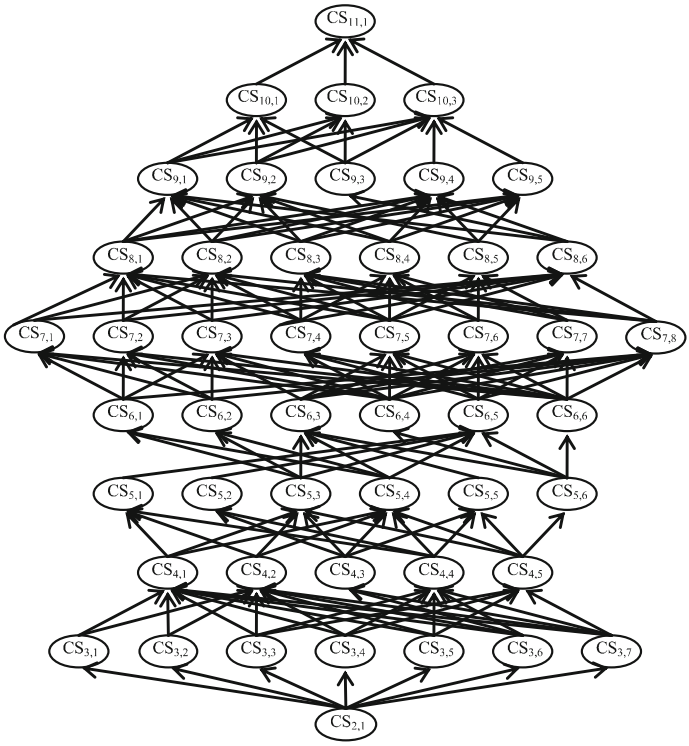


Fig. 8.3 Hasse diagram of the poset that represents cognitive dependencies between all cognitive skills defined in *ConsScale*

list of cognitive skills included in Table 8.1 refers to generic abilities, which means that they cannot be directly used for accurate assessment. A so-called *ConsScale* instantiation process has to be performed with the aim of effectively evaluating the presence of the cognitive skills considered in each level. In the remainder of this chapter we present the *ConsScale* instantiation for FPS game bots and use it as the basis for the discussion about believability.

8.3.1 The Domain of FPS Game Bots

The target domain of FPS bots has to be defined in terms of a problem-specific ontology in such a way that generic cognitive skills from each *ConsScale* level can be translated into testable behaviour patterns. This ontology is based on two main components: objects and actions. Objects or percepts are what the sensors of the agent can acquire from the (virtual) world or their own (simulated) body. Actions or verbs are what the agent effectors can do.

Fig. 8.4 Screen capture of Unreal Tournament 2004 deathmatch game during the 2K BotPrize 2010 competition

The agents considered in this context are bots designed to compete against each other in a FPS deathmatch game (see Fig. 8.4). The main goal of the game is to kill as many other players as possible during a given period of time. In such a scenario the following objects are considered: players, obstacles, weapons, ammo, and health packs. Analogously, the following actions are taken into account: move, jump, run, turn, damage, fire, and sending chat messages.

The application of the former ontology allows the redefinition of the *ConsScale* cognitive skills, adapting them to the domain of FPS bots. Using the translated FPS-specific cognitive skills, specific behavioural patterns (BP) can be specified (see Table 8.2). These patterns are observable and enable us to actually evaluate the presence of the corresponding cognitive skills in the game bot.

8.3.2 Using ConsScale FPS

Having this ordered list of behavioural patterns, bots can be systematically evaluated. Evaluation is to be carried out primarily by observation. Nevertheless, *ConsScale* also considers the internal inspection of the architecture of the bot in order to make sure that no pre-programmed routines are used to fool the scale [6].

Behavioural patterns from highest levels are difficult to identify and test in relatively simple environments like a FPS game. However, as these high level skills are not expected to be demonstrated in such an environment this issue should not affect the believability assessment. Specifically, levels 9, 10, and 11 would require extremely complex environments, like the real world, in order to be satisfactorily tested. Therefore, we can conclude that a FPS believable bot does not really need to achieve level 9. In other words, a bots is expected to

Table 8.2 *ConsScale* FPS instantiation

Cognitive skills ($CS_{i,j}$)	Behavioural patterns ($BP_{i,j}$)
$CS_{2,1}$: Reflexes	$BP_{2,1}$: Basic reflexes, as the ability to back up whenever the bot bumps into another bot or object
$CS_{3,1}$: Ability to learn new simple behaviours adapted to the game	$BP_{3,1}$: Basic behaviours that help the bot reaching better scores, like shooting other players when they are detected
$CS_{3,2}$: Ability to use self state (health, ammo, etc.) to learn new adapted behaviours	$BP_{3,2}$: Looking for health packs when health level is low or looking for ammo when needed
$CS_{3,3}$: Ability to ignore sensory input not critical to current task	$BP_{3,3}$: Ignoring detected ammo reloading kits when involved in a firefight and no more ammo is needed
$CS_{3,4}$: Ability to discard actions not suitable for current situation	$BP_{3,4}$: Actions like firing to walls when running away from an enemy are considered useless and avoided
$CS_{3,5}$: Ability to select what information worth remembering (accessed from memory)	$BP_{3,5}$: When the bot is in need of ammo, it access its memory to get the position of previously seen ammo packs, then it goes directly to pick up the closest one
$CS_{3,6}$: Ability to evaluate other players as friends or enemies. Ability to evaluate the benefits obtained by different ammo or health packs.	$BP_{3,6}$: Bot does not attack friends. Healing and re-arming is performed quickly by selecting the best health and ammo packs
$CS_{3,7}$: Ability to select what information should be stored in memory	$BP_{3,7}$: The position of health or ammo packs that could be needed later are stored in memory. The bot goes directly to a remembered position when it needs a pack (see $BP_{3,5}$)
$CS_{4,1}$: Ability to learn from trial and error	$BP_{4,1}$: The bot identifies other players as friends or enemies by trial and error. If a player currently considered as friend (see $BP_{3,6}$) starts attacking the bot, it is now considered as enemy and the corresponding adaptive behaviours are performed (running away or shooting)
$CS_{4,2}$: Ability to adapt behaviour to specific targets	$BP_{4,2}$: The bot shows directed and sustained behaviour towards enemies, like following and shooting them or running away from them
$CS_{4,3}$: Ability to evaluate own's performance in combat	$BP_{4,3}$: Actions that are not contributing to the expected goal are discarded. For instance, running away behaviour is changed by another when this behaviour is not contributing to diminish damage
$CS_{4,4}$: Basic ability to plan next movements	$BP_{4,4}$: Bot shows a coherent sequence of actions planned in order to reach certain goal. For instance, leaving a firefight for re-arming and then going back to combat

Table 8.2 (continued)

Cognitive skills ($CS_{i,j}$)	Behavioural patterns ($BP_{i,j}$)
$CS_{4,5}$: Ability to keep a depictive [1] representation of objects in the game, i.e. representation in a sensorimotor grounded manner	$BP_{4,5}$: The bot is able to effectively locate objects and calculate relative positions despite of its changing body and sensor positions (see $BP_{4,2}$). Bot shows a good shooting accuracy
$CS_{5,1}$: Ability to interleave between game tasks	
$CS_{5,2}$: Ability to pursue several game goals	
$CS_{5,3}$: Ability to evaluate performance in relation with the accomplishment of several game goals	$BP_{5,1-3}$: Behaviours interrupted due to certain circumstances are later resumed (see $BP_{4,4}$). For instance, a firefight is eluded because the bot is in need of healing, after getting a health pack, the bot resumes the attack. Additionally, the bot estimates to what extent goals are being accomplished depending on strategies being used. Effective behaviours are repeated more frequently than behaviours that lead to poor results
$CS_{5,4}$: Ability to learn based on game experience	$BP_{5,4}$: Evaluation performed according to $CS_{5,3}$ is used to select most promising strategies (see $BP_{5,3}$). For instance, the bot learns to use most destructive weapons when they are available
$CS_{5,5}$: Ability to plan actions taking into account all active game goals	$BP_{5,5}$: Actions are effectively interleaved as required for the accomplishment of multiple active goals. For instance, trajectory is slightly modified whilst chasing and shooting an enemy in order to pick up some ammo packs available in the surroundings
$CS_{5,6}$: Ability to generate and select explicit representations of the world with grounded meaning	$BP_{5,6}$: The bot autonomously generates internal high-level representations of the events taking place around it, for instance, some percepts indicating a dangerous place in a map are automatically generated by the bot (without being explicitly pre-programmed)
$CS_{6,1}$: Ability to assess global self-status as an actor in the game. This represents functional aspects of emotions	$BP_{6,1-3}$: The bot enters a particular state depending on self-status assessment. Global behaviour is biased by this state; for instance, if health is very low and no health packs are available, the bot tends to behave as if it was scared, avoiding any risk
$CS_{6,2}$: Ability to adapt control mechanism to current status	
$CS_{6,3}$: Ability to keep a representation of emotions as described in $CS_{6,1}$ [7]	
$CS_{6,4}$: Ability to keep an accurate representation of player's body	$BP_{6,4}$: The bot control its position, gesture and orientation effectively. For instance, it is able to coordinate its sensorimotor systems to run in one direction while shooting to another relative direction at the same time

Table 8.2 (continued)

Cognitive skills ($CS_{i,j}$)	Behavioural patterns ($BP_{i,j}$)
$CS_{6,5}$: Ability to learn abstract concepts related to the game	$BP_{6,5}$: Intelligent decisions indicate that specific knowledge about the game has been learnt. For instance, the bot tends to attack lonely enemies and run away from groups of enemies
$CS_{6,6}$: Ability to represent a flow of integrated percepts including self-status	$BP_{6,6}$: The bot autonomously generates a sequence of high-level representations that summarizes current situation. For instance, "I am being attacked and I can't get rid of my enemies"
$CS_{7,1}$: Ability to maintain a model of self and a second order representation of the relation between the self and perceived game action	$BP_{7,1-6}$: The behaviour of the bot indicates that a sense of self is present. Decisions are not taken just as a function of player state (health, ammo, etc.), but based on a rich model of self which constitutes the basis for Theory of Mind capabilities [13]. The bot is able to recognize itself and the consequences of its own actions. In other words, a sense of agency is developed. Possible behaviour tests include mirror test derivatives as discussed in [8]. Behaviour is also modulated by the ability of the bot to foresee (to imagine or internally simulate) the emotional outcome of a planned action. Therefore, new behaviours appear as a result of advance planning mechanism including imagination. For instance, the bot develop new strategies to attack enemies that have not been learned using reinforcement but imagination.
$CS_{7,2}$: Ability to maintain an analogous second order representation of the relation between the self and bot actions	
$CS_{7,3}$: Ability to maintain a second order representation of the relation between feelings and self	
$CS_{7,4}$: Ability to self-recognize as a player in the game	
$CS_{7,5}$: Ability to make plans including the model of self as an actor	
$CS_{7,6}$: Ability to imagine the outcome of planned actions in terms of self	
$CS_{7,7}$: Ability to use existing game objects as tools (note that support for using weapons and vehicles is native in the game, so their usage cannot be regarded as a bot cognitive capability)	$BP_{7,7}$: The bot manages to use some object as a means to achieve its objectives. For instance, using a movable object, like a box or a barrel, as an improvised shield

Table 8.2 (continued)

Cognitive skills ($CS_{i,j}$)	Behavioural patterns ($BP_{i,j}$)
$CS_{7,8}$: Ability to represent and self-report mental content (continuous inner flow of percepts)	$BP_{7,8}$: The bot internally plays a "movie" as a sequence of percepts representing the events taking place in the surroundings, including own body actions. That movie is also used as input for decision making
$CS_{8,1}$: Ability to model other players as intentional selves	$BP_{8,1}$: As other players are identified and modeled as selves, their movements can be predicted. The bot put itself in the place of another player to predict next actions of an opponent. Then, the behaviour of the bot is shaped not only according to present sensory data but also using opponent's predicted movements. For instance, the bot predicts the possible escape path of an enemy and make the necessary moves to block it
$CS_{8,2}$: Ability to learn from other players by imitation	$BP_{8,2}$: As the bot can manage both the model of self and models of others, it can also establish analogies and learn strategies by observing other bots. For instance, the bot can acquire new attack strategies developed by human players participating in the same game
$CS_{8,3}$: Ability to collaborate with other players to get better scores	
$CS_{8,4}$: Ability to make plans including the models of other players as actors in the plans (intersubjectivity)	$BP_{8,3-4}$: Social behaviours like forming groups that collaborate in firefights
$CS_{8,5}$: Ability to build new tools than can be used to achieve game goals	$BP_{8,5}$: The bot combines several objects in order to build a new compound object that can be used either for defense or attack. For instance, building an improvised barricade made of a number of objects arranged along a line
$CS_{8,6}$: Inner imagery is enriched with mental content related to the model of others	$BP_{8,6}$: The bot keeps an updated models of other players and use these models to build a richer inner imagery representing current world state. For instance, "I'm attacking player XYZ, who is usually scared of me"
$CS_{9,1}$: Ability to develop Machiavellian strategies as part of the game play	$BP_{9,1}$: The bot is able to reason about opponents' Theory of Mind, i.e. "I know you know I know". Therefore, it shows social intelligent behaviours like preparing an ambuscade
$CS_{9,2}$: Learning of new Machiavellian strategies	$BP_{9,2}$: The bot learns new Machiavellian strategies that have not been pre-programmed, for instance new ways of deceiving enemies

Table 8.2 (continued)

Cognitive skills ($CS_{i,j}$)	Behavioural patterns ($BP_{i,j}$)
$CS_{9,3}$: Ability to report mental content	$BP_{9,3}$: The bot uses game's inbuilt chat system to coherently report its inner mental state
$CS_{9,4}$: Ability to form cultural groups	$BP_{9,4}$: Behavioural profiles associated with culture would require more complex environments. However, clues of cultural organization might be observed in groups of organized bots
$CS_{9,5}$: Ability to modify the environment to serve bot's needs	$BP_{9,5}$: Like in $BP_{9,4}$ complex behaviours associated with these skills requires more complex environments. Nevertheless, bot autonomously using movable objects of the map to build a secured base would be a hallmark of this skill
$CS_{10,1}$: Ability to produce accurate verbal report	$BP_{10,1}$: The bot generates complex linguistic messages using correct grammar and accurate meaning
$CS_{10,2}$: Ability to pass an FPS adapted version of the Turing Test	$BP_{10,2}$: The bot will pass an adapted Turing Test, like the one proposed in the BotPrize competition [9]. Also classical Turing Test-using game chat could be passed
$CS_{10,3}$: Ability to develop civilizations and technology	$BP_{10,3}$: Like in $BP_{9,4}$ complex behaviours associated with these skills requires more complex environments
$CS_{11,1}$: Ability to manage several streams of consciousness	$BP_{11,1}$: For instance, the bot is able to keep a complex explicit conversations over the chat with human beings (all passing the Turing Test, and fight enemies effectively at the very same time. All conscious threads share knowledge, i.e. parallel chats could comment on the ongoing deathmap game)

be more believable as its cognitive development level is higher from the point of view of *ConsScale*, but it does not need to reach levels 9 or 10 for optimal believability.

Another issue related with the assessment of believability using *ConsScale* is the existence of a dynamic level of cognitive development in bots. As learning processes take place over time, the same bot can be expected to show different developmental stages over time, i.e. different *ConsScale* cognitive profiles over time. Taking this into account, *ConsScale* can also be used to assess the learning progression of a bot towards human-like cognitive capabilities.

8.4 Believability and the Degree of Cognitive Development

In this section we explore the possible correlations between the degree of cognitive development of a bot—as defined in *ConsScale* FPS—and the level of believability assessed by a human observer. Our initial working hypothesis is that high degrees of cognitive development should correlate with high ratios of believability. The main point here is that believability is usually assessed by a human observer, who is not explicitly checking for the behavioural patterns defined in *ConsScale* FPS. Nevertheless, we argue that humans naturally perform a covert recognition of these patterns, thus identifying human-like behaviour when they perceive the kind of behaviours described in *ConsScale*.

In order to shed some light on the former hypothesis we have used the main results from the third edition of the 2K BotPrize competition,[3] held in August 2010 in Copenhagen, at the IEEE Conference on Computational Intelligence and Games. First of all, we have analyzed the cognitive profiles of the best three entries in this competition, and we have compared their degree of cognitive development—according to *ConsScale* FPS—with their degree of humanness—according to the human judges. Additionally, we have tried to understand the assessment process carried out by judges during the competition with the aim to discover what specific skills they were hoping to find in a human-like bot. Finally, we have compared the collected data with the current definition of *ConsScale* FPS, looking for important skills that might be missing, or skills already considered in the scale but hardly appreciated by judges.

8.4.1 Cognitive Profile of State of the Art FPS Bots

We have analyzed a small but significant sample of three FPS bots using the *ConsScale* FPS framework. As mentioned above, the selected bots are the best three entries from the BotPrize 2010 competition: ICE-2010 (third place), UT^2 (second place—see Chap. 6), and CC-Bot2 (first place—see Chap. 7).

[3] http://botprize.org/

ICE-2010 bot is based on ICE-2009 and ICE-UT@RITS [10] (second place in BotPrize 2009 and BotPrize 2008 respectively). This bot uses NeuroEvolution of Augmenting Topologies (NEAT) for selecting weapons and implementing two different attack modes. Training of NEAT was performed using fuzzified input features obtained from human play logs.

The UTˆ2 bot is endowed with a battle controller based on multiobjective neuroevolution (using genetic algorithms to evolve neural network controllers). It also uses human trace data to replay short relative paths when it needs to get unstuck (see Chaps. 5 and 6).

CC-Bot2 is based on the CERA-CRANIUM cognitive architecture [3]. It implements several basic cognitive functions, like an attention mechanism and a shared short term memory system that permits the generation of percepts and simple behaviours (see Chap. 7).

Figure 8.5 shows the *ConsScale* cognitive profile of these three bots. All the bots are classified as level 2 agents (*reactive*) because none of them completely fulfil level 3. The CQS value for the three bots is very low, slightly over 0.18, which is the CQS of a pure reactive agent; and rather far from 2.22, the CQS of a pure level 3 agent (*adaptive*). ICE-2010 has the higher CQS (0.26) because it has the highest degree of achievement in level 3. However, it does worse in immediately higher levels. The reason why its CQS is slightly higher is because the *ConsScale* quantitative measure rewards those agents that follow the roadmap proposed by the scale, i.e. fulfilling lower levels in the first place, so higher functions can benefit more from inter-function synergies (see Sect. 8.2.3). Looking at the three cognitive profiles (see Fig. 8.5) it is clear that UTˆ2 and CC-Bot2 are better than ICE-2010 at levels 4, 5, and 6. This would explain the BotPrize results if judges focused more on the behavioural patterns associated with these particular levels (see Sect. 8.4.2).

Looking at the comparison of the three cognitive profiles, it is also interesting to note that all bots have very similar profiles (nearly identical in the case of UTˆ2 and CC-Bot2). This is also consistent with BotPrize results, where the humanness ratio of these bots differed by an average of 5.6 %.

In order to analyze in detail the possible correlations between *ConsScale* evaluation and BotPrize judging we have to look at the L_i scores. Although the CQS

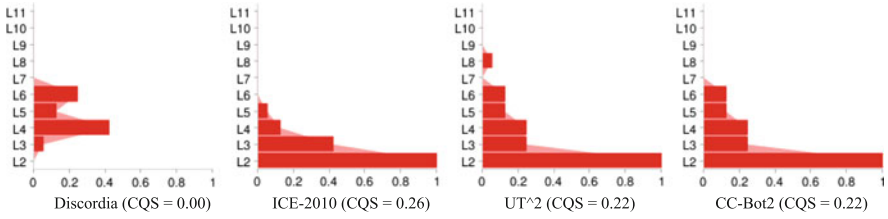


Fig. 8.5 *ConsScale* FPS cognitive profile of the best four entries of the 2K BotPrize Competition 2010

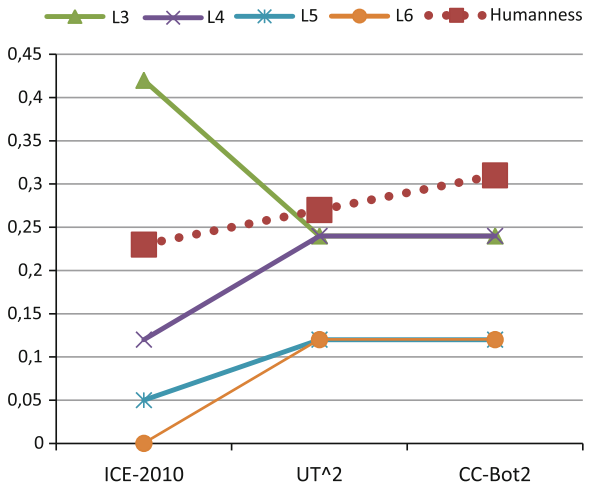


Fig. 8.6 Joint representation of the degree of fulfilment of *ConsScale* levels 3 (L_3), 4 (L_4), 5 (L_5), and 6 (L_6) and the ratio of humanness in the BotPrize competition of bots ICE-2010, UT^2 , and CC-Bot2

represents an integrative and unified measure, its exponential nature (see Fig. 8.1) would distort the correlation analysis. Taking into account that in this case significant L_i scores are between levels 3 and 6, we have focused on these particular levels (see Fig. 8.6).

When comparing the degree of fulfilment of *ConsScale* levels 3–6 and the BotPrize results, it seems to be some correlation between perception of humanness and the appearance of behavioural patterns associated to these levels. Although ICE-2010 bot outperforms the others in level 3, its results in higher levels are poorer. In fact, apart from the direct correlation that can be observed for levels 4, 5 and 6, according to the overall figures, humanness positively correlates with average fulfilment of all *ConsScale* levels. In order to obtain a clearer view, Fig. 8.7 shows the average fulfilment of the bots at levels 3–6 jointly with their humanness ratio. Nothing specific can be said about higher levels because the bots being studied barely fulfil any cognitive skill above level 6.

In general, from the point of view of *ConsScale*, there is a lot of work to do in order to develop cognitively advanced bots. Specifically, we think the typical expectations of human players are located within levels 3–6. Nevertheless, the specific expectations to be taken into account in assessment tasks like the BotPrize competition are those of the competition judges. This point is discussed in the next section.

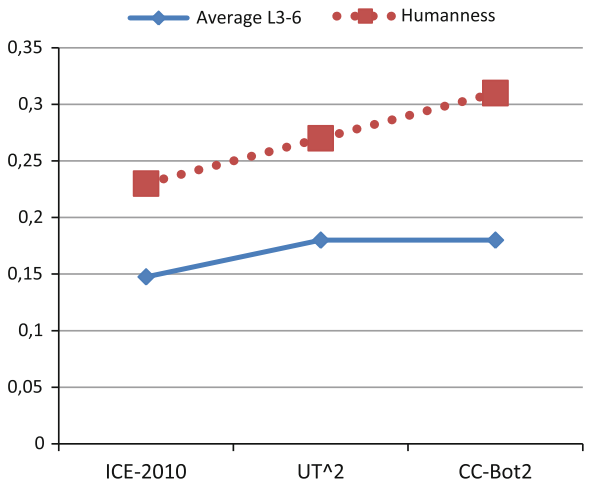


Fig. 8.7 Joint representation of the average degree of fulfilment of *ConsScale* levels 3–6 and the ratio of humanness in the BotPrize competition of bots ICE-2010, UT^2 , and CC-Bot2

8.4.2 Recognition of Human-Like Behaviour

When judges are asked regarding the list of behavioural profiles described in *ConsScale* FPS, they generally think they are interesting and appropriate, but difficult to judge in most of the cases. In the particular case of the BotPrize 2010 competition, it is hard to judge most of the subtler points at the same time that the human player (and also judge) needs to be running around and shooting while trying to avoid being shot.

In order to understand what specific skills judges might be looking at, we have asked them to identify the behavioural patterns from the *ConsScale* FPS list that they took into account during the assessing task. Figure 8.8 shows an integrated cognitive

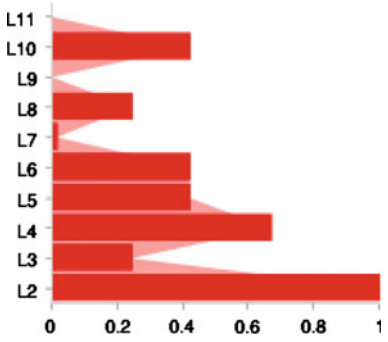


Fig. 8.8 This *ConsScale* cognitive profile represents all cognitive skills considered by at least one judge in the BotPrize 2010 competition

profile that combines the feedback obtained from a sample of BotPrize competition judges.

As represented in Fig. 8.8, human judges do not recognize humanness as a *ConsScale* level 10 agent. In fact, their requirements for human-like behaviour are much lower. Interestingly, although they consider some level 10 skills, they seem to focus on levels 4, 5, and 6. This explains why UTˆ2 and CC-Bot2 bots obtained the best results of the competition (see Fig. 8.5).

8.5 Conclusions

Believability assessment is completely based on judgments from human observers. Therefore, obtaining information about how exactly they perceive humanness is very important for the design of believable bots. However, human-like behaviours recognition is usually a covert and complex mental process, which can be difficult to discover and translate into directly usable rules. In this context, having a tool like *ConsScale* FPS might be helpful in two different ways: on one hand, the list of cognitive skills and associated behavioural patterns described in *ConsScale* can help the human observer to point out what aspects are more important for believability; on the other hand, the cognitive hierarchy established in *ConsScale* can be considered a roadmap for the design of human-like bots.

Using *ConsScale* as reference framework, we argue that the idea of cognitive integration, based on the definition of cognitive dependencies, is the key to the successful design of more human-like bots. There is no point in directly trying to simulate high cognitive abilities if we do not identify their lower cognitive requirements in the first place.

The analysis carried out in this chapter clearly indicates that much more work still needs to be done in the domain of believability. From the point of view of *ConsScale*, progressing on to the development of level 3 and level 4 bots seems to be the right way forward.

Additionally, *ConsScale* might be instantiated and adapted to other game genres, like role playing or life simulation games, where believability could be assessed in a different way.

Finally, typical human error-making is an interesting issue, which has not been yet addressed in *ConsScale*, and seems to be an important aspect for believability. Finding the typically human balance between perfect accuracy and too poor performance during game play is also an open problem.

Acknowledgments We wish to thank Igor Karpov, Jacob Schrum, and Risto Miikulainen from the University of Texas, Austin for providing feedback about the UTˆ2 bot; Akihiro Kojima, Daichi Hirono, Takumi Sato, Seiji Murakami, and Ruck Thawonmas from Ritsumeikan University, Japan for providing feedback about the ICE-2010 bot; and Casey Rosenthal and Clare Bates Congdon from the University of Southern Maine for providing feedback about the Discordia bot. We are also indebted to Philip Hingston, Mike Preuss, and Simon Lucas for their feedback about the BotPrize judging protocol.

References

1. Aleksander, I., Dunmall, B.: Axioms and tests for the presence of minimal consciousness in agents. J. Conscious. Stud. **10**(4–5), 7–18 (2003)
2. Arrabales, R., Ledezma, A., Sanchis, A.: Assessing and characterizing the cognitive power of machine consciousness implementations. In: Biologically Inspired Cognitive Architectures II, AAAI Fall Symposium Series, vol. 1, pp. 16–21 (2009)
3. Arrabales, R., Ledezma, A., Sanchis, A.: CERA-CRANIUM: a test bed for machine consciousness research. In: International Workshop on Machine Consciousness (2009)
4. Arrabales, R., Ledezma, A., Sanchis, A.: Establishing a roadmap and metrics for conscious machines development. In: Baciu, G., Wang, Y., Yao, Y.Y., Kinsner, W., Chan, K., Zadeh, L.A. (eds.) Proceedings of the 8th IEEE International Conference on Cognitive Informatics, pp. 94–101 (2009)
5. Arrabales, R., Ledezma, A., Sanchis, A.: The cognitive development of machine consciousness implementations. Int. J. Mach. Conscious. **2**(2), 213–225 (2010)
6. Arrabales, R., Ledezma, A., Sanchis, A.: ConsScale: a pragmatic scale for measuring the level of consciousness in artificial agents. J. Conscious. Stud. **17**(3–4), 131–164 (2010)
7. Damasio, A.R.: The Feeling of What Happens: Body and Emotion in the Making of Consciousness. Heinemann, London (1999)
8. Haikonen, P.O.A.: Reflections of consciousness: the mirror test. In: Proceedings of the 2007 AAAI Fall Symposium on Consciousness and Artificial Intelligence, pp. 67–71 (2007)
9. Hingston, P.: A Turing Test for computer game bots. In. IEEE Transactions on Computational Intelligence and AI in Games, vol. 1, pp. 169–186 (2009)
10. Hirono, D., Thawonmas, R.: Implementation of a human-like bot in a first person shooter: second place bot at BotPrize 2008. In: Asia Simulation Conference (2009)
11. Neisser, U.: Cognitive Psychology. Meredith, New York (1967)
12. Turing, A.: Computing Machinery and Intelligence. Mind, London (1950)
13. Vygotsky, L.S.: Mind in Society: The Development of Higher Psychological Processes. Harvard University Press, New York (1980)

Chapter 9
Assessing Believability

Julian Togelius, Georgios N. Yannakakis, Sergey Karakovskiy and Noor Shaker

Abstract We discuss what it means for a non-player character (NPC) to be believable or human-like, and how we can accurately assess believability. We argue that participatory observation, where the human assessing believability takes part in the game, is prone to distortion effects. For many games, a fairer (or at least complementary) assessment might be made by an external observer that does not participate in the game, through comparing and ranking the performance of human and non-human agents playing a game. This assessment philosophy was embodied in the Turing Test track of the recent Mario AI Championship, where non-expert bystanders evaluated the human-likeness of several agents and humans playing a version of *Super Mario Bros*. We analyze the results of this competition. Finally, we discuss the possibilities for forming models of believability and of maximizing believability through adjusting game content rather than NPC control logic.

9.1 Introduction

What exactly is believability in a game, what is it good for and how can it be achieved? These are complex but important questions that we will not claim to be able to fully answer. However, in this chapter we will address all these questions from the

J. Togelius (✉) · G. N. Yannakakis · N. Shaker
IT University of Copenhagen, Rued Langaards Vej 7, 2300 Copenhagen, Denmark
e-mail: juto@itu.dk

G. N. Yannakakis
e-mail: yannakakis@itu.dk

N. Shaker
e-mail: nosh@itu.dk

S. Karakovskiy
St. Petersburg State University, Universitetskii prospekt 35,
Petergof, Saint-Petersburg 198504, Russia
e-mail: sergey@marioai.com

P. Hingston (ed.), *Believable Bots*, DOI: 10.1007/978-3-642-32323-2_9,

perspective of *believability assessment*, i.e. how we can accurately judge the believability of a game character. As believability is a fundamentally phenomenological construct, we believe an assessment-based perspective to be appropriate for shedding light on the nature of believability.

In the following, we analyze believability in the context of games, and reason about the complexity of assessing believability and the concerns behind different approaches. We argue that in many cases, believability is better assessed from a third-person perspective rather than a first-person perspective, i.e. where the assessor is not a participant in the game. As an example of third-person believability assessment, we report the results of the Mario AI Turing Test, which was carried out during the Asia Games Show (in collaboration with the IEEE Games Innovation Conference) using uninformed observers as assessors. The discussion about assessing believability is then used to inform a further discussion about how to achieve believability; we propose a solution based on modelling and optimization, analogous to an approach that has been used successfully to maximize player experience. Throughout the chapter we will illustrate the concepts we discuss through how they apply to *Super Mario Bros*, Nintendo's seminal platform game from 1985.

9.2 What Is Believability?

As far as we can tell, there is no generally agreed or precise definition of believability. Instead, we have a family of related meanings denoted by the same word, somewhat similar to the situation for the word "intelligence". In trying to identify these meanings, we can start with the obvious linguistic fact that believability means that something can be believed by someone. As we are talking about believability in the rather restricted domain of computer game bots or characters, we can add that something about the character can be believed by someone. We might constrain ourselves further by adding the "is real" or "is plausible" to the equation, so that we get "someone believes that some character or bot is real". This leaves us with two broad classes of examples:

- *Character believability*: Someone believes that the character/bot *itself* is real, i.e. an actual living being (or actual autonomous robot etc.)
- *Player believability*: Someone believes that the *player* controlling the character/bot is real, i.e. that a human is playing as that character instead of the character being computer-controlled.

Character believability implies a very high degree of realism; characters can be "realistic" in certain respects (such as textures, movement patterns and dialogue) without having any notable character believability. Viewed this way, character believability currently seems to be reserved for big-budget non-interactive movies (e.g. Hollywood productions such as *Lord of the Rings* where certain entirely computer-generated characters look as real as if they had been human actors in a

costume). Hardly any games, or interactive media of any sort, can reasonably aspire to character believability within this technical generation. On the way towards true character believability lurks the well-known problem of the *uncanny valley*, much discussed in humanoid robotics research: almost, but not completely, believable characters tend to be "creepy" [1] and elicit negative emotions in humans. A number of studies has investigated and, in part, confirmed the uncanny valley theory in virtual characters within games [2, 3] while a large volume of work has focused on the impact of virtual character detail on players' self-reported presence [4] or observers' impressions of character facial images [5]. We do not know whether the uncanny valley phenomenon exists for player believability as well, and there have to our knowledge not been any thorough investigations of this, but some arguments have been made that the phenomenon does indeed exist [6]. The discussion in the rest of the paper will chiefly apply to player believability rather than character believability.

Player believability presupposes that the observer knows that the character on the screen is not real – that it is just a graphical representation of digital processes inside a computer. (This is certainly the case in Super Mario Bros.) Importantly, this means that most aspects of animation and graphics rendering are unimportant for player believability. However, the observer believes that a human has an ongoing input to and (at least partial) control over these processes, and that the human's control is interactive in the sense that the human is aware of what the character is doing in the game.

There are many games and bots that are not designed to be believable, neither in the sense of character believability nor player believability. To begin with, there are game genres completely devoid of NPCs (e.g. puzzle games). There are also games where characters are intended to be predictable and "robotic" in their actions, for example many shooting gallery games and platform games. For example, in Super Mario Bros the most common NPC creatures move with constant speed along a platform and reverse direction or simply fall down when reaching the end of the platform. In other games, there is little chance for the player to assess believability because each NPC is typically only encountered for a few seconds, and then in the middle of a rather chaotic situation; this is the case for the single-player campaign of many FPS games. In general, the game design has considerable influence on the believability of NPCs: either the design showcases the bot and its AI or the game is designed so that the bot's stupidity (or non human-likeness) is absorbed.

Even when the player has a fair chance to observe an NPC for some time and assess its believability, it is clear that player believability is harder to achieve in some games than others. One important factor for the difficulty of achieving player believability is the amount and type of information required from the player (human or algorithm) to effectively play the game. As an example of the importance of the amount of information, chess only requires a few bits of information transferred from the player every turn to select which piece to move to which position (a low-bandwidth channel in the vocabulary of information theory) whereas a first-person shooter such as *Unreal Tournament* requires the player to continuously move the mouse with one hand and tap keys on the keyboard with the other, often performing several actions per second. The much higher communication bandwidth between player and game goes some

way to explain the relative difficulty of creating believable bots for FPS games. In terms of control bandwidth, Super Mario Bros has more in common with the typical FPS than with a board game; although relatively few actions are available to the player at each moment, a player during normal gameplay presses more than one key a second.

As an example of the influence of the type of information required, creating a believable bot for a complex strategy game such as *Civilization* is hard, although there are moderately successful attempts; however, creating a believable bot for a digital version of the strategy game *Diplomacy* is orders of magnitude harder, even though the action space of the latter game is smaller [7]. The key difference here is that text-based (or verbal) communication between players is crucial in Diplomacy, whereas it is often not even possible in Civilization. In terms of the type of information supplied by the player, Super Mario Bros should be relatively easy to achieve character believability in: the player controls the character through a combination of five discrete button presses (left, right, down, A, B). No text input nor continuous input is possible.

Another important question is "believable for whom"? In general, an experienced player will have a much easier time distinguishing between a human-controlled and a computer-controlled character. This is due both to the experienced player having better knowledge about the rules and possible actions in this particular game, or in games in general, and to the experienced player having a larger (though often implicit) knowledge of the patterns of actions exhibited by the artificial intelligence routines found in the game at hand, or in games in general.

It is clear that non-player characters with player believability can bring major advantages for a game. Many games become much more engaging for players who believe that they are playing against fellow human players. Several authors have argued that the appearance of human intelligence and overall human-likeness that adds value to a computer controlled character and overall, increases the quality of gameplay [8, 9]. There seems to be multiple reasons for why games against humans become more engaging, including the belief that humans are less predictable than computers, that the player through gameplay is capable of inflicting real joy or disappointment in other humans, and the sense of having company in what one does.

Believability—similar to emotions such as frustration and cognitive processes such as attention—is an artificial construct with fuzzy boundaries. These properties make the assessment and computational modelling of believability far from trivial. Before discussing the complex nature of detecting and assessing believability in Sect. 9.3 we will briefly analyze the relationship between player emotions and believability.

9.2.1 Believability, Player Emotions and Cognition

Game believability is a critical subcomponent of player experience. It can be linked to a stream of player emotions, which may be active simultaneously, usually triggered

by events occurring during gameplay but also related to cognitive and behavioural process during gameplay. Games incorporating believable elements, such as bot behaviour, can elicit particular emotional responses to a player which in turn may affect the player's attention level and gaze patterns, reflect on the player's facial expression and even cause bodily alterations (i.e. the player's physiology).

Research in game artificial intelligence is based on several empirical assumptions about believability, human cognition, human-machine interaction, player satisfaction and fun. The primary hypothesis of most studies in the literature is that the generation of believable, human-like opponents [10] leads to increased player enjoyment. While there are indications to support such a hypothesis (e.g. the vast number of multi-player on-line games played) and research endeavours to investigate the relationship between believability of non-player characters (NPCs) and satisfaction of the player [11], there has been no clear evidence that human-like NPC behaviours generate more appealing games.

We need to make clear that we do not necessarily assess player satisfaction or other emotional states by assessing believability; the relationship between the two is complicated, and while it would be interesting and important to clarify it, this is beyond the scope of the current paper.

9.3 Assessing Believability

Believability of character behaviour may be viewed as part of player experience. Player experience [12] in general can be measured via reporting (*subjective*); via monitoring a player's physiological responses [13, 14], tracking a player's body, head and facial motion [15] during play (*objective*); or via logging gameplay statistical data that embed behavioural responses of player experience (*gameplay-based*) [16]. On the same basis, one could attempt to adopt any of the three above-mentioned approaches (or even combinations of those) to capture believability within games.

In this paper we focus on the *subjective* approach for assessing believability and we question whether a player can reliably assess believability his/herself and, in part, question existent player testing schemes.

9.3.1 Subjective Assessment

The most direct way to assess believability is to ask the players themselves about their experience when they face or presented with opponents that need to be assessed [17]. Subjective believability assessment can be based on either players' *free response* during play or on *forced* data retrieved through questionnaires.

Naturally, free-response answers may contain richer information about one's believability notion but are hard to analyze appropriately. An experiment designer may decide to annotate the derived text or verbal response into specific critical words

or phrases which can then be mapped to believability. However, doing so requires strong assumptions about the validity and the importance of the text/speech clusters identified, and make it hard to automate the assessment. On the other hand, forcing game experiment subjects to report believability through a questionnaire, constraining them into specific questionnaire items, yields data that can be easily used for analysis.

Subjective assessment may yield very accurate models of self-reported believability; however, there are quite a few limitations embedded in this approach. First, there is usually significant experimental noise in the responses of subjects; this may be caused by subject learning effects (the subject might find it easier to spot bots after having seen a few) and self-deception. Second, self-reports can be intrusive if questionnaire items are presented during the gameplay sessions [18, 19]. Third, they are sensitive to subjects' memory limitations if players are asked to express their experience after a lengthy game session (post-experience effect).

Numerous studies have shown that self-reports can guide machine learning algorithms for successfully capturing aspects of player experience in prey/predator [20], physical interactive [21], platform [16, 22] and racing [23] games. We argue that similar approaches can be used for the efficient capture of believability within games.

9.3.2 When to Ask?

While efficient methods for minimizing learning effects and self-deception effects have been proposed [24], there is no universally accepted time window within which subjects should be asked to express the level of believability of an NPC. Such a time window should result in a self-reporting process that is both as unobtrusive as possible and suffering from minimal post-experience effects.

Reporting on paper or on a digital questionnaire sheet is the most popular approach to subjective assessment for player experience, either interrupting the subject during gameplay or at the end of a game session. It is straightforward to extend this method to believability assessment. However, a recent attempt on believability assessment, the *2k BotPrize* [25] has focused on making the assessment process part of the game itself. In that study, subjects played a first-person shooter game and were equipped with a special weapon that could be used to distinguish between an AI bot and a human opponent. As a pioneering approach to assessing believability, the BotPrize has received considerable attention, and rightly so. While not directly a subjective approach for believability assessment the innovation of that study is that an in-game element (and not an external questionnaire) defines the platform for the assessment of opponent believability during play. Such an approach initially appears to minimize report intrusiveness. Moreover the ideal interaction time window is set by the player his/herself—the window depends on the interaction time a player spends with particular opponents—bypassing that key protocol design decision.

We argue that in a game setting—such as the FPS scenario of [25]—the believability of opponents cannot be detached from player experience. Thus, assessment during play may turn out to be highly intrusive for both player experience and the assessment of players, while gameplay statistics collected this way may contain experimental artefacts that are difficult to detect and correct for.

Results obtained from the 2010 2k BotPrize during the 2010 IEEE CIG conference corroborate our hypothesis. In particular, it was apparent that some players focused on the task of believability assessment while others focused more on gameplay. Believability assessment done this way introduces a new game mechanic that may appeal to some players. The result was that a human player was characterized as the least human player (even less human than all AI opponents) in the game since he had adopted a strategy seeking to excel to that game mechanic. Some judges raised complaints that the experiment was neither a game (the game was way too intrusive to elicit genuine gameplay experience) nor a proper experimental protocol (since some judges aimed to excel in appearing less believable), but rather a hybrid between the two. Our hypothesis is that during-play believability assessment entails protocol design flaws that do not allow for a reliable evaluation of gameplay believability. Thus, when designing the Turing Test track of the Mario AI Championship (our own attempt at believability assessment) we chose to deviate from the approach taken in the BotPrize and use a protocol where judges were asked questions only after a level had been played.

9.3.3 How to Ask?

Forced questionnaires could vary from simple tick boxes to multiple choice items. Both the questions and the answers provided could vary from single words to sentences; even though, generally, short and clear question-and-answer items are preferred since lengthy questionnaire items may challenge short-term memory and cognitive load of the subject. Again taking our cue from methods for assessing player experience, one could identify three types of forced questionnaires for believability assessment:

1. Boolean: subjects have a single boolean answer choice (e.g. *is this believable?*, or, *is this a human playing?*). While this question type is direct and clear, it does not provide with rich information for further analysis.
2. Ranking: subjects are asked to answer questionnaire items given in a ranking/scaling form (e.g. *how believable was that?*). For the use of similar questionnaires in player experience assessment, see [14, 18, 19].
3. Preference: subjects are asked to compare the believability level of two or more sessions of the game (e.g. *which one was more believable?*). For the use of similar questionnaires in player experience assessment, see [23, 24, 26].

There is no single universally accepted approach for questionnaire type even though preliminary results in various user studies suggest that there is an inconsistency

between the three questionnaire types when used for assessing player experience. The main disadvantage of ranking questionnaires is that they do not control for the subjective notion of believability across subjects. On the other hand, pairwise preference and boolean questions can minimize subjects' subjective notions of scaling, allow a fair comparison between the answers of different subjects and, thereby, may assist towards a more accurate and subjective capture of believability. For simplicity, we opted for boolean questions in the Mario AI Championship. Being the first time we ran the competition and unfamiliarity with the venue, we decided that boolean questions minimized the risks of technical and/or linguistic glitches.

9.3.4 First Person Versus Third Person

Subjective assessment may consider both first person reports (self-reports) but also reports expressed indirectly by experts or external observers. Analogies of this relationship can be found in the comparison between self-expressed experience and annotated experience in affective computing studies [12]. While self-expressed experience comes with several limitations such as self-deception, increased gameplay cognitive and short-term memory load, annotated data (if in large sample sizes) can effectively encapsulate notions of player experience.

In first person assessment, the player is part of gameplay and the same gameplay is the elicitor of his/her player experience. As the player is engaged in playing and forms a vital component of the game-player interaction he/she thereby influences the degree of believability that emerges from that relationship. Thus, believability, gameplay and player experience are interconnected components of the interactive experience which are hard to separate from each other.

In third person assessment, on the other hand, the player does not play the game his/herself as she is the observer of the playing experience and the gameplay. While one could claim that an observer is not engaged in the *true* experience of gameplay believability (rather in a *quasi*-experience [27]) she is able to concentrate more on the assessment of believability via a higher cognitive focus on the task. Because of the aforementioned limitations of the first person approach our research hypothesis is that third person assessment may lead to more accurate believability assessment. We therefore chose to use third-person assessment in the Mario AI Championship.

It should be added that first-person believability assessment is only possible for games played by at least two players simultaneously. First-person assessment is not possible for a platform game such as Super Mario Bros, at least in the standard version where only a single player character plays at any time, nor for a puzzle game such as *Tetris* or many casual mobile games such as *Angry Birds*. Games for which first-person assessment is possible include FPS games such Unreal Tournament, used in the BotPrize, but also a large variety of strategy games, both real-time (RTS) and turn-based and sports games like racing games, not to mention classic non/digital board games like Chess. Third-person assessment, on the other hand, is possible for all game genres.

As an anecdotal example of the perils of first-person assessment, consider the famous 1997 chess match between reigning world champion Garry Kasparov and the Deep Blue hardware/software. After losing the match, Kasparov complained that the program was playing in a suspiciously human-like manner [28]. It is very likely that the subject could have been affected by one of the many limitations of self-expressed first-person believability assessment, such as self-deception.

9.3.5 Time Required for Assessment

What is the optimal (or minimal) time interval for a user to identify believability? Players in a game have differing perceptual capabilities and cognitive responses. One has to take into account such experimental effects when designing a protocol for assessing believability and to control for them. One way to eliminate the time factor as an artefact from any data collected is to design game sessions that would generate equivalent interaction times with the bot which is under believability assessment. Then players can be asked to express their believability preferences over different game interaction sessions of similar time windows.

The time required to assess believability is also clearly dependent on the game genre. For example, FPS game bots are hard to assess since they often only appear on screen for a few seconds, and therefore their interaction time with the player is often insufficient [25]. On the other end of the spectrum, opponents in RTS games are observed and interacted with for time windows of several minutes or tens of minutes. Believability is not only dependent on the game genre but also on the surrounding content of the bot; an experiment protocol designer needs to cater to this and control for the game content that is present. That in turn will have an influence in the time required to assess believability appropriately.

In the Mario AI Championship, we chose to let the judges observe the game for a duration of 20–30 s, corresponding to the time needed to complete (or fail to complete) a short level, and in the authors' opinion enough to get an idea of the playing style of the player.

9.3.6 The Representation of Believability

Believability is a conceptual construct in a similar way to any other user, cognitive or affective state with unclear boundaries [29]. Given the fuzzy boundaries of believability and the subjective nature of its notion the representation of it is of key importance: should believability be represented within questionnaires as a state or as a continuous value?

It is even possible that believability is best represented as more than one continuous value. For instance, the emotional dimensions of arousal and valence [30, 31] are used very often to represent emotional states with unclear boundaries in affective computing studies; believability could be represented in a similar fashion.

9.4 The Mario AI Championship: Turing Test Track

The Turing Test track of the Mario AI Championship was held during the Asia Games Show 2010[1] in conjunction with the 2010 IEEE Games Innovation Conference.[2] This section presents the competitors, the competition setup and the results obtained.

9.4.1 The Competitors

Five bots and one human player competed in the Turing Test track. The five bots were chosen from among the competitors in the Gameplay track of the competition (thus built to play the game as well as possible rather than in a human-like fashion) and the organizers' own experiments. The chosen bots were of varying sophistication, playing strength, and in particular exhibited different playing styles:

- Robin Baumgarten's A^* Agent: This agent is based on an A* search algorithm in state space and simulates the future trajectory of both itself and enemy NPCs for each considered actions. This agent runs through the levels, almost continuously jumping and shooting fireballs. Detailed information about this agent can be found in [32].
- Slawomir Bojarski's and Clare Bates Congdon's REALM Agent: A rule-based evolutionary computation agent that evolves rule sets based on abstract vocabulary of conditions and actions with an A* component to determine specific keystrokes for each high level action [33]. This agent exhibits a more human-like behaviour than the other agents by starting to jump before reaching the edge of a gap, attacking and avoiding enemies, grasping power-ups and moving in both directions.
- Forward Agent: A very simple heuristic agent that constantly runs left and jumps when it senses that it is in front of a gap or obstacle.
- Forward Jumping Agent: An even simpler agent, that constantly jumps while running left.

[1] http://www.asiagameshow.com/

[2] http://ice-gic.ieee-cesoc.org/2010/

Fig. 9.1 Image from the Mario AI: Turing Test track competition held during the Asia Game Show, 2010

- Erek Speed's Agent: Rule-based controller, evolved with a GA. Maps the whole observation space (22×22) onto the action space, resulting in a genome of more than 100 Mb.
- Human player: Along with the five agents, an extra recording from a human player who was not involved in programming any of the bots or the organization of the competition (Nikolay Sohryakov) was used. This human had a skill in the high intermediate range, and a non-exploratory playing style.

9.4.2 Competition Organization

Prior to the competition event, videos were recorded of the five AI contestants and the human player playing a short level of the competition version of Super Mario Bros. The videos of gameplay were presented to the audience of the Asia Games Show in a random order and the audience voted on whether the player was a human or an algorithm after each video was completed. Each of the 60 observers was asked to vote whether the Mario they just saw playing was controlled by a computer or a human. The "Not decided" option was also available. Each agent was shown at least twice, and the orderings between the agents were varied so as to prevent order effects. A photograph of the competition presenter and the general competition setup is presented in Fig. 9.1.

Table 9.1 Turing Test Track competition results

Super Mario Bros Player	Computer	Human	Not decided
Human player	32	22	6
REALM Bot (evolved ruleset)	30	17	13
Erek Speed (GA)	41	9	10
Robin Baumgarten (A*)	46	6	8
Forward Agent	48	6	6
Forward Jumping Agent	54	0	6

9.4.3 Competition Results

Table 9.1 presents the final results of the Realm competition. 60 persons attended the competition session and voted. The REALM bot was the winner of the competition since it managed to convince 17 of the observers that it was a human. The REALM bot is the bot that comes closest to the 22 "Human" vote baseline of the human player with only 5 votes away while Erek Speed's bot also did rather well gathering 9 human votes. It is worth noting that the human player got more "computer" votes than the REALM bot (32 and 30, respectively) since the REALM bot left 13 spectators undecided—7 more than the undecided observers for the human player.

The competition results illustrate the difficulty of assessing believability even in a game such as Super Mario Bros, with low control bandwidth, simple graphics and easy overview of the play area. While the human player got the most of the human votes those were only 22 out of 60. Given this preliminary experimental protocol it appears that the 3rd person assessment approach is appropriate since believability can be successfully assessed; however, results also show the subjective nature of believability and the complexity that arises when one attempts to assess it.

The majority of experiment observers classified the human player as an AI-controlled bot and, a few observers indicated that the A* bot and the forward agent are controlled by humans. While the forward jumping Mario agent does not appear to be believable, the forward moving agent and the A* bot can still mislead a few observers. So, how did this happen? The core mechanics of platform games promote simple forward moving behaviour combined with jumps when necessary; such a playing behaviour is often followed by average players of this game genre. However, a human player with relatively high skills has been used in this experiment, and this might be a possible explanation for mistaking it for an AI-controlled bot. On the other hand, the A* bot near-optimal performance resembles the behaviour of very few highly skilled platform game players. Agents mimicking any of those playing behaviours can apparently mislead a few observers and be assessed as believable. On the contrary, the forward jumping agent does not convince any observer of its human nature since such gameplay is rarely met in humans playing Super Mario Bros.

9.5 From Assessing to Modelling to Optimizing Believability

Once we have established a reliable measure of believability, we could use supervised learning techniques to create a model from game configuration to believability. This process would be completely analogous to previous work on computational modelling of player experience [12, 34]: a number of game configurations are presented to a set of users, the users judge their believability, and based on this data set (with believability assessments as target values) a model is inferred that predicts believability based on game configuration. The model could use any of several function representations, for example a multilayer perceptron or a decision tree, and be trained with e.g. neuroevolutionary preference modelling (in case of assessments being expressed as preferences) or more standard supervised learning algorithms (in case of scalar assessments). After this model is obtained, optimization algorithms can be used to tune the game so as to maximize predicted believability. In previous work, player experience has been optimized for Super Mario Bros through creating a model from in-game player behaviour and level design parameters to predicted player affective states (such as fun and frustration), and new levels thereafter evolved that maximized predicted player experience [16, 35].

A key design question then becomes how to meaningfully parameterize the game configuration, so that the parameters have bearing on believability and create a tractable search space for the optimization algorithm. The obvious candidate would be the control logic for the character that is to be made believable. In Super Mario Bros itself this would be the main character, Mario. A number of good Mario controllers have been developed and submitted the Gameplay track of the Mario AI Championship, including Robin Baumgarten's A*-based controller that won the 2009 edition of the competition, and Slawomir Bojarski's and Clare Bates Congdon's evolutionary rule-based agent that won the 2010 edition, and also won the Turing Test track of the championship as described above. Both of these have several parameters that could conceivably be optimized for believability (and other modifiers could be introduced, such as a probability of "stopping to think" for a while every now and then), given that a good evaluation function was available.

However, it might be worth considering investigating other alternatives as well. In his classic book *The Sciences of the Artificial*, Herbert Simon describes the complex path of an ant walking on the beach, noting that the ant itself to our best evidence has a very simple "control system", before asking whether the apparent complexity of the ant's path is due to the ant itself or the topology and distribution of objects on the beach [36]. Analogously, we may ask to what extent believable behaviour in an algorithm-controlled game agent comes about from the controller and to what extent it is a product of the environment. It seems entirely probable that optimizing the environment for believability, in conjunction with a sufficiently generic NPC controller, could be every bit as effective as optimizing the NPC controller itself. In Super Mario Bros, this could be done through optimizing the design parameters of the levels. It is conceivable that the best effects are reached through combining NPC controller and level design optimization.

9.6 Conclusion

This chapter has discussed a number of aspects of believability from the perspective of believability assessment. We have outlined a number of important choices to consider when assessing believability, and briefly discussed their pros and cons. Throughout the chapter, we have used the platform game Super Mario Bros as a running example, and discussed what it would mean for a Mario player to be believable, and how believability could be achieved and assessed in this context. We reported the design and results of the Turing Test track of the 2010 Mario AI Championship, which attempted to measure believability in this game, taking a number of design choices that differ markedly from the perhaps most well-known attempt at assessing bot believability, the 2k BotPrize. It is clear that there is much research left to do about how believability can be assessed, modelled and optimized, and its relation to other aspects of player experience. We intend to contribute to this discussion through running further iterations of the Mario AI Championship, improving the competition design using lessons we have learned from last year's competition.

Acknowledgments Thanks to all the participants of the Mario AI Championship: Turing Test track held in the IEEE GIC conference in Hong Kong, December 2010, sponsored by IDSIA in Lugano. This research was supported in part by the European Union FP7 ICT project *SIREN* (project number 258453) and by the Danish Research Agency project *AGameComIn* (project number 274-09-0083).

References

1. Masahiro, M.: Bukimi no tani (the uncanny valley). Energy **7**(4), 33–35 (1970)
2. Schneider, E.: Mapping out the uncanny valley: a multidisciplinary approach. In: ACM SIGGRAPH 2008 posters. SIGGRAPH '08, New York, NY, USA, ACM (2008) 33:1–33:1
3. Schneider, E., Wang, Y., Yang, S.: Exploring the uncanny valley with Japanese video game characters. In: Proceedings of the DIGRA Conference, pp. 546–549 (2007)
4. Vinayagamoorthy, V., Brogni, A., Gillies, M., Slater, M., Steed, A.: An investigation of presence response across variations in visual realism. In: Proceedings of the 7th International Conference on Presence (2004)
5. Seyama, J., Nagayama, R.S.: The uncanny valley: effect of realism on the impression of artificial human faces. Presence **16**(4), 337–351 (2007)
6. Hayward, D.: Uncanny AI: artificial intelligence in the uncanny valley. Gamasutra N/A (2007)
7. Kemmerling, M., Ackermann, N., Beume, N., Preuss, M., Uellenbeck, S., Walz, W.: Is human-like and well playing contradictory for diplomacy bots? In: Proceedings of the IEEE Symposium on Computational Intelligence and Games, pp. 209–216 (2009)
8. Champandard, A.J.: AI Game Development. New Riders Publishing, Indianapolis (2004)
9. Bateman, C., Boon, R.: 21st Century Game Design. Charles River Media, California (2005)
10. Freed, M., Bear, T., Goldman, H., Hyatt, G., Reber, P., Sylvan, A., Tauber, J.: Towards more human-like computer opponents. In: Working Notes of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment, pp. 22–26 (2000)
11. Taatgen, N.A., van Oploo, M., Braaksma, J., Niemantsverdriet, J.: How to construct a believable opponent using cognitive modeling in the game of Set. In: Proceedings of the Fifth International Conference on Cognitive Modeling, pp. 201–206 (2003)

12. Yannakakis, G.N., Togelius, J.: Experience-driven procedural content generation. IEEE Trans. Affect. Comput. **2**(3), 147–161 (2011)
13. Yannakakis, G.N., Hallam, J., Lund, H.H.: Entertainment capture through heart rate activity in physical interactive playgrounds. User Model. User-Adap. Inter. Special Issue: Affective Modeling and Adaptation **18**(1–2), 207–243 (2008)
14. Mandryk, R.L., Inkpen, K.M., Calvert, T.W.: Using psychophysiological techniques to measure user experience with entertainment technologies. Behav. Inf. Technol. (Special Issue on User Experience) **25**(2), 141–158 (2006)
15. Asteriadis, S., Karpouzis, K., Kollias, S.D.: A neuro-fuzzy approach to user attention recognition. In: Proceedings of ICANN, pp. 927–936. Springer (2008)
16. Pedersen, C., Togelius, J., Yannakakis, G.N.: Modeling player experience for content creation. IEEE Trans. Comput. Intell. AI Games **2**(1), 54–67 (2010)
17. Hingston, P.: A Turing Test for computer game bots. IEEE Trans. Comput. Intell. AI in Games **1**(3), 169–186 (2009)
18. Drachen, A., Nacke, L., Yannakakis, G.N., Pedersen, A.L.: Correlation between heart rate, electrodermal activity and player experience in First-Person Shooter games. In: In press for SIGGRAPH 2010, ACM-SIGGRAPH Publishers (2010)
19. Pagulayan, R.J., Keeker, K., Wixon, D., Romero, R.L., Fuller, T.: User-centered design in games. In: Jacko, J.A., Sears, A. (eds.) The HCI Handbook, Lawrence Erlbaum Associates, Mahwah (2003)
20. Yannakakis, G.N., Hallam, J.: Towards capturing and enhancing entertainment in computer games. In: Proceedings of the 4th Hellenic Conference on Artificial Intelligence, Heraklion 2006. Lecture Notes in Artificial Intelligence, vol. 3955, pp. 432–442. Springer (2006)
21. Yannakakis, G.N., Hallam, J.: Real-time game adaptation for optimizing player satisfaction. IEEE Trans. Comput. Intell. AI in Games **1**(2), 121–133 (2009)
22. Pedersen, C., Togelius, J., Yannakakis, G.N.: Modeling player experience in Super Mario Bros. In: Proceedings of the IEEE Symposium on Computational Intelligence and Games, pp. 132–139. Milan, Italy, IEEE (2009)
23. Tognetti, S., Garbarino, M., Bonarini, A., Matteucci, M.: Modeling enjoyment preference from physiological responses in a car racing game. In: Proceedings of the IEEE Conference on Computational Intelligence and Games, pp. 321–328. Copenhagen, Denmark (2010)
24. Yannakakis, G.N.: Preference learning for affective modeling. In: Proceedings of the International Conference on Affective Computing and Intelligent Interaction, pp. 126–131. Amsterdam, The Netherlands, IEEE (2009)
25. Hingston, P.: A new design for a Turing Test for bots. In: Proceedings of the: IEEE Conference on Computational Intelligence and Games, pp. 345–350. Copenhagen, Denmark, IEEE (2010)
26. Yannakakis, G.N., Hallam, J.: Towards optimizing entertainment in computer games. Appl. Artif. Intell. **21**, 933–971 (2007)
27. Walton, K.L.: Mimesis as make-believe. Harvard University Press, Cambridge (1990)
28. Kasparov, G.: The chess master and the computer. The New York Review of Books, New York (2010)
29. Calvo, R.A., Mello, S.D.: Affect detection: an interdisciplinary reveiw of models, methods and their applications. IEEE Trans. Affect. Comput. **1**(1), 18–37 (2010)
30. Feldman, L.: Valence focus and arousal focus: Individual differences in the structure of affective experience. J. Pers. Soc. Psychol. **69**, 53–166 (1995)
31. Russell, J.A.: Core affect and the psychological construction of emotion. Psychol. Rev. **110**, 145–172 (2003)
32. Togelius, J., Karakovskiy, S., Baumgarten, R.: The 2009 Mario AI competition. In: Evolutionary Computation (CEC), 2010 IEEE Congress on, pp. 1–8. IEEE (2010)
33. Bojarski, S., Congdon, C.B.: REALM: a rule-based evolutionary computation agent that learns to play Mario. In: Proceedings of the IEEE Symposium on Computational Intelligence and Games, pp. 83–90. Copenhagen, Denmark, IEEE (2010)
34. Yannakakis, G.N.: How to model and augment player satisfaction: a review. In: Proceedings of the 1st Workshop on Child, Computer and Interaction, Chania, Crete, ACM Press (2008)

35. Shaker, N., Yannakakis, G.N., Togelius, J.: Towards automatic personalized content generation for platform games. In: Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE'10), pp. 63–68. Palo Alto, CA, AAAI Press (2010)
36. Simon, H.: The Sciences of the Artificial. MIT Press, Cambridge (1969)

Chapter 10
Creating a Personality System for RTS Bots

Jacek Mańdziuk and Przemysław Szałaj

Abstract Bots in Real Time Strategy games often play according to predefined scripts, which usually makes their behaviour repetitive and predictable. In this chapter, we discuss a notion of personality for an RTS bot and how it can be used to control a bot's behaviour. We introduce a personality system that allows us to easily create different personalities and we discuss how different components of the system can be identified and defined. The process of personality creation is based on several traits, which describe a general bot's characteristics. It allows us to create a wide variety of consistent personalities with the desired level of randomness, and, at the same time, to precisely control a bot's behaviour by enforcing or preventing certain strategies and techniques.

10.1 Introduction

Real Time Strategy (RTS) is a very popular genre of computer games, which is based on constructing buildings and creating an army that will be led to defeat the opponent. The games mostly take place in a 2D environment consisting of different area types and containing resources, such as gold, crystals, wood or food. Players try to take control over the resources, which are used to build new units and facilities. Moreover, in the majority of games some inventions and upgrades have been introduced, which affect features of game objects, for example by increasing their battle efficiency or creating items that have been unavailable before. Some of the most popular RTS games are Starcraft [1] and the Age of Empires series [2].

J. Mańdziuk (✉) · P. Szałaj
Faculty of Mathematics and Information Science, Warsaw University of Technology,
Plac Politechniki 1, 00-661 Warsaw, Poland
e-mail: mandziuk@mini.pw.edu.pl

P. Szałaj
e-mail: szalajp@mini.pw.edu.pl

P. Hingston (ed.), *Believable Bots*, DOI: 10.1007/978-3-642-32323-2_10

Most games offer both single player mode, in which a player competes against computer rivals—bots—and a multiplayer mode, which is a contest among people only. Since LAN networks and the Internet became popular, multiplayer games gained in popularity. Nowadays, most people prefer multiplayer mode, using single-player mode only in case of a lack of human players available or for practice before entering multiplayer games [16]. Such a situation is caused mostly by the poor performance of bots' AI (Artificial Intelligence). While improvement of AI techniques and growth of computational power of computers allowed us to build AI systems capable of defeating human champions in most classical games, we are still unable to build a competitive AI for RTS games [10]. Worth noticing is the fact that as long as classical games are a kind of a test bed for AI, where the aim from the very beginning has been the optimal game, in the work on strategy games there is a visible division of goals between commercial ones, which have to increase the entertainment value of bots, and scientific, which treat RTS games as an extension of classical games, trying to find ways ensuring finding the optimal solutions [9]. Livingstone in [13] points out that we do not need bots playing in the optimal way, but acting believably; we should, however, distinguish here developing AI techniques to create more intelligent bots from using the RTS games' environment to improve and test AI methods. In the first case, the aim is to make intelligent bots for games, in the second—development of advanced AI methods, able to manifest intelligent behaviour in complex environments. Using RTS games as a test bed is justified by the fact that they offer high complexity, but at the same time it is still possible to describe all their elements in terms of their statistics and dependencies. Moreover, they can be treated as simplified representations of real problems. AI methods formed on the basis of RTS games can later be used in real life problems, for example for military purposes [9].

One of the issues common to many AI approaches to RTS games is *believability*. Game developers are interested in creating credible bots since they increase entertaining value of the game, AI researchers—because believability is an inherent quality of intelligent agents. Even if we decide that it is not optimality, but credibility, that constitutes our aim, creating acceptable bots is still very difficult. Analyzing the world of the game—because of the number of elements existing in it and the relations holding between them—is too difficult at the moment to allow for creating bots that would behave in an intelligent way [10]. In effect, bots act according to predefined strategies or scripts defining their behaviour. One of the drawbacks of such an approach is the necessity of describing a considerable number of rules that would be relevant in as many situations as possible. Another problematic issue, which is much worse from the player's point of view, is the repetitive nature of bot's behaviour. If a bot will make use only of a limited number of actions and will react in the same way in given situations the player will sooner or later learn to foresee bot's moves and, as a result, the game will become much less interesting. In addition, the player will also eventually find weak points in the opponent's strategy, which will cause his interest in the game to decrease even more.

The main aim of this chapter is to suggest a system for a bot's personality. Its main task is to maximize the entertaining value of a game by means of providing

mechanisms that ensure high unpredictability of the game and matching the level of bot's skills with those of the player. The system provides a method for controlling bot's behaviour, thus enabling us to enforce certain actions or forbid them, which can be convenient for a user for instance while practicing different strategies or techniques. In addition, the structure of the system makes it possible to use it as a tool for game creation, for example to balance various statistics and parameters of the game engine.

We should explicitly point out that the proposed system is not an alternative for a bot's AI system, but rather its supplement. It can be used to determine when a given action (e.g. attacking the opponent or establishing the resource base) should be performed, and to determine a general way the action should be executed—for example, to characterize how strong an army should be used in an attack or what kind of fortifications should be built in the newly created base. Nevertheless, the exact way of performing these tasks, for example a choice of a base to attack, the tactic used in a battle or the arrangement of fortifications should be controlled by AI modules. This seems to be a sensible approach—aggressive players tend to lead to battles more often than passive ones, brave players could attack armies that are stronger than their own—but when it comes to the battle, it is more a matter of a tactics than personality (assuming that the players play rationally).

10.2 Believability in RTS Games

In this section we will discuss the notion of believability in the context of RTS bots and try to list relevant—from the player's perspective—features that a credible bot should possess. We will also describe different types of players as well as techniques used by them and discuss differences between bots' and humans' ways of playing RTS games.

10.2.1 Rock-Scissor-Paper Rule

The complexity of strategy games renders a lot of possible strategies for a player to choose from. For the game to be interesting they should be carefully balanced and for every strategy some counter-strategies should exist. Such a requirement assures that winning the game is not only a matter of a well-executed strategy, but also requires skillful adaptation to the opponent actions. Such balancing, by analogy with the child's game is often called *the rock-scissor-paper rule*.

This rule is applied not only at the strategic level. Another form of its implementation is the differences between types of units. In most games, units can be divided into different formations, such as infantry, cavalry, ranged units or ships. Each formation has its pros and cons, but none of them is efficient enough against all the others. For example, ranged units, such as archers, are effective against the infantry,

but they are also vulnerable to the cavalry. Cavalry is generally efficient versus most of the units, but there are some infantry units, like pikemen, against whom cavalry is very weak. In some games units are also split into two categories: light and heavy ones, with their relative efficiency adjusted more precisely (usually with military and historic accuracy). What is more, some units may have special abilities or skills, which should also be properly balanced.

10.2.2 Player Types

People playing computer games differ greatly, ranging from those who play very rarely to professional players, who spend many hours each day playing the same game. Naturally, they differ not only in skills possessed or attitude to playing games, but also in their expectations. For the needs of this chapter, we will define three types of players:

- **Amateurs** are players that play rather rarely and poorly. They may be somewhat familiar with the concept of RTS games, but are not used to the given game and are still learning about different types of objects (i.e. units or inventions) and their usage.
- **Regular players** have some experience with games and understand RTS games mechanics. They are able to formulate and execute simple strategies and tactics and to adjust them to their opponent's actions. They possess some knowledge about the given game, which allows them to efficiently develop their base, raise armies and lead them into a fight.
- **Experts** possess excellent, deep knowledge of the game. They had great knowledge of game world objects and are familiar with their statistics. They know vast majority of the strategies that can be used in the game. Experts are very familiar with the game's user interface and keyboard shortcuts, which enables them to execute actions very quickly.

Of course, these categories are game-specific, and an expert player in one game could play on a regular or even an amateur level in another.

10.2.3 Advanced Techniques

One of the advantages of experts over regular players is their extensive knowledge about the game. For example, regular players usually act according to some general strategies, which are not very detailed, as they just describe an overall direction of development and actions. Experts, on the other hand, develop a set of very precise and detailed strategies. A typical example of such strategies are *build orders*. A build order precisely describes the order in which units and buildings should be created

at the beginning of a game. The order is chosen specifically to achieve the required level of development as soon as possible, which sometimes yields huge advantage. Different build orders serve different goals so the optimal build order that is suitable for all strategies does not exist. By observing initial actions of his opponent an experienced player can infer his next moves quite precisely, which allows him to react accordingly in advance.

Similar rules apply on the tactical level. For example, experts have learned to exploit small differences between different units (like their speed or attack range) to formulate very effective tactics. Sometimes they are only generally outlined, but it happens that they are very precise. In order to better illustrate the level of details that is sometimes achieved we will shortly mention two such techniques, called *the Patrol Method* and *the Chinese Triangle* [5]. These techniques comes from StarCraft and describes a method of controlling a group of *Mutalisks* in a fight against *Scourges*. The first one describes the situation when a player who controls Mutalisks tries to gain distance of one and a half Mutalisk from the opponent's Scourges and then issues a Patrol order as a target choosing (by clicking) the area right in front of his Mutalisks group. Because of game latency, before the order reaches the game engine, the Mutalisks move to the previously marked place and while performing the Patrol order they turn around and engage the opponent's units. At this very moment, the player should give a Move order aiming at an distant area behind the Mutalisks, which will cause them to turn back again and fly away. This procedure can be repeated several times, until enemy units are destroyed. The Chinese Triangle technique is similar, however it requires not two, but four clicks at right angles to the units participating in the skirmish.

These examples perfectly show how precise some techniques are. Distances and angles used are adjusted so that Mutalisks are able to attack an enemy and to retreat to a safe distance without taking any losses. Of course, a proper execution of this technique requires high manual skills and precision. In fact, in some games the speed of the interaction is so important that it is measured and used as an approximation of player's skills. For example, in StarCraft the speed of a player is usually expressed as the *Actions Per Minute* (or APM for short) coefficient. Typical APM ranges from 50 for casual players to about 300 for the experts [3, 4].

Another category of advanced techniques are those taking advantage of the game engine flaws or exploiting its features in a manner unintended by the game developers. A perfect example of this technique is *stacking* [6]. It exploits the mechanism of formation holding included in StarCraft: if units in a selected group are close enough to each other, they will act as a whole while they move—keeping initial distances and relations among them; in the other case, when units are scattered over the map, they will gather around the destination point, partly overlapping and stacking on each other (hence the name of technique) [7]. Stacking allows one to maneuver groups of units very precisely (which is required by techniques like the Chinese Triangle) and makes it harder, or even impossible for the opponent to pick the weakest unit in a group as a target to attack. The trick to exploit this mechanism is to add an additional unit to a selected group, usually very distant and slow or immobilized, just for the

engine to treat the group as scattered, and, as a result, to have units continuously stacked.

The last example of an advanced technique is remarkably different from the previous ones. This technique, used, among others, in the *Age of Empires* series, relies on building palisades and walls around the resource locations that are close to the opponent's base. This is a way to force him to gather resources away from his main base, which makes his resource bases more vulnerable to attacks, or to time-consuming razing of fortifications with units not suited for this (as siege weapons are not available in the early stages of the game). It may be surprising to qualify such a technique as advanced, when, in opposition to the previous ones, it does not require special manual skills or good knowledge of the game. This classification was made on the basis of unnaturalness of particular techniques. While common players usually use objects in the game according to their primary functions (with walls used to defend bases or to block strategic passages), the experts try to gain advantage over their opponent and achieve certain goals with any means possible.

The above examples show that the difference between experts and common players in RTS games lies not only in mastering certain skills, but also—and in some cases mainly—in a better use of techniques specific for a given game.

10.2.4 Difficulty Levels

For a game to be satisfying for players with different skills mechanisms of difficulty adjustment are required. A classic solution is to introduce several difficulty levels. Two approaches can be distinguished here. The simpler one, most commonly used, does not affect bot's behaviour, and only modifies its certain characteristics (e.g. resource gathering speed or attack efficiency of units). Though this approach generally fulfils its purpose and appropriately lowers or raises game's difficulty, it is not perfect. As shown earlier, the experts' style of play is highly specialized and differs considerably from the regular player's style; the difference is not only quantitative but also qualitative. In other words—a *believable* bot playing on the expert's level should use techniques and strategies used by experts. This point of view is realized by the second approach, where the bot's behaviour is modified e.g. by introducing new strategies and techniques or by improving reasoning mechanisms. This way, the bot not only imitates experts better, but also generally becomes more effective.

Adapting to beginner's level can be achieved similarly, but it is not so interesting, as beginner players have less expectations from bots, and furthermore, that kind of adaptation is usually much easier to achieve. It is usually enough to put a cap limit on the number of units and buildings controlled by the bot, so it can develop only to some extent, or limit the frequency and strength of its attacks, leaving the initiative to the player.

Setting a fixed number of difficulty levels may lead to a situation where a given level is too easy for the player, while the next one is too difficult. To avoid such a situation more levels can be created thus making differences between them smaller.

Alternatively, the difficulty level can be specified as a continuous parameter, allowing more precise fitting to the (human) player's skills and, in perspective, bringing him more satisfaction from playing.

In the last years some research was done on the automatic adaptation of the bot's level of play to the player's skills. By in-game adjustment of the level of challenge the game presents we can assure that the game is neither too easy nor too difficult for the player, which makes it more entertaining. Those methods are of limited use in RTS games, as level of interaction in those games is rather low in comparison to other, more action-oriented genres, like FPS (First Person Shooter) games [14]. Most of the actions executed by the players (i.e. building an army, researching technologies) are hidden from each other thanks to the fog of war, and only their effects are visible (directly, like being attacked by an army, or indirectly, like the opponent's unit becoming more effective than ours) only later in the game.

10.2.5 Believability

Let us try to describe a bot's *believability* more precisely. Naturally, it would be perfect if a bot was able to imitate a human to the extent where the player could not tell whether he is playing against the bot or the human player. In most RTS games, however, it is very easy to identify a bot. The first hints are very low reaction time and high playing speed, in which bots have an intrinsic advantage over humans. Of course, the decision whether a given speed is achievable by a human is subjective and depends greatly on a player—it will vary considerably for a normal player with APM 50 and for an expert with APM of 300, but even experts are not able to precisely control dozens of units fighting in a few different battles simultaneously. Another symptom betraying a bot is the specificity of his high-level strategic actions. The main factors here are the repeatability of actions and the lack of adaptation to the opponent's strategy and the situation on the map. Another sign is an absence of reaction for unsuspected events. For example, big units sometimes get stuck between buildings or trees when moving between two points. Such errors concern both human and computer players, as pathfinding algorithms are usually identical for all players, but a human player will eventually spot and fix them (e.g. by choosing a different path), while bots often fail to notice the problem.

On the other hand, considering the fact that the main purpose of a bot is to provide an entertainment for players, a perfect imitation of the human game-style may not be necessary. Of course, if a bot were playing so well (in terms of action believability, not effectiveness) that it would be taken for a human, it would probably meet its entertaining purpose in a high degree (at least for players with comparable skills), but perhaps it could be enough for bots to play in a way that the player could

enjoy, even knowing that his opponent is a bot. With a reference to the Turing Test[1], creating a conversational bot which could successfully deceive a human would be a huge achievement in artificial intelligence, but it would be a great success even if it could just converse sensibly. Adding limitations like causing a conversational bot to refuse to perform complicated mathematical calculations (in case of RTS bots this could correspond to, for example, limiting the speed of giving orders) increases imitation level, and accordingly the chances to deceive a human player, however, it does not introduce any new quality to the bot's behaviour.

These intuitive conclusions are largely confirmed by a study made by Sweetser et al. [16]. They questionnaired a number of gamers to determine the importance of different aspects of games (of various genres). Their study yields several interesting conclusions. Firstly, there is a reasonable number of gamers that prefer to play with bots rather than with other humans. They do so mostly for convenience (bots are always available, they do not argue about matches settings etc.) and training (they feel that they are not good enough to play with other human players). Secondly, those who prefer playing with humans gave three main reasons for their preferences—opponents' intelligence, social interaction and behaviour realism (in the order of importance). Realistic behaviours were significantly less important for study subjects than the remaining two factors. As it is very unlikely that we will be able to reproduce the social interaction within a group of friends using bots, the results of the study suggest that we should concentrate more on creating intelligent bots than on assuring realistic behaviours.

10.2.6 Summary

In this section we have shortly discussed various issues related to RTS games:

- Balancing of strategies and units,
- Difficulty levels,
- Advanced players techniques, including:
  - precise use of game objects statistics and game engine features,
  - taking advantage of game engine flaws or its features in a manner unintended by its creators,
  - use of game objects in a manner unintended by its creators.

While appropriate balancing and introduction of difficulty levels help to raise the entertaining value of the game, adaptation of advanced techniques influences mostly believability. We will address these issues again later, when describing the personality system, and we will try to show how it can help to deal with them.

[1] For an in-depth discussion of Turing Test variants and the meaning of intelligence in computer games see [12, 13].

10.3 Personality System

Before we start describing the structure of the system, we will briefly mention goals that we had in mind while constructing it. It should make the construction of the system as well as some of the solutions we have adapted more justified and easier to understand.

Our main goal was to create a system able to imitate players possessing different skills and representing different game-styles, which would potentially increase the entertaining value of a bot. The general idea was to make it possible to modify general features of the bot (which can play more or less aggressively, pay more or less attention to economic and technological development, etc.) as well as to control particular behaviours, for example the strength of fortifications to be built or time when the first attack is launched. Naturally, diversity of actions should be balanced with their effectiveness. We do not want a bot's actions to be just random—carrying out a reasonable strategy requires at least minimal coordination and consistency of actions. Lowering difficulty level is not the only problem of over-randomizing—if a bot's actions are hardly justified from the strategic point of view, then its credibility suffers.

Another requirement was the possibility to create personalities corresponding to specified difficulty level.

Even though the system is of general purpose, it should be adequately adjusted to a chosen game. Differences between RTS games are often significant, so we decided not to base on any particular game while describing the system. Instead, we present typical examples for RTS games scenarios and on these examples we show how the possible adjustment may look, and also how it is possible to make the system more detailed if needed. It allows one to adjust the system appropriately and to achieve a desired level of complexity.

While designing the system, we took into consideration that the players do not perceive a bot's decision making mechanism, they may only observe their results (for example, they may notice a bot's army moving towards one of their bases, but they do not know why a bot had chosen this particular base and how the size of army was determined). Creating a highly advanced bot's behavioural system would be most certainly very time consuming and it might be unrewarding if it did not make noticeable difference compared to a simpler system. On the other hand, even if the player did not notice the difference in a bot's individual actions after the introduction of the new system, it might result in a difficulty level increase; firstly, because the sum of many little advantages (unnoticeable individually) might, eventually, become significant, secondly—because the meaning of some very carefully thought actions could be unclear to the player at first. The conclusion is that the balance between simplicity of the system and its capabilities should be kept.

10.3.1 Introduction to the System

Analysing a typical course of a game, certain actions such as building fortifications, establishing resource bases, researching technologies or attacking the opponents bases can be distinguished. These actions (common to majority of RTS games) will be called tasks. To each such task a policy a set of parameters that describe the desired manner of executing it is assigned. For example, the policy concerning attacks on the opponents bases can contain two parameters the first defining when an attack should be launched and the second describing a required strength of the army. In simplification, a complete set of policies (one for each task) describes the entirety of bots behaviours, and so such a set will be called a bots personality. From a technical point of view, a personality is just an object with a specific internal structure set of different parameters grouped into policies.

In short, the system has to generate a personality which can be then transferred to the Al bot modules, which can use it in their decision-making process. Obviously, it is not the task of the system to generate just random personality, we want them to be reasonable and believable, while picking policies' parameters at random may lead to a weak play and unnatural random-like behaviour.

Firstly, we will focus on policies and their parameters, describing how to choose them and how they can be utilised by AI modules and to what extent they can control bot behaviour. Afterwards we will describe the process of personality generation.

10.3.2 Profile

Some parameters that we can use in policies are so general that they could appear in more than one policy. For example, for tasks such as *attacking opponent's base* or *attacking groups of opponent's units* it would be worthwhile to determine a *courage* parameter—a minimal ratio of our's and our opponent's units strength required to execute an attack (the lower the ratio, the braver the attacks would be). Defining this parameter for each task individually has an obvious advantage—increasing styles of play diversity by imitating personal players' preferences (in this example, a player can prefer battles with siege weapons and readily lead to them, while another player would avoid them, waiting for the opponent to come out to an open field). On the other hand, our system is bound to allow easy creation of different types of players—an easy and intuitive way to do so is to allow one to set a bot's courage or aggressiveness levels, which would affect the bot's overall behaviour. From this point of view sharing those parameters seems to be reasonable.

Thus, a special policy—a *profile*—is introduced. The profile can be perceived as the character of the bot. Its aim is to briefly describe features of a bot referring to its global behaviour, as opposed to particular (local) tasks. The influence of these features does not have to be decisive nor have they to be apparent in particular actions, they should however project on the overall behaviour of a bot (up to some point). The

profile may contain rather specific attributes, which do not fall into any of policies, like preferences for unit types, as well as general ones, like aforementioned *bravery*. AI modules should take into consideration values from both policies and profile when making their decisions. This way it is possible to control a bot's behaviour both globally, by changing parameters in a profile, and locally, by changing the right policy.

Examples of the features that can belong to a profile include:

- **courage**—has been previously described as the minimum ratio of strength of our army to the strength of the opponent's army that allows a bot to launch an attack.
- **cowardice**—can be described similarly to courage—as a ratio of armies strength at which our army would run away from the battlefield.
- **expansiveness**—can be described as a tendency of a bot to step into his enemy territory accompanying building new objects or settling resource bases. With a low level of expansiveness, a bot will play cautiously and will limit his activity to the areas controlled by themselves. On the contrary, with high expansiveness a bot will be playing aggressively and will invade territory of an enemy if it will be lucrative (attractive locations may include rich resources locations or strategically important regions).
- **quantity/quality ratio**—describes a tendency of a bot to create a massive armies, as opposed to smaller but more effective ones (e.g. by technological development or by choosing more expensive, upgraded equivalents of the primary units).

Other important features worth mentioning are preferences for unit types. According to Sect. 10.2.1, the army created by the player should consist of various formations. However, the proportion between them is not strictly defined. Individual preferences of the player have crucial importance here and decide about the real power of each formation (e.g. by efficient use of a formation's properties). To simulate such preferences, we can define desirable proportion of formations in the profile. Then, when building a new unit, we can count the current ratios of the formations and compare them with the desirable ones. Thus, we can choose the best fitted formation. Thanks to that, we can ensure that armies created by a bot will be more reasonable when it comes to the strength of each formation. By randomizing these parameters at the beginning of each game, we can provide variability among subsequent games. This goes beyond purely visual meaning (as a bot's armies will consist of different units) since it also has an impact on the difficulty level. If the armies created by bots were always the same, the player would easily find the optimal counter-army (for example if a bot was always building lots of cavalry, the player would anticipate that they would need a lot of phalanx or pikemen). Our solution largely eliminates this problem.

These preferences may also affect developed technologies and created objects. Obviously, it is reasonable to invest more in technologies related to strongly preferred units. Similarly, it would be wise to possess more buildings that are able to train the preferred units, so that the time of unit production is minimized (usually, a building may train only one unit at a time, so a small number of buildings can hinder production speed).

We can go one step further and notice that players often have their own preferences for some units. We can model them similarly to formations—all we need to do is define the preference ratio for each unit type. If differences between units are not significant (with respect to their statistics), such a change may have only visual importance—generally, the bigger the differences between units, the greater the importance of the preferences. Additionally, some units can possess unique skills or statistics modifiers that depend on their target (like a pikemen against a cavalry). In this situation, the choice of an appropriate counter-army can be difficult as the relations between units can be complex.

Profile parameters could be also modified during the game which may give some interesting results. For example, changing the expansiveness of a bot from high to low may lead to a situation in which a bot starts to play aggressively, conquers a few strategically important objects on the enemy territory with a few aggressive, risky moves and then calms down trying to maintain advantage gained—which is strategically well-justified. As a result, we can easily exceed the simple expansive-passive scheme and making it harder for the player to predict a bot's moves.

An example of a situation, where such changes may be of high importance is adjusting formations preferences. The result of randomizing the preferences at the beginning of the game is that the player will not know the composition of the bot's army in advance. However, without changing them during the game the player will quickly learn it simply by observing the bot's armies. Introducing randomized changes partly alleviates this problem. We can go one step further, and adjust those preferences based on the analysis of the player's armies. This can be achieved by reversing the proportion of formations (for example, if the player has more archers, a bot should build more cavalry—see Sect. 10.2.1) or by more sophisticated mechanisms, such as self-organizing maps that were used in [8] to find an optimal counter-army given a set of the opponent's units as an input.

10.3.3 Policies

As the choice of policies and their parameters is game dependent, we will now discuss them in greater details, showing how they and their parameters can be chosen.

In many RTS games one of the most important things in the early stage of the game is setting up an effective economy that will provide the player with resources required for further development. This is why the beginnings of such games usually resemble each other: for a time only workers capable of gathering resources are being created and only after obtaining a number of them the player moves on to building other units, constructing buildings etc. In the following analysis we will call such actions *the initial development of economy* or simply *the initial development*. Usually the player does not stop creating workers after the initial development stage, but rather keeps on producing them until a desired number is attained, or sometimes even throughout the game. Simplifying the issue a bit, the creation of workers can be described with the help of three parameters: the number of workers necessary

for completing the initial development, the desired number of workers (the number of workers at attaining which the player stops producing them) and the parameter describing the right time for producing a worker. This last parameter can be defined as an amount of game time that has to elapse between two subsequent worker creation processes.

Another feature common to most RTS games is attacking enemy bases. Generally speaking, describing such a task requires setting the time when an attack should be launched and the strength of the army required to do it. It should also be remembered that the first attack should differentiate from the following ones. In the case of the first attack, defining the temporal parameter can be done in a couple of ways. We can simply describe the moment of an attack by relating it to a chosen event, e.g. completing the initial development, building a given construction (for example stables, so that cavalry can join the army) or a hostile attack (we hand the initiative over to the enemy and wait for a chance for a counter strike). If we decide to treat the beginning of a given game as the starting point, the moment of an attack can also be stated in absolute values, e.g. we always attack when n minutes have elapsed since the beginning of the game. Another option is to define a condition on which the attack is launched, e.g. the number of units or fortifications we have (to avoid leaving our base defenceless). Defining the required strength of the army does not allow for so many possibilities—it is only possible to define a required number of units or their joint strength.

The above definition may lead to various problems. It may well happen, for example, that the condition for launching an attack has been fulfilled when we are still at the initial development stage (e.g. when the condition was defined as a given amount of time since the beginning of the game and the initial number of workers is big), and we have neither military units nor buildings at our disposal. What follows is a clash between policies—following one of them (attacking the enemy) violates the other (building workers), according to which we should be still producing workers. Solving such conflicts lies outside the scope of our system.

The parameters used in the above examples are quite diversified with respect to their structure—some of them are just numbers (e.g. a number of workers), others are more complex and consist of several values, e.g. temporal parameters that consist of two values: an amount of time, expressed in game ticks or seconds, and a label of the referenced event. What is more, policy parameters can be assigned values of different types—be it a single number, a set of objects or a condition. All of them will be treated (and referred to) as conditions in our system.

The conditions we have discussed so far involve mainly checking a given game feature, e.g. the time that has elapsed since a given event or calculating and summing up the strength of battle units, with a notable exception of the *being attacked by the enemy* condition, which requires some abstraction—mainly the notion of being attacked and recognition of such an event. When applied with care, those simple conditions should suffice to create a satisfying degree of unpredictability, but we would also like the bot's actions to "look intelligent". In such a case, it is desirable to introduce more complicated conditions that rely on the bot's AI mechanisms.

To describe what is meant by that, we shall examine the case of constructing fortifications adjacent to a newly built resource base, with the assumption that we defend every base with a previously chosen, fixed set of fortifications. It may well turn out (especially, if we decide to start building our bases early in the game) that in the initial stage of the game, when our economic development is of crucial importance, we are spending too much time and resources on building fortifications that are too powerful in relation to the strength of units that are at our enemy's disposal at that stage of the game. A partial solution to this problem can be to set out intervals at which subsequent elements of fortifications should be built, which will help to distribute the overall cost more evenly. This solution, however, will not work out in the later stages of the game, when resource bases become vitally important and it is usually necessary to build the fortifications as soon as possible to forestall the enemy's attacks aimed at recapturing the bases. What is more, a given base may be located in a place the access to which is only possible from the bot's main base, which makes the construction of fortifications quite useless (unless there is a possibility of attacking the place from the air or from the sea). What follows, is that a bot should be able to estimate the importance of a given base as well as the probability of its having been attacked by the enemy, which is dependent not only on its importance, but also on other factors, like base location or terrain configuration. The enemy's way of playing should be also taken into account—if they continuously attack our resource bases we should build our fortifications as soon as possible.

In the light of the arguments presented above, it stands to reason that a relevant decision-making process should be based on an advanced reasoning rather than on fixed rules. As it was shown in the previous example, such reasoning may be applied to the stage of the game and its development (e.g. the distribution of areas controlled by players), the analysis of the game map (places vulnerable to attacks, amount and distribution of resources) or even the enemy's behaviour. It should be noted here, however, that the aim of our system is neither to carry out nor to characterize such reasoning. It is also unjustifiable to define policies by means of detailed parameters describing the features mentioned. For example, a policy for building fortifications should not be described by means of specifying a relation holding between the strength of the fortifications and the game phase or the strength of the enemy forces. Parameters that are general in nature should be used instead; hence they should be—if possible—normalized. Such a system should remain functional not only after changes in the features influencing the bot's reasoning (e.g. a drastic change in the units' statistics or the way of calculating the overall strength of the army), but also after a complete replacement of bot's AI. Obviously, it is not always possible to do that, e.g. if we decide to take a given stage of the game as a condition it would make no sense to normalize it to, say, the interval [0, 1].

Defining all the parameters as conditions, while redundant at first sight, may provide us with a whole range of possibilities of modifying and improving a bot's behaviour. To exemplify this, let us have a look at the policy of creating workers, which has a predefined maximal number of workers. Instead of defining this parameter as a given number, we can define it as a relation between the number of our workers and the workers of the enemy. Such a relation can be described by means

of a coefficient (e.g. we want the number of our workers to be a $k\%$ of the enemy workers number), constant (e.g. we want to have k more workers than the enemy) or even a more complex function, dependent on, e.g. the game phase, the state of the world, the level of difficulty or personality features.

Another example of a complex condition may be making this parameter dependent on the actual need for resources. If our economy is in a good shape and we are gaining resources faster than we can spend them, building new workers may be unnecessary at the moment. However, such an approach requires a more detailed analysis—taking into account future spending, so that we can prepare for a possibly increasing demand for resources in the future. Even though the complexity of such an analysis may make its implementation quite unprofitable, despite its obvious efficacy, the presented system assures the possibility to use arbitrarily complex conditions.

To achieve greater control over the bot's behaviour it is convenient to combine different conditions. For example, we can define two or more conditions and specify that all of them must be fulfilled to execute an action, making the condition stronger, or that it is enough if one of them is fulfilled, making it weaker. By nesting such compound conditions we can describe conditions as logical sentences with variables being single conditions. The use of such conditions allows us to create a bot whose behaviour is based on different factors and is therefore more flexible. One example of such use would be to define—as above—the maximal number of workers for a bot to be a function of the number of his opponent's workers, but to introduce also an upper limit for this number. Another example would be to specify that we should launch an attack only when our main base is fortified and our army is stronger than the opponent's army. This way we can define behaviours that are more natural and reasonable.

We mentioned above two policies concerning attacking the enemy: one of them concerning the first attack, the other—subsequent attacks. Such a division may prove insufficient in case of more complex games. Simply, the policies could describe two aspects: the moment when an attack should be launched and the required strength of the army. But if the enemy's base is well-defended then—unless we have a decisive advantage or we are able to prepare a landing operation avoiding the defence lines—our army should be strengthened by siege engines. And while it may be hoped that a certain strategy of creating an army will ensure the presence of siege engines in virtually any army, it may be better—for the sake of credibility—to make sure that this actually takes place (it may happen, for example, that our army will be created out of the previously defeated army remnants that did not include siege engines). In order to do that we should define not only the minimum strength of ordinary units, but also set a fixed number or desired strength of siege engines.

If we decide, however, to introduce this parameter to a policy describing the way of attacking, we can encounter a situation in which siege engines will be present in all our attacks, even those against targets consisting solely of enemy units. This would have a very undesirable effect of reducing the credibility (of the game), but also will lower a bot's efficacy, since siege engines are usually much slower than other units and vulnerable to damage (so we are only putting them at unnecessary risk). In addition, adding a requirement concerning the presence of siege engines in

an army prolongs its creation: siege engines usually require a special building to be built and some technologies to be invented. All this renders quick attacks impossible. It is then sensible to introduce a new policy, defining the way of attacking fortified places. In the process of planning such an operation a bot's AI should decide if this policy should be put into effect, for example on the basis of data gathered by scouts.

In the case of more complex games with water/air units it may happen that the mere fact of having the required number of units is not enough—and that we need to transport them to an isolated place, say, an island or an area protected by mountains. In such a case, we need transporting units (be they air or water)—and, as they are usually easily damaged, we should assume the existence of some units to protect them as well. As a result, there appears a need for one more policy—this time for carrying out desant operations.

We can keep on adding new policies to deal with increasingly detailed situations. It is difficult to define an immutable stopping point at which we should terminate our quest for precision—it depends to a large extent on the game itself, but also on how detailed the bot's AI is. If we have a ready-made AI at our disposal and we know that it distinguishes a category of e.g. *attacks against fortified positions*, then the policies we create may aim at complementing the already existing system. Otherwise, we have to rely on our intuition or expert knowledge, or create the system incrementally, i.e. initially build a simple system and enrich it along with the needs.

Some tasks may require breaking down into smaller subtasks, even if the subtasks would remain unchanged in terms of their structure. For example, building fortifications for the main base and resource bases are identical if one takes into account the actions that have to be undertaken (building a given number of buildings of a given type, such as defence towers or fortresses on a given area). But from the point of view of the bot's behaviour designer these actions are different. First of all, the reason for creating the buildings is different in both situations—the fortifications of the main base should be much stronger as they are protecting our most vital technological and military facilities, so their main purpose is to withstand even heavy attacks of the enemy. Secondly, the starting conditions are not the same. While the resource base fortification usually starts immediately when its construction is completed, the process of fortifying the main base usually starts much later, since at the beginning of the game the player's economy is simply too weak to provide the resources necessary for building expensive fortifications. It may also happen that building fortifications requires an advanced technology which is unavailable at the beginning of the game.

10.3.4 Personality Creation

The above discussion should suffice to conduct the analysis of the game and to preliminarily distinguish tasks and their parameters that would fit into our needs best. Now we will focus on generating a personality.

As mentioned previously, one of the system's objectives is to provide a player with a detailed control over the bot's behaviour. Depending on the player's experience with

the game he may perceive certain behaviours of the bot to be unnatural or undesirable. For example, frequent and strong attacks can be interesting for advanced players, however they can easily daunt amateur players, who are unable to prepare for them. Of course it also acts in the opposite direction—behaviours typical for amateurs (e.g. slow and static development) are not demanding enough for experts. Hence, there exists the need to adjust a bot's behaviours to the player's skills. This control is established by setting parameters included in two objects:

- **Traits**—parameters describing the bot's general attitude. Examples of these parameters are significance attached to economic, military and technological development, building defences, quick base development or aggressiveness. Additionally, we can place here values that we intend to use in a profile. As it will be shown, it is also worthwhile to add a difficulty level parameter here (which is not related to a bot's attitude),
- **Bot settings**—settings describing the bot that are not a part of traits because of their nature, as they concern more specific actions, like flags defining whether a bot can employ particular techniques such as the Chinese Triangle or more complex strategies that the bot can follow.

The general idea for creation of a personality is to consider each policy separately and describe its parameters in terms of values they can take. Specifically, we would like to define policies' parameters values as a functions of traits, so that modifying traits would modify also personality. For example, we can describe a time elapsed condition with a type of the event and a function returning a number of ticks. We will call such descriptions *metaconditions*. As it was described earlier, some policies' parameters can be described with conditions of different types. Thus, to each parameter we can assign a set of metaconditions—we will call it a *parameter description*. A set of parameter descriptions for each policy parameter will be called a *metapolicy*. To better illustrate these new concepts they are presented in Fig. 10.1. An example of the metapolicy is presented in Fig. 10.2. Now, given a metapolicy we can easily create a corresponding policy. We simply choose one metacondition from each parameter description and create a condition from it—resulting conditions will constitute a new policy. This is repeated for all policy-metapolicy pairs. Including a difficulty level to the traits allows us to adjust personality to the player's skills more precisely.

Considering policies separately has one serious drawback, which we will present on an example of typical RTS strategy called *rushing*. The idea of the strategy is to attack the opponent very early, when he does not have any (or hardly any) defensive structures or military units to defend itself. It is inadequate to characterize rushing by simply specifying that (for example) the first attack should be launched as soon as the initial development has been completed, as it is not known in advance when it is going to happen (and it may be late in the game). Thus, the initial stage duration needs to be restricted as well. All in all, apart from the *first enemy attack* metapolicy, the *workers* metapolicy has to be explicitly specified to ensure that the initial development phase will be relatively short. Additionally, some policies may have an impact on the chosen strategy, even though they are not visibly related. For example, when rushing, the main goal is to build a required number of units as quickly as possible,

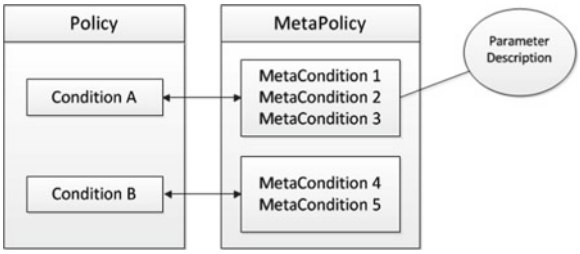


Fig. 10.1 A relation between a policy and a corresponding metapolicy. For each policy's parameter there is a corresponding parameter description. Parameter descriptions consist of a number of metaconditions

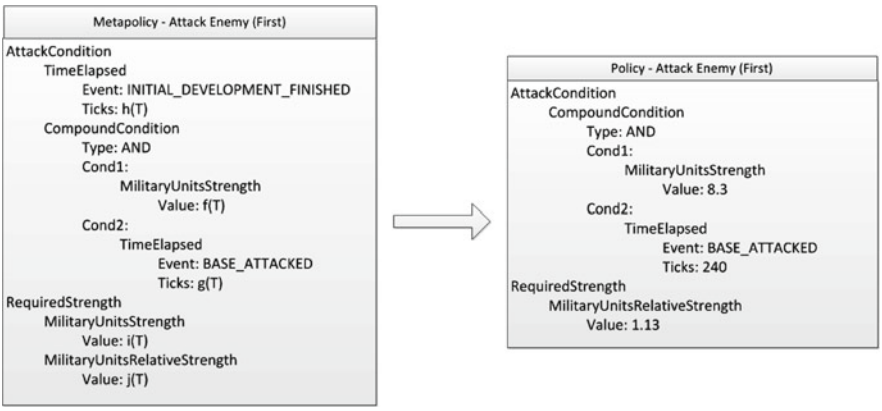


Fig. 10.2 An example of a metapolicy and policy generated from it. In the former, *AttackCondition* and *RequiredStrength* are parameter conditions and *TimeElapsed*, *CompoundCondition*, *Military UnitsStrength* and *MilitaryUnitsRelativeStrength* are metaconditions. f, g, h, i and j are functions taking T (traits) as an input

and so researching any technologies is not only unnecessary, but it also interferes with this goal, as it requires resources which are essential for building units. What is then needed is coordination of several metapolicies. Even though, when metapolicies are properly defined it may be possible to create a rushing bot by setting traits accurately, it is uncomfortable and unsatisfactory since it limits our possibility to control traits. To allow for such coordination *metapolicy sets* were introduced. These are sets of a certain number of metapolicies which function as a whole. Their application to the bot creation process is very simple: knowing a metapolicy set that is to be used, all that needs to be done is to check—while iterating over metapolicies—whether for a current metapolicy there exists a metapolicy of the same type defined in the metapolicy set—if yes, it is employed to generate a policy, if not—a "default" metapolicy is used instead.

Such sets may be used in two ways—firstly, we may allow players to turn corresponding strategies on/off through the corresponding option in the bot settings. It would make it easy to allow player to choose a specific strategy, so that he can

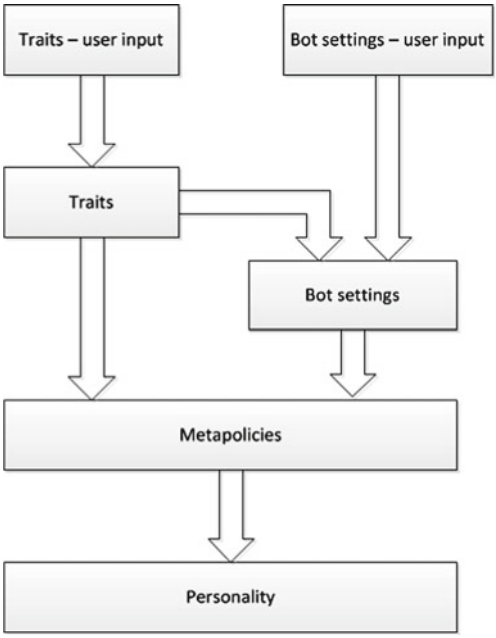


Fig. 10.3 Scheme of a personality creation process

practise his play against it, and, at the same time, he would be able to disable strategies that he is uncomfortable with. Secondly, if a player has omitted this option (or it is not available) it can be turned on with a certain probability.

The process of creating a personality is outlined in Fig. 10.3. It consists of four phases:

1. User input—the user (player) is given an opportunity to specify traits and bot settings that he is interested in. The manner in which the user can modify these values is not of great importance. For example, the user can be presented with a number of controls that allow him to set parameters to desired values (e.g. using sliders to set values such as a difficulty level or aggressiveness) and checkboxes to enable or disable certain behaviours, or, alternatively, the user can set an interval, from which a given value will be chosen randomly. To speed up the process he may also be presented with a list of predefined settings he can choose from and—eventually—modify them according to his needs. Regardless of the method used, the user should be given the opportunity to omit
certain settings.
2. Supplementing traits—settings chosen by the player are now processed and the omitted parameters are supplemented. Again, as far as the process is concerned, the exact method of supplementing those values has no significance. The simplest

solution uses values chosen randomly from the uniform distribution from an appropriate interval, more complex may use for example normal distribution with appropriately matched parameters.
3. Supplementing bot settings—similar to the previous stage, however, traits and bot settings that were set can be taken into consideration while supplementing omitted values. For instance, if a player omitted a setting concerning techniques that are employed by the bot, their value can depend on the difficulty level and aggressiveness (as a player using advanced techniques may seem to be more aggressive). It is important here to be careful and enable some techniques only for adequately advanced players. Otherwise, if the player is not familiar with a given technique, they may think that the bot is cheating or that the game is not fair.
4. Creating personality—a bot's personality is created as previously described. As already mentioned, when the personality is created it should be passed to AI modules.

Creating a personality in this manner allows us to generate personalities with desired characteristics on the one hand (with the assumption that metapolicies are well defined), and, on the other, to generate a number of bots behaving diversely (when some the user settings values are left unset).

10.3.5 Advanced Applications

In the simplest case the system can be used as a tool that decides which actions should be performed. It can be also used in a more advanced manner, for example to run different simulations. In [11] the Monte Carlo method was used to choose the most promising strategy in a simple *capture the flag* game. To give another example, simulations were used in [15] to find a Nash equilibrium approximation to solve the problem of commanding armies in a simplified environment corresponding to a typical RTS games. Such methods require a certain number of strategies available. The best ones of them are being determined by comparing their outcomes (in a way depending on a method). The simplest approach to define such strategies is to generate them randomly. This, however, forces us to generate a reasonable number of them (as the number of strategically good moves is usually small compared to the number of all possible moves), which increases the time required to perform simulations. Time restrictions imposed on the simulations lead to shortening of a single simulation time or enforcing the use of a higher abstraction level. Either way, this leads to a quality drop of the results. This is the reason for using predefined strategies for describing typical actions. However, such a solution has the same disadvantage as scripting—repeatability—which is still the problem for the game's playability, even if the strategies chosen are optimal.

Our system helps to partially overcome these problems. Generally, when using such methods at a given game stage we identify a set of possible actions, like attacking an enemy, building resource base or raising army and then, repeatedly, choose one of

them as a current action: characterize the method of its execution, run the simulation and evaluate the outcome. Predefined metapolicies can be used to characterize the execution method, so that we gain a convenient tool for such methods' generation. In order to demonstrate another advantage, let us imagine that at a given point of a game we control only a few weak military units, while our opponent possesses a large army. In such a situation launching an attack is not sensible. Monte Carlo methods, however, tend to discover the unexplored possibilities, so they will potentially try many possible variants of attacks. When using policies to describe tasks we can check whether their conditions are fulfilled (for example, conditions for attacking an enemy may require a certain army strength). Most probably, some of those conditions will not be fulfilled, so we can reject simulating corresponding actions and thus limit the total number of simulations needed, which shortens the overall time requirements.

10.4 Implementation

To test the system in practice and to verify our assumptions, a simple bot which based its actions on the personality system was implemented. This section will begin with the description of test environments employed during the system development. Subsequently, the implementation of the system and a bot will be described.

10.4.1 Test Environments

At first the system has been developed in a testing environment created specially for this purpose. In the general assumptions it was based on the *Age of Empires* game. It involved the following elements:

- Three types of resources—wood, gold, and stone,
- Several types of buildings including:
  - main building, in which workers were built. Also, it possessed the ability to shoot arrows at the enemy units (to prevent rushing),
  - three types of military buildings: barracks, archery range and stables, which correspond to different formations: infantry, archers and cavalry,
  - two types of defensive structures: towers and fortresses,
- Six types of units—2 for each formation. The units possessed statistics such as hit points, armour, speed, attack strength and attack range. Additionally, they had modifiers of attack and defence ascribed to every formation,
- Six types of upgrades—attack and defence upgrades available for every formation. Every upgrade had 5 levels; each of them increased the corresponding modifier by a constant value. Upgrades influenced all units of the appropriate formation.

A default bot's behaviour for basic tasks such as pathfinding, attacking the opponent or gathering resources was built-in in the game engine. A screenshot from the environment is given in Fig. 10.4.

One of the major problems with using this environment was the lack of the possibility to assess the bots performance. We have tried to use the personality system itself to find the appropriate bots but it was a moderate success (the experiments we have conducted are described in Sect. 10.5.1), and it was still unknown whether the bot would play well against bots based on different techniques. Also, another problem of different nature arose—how to evaluate the believability and entertaining value of a bot which plays in a game that nobody knows (despite the fact that our environment imitated the game *Age of Empires*, differences in the game mechanics and object statistics made the game quite different). Because of these problems (in connection with some technical problems concerning the game engine) the decision to change the test environment was made.

As a new test environment one of the most popular RTS games—StarCraft—was chosen (a screenshot is given in Fig. 10.5). This choice has some key advantages:

- **simplicity of bot creation**—BroodWar API (BWAPI) framework enables relatively easy and fast creation of AI modules,
- **ease of assessing bots effectiveness**—a created bot can be tested during the game with the built-in bot, in the game with people, and also with other bots created using BWAPI,
- **ease of assessing bots reliability**—the popularity of the game make it relatively easy to find experienced players, who are able to assess a bots believability using their knowledge about the game and experience they have gained while playing with other players.

The choice of StarCraft as the test environment has also its drawbacks. The most important one (when taking into consideration the described system) is probably uniqueness. There are three different races in StarCraft, each of them having its own unique units and buildings. The manner of playing depends largely on the choice of the race both ours and our opponents (as the strategies successful against one race may be completely ineffective against the others). Thus, there is a greater need to adapt to the game style of the opponent. This uniqueness is also evident in differences between units themselves. Unlike games such as Age of Empires, where a series of units is available for each formation (that of course differ in statistics, but have similar characteristics and purpose), in StarCraft it is difficult to distinguish any division of units, as each of them has strong individual properties.

It is then a difficult task to describe the desired bots game using policies, as they need to become more specific. In other words, when creating bots the focus should shift from "what to do" to "how to do"—for example, to precise control of the units and an appropriate use of their properties. It is especially important when creating bots whose purpose is to play with experts, who not only can use these features skillfully, but also employ advanced techniques.

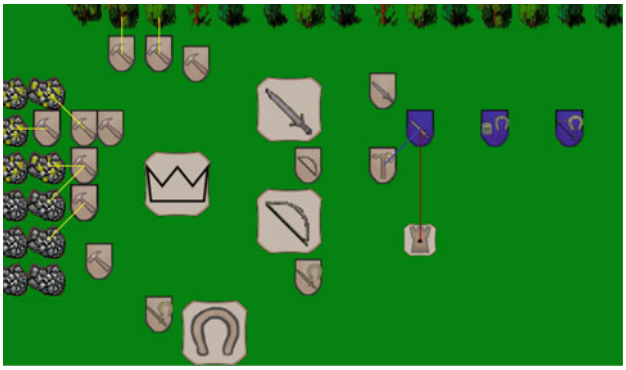


Fig. 10.4 First test environment. A *white* player base is attacked by a *blue* player

Fig. 10.5 A screenshot from StarCraft, the second test environment

To simplify the system development we have decided to choose *Zerg* as a race. We have also decided to limit unit types used to the simplest ones: *Drones*, *Overlords*, *Zerglings*, *Hydralisks* and *Mutalisks*.

10.4.2 Personality

The described system was used in two very different environments. This required appropriate choice of parameters and policies for each of them, to reflect the environment specificity to highest possible degree. We now present these policies

and conditions that we applied to the second version. Afterwards, we will briefly describe the main changes that occurred when switching between environments.

The following traits were introduced:

- Aggressiveness
- Massive
- Defense
- Economy
- Technology
- Quick Development

Aggressivness and Massive were used mainly to specify a manner in which bots was attacking an enemy. Agressivness referred mainly to the frequency of the attacks, and Massive to the size of the armies that were used. Defense, Economy and Technology described the importance of corresponding activities (building fortifications, creation of workers and resource bases, researching upgrades). Quick Development was used as a general indicator of how soon in the game a bot tried to create new buildings and resource bases.

These traits were used to describe metaconditions of following types:

- **Military strength**—represented by a type of units (*All*, *Unassigned*) and a number—strength of military units of given type (for simplicity, unit strength was calculated based on unit's hitpoints) ,
- **Time elapsed**—represented by a type of an event (which could be, among the others, *Game phase changed*, *Resource Base established*, *Enemy base atacked*, *Base attacked by an enemy*, *Building created*), type (*After*, *Before*) and a number of ticks,
- **Unit count**—containing a type of units (an actual unit type, like Hydralisk or Overlord, or *Military*, which contained all the units except for Drones and Overlords) and an integer value,
- **Building count**—represented by a type (*Any*, *Fortifications*, or an actual building type, such as Hatchery or Extractor) and a number,
- **Upgrade count**—containing a single value (representing a number of upgrades that were researched),
- **Resources**—containing a single value and a type (*All below*, *All above*, *Any above*, *Any below*),
- **Composite**—described by a type (*And*, *Or*) and a list of conditions,
- **Const**—with two possible values: *True* and *False*.

These metaconditions, in turn, were used to specify metapolicies such as:

- **Attack enemy** (*First* and *Next*)
  - Launch condition—specifies when an attack should be launched,
  - Required strength—specifies a required strength of the army,
  - Maximal strength—describes the maximal strength of the army that can be used in an attack,

- **Establish resource base** (*First* and *Next*)
  - Establish condition—describes a condition to create a new resource base,
  - Stop condition (for *Next* only)—defines when to stop creating new bases,
- **Fortifications creation** (*Main base* and *Resource base*)
  - Start condition—specifies when to build fortifications,
  - Strength—describes how strong they should be,
- **Units creation**
  - Workers on minerals—specifies how many workers should be used to collect minerals,
  - Workers on gas—specifies how many workers should be used to collect Vespene gas,
  - Units limit—describes when to stop creation of military units,
- **Building creation**
  - Morph to Lair—defines when to *morph* (transform) Hatchery into its more advanced version—a Lair,
  - Build Evolution Chamber, Build Spire—specifies when specific building should be built,
- **Hatchery creation** (*First* and *Next*)
  - Build condition—defines when to build a Hatchery,
  - Stop condition (for *Next* only)—specifies a condition to stop building Hatcheries.

One of the major changes was introduced to the algorithm for creating buildings. In the first environment, the units were created in military buildings. Each building could construct one unit at a time, so the speed of unit construction was much limited by their number. The policy for creation of buildings described when to construct the first military building, when to increase the number of buildings and when to stop creating new ones. The type of building to be constructed was selected by the bot's AI mechanisms. In StarCraft units are created only in the main building (Hatchery), however, to construct the unit, a player needs to possess a proper military building (for example, to produce Hydralisks, a Hydralisk Den is needed). We have therefore adjusted the policies to describe when to build new Hatcheries, and separately when to construct buildings of other types.

Other policies, especially those of general significance, remained almost unchanged. In some cases proper adaptation of their application was necessary. For example, in our first environment units could freely overlap (there could be arbitrarily many units on one map field) and each could gather resources independently of the others. However in StarCraft only one unit may gather minerals from one minerals deposit at a given time. Therefore while in our first environment only the total number of workers mattered, and their distribution between resource bases had no practical influence on the game, for StarCraft we had to consider the number of workers for

each resource base separately, because too many drones in one base could result in them stalling and waiting in queue for their turn instead of working.

Another change was in the policy for creating units. At first, it was described similarly to that for buildings specifying when to start, continue and stop creating units. In StarCraft we changed our approach and came to the conclusion that units should just be created whenever it is possible.

10.4.3 The Bot

In the description of the bot we present the version that was used in the second test environment, which slightly differs from the earlier version. The structure of the bot is presented in Fig. 10.6. The central part of the bot is a goal-oriented *Strategic Manager*. It uses a personality created by the personality system to determine which tasks should be currently executed. Strategic Manager creates a goal for the selected task, assigns a priority to it, and pushes it to a *Goal Queue* (in fact, it is a priority list, not a queue). Next, Strategic Manager iterates through the Goal Queue and depending on the type of the goal and the state it takes appropriate action. The Strategic Manager is assisted by lower level managers: Military, Economic and Development Managers. For example, the Military Manager manages military units, selects the type of units to be built, and finds the best place to attack the opponent etc. Each of the managers has access to the game objects that store information about the state of the game and the map, and two support tools: an *influence map* [17] and a *resource tree* [18].

A *Build Queue* is a priority queue that stores information about objects to be constructed (for buildings) or invented (for upgrades). If at a given moment the Build Queue is empty, the Strategic Manager, depending on several factors, selects a unit that should be built. One of these factors is the demand for workers—each of the bases sets out how much it needs a new worker (as a value from the interval [0, 1]). The base with the highest demand is selected, and the demand is compared with the demand for military units, provided by the Military Manager. Depending on which of these values is higher, either a worker or a military unit is built. The demand for workers is based on the current and target number of workers and is modified by the Economy parameter of the bot's personality. Thatnks to that, bots with a high setting of Economy parameter develop their economy much faster.

To decide what type of military unit should be built, a *Resource tree* is used. It is a simple structure enabling to calculate the proportions between different formations and units within them. The way preferences can be used for different formations was described in Sect. 10.3.2. The use of resource tree was more natural in the first of the used environments, where the division into formations was very clear, and we could make the choice of not only units, but also buildings and technologies based on the player's preferences easily. In StarCraft, due to the problems with finding appropriate classes for formations, we simplified the resource tree structure and used only preferences for the specific types of units.

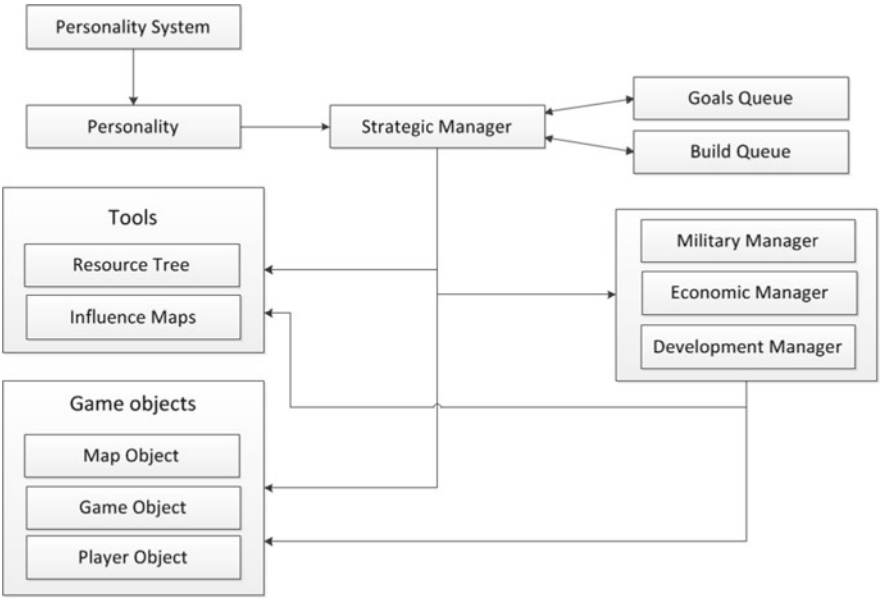


Fig. 10.6 Bot structure

Decision making is also supported by an *Influence map*. Its cells contain information about the value of units (separately for military units and workers), value of buildings, strength of fortifications and amount of resources of each type. This map is used for example for finding locations for new resource bases or areas where to attack the enemy. To reach the final decision a summary map is created based on the influence map. It is a linear combination of various data from the map, with the parameter values from the bot's personality as coefficients, resulting in only one numerical value for each of the map cells. For example, when looking for a place to establish a new resource base we can consider the amount of resources and the enemy presence in potential locations. We can choose to prefer wealthy locations, even if they are close to the enemy, or to settle with smaller, but safer ones. For example, the parameter of Expansiveness described in Sect. 10.3.2 can be used to modify a weight assigned to enemy's units and structures. Similar use of other parameters caused the bot's behaviour to reflect its assigned personality to a high degree.

10.5 Results

To verify the system a number of experiments was conducted. Their design and results will be now described.

Table 10.1 A tournament for identifying an optimal number of workers

$N_{init} \backslash N_{goal}$	16	21	26	31	Sum
9	3	3	2	2	10
14	7	8.5	12	9	36.5
19	10.5	12.5	9	7	39
24	5.5	6.5	12.5	10	34.5

Numbers denotes points scored. Each setting was used in 15 games

10.5.1 Identifying Good Condition Values

As already mentioned, we have tried to use the system to create a set of well-playing bots that could serve as a reference. A series of tournaments was organized to identify "good" values for policy parameters. Each experiment was designed to verify values for one or two parameters. We have created a number of almost identical personalities that differed only in the values set to parameters in question, to which we have assigned some arbitrarily chosen values. These personalities become our experiments subjects and were assessed in a round-robin tournament. For every victory the winner was awarded with one point. To reduce the matches duration a time limit was introduced (50,000 game ticks), after which the match was called a draw and each player was awarded with 0.5 point.

In order to present the method in greater details, a simple experiment will be described. It was designed to identify appropriate values of the *workers* policy—specifically, for the conditions of the initial phase completion and stopping workers production. Only a simple conditions of having a fixed number of workers were taken into consideration. For both of these conditions four test values were chosen ($N_{init} \in \{9, 14, 19, 24\}$ and $N_{goal} \in \{16, 21, 26, 31\}$, respectively), resulting in 16 different configurations. The results of the experiment for one of the base personalities used are presented in Table 10.1. It can be easily noticed that the results for $N_{init} = 9$ are significantly worse than for the other values, hence, it can be safely assumed that this value was too low. But differences among the remaining scores were relatively small (and they varied for different personalities used), and there was no clear correlation between parameters tested and the results.

The experiments were a moderate success—even though most of them allowed to slightly limit the interval of "good" values for conditions, information concerning the remaining conditions that was acquired in these experiments has not been of great practical importance. On the other hand, such experiments can be used to balance game objects' statistics. For example, we can create identical personalities with only preferences for different formations and/or units changed and run a tournament to check whether some of them have an advantage over the others.

10.5.2 Describing Custom Strategies

The first experiment run in the StarCraft environment was to examine how accurately (if at all) custom strategies can be reproduced. A few players (they could be described as regular players) were asked to describe in their own words strategies they are acquainted with—with the emphasis on those they use themselves. Some examples of following answers are (translated into English): "I build an exp[2] or two, then I attack my enemy's exps if he has any", "I build drones to the second Overlord, then I build an exp and simultaneously I begin to raise an army, I attack when I have at least two or three groups of Hydralisks, "Primarily I try to destroy my enemy's exps", "I gather a group of Hydralisks and build an exp after the third, sometimes the fourth Hatchery", "I make upgrades only when I have too much resources", "I make a speed upgrade for Hydralisks as quickly as possible".

The following observations were made after a short analysis of the received answers:

- The majority of the descriptions could be embedded in the structure of tasks and policies that the system is based on—they refer to such activities as launching attacks or establishing resource bases, including when a given action is performed or which condition needs to be fulfilled. Most often those descriptions mentioned particular events and numbers of units of a given type that a player has,
- The conditions concerning the numbers of units were quite general: "two or three groups", "few [units]", "until I have enough [units]",
- The descriptions mainly concerned the initial phase of the game i.e. the first attack and the establishment of a resource base. Descriptions concerning mid-game contained such statements as: "I keep track of what my opponent is doing" or just "it depends",
- Specific activities are sometimes present in the descriptions, e.g. a discovery of a particular technology.

It seems as if the division into tasks as well as treating them as a certain whole is natural, and thus desirable behaviour of a player can be intuitively described in terms of policies. It seems also natural to distinguish activities that are to be made for the first time (for example the first attack from the policy describing all attacks). While the majority of players indicated when they launch their attack quite precisely (before or after building a resource base or fortifications), they did not formulate any explicit rules concerning the following attacks. They explained—quite aptly—that it depends on the current situation on the map.

A certain imperfection of the system became apparent—it does not allow us to strictly control the type and order of objects created (e.g. units) or technologies discovered. It is possible to try to do that for example by manipulating preferences, but strict control is difficult or even impossible if a player wants to perform several actions in a particular order. A better solution would be, for example, to define a policy describing researching technologies—such a policy could

[2] *exp* a short for *expansion*, which is what resource bases in StarCraft are sometimes called.

Table 10.2 A tournament for testing effectiveness

Enemy race	First series	Second series
Zerg	8	9
Protoss	7	9
Terran	3	8

The number of our bot wins is shown (out of 10 matches). In the first series a single personality was employed in all the matches. In the second one, for each race a specially created personality was used

contain conditions concerning the beginning of the discovery of a few key technologies, and a general conditions concerning other technologies. Another possible solution is to simply provide a list of technologies that are to be researched in the desired order.

A few strategies were chosen from the received descriptions (so that they presented a wide range of behaviours and differed between each other) and were described within the system. As expected, it was a relatively simple task to determine the policies; the majority of behaviours could be described using uncomplicated conditions that we have implemented. However, some of the strategies, such as the one concerning the fastest possible attack on the opponents resource bases, required certain modifications both of the personality system (adding a new event—*building a resource base by the opponent*), and of a bot (recognizing this event and an appropriate response to it).

Bots created this way were then verified: a series of games with the default bot was run for each bot. Their behaviours were observed to check their compliance with the adopted strategy. Random values were chosen for those aspects which had not been described or did not have any significance for a strategy. These experiments revealed the imperfection of the goal management algorithm, which sometimes hampered accomplishing the following goals in the situation when it was possible to fulfil them without any adverse influence on the current goal. Despite this drawback, it was observed that the behaviours of the bots quite accurately corresponded with the custom strategies, thus the conclusion that the system is suitable for simple strategy modelling was made.

10.5.3 Effectiveness Testing

The effectiveness of the bot was verified in a series of games against the default StarCraft bot. We have arbitrarily chosen values of traits that were used in all matches. Ten matches were played against each of the races. The results of the experiment are shown in Table 10.2.

The differences between the results for different races can be easily explained by aforementioned distinctions drawn between races. Terrans, with their great defensive capabilities proved to be a very tough opponent. A Terran player usually used a

few Bunkers, strong structures capable of holding an enemy attack for a long time, together with Siege Tanks in a Siege mode, with their splash damage decimating large groups of our units. Protoss and Zerg races have relatively weak defences, and even small groups of units are able to destroy them. In the case of matches played against a Zerg player, the only matches ending in defeat were those in which we were rushed. As for the matches with a Protoss player, we were usually able to win quickly with a series of fast attacks. In the remaining matches, however, the Protoss player was able to survive them and then—after creating more advanced units—to obtain a victory.

After this experiment we have tried to adjust traits manually in order to create personalities that are effective against a specific race. It turned out to be relatively easy, and the observations from the previous matches were of key importance here. The experiment was repeated, this time using different personality for each race, and we were able to win the majority of matches.

10.5.4 Believability Testing

The last of our experiments consisted of a series of games played between the bot and a few human players. Its purpose was to test the receipt of the bot's AI by humans in two aspects. First, we wanted to check whether the bot was perceived as credible (meaning: could the bot's AI be distinguished from the way a human would play the game). Secondly, we wanted to check whether players will be able to notice the difference between bots using different personalities (a positive response to this question would indicate that the described system is suitable for modeling various personalities). The experiment design was as follows: we chose four different sets of traits. One of them was used as a point of reference—all the parameter values has been set to 0.5. Each player played five matches, the first two with the reference bot, the next three with the others. After each game the players completed a short survey, in which the seven-point Likert scale responses were noted in several questions, including:

- the extent to which the current match was similar to the previous one,
- how much the bot's AI resembled the way a human would play the game (overall, method of attack, base building, resource bases expansion),
- to what extent the bot's AI could be described as:
  - difficult to win with,
  - playing aggressively,
  - concerned with the defence,
  - caring about the economy,
  - caring about the technological development.

Pretty soon it turned out that most of the players were able to relatively quickly overcome our bot. From the observation of game replays we concluded that one of the main reasons was the way our bot attacked enemy bases. Attacking consisted of selecting a number of units and issuing the command *Attack*, specifying the center of the opponent's base as target. Groups of units marching through the map gradually stretched, until finally all the units went in a single line. Such stretched group of units usually became an easy target for a condensed group of enemy units, which was able to destroy the marching units one after another as they were arriving, with very small losses. This way, the players gained a large advantage which they were able to quickly turn into victory. Another problem was ineffective planning of construction of units and buildings. Even in situations where the actions of the bot and players were very similar, the players were often able to achieve their goals more quickly and efficiently.

These problems lead us to a decision to stop the experiment, deciding that the bot's poor performance would have a too big an influence on the obtained results, and that results obtained from games where the players were able to defeat the bot so quickly would be meaningless. In fact, in most games the players did not see the difference between bot personalities using different sizes of armies (which for some games were significant, in one game the bot used armies consisting of 12 units, in the subsequent one—of 25 units). The players also had the tendency to regard as more aggressive these bots which attacked their weakly defended resource bases, even when these attacks were rarer and used smaller armies that attacks targeted at their well defended main bases or big groups of units. Therefore, it seems that the control of aggressiveness should rely primarily on the proper selection of targets and the size of the army (relative to the opponent's strength), and not on the frequency and absolute size of the attacking army. While this is consistent with our initial assumption that the personality should be only an addition to the AI, we initially expected that the results obtained by using simple absolute conditions not requiring deeper analysis of game state would be much better.

It is quite an interesting observation that some of the players assessed the credibility of some of the aspects of bot's behaviour by comparing it to their own actions. For example, they justified that the way the bot built its fortifications or resource bases was credible to them, because they played similarly (in terms of a number of fortifications constructed and bases established). Although this rule does not seem to work the other way (no one justified the incredibility of the bot by pointing out discrepancies in the number of bases or buildings constructed by the bot compared to how the player played), it seems to contain some indication that the creation of credible bots can be based on mimicking the human's way of play.[3]

[3] It is interesting that such imitation may have an impact not only on the credibility, but also effectiveness of the bot. The winner of one of the tournaments (Tournament 3: Tech-Limited Game) during StarCraft AI Competition at AIIDE 2010 was *Mimic Bot*, which tried to imitate the movements of the enemy. Results of the tournament can be found on http://eis.ucsc.edu/Tournament3Results.

10.6 Conclusions and Future Plans

This chapter consists of four parts. In the first one, we discussed various issues related to players and their skills. This discussion is by no means complete, its goal is rather to signal some problems that should be considered when creating believable RTS bots. That part ended with a short discussion of the notion of believability.

In the second part, we described a personality system built in accordance with the issues discussed beforehand. A personality consists of policies, which describe a manner of execution of different tasks and a profile—a special policy that provides general bot characteristics. Using the selected examples we discussed how one can distinguish different tasks and corresponding policies and choose adequate conditions to describe them.

In the third part, we briefly described test environments, components of the personality system we have used and the structure of the bot, providing also some details about the implementation.

In the last part, the results of the experiments conducted were presented.

The results of the experiments show that the personality system can be used to model custom strategies, even though there are some limitations. The process of personality creation allows us to easily create bots with the desired general characteristic by using traits. By using multiple metaconditions for parameter descriptions we may cause the system to create different personalities for the same sets of traits. Controlled randomization of trait values allows us to keep unpredictability at the desired level.

Verification of the ability of the system to create credible and entertaining bots is troublesome, because its behaviour is dependent on the bot's AI systems. One of the crucial tasks for believable RTS bots, i.e., the ability to intelligently react to game state changes and to adapt to opponent moves, lies outside the scope of the personality system. In our case, the simplicity of the bot had great impact on the human perception of the bot behaviour. Throughout the chapter we gave examples of how advanced conditions can be used to achieve greater control over the bot behaviours and how to adjust difficulty level to the players' skills. Their implementation would require adequately advanced AI mechanisms for game state analysis. In the future, we plan to focus on implementation of such mechanisms and construction of a more advanced bot that would allow us to utilize the personality system to its full capabilities.

Thanks to the simple structure of the system and its flexibility there are numerous possible extensions and improvements. One of them is to assign to each metacondition within a given parameter description the probability of being selected, so that more typical conditions would be chosen more frequently. We can take one more step and assign a probability distribution to the metacondition's values range. This distribution could be also related to the selected level of difficulty.

Another direction for further research is applying evolutionary algorithms to find good strategies. In a population comprising of personalities, mutation and crossover operators have obvious meaning—modifying some of the policies for mutation and policies exchange between two individuals for crossover. Naturally, we would not be

interested in finding a globally optimal individual (even if it exists), but to find a set of good individuals. They could be analyzed later and, on that basis, crucial policies and their parameters could be determined, allowing us to create more effective strategies.

Acknowledgments Special thanks are due to Marta Buchlovská for her help with the design of the last experiment. Also, thanks to all the participants of that tournament.

References

1. http://us.blizzard.com/en-us/games/sc/. Accessed Jan 2011
2. http://www.microsoft.com/games/empires/. Accessed Jan 2011
3. http://starcraft.wikia.com/wiki/Actions_per_minute
4. http://arstechnica.com/gaming/news/2010/07/excellence-of-execution-video-of-starcraft-mastery.ars
5. http://wiki.teamliquid.net/starcraft/Mutalisk_vs_Scourge_Control#Method_2_:_The_Chinese_Triangle_Method. Accessed Jan 2011
6. http://wiki.teamliquid.net/starcraft/Mutalisk_Harassment#Grouping
7. http://wiki.teamliquid.net/starcraft/Magic_Boxes
8. Beume, N., Hein, T., Naujoks, B., Piatkowski, N., Preuss, M., Wessing, S.: Intelligent anti-grouping in real-time strategy games. In: IEEE Symposium on Computational Intelligence and Games, pp. 63–70 (2008)
9. Buro, M.: Call for AI research in RTS games. In: Proceedings of the AAAI Workshop on AI in Games, pp. 139–141 (2004)
10. Buro, M., Furtak, T.M.: RTS games and real-time AI research. In: Proceedings of the Behavior Representation in Modeling and Simulation Conference, pp. 51–58 (2004)
11. Chung, M., Buro, M., Schaeffer, J.: Monte Carlo planning in RTS games, In: IEEE Symposium on Computational Intelligence and Games (2005)
12. Hingston, P.: A Turing test for computer game bots. IEEE Trans. Comput. Intell. AI Games **1**(3), 169–186 (2009)
13. Livingstone, D.: Turing's test and believable AI in games. Comput. Entertainment **4**(1), Article 6 (2006)
14. Olesen, J.K., Yannakakis, G.N., Hallam, J.: Real-time challenge balance in an RTS game using rtNEAT. In: IEEE Symposium On Computational Intelligence and Games, pp. 87–94 (2008)
15. Sailer, F., Buro, M., Lanctot, M.: Adversarial planning through strategy simulation. In: IEEE Symposium on Computational Intelligence and Games, pp. 80–87 (2007)
16. Sweetser, P., Johnson, D., Sweetser, J., Wiles, J.: Creating engaging artificial characters for games. In: Proceedings of the Second International Conference on Entertainment Computing, pp. 1–8 (2003)
17. Tozour, P.: Influence mapping. In: Deloura, M. (ed.) Game Programming Gems 2, pp. 287–297. Charles River Media, Hingham (2001)
18. Tozour, P.: Strategic assessment techniques. In: Deloura, M. (ed.) Game Programming Gems 2, pp. 298–306. Charles River Media, Hingham (2001)

Chapter 11
Making Diplomacy Bots Individual

Markus Kemmerling, Niels Ackermann and Mike Preuss

Abstract Diplomacy is a round-based strategy game with simple rules but a real-time component as players move in parallel. It also emphasizes negotiation between players, which is difficult to realize in a bot but essential to achieve a human-like playing style. In a previous work, we found that in Turing Tests, players mainly use three usual shortcomings of current bot implementations to identify them as computer players, a certain level of playing strength which makes planning necessary, the avoidance of mistakes, that is moves a human most likely would not use, and a meaningful communication. According to previous results, it seems to be especially hard to combine well-playing with a human-like move style. While the communication problem has already been treated successfully at least for short games, currently known CI-based bots do not plan ahead. We present a planning Diplomacy bot which employs the negotiation kernel of an already existing bot and apply our believability measure technique in a new and interesting way. Instead of learning how to minimize the number of bad moves according to a mixture of games of several players—this had proved difficult as different players regard different moves as bad or computer-like—we go a step into the direction of mimicking human player styles by using only saved games of one person each. We thus effectively create a bot which is playing well, including planning, uses basic communication and partly inherits the playing style of a specific human player. The different obtained bots are compared according to playing strength and believability.

M. Kemmerling (✉)
Robotics Research Institute, Section Information Technology,
Technische Universität Dortmund, Dortmund, Germany
e-mail: markus.kemmerling@tu-dortmund.de

N. Ackermann · M. Preuss
Chair of Algorithm Engineering, Computational Intelligence Group,
Department of Computer Science, Technische Universität Dortmund,
Dortmund, Germany

M. Preuss
e-mail: mike.preuss@tu-dortmund.de

P. Hingston (ed.), *Believable Bots*, DOI: 10.1007/978-3-642-32323-2_11,

11.1 Introduction

Diplomacy is a round-based strategy game with simple rules but a real-time component as players move in parallel. From the game mechanics viewpoint, it may thus be regarded as most simplistic *realtime strategy* (RTS) test game. However, playing Diplomacy against computer programs (called 'bots' in the following) still has a remarkably different character from playing against humans as the game strongly emphasizes negotiation. Different from the diplomacy options known from computer strategy games, one is completely free to suggest very complex plans and keep the promises—or not. This level of interaction is surely very hard to attain for a computer program, maybe impossible in the foreseeable future. But even if language is restricted to a computer-interpretable vocabulary (as e.g. realised in the Diplomacy AI Development Environment (DAIDE)[1]), computer players have to cope with terms like trust and betrayal and need to predict what the other six opponents are up to. Not only in the next move, but also over the next few rounds. Human players heavily utilize their intuition for deciding when to trust whom, but this fallback decision help is unavailable for bots. Nevertheless, appearing to be intelligent in negotiations is essential to achieve a playing style that makes the game interesting for human opponents. Surprisingly, this is possible at least for rather short games as they are usually played by humans (about 5–7 years which makes 10–14 rounds). In a previous work [12], we found that in Turing Tests, players mainly rely on three shortcomings of current bot implementations to identify them as computer players, namely:

- a certain level of playing strength which makes planning necessary
- the avoidance of mistakes, that is moves a human most likely would not use
- meaningful communication.

According to our previous findings, it seems to be especially hard to combine well-playing with a human-like move style. Possibly, this is just a matter of effort because it causes a lot of work to make a bot play well and the same holds true if it shall even communicate well. However, there are certainly also conflicts between strategy finding and negotiating, the most notable one is timing. Concrete strategical planning of the next move is only possible after the last round's moves have been resolved by the game because one cannot even be sure that the own move set can be executed due to conflicts with the other players. However, round times are usually limited, so not too much time can be spent on finding a good move set for the next round before starting negotiating. Additionally, agreements made concerning single moves can influence all moves designated for the next round, in the extreme case requiring a completely new strategy.

While the communication problem has already been treated successfully at least for short games, currently known *Computational Intelligence* based bots do not plan ahead. We present a planning Diplomacy bot which employs the negotiation kernel of an already existing bot and apply our believability measure technique in a new and interesting way. Instead of learning how to minimize the number of bad moves

[1] http://www.daide.org.uk

according to a mixture of games of several players—this had proved difficult as different players regard different moves as bad or computer-like—we go a step into the direction of mimicking specific human player styles by using only saved games of one person to generate one style. We thus effectively create a bot which is playing well, including planning, uses basic communication and partly inherits the playing style of a specific human player. We do not proceed to largely change bot behaviours in order to mimic a human playing style better, but at least we provide a number of variants of our bot so that the one with the highest believability count can be chosen. Of course, one could turn the adaptation to one human player into an optimization problem and approach it with many more degrees of freedom, but this is out of the focus of this paper. Moreover, playing experience tells us that for a human, it may be much more interesting to deal with several different bots than facing six times the same bot, regardless how believable it is.

However, at first we start with a small primer for the Diplomacy game (Sect. 11.1.1), followed by an overview of the related work, a recently developed complexity estimation, and some thoughts about how to obtain believability in a diplomacy bot (Sects. 11.1.2–11.1.4). Some of this material is taken from [13] for sake of completeness. Section 11.2 introduces the Stratogiator, our base bot, and describes recent extensions concerning map and move weights and the look-ahead planning. We then proceed to the core questions of the paper, namely if the planning Diplomacy bot is still as believable as its non-planning predecessor and if we can move into the direction of a bot that is highly believable for one specific player.

11.1.1 Diplomacy

The board game Diplomacy[2] is a strategy war game focusing on informal negotiations between players who may reach agreements on moves, contracts and alliances which are not binding. Action takes place in Europe just before World War I. The goal is to conquer most of Europe either alone or together with allies. The game was created by Calhamer [4] in 1954 and was/is published by Games Research (in 1961), Avalon Hill (in 1976), Hasbro's Avalon Hill division (in 1999), Wizards of the Coast (in 2008) and others.

The standard board (there are variants of the game on other boards,but these are disregarded here) is divided into 56 land and 19 sea regions (provinces) whereas 34 land regions contain supply centres, which can support one army or fleet each. Each of the players commands the units of one of the factions Austria (AUS), England (ENG), France (FRA), Germany (GER), Italy (ITA), Russia (RUS), and Turkey (TUR). The game ends when a single faction or an alliance of countries attains supremacy with at least 18 supply centres.

Diplomacy consists of two rounds (spring and fall) per year, starting from 1901. When playing with humans, each round consists of a negotiation and a movement

[2] http://www.wizards.com/default.asp?x=ah/prod/diplomacy

phase (negotiation phases are disabled in no-press games). In a computer environment, one can wait until the last player submits a set of moves or one employs a fixed deadline until all move sets have to be delivered. Negotiations are in principle not bound to these deadlines but can go on concurrently. However, they can of course not influence strategy selection for one round after the deadline.

In every round, a unit can execute one of four possible move types: either hold its position, move (which is an attack if the move goes to a region that is neutral or belongs to another faction), or support another unit in moving or holding. There are two unit types, armies and fleets, and armies can only move onto land regions, whereas fleets are allowed on sea and coastal land regions. Armies and fleets can execute an additional specific move type together, the convoy, which means that one or multiple fleets 'carry' an army as if they were a bridge with no movement cost. As this move type is usually only important for England, many bots do not implement it.

The complexity of Diplomacy mainly stems from the simultaneous execution of moves. As in each province one unit can remain at most, conflicts are resolved by the majority rule, that is, a province can be conquered only with a superiority of units against an enemy. New supply centres may be conquered in fall moves only and the number of units is adapted to the number of supply centres in winter. Winter is an additional phase subsequent to fall in which units may be build or have to be removed. New units can only be build in one of the 22 home supply centres which are a subset of all supply centres, and every faction can only build in its own home supply centres. The game does not incorporate any random effects. However, interactions between different factions during movement resolution can be quite complex as all supports and convoys are prevented if the supporting or convoying unit is attacked. This is an indirect possibility to collaborate with another player. A more direct one consists in giving supports to an attacking unit. Whether indirect support may look as having happened by chance, the direct support suggests that the two players closely work together and will probably continue doing so during the next rounds.

Besides the Hasbro Diplomacy (computer) game, written by Microprose and released in 2000, there exist several computer based versions of Diplomacy, more or less supporting the development of Artificial Intelligence (AI). The best known of these is DAIDE that was started in 2002 and consists of a communication protocol and a language (syntax and semantics) for negotiations and instructions. The diplomacy server (AIserver) and a human interface program (AImapper) allow for games played by bots and humans.

11.1.2 Related Work

Works on Diplomacy bots date back to the 1990s and usually focus on either the negotiation aspect (e.g. Kraus et al. [14] suggest the *Diplomat* bot which is modeled as a *Multi-Agent System* (MAS)) or rather on the strategy aspect which is often approached with game theoretic techniques as suggested by Loeb [16] (introducing the *Bordeaux Diplomat*). After some years of standstill in publication history,

Shapiro et al. [22] present their bot concept of bottom-up self-learning Diplomacy Strategy by means of *Temporal Difference Learning* (TDL) and pattern-weights. Shaheed [21] suggests another bot named *Diplomat*, combining strategy and negotiation with an opponent model, but without look-ahead into the future. Booijink [3] establishes an *Evolutionary Algorithm* (EA) based strategy forming module for his MAS representation of Diplomacy, without negotiation component. Diplomacy games without negotiation are usually called *no-press* games. Johansson and Håård [10] also focus on the no-press variant, suggesting the HA AI bot, which establishes strategies by valuing provinces and unit threats and also provides different bot characters (berzerk and vanilla) with different properties in range of sight and blurring strength.[3] Johansson [9] further discusses similarities and differences of different strategic board games as Diplomacy and Risk and the consequences for the bot design. Keaveney and O'Riordan [11] continue this reasoning and suggest to use these games as basis of an abstract realtime strategy game model.

Currently, the interest in Diplomacy as game and/or testbed for agent models seems to be renewed, as Fabregues and Sierra [6, 7] state. Presumably, this is because modern AI techniques have improved to a level that makes them more capable to deal with negotiation and opponent modeling in near real world situations. This direction has also been taken up by others. Where Ribeiro et al. [20] mainly focus on the different personalities a bot may express, Kemmerling et al. [12] provide the *Stragotiator*, a bot that has been reported to sometimes trick humans in a Turing Test (at least for short games). It has been developed with focus on believability and negotiations, modeling a specific human playing style described by Windsor [26] as *Classicist*. The Classicist is a loyal team-player, tries to maintain the groups welfare and does not violate contracts. While the Diplominator [25] was used as initial codeframe, he was enhanced according to the archetype of the Bordeaux Diplomat [16] and by using an EA for movement optimization. The obtained bot was later improved by Ackermann [1] who adopted an evolutionary movement planning as additional module calculating current opponent moves approximately to look one step ahead. However, the best playing no-press bot to date is van Hal's *Albert* [24] which uses a mixture of clever pruning techniques in a main algorithm that may be described as competitive coevolution.

11.1.3 Diplomacy Complexity

In Diplomacy, all seven players set up their next moves simultaneously, without knowledge about the moves of the others. This makes the game a highly complex game in terms of possible moves. Booijiink [3] already reasons that Diplomacy is more complex than *Go*. According to Loeb [16], there are approximately 4.4×10^{15} possible moves in the first round. Shapiro et al. [22] give an estimate for the scenario of all units with $\approx 1.7 \times 10^{27}$ possible moves. Making use of some simplifying

[3] Blurring distributes province values to the neighbouring provinces by gradual decrease.

assumptions, Kemmerling et al. [13] develop a more accurate estimate we recapitulate here because to our knowledge, it gives the best impression of how many move possibilities are available for any round, given that the number of currently existing units is known.

In standard Diplomacy, 75 regions exist, of which 19 are sea, 42 coastal, and 14 interior provinces. The number of possible unit placings on the map are given in Eq. 11.1, ignoring unit affiliations to factions and using exchangeability of individual units. However, it take account of coastal provinces which may be occupied either by a fleet or an army.

$$P(n) = \sum_{u=0}^{max\{n,33\}} \binom{42}{n-u} \cdot 2^{n-u} \cdot \binom{33}{u} \tag{11.1}$$

The number of placements of $n \leq 34$ units without repetition is given by $P(n)$, $u \leq 19 + 14 = 33$ is the number of units on non-coastal provinces and $n - u$ on coastal provinces. On non-coastal provinces, the unit type is determined by the region type, otherwise two possibilities exist. Placing u units on 33 provinces without replacement the alternatives are given by $\binom{33}{u}$. The remaining $n - u$ units have to be distributed to the coastal provinces with $\binom{42}{n-u}$ possibilities, where 2^{n-u} is the number of possible army-fleet combinations. For $n = 34$, that is the maximum number of units on the board, Eq. 11.1 results in a number of 4.09×10^{27}. Normally, the maximum number of units is available after the fourth round.

For moves, they assume complete move information concerning units taking part in any support or convoy operation. Let p_i be the set of neighbours of any province i. Then Eq. 11.2 gives the common neighbours of provinces i and l.

$$\varphi_{i,l} = \{p_i \cap p_l\} \setminus \{i, l\} \tag{11.2}$$

Equation 11.3 gives the number of different moves a unit may execute without considering convoys. This number $M(i)$ is calculated by adding up one hold 'move' plus $|p_i|$ moves to neighbouring provinces, plus a hold support for all occupied neighbouring provinces, plus a move support for all common neighbours of i and l, plus move supports for indirect neighbours.

$$M(i) = 1 + |p_i| + \sum_{l \in p_i} \delta_l \cdot (|\varphi_{i,l}| + 1) + \sum_{k \in p_i} \sum_{\{j \in p_k \setminus p_i | j \neq i\}} \delta_j \cdot |\varphi_{i,j}| \tag{11.3}$$

The set $\{j \in p_k \setminus p_i | j \neq i\}$ with $k \in p_i$ characterizes the indirect neighbours of i (excluding $j = i$, the province itself, and the direct neighbours p_i). Employing operator δ_i (gives 1 if the province is occupied and 0 otherwise), the sum is build over all neighbours $k \in p_i$, conditional on occupation ($\delta_j = 1$), in this case $|\varphi_{i,j}|$ move supports are possible.

Not every move of neighbours can be supported. Neighbouring units may execute one of $|p_l| + 1$ moves (one hold or support), indirect neighbours $|p_j| + 1$ moves (one

hold), leading to Eq. 11.4.

$$M_{\text{corr}}(i) = 1 + |p_i| + \sum_{l \in p_i} \delta_l \frac{|\varphi_{i,l}| + 1}{|p_l| + 1} + \sum_{k \in p_i} \sum_{\{j \in p_k \setminus p_i | j \neq i\}} \delta_j \cdot \frac{|\varphi_{k,j}|}{|p_j| + 1} \quad (11.4)$$

Using the mean number of neighbours (4.31), common neighbours of neighbours (1.91), and common neighbours of indirect neighbours (1.41) and assuming that 33 other units are placed on the remaining 74 provinces, Eq. 11.5 is obtained.

$$M_{\text{corr}}(i) = 1 + 4.31 + \frac{33}{74} \cdot \left(\frac{1.91 + 1}{4.31 + 1} \right) + \frac{33}{74} \cdot \frac{1.41}{4.31 + 1} = 5.67 \quad (11.5)$$

With all 34 units, the (maximum) number of different moves results to:

$$5.67^{34} = 4.19 \times 10^{25} \quad (11.6)$$

11.1.4 Measuring Believability

Negotiation is a major component of the game Diplomacy and an important fun factor. Bots developed concentrating on the tactical playing skills mostly lack the ability to communicate with other players. Even if those bots are more fun to play against than weak playing bots without communication, they are still not *believable* [15] as they lack any close cooperation with human players. The desire of human players to interact with one another is shown by the success of *massively multiplayer online role-playing games* (MMORPG) like World of Warcraft. For Diplomacy, the original (board) game strongly focuses on the negotiation aspect so that the desire to cooperate with other players may be even stronger. No-press variants simply make a completely different game. However, this is the only aspect of the game handled reasonably well by current bots.

In order to fulfil the desire for interaction, the emulation of human-like behaviour of NPCs in single player games is necessary. Unfortunately, this task is not simple as believability can not as easily be modeled as score-hunting tasks in shooters where the success of an NPC rises with the number of defeated enemies.

Measuring believability in Diplomacy is also non-trivial. Test games against human players similar to the well known Turing Test [23] are time consuming and only convincing in greater number. Kemmerling et al. [12], inspired by the approaches of Yannakakis and Hallam ([27] and [28]), observed players and bots and identified certain patterns which are rewarded or punished and subsumed in formulas to generate a believability measure without the need of human interaction. This patterns are shown in Table 11.1. Each pattern was implemented as a separate

Table 11.1 Believability aspects (modules) included in the measure

Module name	Unweighted domain	Meaning
Purposefulness	$[-\infty, 0]$	Identifies when a player may reach an unoccupied supply centre but does not capture it
ChainMove	$[-\infty, 0]$	Counts chain-like moving units with quadratic punishment in their number
SelfSeizing	$[-\infty, 0]$	Identifies where more than one unit of one faction tries to stay in or move to one province
JumpingJack	$[-\infty, 0]$	Counts (with quadratic weighting) toggling of one unit between two positions
DeadLock	$[-\infty, 0]$	Counts repeated unsuccessful moves with quadratic punishment
ClusterDisconnect	$[-\infty, 0]$	Counts the number of isolated units (distance of more than three provinces to the next unit of the same faction) with quadratic punishment
SameMessage	$[-\infty, 0]$	Punishes repeatedly sent peace proposals
XDOToNeighbour	$[-\infty, \infty]$	Counts move proposals to faction of nearby units and punishes proposals to faction far away
AcceptedXDO	$[0, 100]$	Calculates the ratio of realized and agreed move proposals
AcceptedMessage	$[0, 100]$	Calculates the ratio of proposals accepted by the negotiating party

All aspects are supposed to represent good believability with high values and non human-like behaviour with low values

module with different domains for the calculated believability values. This modules, and their results, are weighted to adapt the measure to a corpus of played and humanly evaluated games to match the human's relative assessment as close as possible. Several games were conducted with one human player and different bots for which the human gave a believability ranking. To derive a general measure, not only one person but several testers were consulted. The weights were optimized to calculate believability values for the test games resulting in rankings closely matches the human ones.

For optimization, they utilized a standard $(\mu + \lambda)$-EA that operates on the weights as search space. By cubing the weights before evaluation, the large absolute differences in values for the single behaviours was reflected and the EA was enabled to modify weights on different scales while keeping a singular mutation step size. The initial population was created from Gaussian distributed random numbers with mean 0 and standard deviation 1. Mutation was done by adding a random number from $[-\sigma, \sigma]$ to every weight while recombination was done by choosing each weight from any of the two parents with probability $\frac{1}{2}$. The parameters were tuned manually resulting in $\mu = 50$, $\lambda = 50$, recombination probability 0.7 and step size $\sigma = 4.0$. Figure 11.1 shows the optimized weights. Please note that a negative sign reverts the naive interpretation of human and non-human behaviour. The last two modules had been disabled manually as the measured values

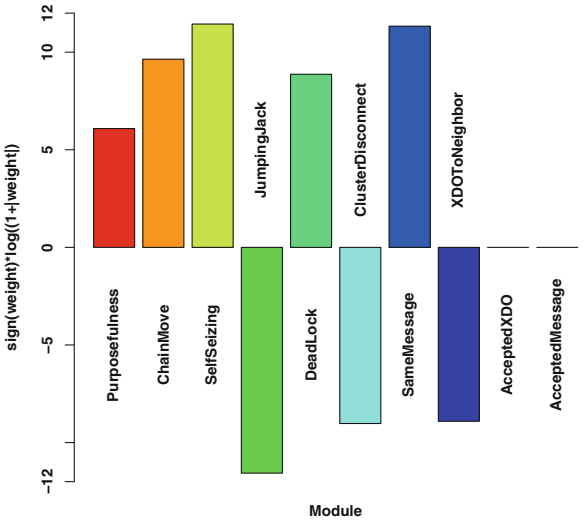


Fig. 11.1 Original weight set from [12], plotted in logarithmic scale (needed for comparison)

did not represent the players experience well (due to bugs in the implementation). Following optimization, the resulting weight set can then be used to measure believability in new games, even if no human participated. This is very important as otherwise, changing bots or even testing many variants becomes infeasible due to time constraints.

Note that although the bot behaviours we set up as basic modules of a believability measure are targeted at the Diplomacy game, the concept itself together with several of these behaviours could also be employed for another strategic game. One may also think of the different modules as identificators for different cognitive abilities, an approach that is featured in the ConsScale FPS chapter of this book (Chap. 8).

11.2 The Stragotiator

The *Stragotiator*[4] is a bot for the game Diplomacy. It is implemented in Java and designed to be used within DAIDE. As we employ this bot to realize a look-ahead planning and finally intend to establish variants of it with high believability, we first summarize the development stages and provide an insight into the used techniques.

[4] The Stragotiator is available at http://www.irf.tu-dortmund.de/cms/de/IT/Mitarbeiter/Wissenschaftliche_Mitarbeiter/Kemmerling.html.

11.2.1 Basis of Believability

The Stragotiator project started in 2009 as student project. In contrast to other existing bots, the intended goal was the development of a human-like bot that is fun to play against (or to collaborate with). Consequently, the focus in the Stragotiator project lay on good negotiation capabilities in order to interact with other players and to appear mostly human. This resulted in a bot capable of press level 20 of the DAIDE language while all other bots known in 2009 were capable of press level 10 at most.

To imitate human-like behaviour, the *Classicist* player type as described by Windsor [26] was modeled. The Classicist is a loyal team-player, tries to maintain the groups welfare and to win with his allies. He does not break alliances or promises but is unforgiving regarding betrayal. The human-like behaviour of the Stragotiator was approved in a Turing Test, where it was not only the bot with the most human-like behaviour but also the only bot who was sometimes regarded as human.

While the *Diplominator* (see Webb et al. [25]) had been used as initial code-frame, it was enhanced according to the archetype of the *Bordeaux Diplomat* (see Loeb [16, 17] among others) inheriting the modular structure. The main components are the STRAtegy core and the neGOTIATOR that have given the project its name as well as an opponent modeling module that are described in Kemmerling et al. [12].

11.2.1.1 Opponent Modeling

The opponent modeling module maintains a friend-foe matrix and an opponent's intelligence model. The friend-foe matrix is a 7×7 matrix with real-valued entries indicating the relationship between two factions. Its functionality is oriented towards the description of Loeb [17] and implements a subset of his suggestions. When two or more factions order their units to the same province, they are considered to attack each other and their friend-foe value is decreased. Low matrix entries indicate hostilities while high values indicate friendships. Additionally, the move directions are taken into account. When a faction orders their units towards a supply centre of an other faction, the behaviour is considered to be inimical. On the other hand, when a faction orders their units away from a foreign supply centre, it is considered as peaceful behaviour. Since the relationship between two factions is assumed to be symmetrical, the matrix's symmetry is always maintained. Furthermore, the friend-foe values are never cleared but updated. This results in a basic history to consider not only the current actions.

The intelligence model is a real-valued array containing an intelligence estimation for each faction. The values are calculated with the strategy core evaluating the opponent's moves from its point of view. Thus, the Stragotiator deems an opponent the more intelligent, the more similar its moves are to the ones it would itself employ in his situation (cp. Loeb [16]). It may thus rather be considered a 'similarity model' or 'compatibility model'.

11.2.1.2 Negotiator

The negotiation module is responsible for all interactions with other players. It initiate proposals, responds to request and decides about peace and ally pacts. Messages up to press level 20 can be handled, with few features missing, as demilitarized zones.

The decision whether or not to request or accept peace or alliance pacts depends on the friend-foe matrix, the intelligence of the opponent and the trust in him. After contracting such a pact, the strategy core will calculate moves for all units in the party, including that of his contracting parties. Based on this calculation, movements will be suggested to the partnering factions (the so-called XDOs).

If an opponent proposes moves, the negotiation module first checks whether the sender is trustable as described in Loeb [16]. If the other player is strongly trustable, the proposal is accepted with a probability of 0.75. If the sender is not trustworthy, the proposal is rejected with a probability of 0.75. If the decision cannot be taken according to this heuristic, the move is evaluated by the strategy core and accepted if the move is promising.

Trust towards other players is modeled with a linguistic variable with four fuzzy sets, namely mistrust, neutral, trust, and full trust. Whether the Stragotiator relies on a faction is determined by the set to which a faction specific trust value belongs to. The trust value increases when an ensured move (an accepted suggestion) was indeed executed or a good move is proposed. It decreases when an ensured move was not executed, a bad move is proposed or a peace/alliance pact that excludes the sending faction is proposed. Defuzzification into a discrete trust level is done according to maximum membership to one of the four classes.

11.2.1.3 Strategy Core

The strategy core calculates and evaluates moves for any set of factions. In order to use this module in various contexts (e.g. within the opponent modeling), it does not use knowledge about the faction actually played. However, it knows about allied, friendly (peace contract), neutral and enemy factions. This enables move calculation for mixed factions, seeking the advantage of all allied factions at once, including the actually played faction, as suggested by the implemented Classicist character.

The strategy core first checks for each unit of an allied faction if a move must be calculated. According to negotiations there can exists units for which a move was already selected. For all other units the possible target provinces are listed and the one with the highest value is chosen. A possible target province can be the province the unit stands on and its adjacencies. Peace treaties and demilitarized zones are considered in this step. Table 11.2 shows the different values. If the unit already stands on the selected target province, the unit is ordered to hold. Otherwise it is checked whether a support can be ordered. If an other unit was already ordered to move to the selected province or to hold there, than the unit is ordered to support it. If no support is possible, the unit is order to move to the target province. A move is automatically an attack when the target is owned or conquered by an other faction.

Table 11.2 Province values according to support centre status, ownership and nearby units, version 1

Province status	Value
Own home supply centre owned or endangered by enemy	50
Supply centre owned by us, enemy unit nearby	20
Supply centre owned by enemy, neutral faction or no one	10
Supply centre owned by a faction in peace status	−18
Supply centre owned by a faction in ally status	−20
Normal province occupied by unit of enemy faction	10
Normal province occupied by unit of faction in peace status	−18
Normal province occupied by unit of faction in ally status	−20
All other cases	0

Since contracts are considered when the target province is selected, there is no way to attack a befriended faction.

As mentioned, in order to calculate moves the game map is evaluated by the strategy core. A value is assigned to each province as shown in Table 11.2. To recapture or defend own home supply centres is of major importance, followed by defending normal supply centres and conquering new ones. Negative values forcing the Stragotiator to stay away from factions in peace or ally status. 'Endangered by' means that a unit exists that can move into the province in a single step.

The evaluation of a move depends on the value of its target province and the environmental conditions. The value of an attacking support move is increased by the number of defenders multiplied with ten and decreased by the constant value 100 if there is no defender. An attack gets the value of the target province multiplied with 1.2 if a neutral faction is attacked or with 1.5 if an enemy is attacked. For the first round, moves are not calculated but randomly chosen from common opening books (see Hand [8]).

11.2.2 Evolutionary Move Optimization

Kemmerling et al. [12] improved the playing capabilities of the Stragotiator while not decreasing the believability. This was achieved by adopting an EA (see e.g. [5]) for move set optimization and a more sophisticated move evaluation and map weighting. Preservation of believability was attested with a quantitative believability measure derived from believability rankings multiple human players gave after test games as described in Sect. 11.1.4. In the following we call the obtained bot *Markus*.

The employed optimization algorithm is a $(\mu + \lambda)$-EA with $\mu = 20$ parents and $\lambda = 40$ offspring. It utilizes a uniform recombination probability of 0.4 and is run for up to 500 generations. Mutation of a move set is done by means of picking each unit with probability $1/\#units$ and change its destination. The new destination

Table 11.3 Improved province values according to support centre status and nearby units, version 2

Province status	Value
Own home supply centre owned or endangered by enemy	245
Own home supply centre, neutral unit nearby	245
Own home supply centre, all other cases	40
Supply centre owned by us or a faction in peace/ally status, endangered by enemy	140
Supply centre owned by us or a faction in peace/ally status, endangered by neutral faction	140
Supply centre owned by us or a faction in peace/ally status, all other cases	40
Supply centre without owner, enemy unit nearby	110
Supply centre without owner, occupied by enemy unit	115
Supply centre without owner, occupied by unit of neutral faction	95
Supply centre without owner, all other cases	100
Supply centre owned by enemy, occupied by unit of neutral faction	130
Supply centre owned by enemy, all other cases	140
Supply centre owned by neutral faction	90
Normal province, occupied by unit of enemy faction	50
All other cases	19

decides the new order type: hold if it is the source province, support if a friendly unit holds or attacks the province, or else attack. The starting point of the optimization is created by the strategy core as described in Sect. 11.2.1.3. In contrast to the basic Stragotiator, the province values are adapted to the ones given in Table 11.3. The adaptation followed from observing the behaviour of the bot in various situations, intending e.g. that home supply centres should be heavily defended if threatened, or free to build on, if not. The province values may be interpreted in the same way as done before. However, they are chosen more fine-grained.

Moreover, an influence map that distributes the worth w of important provinces in weakened form onto their neighbours was introduced in order to increase the attractiveness of provinces which open up possibilities of attacking valuable adjacent provinces. This blurring is done iteratively in four steps $k = 1, \ldots, 4$ according to Eq. 11.7. The factors $g(k)$ correspond to an exponential decrease of the influence on neighbouring provinces in relation to their distance k to the current province.

$$w_k(p) = w_{k-1}(p) + \sum_{\forall \text{ neighbours } p_i \text{ of } p} g(k) \cdot w_{k-1}(p_i) \tag{11.7}$$

$$\text{with} \quad g_{\text{spring}}(k) = \{0.12, 0.022, 0.01, 0.004\},$$
$$g_{\text{fall}}(k) = \{0.04, 0.022, 0.01, 0.004\}.$$

The movement evaluation was updated in order to consider possible superiority regarding a move's target province. Therefore, possible defenders or attackers (immediate neighbourhood of the target province) are taken into account. These may in turn be neutralized via cutting of their support by attacking them, or in equal terms,

Table 11.4 Province values derived for the planning Stragotiator, version 3

Province status	Value
Own home supply centre owned or occupied by enemy	180
Own home supply centre owned by us, not occupied or endangered by some faction	0
Own home supply centre owned by us, all other cases	20
Supply centre owned by us, occupied by an enemy	120
Supply centre owned by us, endangered by an enemy, possible to defend	80
Supply centre owned by us, endangered by an enemy, not possible to defend	25
Supply centre owned by us, not occupied or endangered by some faction	0
Supply centre owned by us, all other cases	25
Supply centre without owner, not occupied or endangered by some faction	250
Supply centre without owner, all other cases	100
Home supply centre of an enemy, owned by him	144
Supply centre owned by enemy	120
Home supply centre of a neutral faction owned by itself	96
Supply centre owned by neutral faction	80
Home supply centre of a faction in peace status owned by him	16
Supply centre owned by a faction in peace status	20
Home supply centre of a faction in ally status owned by him	8
Supply centre owned by a faction in ally status	10
Normal province	0

supporting the moving or holding unit. For successfully moving into a province, there must be at least one more unit supporting the attack than supporting the defence. For holding a province, equal terms are sufficient. Equations 11.8 and 11.9 denote how the value of a move is derived, where $w(p)$ is the previously computed worth of the move's target province p. Please note, that a supporting unit does not move and is therefore treated as implicit hold.

$$move(p) = \begin{cases} f(w(p)) & \text{if \#supporting units} \geq 1 \\ f(w(p))/2 & \text{else} \end{cases} \tag{11.8}$$

$$f(w(p)) = \begin{cases} w(p)/(1 + \#\text{def.} - (\#\text{sup.} + \#\text{attacked def.})) & \text{for attack} \\ w(p)/(\#\text{att. units} - \#\text{sup.} + \#\text{attacked att.}) & \text{for hold/support} \end{cases} \tag{11.9}$$

11.2.3 Evolutionary Move Planning

There are certainly many ways to integrate planning into a Diplomacy bot, largely depending on the complexity of the opponent model one employs. This could be as complex as a full-grown *theory of mind* [19] which could also consider expectation

of expectations as formulated as sociological concept by Luhmann [18]. In such a setting, one would even take into account that behaviours may be changed because a bot (or human) has a concrete expectation of what his opponents expect him to do. However, our opponent model is much less complex (see Sect. 11.2.1.1). It just accounts for the experiences with a faction during the game in a tit-for-tat [2] style. We also doubt that using very complex opponent models makes sense in Diplomacy as one always has to consider several players at once (usually there are at least three neighbouring factions).

In his thesis, Ackermann [1] recently integrated an evolutionary planning technique to look one step ahead and Kemmerling et al. [13] observed that this again rises the playing strength of the Stragotiator. In the following, the obtained bot is called *Niels*.

In a first step, the EA described in Sect. 11.2.2 is used to calculate at most 20 move combinations with a value of at least 85 % of the best combination. Then, the strategy core and another EA instance are employed to calculate the best moves for all remaining factions. This can be done for every single faction separately or for all factions at once (this is later on referred to as *option 1*, meaning that we assume that all enemies and neutral factions play together).

Moreover, the outcome of each move combination saved in the first step is calculated regarding the obtained enemy moves. Then, several EA instances are adopted to calculate for all distinguished board states the best next move combination for own and allied units. The value of the best obtained move combination for a particular board state is added to the value of the move combination that lead to this state (assuming the other factions act as expected). The combination with the highest accumulated value is chosen for execution. Move combinations are evaluated with a new target function utilizing the third version of province ratings as depicted in Table 11.4. See Kemmerling et al. [13] for a detailed description on how to distribute computing time to the EA instances in the last step.

From another point of view, several move combinations for own and allied units are calculated, then selecting the one that leads to the most advantageous game state for the next round regarding a specific opponent behaviour. If the opponent reacts in a completely unforeseeable way, obtained quality values may not be very realistic.

Equations 11.10–11.12 show a simplified version of the used move rating function $move(p)$, where $w(p)$ is the worth of the move's target province p. α, β, γ and δ are parameters to fine-tune the bot's behaviour. They were manually set and depend on superiority and province status. Please see Ackermann [1] for a detailed illustration of the functions.

$$move(p) = \begin{cases} (a(p) + \alpha)^2 \cdot \beta \cdot w(p) & \text{for attack} \\ (h(p) + \gamma)^2 \cdot \delta \cdot w(p) & \text{for hold/support} \end{cases} \tag{11.10}$$

$$a(p) = \frac{\min\{1 + \#\text{sup.}, 1 + \#\text{def.} - \#\text{attacked def.}\}}{1 + \#\text{def.} - \#\text{attacked def.}} \tag{11.11}$$

$$h(p) = \frac{\min\{1 + \#\text{sup.}, \max\{\#\text{att. units} - \#\text{attacked att.}, 1\}\}}{\max\{\#\text{att. units} - \#\text{attacked att.}, 1\}} \tag{11.12}$$

The denominator in $a(p)$ prefers provinces with few defenders, taking into account that support can be cut off. Using the minimum in the numerator prevents a preferential treatment of unnecessary supports. $h(p)$ was constructed to mirror $a(p)$.

Table 11.4 shows the used province values. Furthermore, an influence map similar to the one described in Sect. 11.2.2 is used. Reviewing the evolution of the province value tables from version 1 to 3, one instantly perceives that they get longer and longer. As stated before, this mostly stems from corrections done to prevent 'stupid' behaviour of the bot. However, it is not clear if the tables are already 'optimal' in some sense. A reasonable next step would be to learn values for the concrete provinces on the map.

11.3 Believability and Individualization

The playing skills of the planning Stragotiator have been investigated in Kemmerling et al. [13], taking van Hal's Albert as reference. It was shown that playing strength mostly increases with respect to the non-planning Stragotiator, especially in the first rounds. Considering that improving the playing strength was the main reason for inventing the planning module, it was of course the most crucial question to answer if this is indeed the case. Since the Stragotiator was originally created in order to behave most believably, the next and still open task is to examine the impact of the planning module on the Stragotiator's believability. We have approached this task in two different ways. Kemmerling et al. [12] presented a technique to measure the believability of Diplomacy bots. In Sect. 11.3.1, we will show the influence of the planning module on believability by comparing the planning and the non-planning Stragotiator. For this first investigation, we have used the measure developed in [12]. Of course, it is a desired result to see that the increase in playing strength does not result in a big loss in believability at the same time. However, as already stated in [12], this believability measure comes with some problems, one of which is that the original game data stems from several human players and there may not be something like 'the' human playing style.

Currently, the behaviour of the Stragotiator cannot be changed completely just by changing some external parameters, thus we cannot run some optimization method to 'fit' the playing style to the believability measure. However, the Stragotiator is composed of several separate modules which exist in various versions. Since one can combine these modules and their versions freely, we are able to construct many

different Stragotiator variants. In order to mimic a specific human playing style, we first derive new individualized believability measurements in Sect. 11.3.2. This measurements are then used to test which Stragotiator variant resembles most the individual concept of believability of one human player.

Kemmerling et al. [12] compared two versions of the Stragotiator and the *NICE* bot. The Albert bot was not included in the test because at that time, the available version showed stability problems. The NICE bot is a communication-enhanced version of the Diplominator [25] that was never released to the public. Even though the Diplominator is available, a direct comparison of the Stragotiator with a well-established bot was never done. We will fill this gap by including the current version of Albert in our investigations, knowing that it is (meanwhile) a very able competitor. Progress made in Diplomacy bot AI during the last years has been noticeable.

11.3.1 Believability of Planning

To investigate the believability of the new planning Stragotiator, we performed over 600 games with Albert '-d37', the bots introduced by [12] (non-planning Stragotiator, see Sect. 11.2.2), called Markus, and [13] (planning Stragotiator, see Sect. 11.2.3), called Niels, with option 1 enabled. Each game comprises of the 6 years from 1901 to 1906, which is a usual game length for humans, and the bot for each faction was selected randomly from this set of three.

Albert [24] is probably the best currently available bot regarding playing strength. But more important for this investigation, it is the only known bot capable of press levels 10, 20, and 30, according to its web page. Unfortunately, not all messages defined in the upper levels are implemented. Especially, the important movement proposals (XDO) are neither sent, nor understand by the bot.

We used Albert to achieve a neutral impression of the believability of both Stragotiators. The option '-d37' corresponds to a parameter named *thinking depth* which adjusts the tree depth of Albert's internal data structures employed to predict each faction's best moves. It defaults to '-d50'. Albert iteratively adjusts the probability of any order set being selected for each faction, resulting in a very time consuming procedure. To enable submitting orders within given time limits, this parameter shall be well considered. According to Albert's README, '-d20' or '-d30' should already yield good results, and still be relatively quick. Our selection of '-d37' is the outcome of various pre-experimental test games with chosen time limits of 100, 5, and 3 s for movements, retreats and builds, respectively.

Employing the measure derived in [12], we obtained the believability values depicted in Fig. 11.2. Focusing on the first to third quartile, we cannot discover a clear advantage for any bot. Values and even distributions of all bots are comparable. However, the whiskers show slightly more variance for the planning Stragotiator Niels. Taking into account the outliers, one may get the impression that the Stragotiators, especially Niels, are sometimes much more believable than the Albert. Both have more positive than negative outliers. Although we can not speak of an advantage

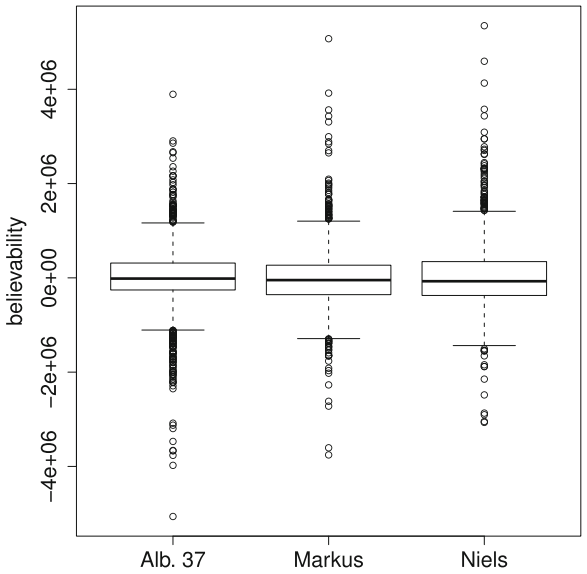


Fig. 11.2 Believability of Albert '-d37', the Stragotiator with move optimization (Markus) and with evolutionary planning (Niels)

of the Stragotiator over the Albert, we can conclude that the planning module does not violate the believability of the Stragotiator.

Unfortunately, due to some problems in the experiments, the weights for the believability measuring modules AcceptedXDO and AcceptedMessage had been set to zero in [12]. This disables the majority of negotiation based believability calculation, leading to a disproportional weighting of movements and tactics. Since the Albert was developed with focus on playing strength and not on negotiation while the Stragotiator was originally build to imitate a specific human playing style with main focus on negotiation, the used measure may not be very suitable for comparing the Albert and the Stragotiator.

11.3.2 Deriving New Measurements

The measure used in Sect. 11.3.1 has two disadvantages. At first, it disregards the modules AcceptedXDO and AcceptedMessage. Second, [12] tried to derive a general measure mirroring the opinions of many persons. However, this may not be very meaningful as different human players may have indeed very different preferences. In this section, we derive two individualized measures each matching the opinion of only a single person which should be much more consistent.

We chose two players, A and B, with some experience in the game Diplomacy and with the DAIDE AImapper (the GUI that may be used for playing as well as for observing games). They played six and seven games respectively against two Alberts with default thinking depth '-d50', two Stragotiators called Markus (non-planning, see Sect. 11.2.2), and two Stragotiators called Niels with option 1 enabled (planning, see Sect. 11.2.3). Round times were set to 120 s for the movement phase, 45 s for retreats and 30 s for building. After each game, all bots were ranked according to their believability. The games were evaluated with the unweighted (all weights set to 1) believable measure comprising of the modules listed in Table 11.1. Then, the EA described in Sect. 11.1.4 was used to optimize the weights of all modules so that the believable results of each game matches the given ranking well. To obtain one measure per player, the optimization was done for each player separately. The EA employed the manually tuned parameters $\mu = 50$, $\lambda = 400$, recombination probability 0.7 and step size $\sigma = 4.0$, differing from the parameters in [12] only in the offspring size λ. However, algorithm performance seems not very sensible to parameter changes.

For both players, one of the best obtained weight sets with only 20 and 15 ordering mistakes respectively is presented in Fig. 11.3 in logarithmic scale. This weights are be used in the following investigation. In total we obtained 54 weight sets with 20 mistakes for the six games of player A. All have in common that AcceptedMessage always gets high positive while ChainMove, SelfSeizing, and JumpingJack get negative weights. This demonstrates the importance of this behaviours for player A. Especially AcceptedMessage seems to be very important for obtaining a good ranking. Some of the weights (e.g. the one for Purposefulness) are rather small in all optimal weight sets. Even though the module is apparently not very significant, we can not disable it as the result for player B shows. For the seven games of player B we obtained 13 weight sets with 15 mistakes. This sets differ substantially from the ones obtained for player A. Purposefulness, ChainMove, AcceptedXDO, and AcceptedMessage always get positive and while ClusterDisconnect and XDOToNeighbors get negative weights.

Only AcceptedMessage gets high positive weights in all sets for both players. This demonstrates the importance of well considered negotiation. Even if a player creates a flood of messages it is not believable when they are nonsense and will always be rejected by the negotiation party. For player A, the execution of chain moves and toggling of one unit is not unbelievable as it was originally considered at the development time of the measure. In the opposite, player B considers chain moves unbelievable while it is also believable to send XDOs not only to neighbours and to divide own units. For both game sets, some modules sometimes get positive and sometimes negative weights, showing the need for more games as a basis. Since each game takes up to one hour and all games must be done by a single person, it is difficult to get hold of a larger number of games. However, it is obvious that both players take different aspects into account when judging the believability of their opponents.

From the comments of the game testers, we know that the weight set fairly well reflects the observed course of the test games. Player A is oriented at a very high-level

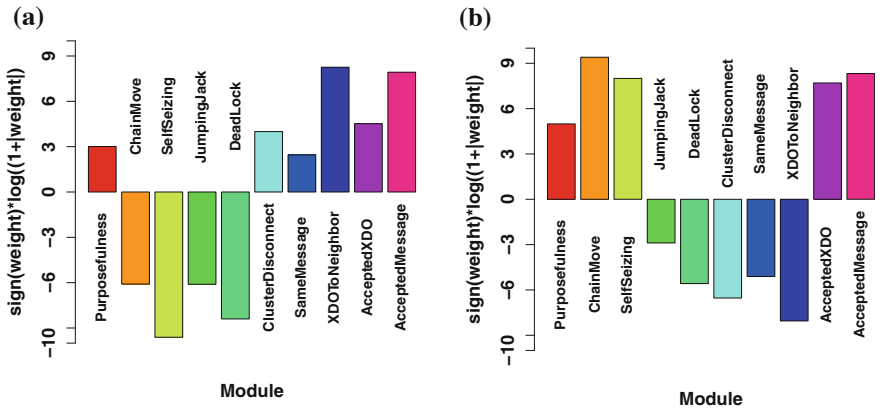


Fig. 11.3 The best attained (**a**) weight sets of our new believability measures, plotted in logarithmic scale (denoted as *measure-2a* for player A and *measure-2b* (**b**) for player B. Note that nearly all message related modules get positive values)

strategic layout of the game, by selecting one or two of the neighbouring factions as allies and opportunistically or pro-actively trying to crush one enemy at a time while keeping the other possible fronts stable. He found it most believable if a good collaboration with some bots (usually representing one or two of the neighbouring factions) could be established. This included combined attacks and also successful combined defenses against another attacking faction. Of course, rather trivial bot features as delayed answers also played a role, as well as consistency. In the rearview, it was identified that the best collaborations could be established with Stragotiator bots, however with a sometimes too large number of move suggestions. Interestingly, the Albert bot tends to punish breaking DMZ agreements harshly by refusing any collaboration afterwards.The second game tester describes himself as aggressive player with a risk aversion focusing on strategical game play. He tries to conquer many supply centres as soon as possible. However, he does not rely on luck but collects a definite majority of units before attacking. He is willing to loose one supply centre in favor to conquer an other in a strategically better position. He found it most believable if an opponent reveals a great amount of playing strength and tactic skill. Even if he likes to utilize XDOs to gain advantages in the positional play, he considers bots that do not send or react to XDOs as believable as long as they play well enough. That is why he considers the Albert bot in six of seven games to be most believable. According to the test persons, the games have been a very interesting experience as especially the mix of different styles leads to challenging games which are far from boring. This opens an interesting line of research for the future, namely considering ensembles of bots instead of a single bot as they enable a much richer game experience than playing against 6 relatively similar bots, even if they are very well developed concerning negotiation. However, we disregard this direction for now and defer it to future research.

11.3.3 Individualizing of Believable Bots

We now rate the believability of different versions of the Stragotiator according to the measurements derived in Sect. 11.3.2 (*measure-2a* and *measure-2b*). At first, we employ the bots introduced in [12] (see Sect. 11.2.2), called *Markus*, and [13] (see Sect. 11.2.3), called *Niels*. Furthermore, we created hybrid bots with the move rating function of Markus and the map evaluation function of Niels, called *Mike*, and vice versa, called *Wolfgang*. All four combinations of the two functions are investigated with and without the planning discussed in Sect. 11.2.3, also enabling and disabling option 1 for planning. To obtain a good impression of the believability of the Stragotiator, the Albert bot was included with thinking depth '-d26' and '-d37' in the experiments, thereby deliberately making it a bit weaker than the default configuration. The '-d26' option value results from an experiment discussed in [13]. According to larger movement time in this experiment, we also increased Albert's thinking depth to '-d37'.

To measure the believability, we conducted more than 2,800 games with press level 30, and 100, 5 and 3 s as time limits for movements, retreats and builds, respectively (this is comparable to the human test games, taking into account that bots need a bit less time to actually send messages concerning orders, retreats and builds). Again, each game comprises of 6 years from 1901 to 1906. The bots were selected randomly from the described set of six for each game.

Tables 11.5 and 11.6 show the median of the calculated believabilities for the Stragotiators. For Albert '-d26' and '-d37', we measured a median believability of −19,470 and −22,560 with measure-2a and 24,750 and 18,870 with measure-2b, respectively. Since the measures have different, non-normalized weights, we can not compare results obtained with different measures. Even with normalized weights, a comparison is not advisable because some modules aggregated in the measure implement non-linear functions and the weights are optimized according to believability ranks and not believability ratios. Therefore, we show the rankings according to the median believability values in addition.

According to measure-2a Niels with planning exhibits the greatest believability and Albert the lowest. Only considering the planning bots, the believability decreases when option 1 is enabled, usually to around the level of the non-planning bots. Comparing non-planning and planning bots, planning increases the believability (with exception of Mike). In most cases, the map evaluation function of Sect. 11.2.3 is preferable over the one described in Sect. 11.2.2. The same holds true for the two move evaluation functions.

Measure-2b provides a different results. Player B pay heed to the playing strength and the tactic that his opponent shows. Therefore, measure-2b ranks the Albert bot among the most believable bots but still more unbelievable as Niels with planning. Again, option 1 decreases the believability when comparing the planning bots while the planning increases the believability when comparing non-planning and planning bots. In all cases, the map evaluation function of Sect. 11.2.3 is preferable over the

Table 11.5 Median believability according to measure-2a

	Markus	Mike	Wolfgang	Niels
Non planning	47,060 (11)	74,870 (02)	69,370 (05)	62,970 (08)
Planning	54,800 (10)	65,830 (06)	74,860 (03)	85,930 (01)
Option 1	36,870 (12)	61,860 (09)	63,450 (07)	69,850 (04)

The value in brackets depicts the believability rank. Albert '-d26' obtained −19,470 (13) and Albert '-d37' −22,560 (14)

Table 11.6 Median believability according to measure-2b

	Markus	Mike	Wolfgang	Niels
Non planning	−59,530 (12)	−31,480 (09)	−78,370 (14)	−36,230 (10)
Planning	−22,190 (07)	8,052 (06)	20,370 (03)	73,450 (01)
Option 1	−60,470 (13)	−27,200 (08)	−50,560 (11)	10,860 (05)

The value in brackets depicts the believability rank. Albert '-d26' obtained 24,750 (02) and Albert '-d37' 18,870 (04)

one described in Sect. 11.2.2. In most cases, the same holds true for the two move evaluation functions.

According to the computed values of both measures, the human tester should be most satisfied with the Stragotiator in its original planning version (Niels), with option 1 disabled.

The results demonstrate that there is more than one definition for believability. Even so, the Stragotiator is always a good choice. With both differing measurements which now consider press, the planning Stragotiator Niels is much more believable than the Albert. However, one has to mention that Albert has not been designed to behave very human-like but only play well. Measure-2b reflects the view of a player who considers a strong opponent as believable, resulting in a good rank for the Albert bot. Other players, like game tester A, do not consider playing strength as so believable. This is reasonable since not all human gamers play very well. In contrast, the Stragotiator was developed with intent to create a bot that is at first believable. The success of this effort can be clearly seen in the results of both measures. Especially with measure-2a, the Stragotiator in all its versions achieved a better rank than the Albert bot. It shall be noted that the Stragotiator's planning module was developed with focus on improved playing strength and does not strongly follow the characteristics of the Classicist playing style. The negotiation module is not affected by the planning module, but the planning module does not consider negotiation outcomes. More precisely, the negotiation module accepts and asks for contracts and proposes moves as described in Sect. 11.2.2, but the planning module does not consider move proposals rejected by others and moves assured by the negotiation module. Only peace and alliance contracts made in previous rounds are respected.

With this insight in the internal behaviour of the Stragotiator, the results hint to the necessity of planning to build a believable Diplomacy bot. Although Kemmerling et al. [13] showed that option 1 results in an advantage by means of playing strength, it

seems important to calculate moves for each enemy faction separately in the planning module. According to our experience, this is exactly the case for human players. Human players usually follow two plans: a short-time (one or two rounds ahead) and a long-time plan (several rounds ahead). Furthermore, they always try to guess which factions play as a collaborating party and who is enemy of whom. In future research, it would be interesting to consider the calculations and decisions of the opponent modeling and the negotiation module for planning.

11.4 Conclusions

In this chapter, we have provided insight into our believable Diplomacy bot, the Stragotiator. We have presented the history of the Stragotiator, the implemented techniques and reported the steps undertaken in order to improve playing strength as well as believability. The last of these steps was the implementation of an EA based planning module enabling the Stragotiator to look one step ahead. The obtained planning Stragotiator was validated with our believability measure technique in a new and interesting way, now correctly incorporating messages and move order suggestions. Instead of deriving a general believability measure for Diplomacy (which may indeed be difficult if not impossible), we tailored our measure to meet the preferences of two specific test players, thereby deriving individualized and indeed different instantiations of the measure. We employed these customized measures to select the parametrization of the Stragotiator that match these preferences best. We thus effectively created a bot which is playing well, including planning, uses basic communication and partly inherits the playing style of a specific human player (or better, the expected opponent playing style for this player). However, more tasks have turned up in the course of this investigation. We identified the need for a greater data basis for training our measure to be able to select important believability characteristics. Furthermore, we presented some ideas to improve the planning module that may result in an even more believable bot. Finally, according to the report of our test player, it may be worthwhile to consider bot ensembles instead of single bots in order to create an even richer playing experience.

References

1. Ackermann, N.: Evolutionäre Spielbaumsuche für das Strategiespiel Diplomacy. Dipl. Inf. Diploma Thesis, Department of Computer Science, TU Dortmund (2010)
2. Axelrod, R.M.: The Evolution of Cooperation. Basic Books, New York (1984)
3. Booijink, B.: Strategy evolution and resolution deduction in diplomacy. Master's Thesis, University of Groningen, Artificial Intelligence (2005)
4. Calhamer, A.B.: Calhamer on Diplomacy: The Boardgame Diplomacy and Diplomatic History. Authorhouse, Bloomington (1999)
5. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer, Heidelberg (2003)

6. Fàbregues, A., Sierra, C.: Diplomacy game: the test bed. PerAdaMagazine (2009). http://www.perada-magazine.eu/view.php?source=1761-2009-08-03
7. Fàbregues, A., Sierra, C.: A testbed for multiagent systems. Technical report, IIIA-CSIC, Bellaterra, Barcelona (2009)
8. Hand, M.: The diplomatic pouch: the library of diplomacy openings (1995). http://www.diplom.org/Online/Openings/textversion/
9. Johansson, S.J.: On using multi-agent systems in playing board games. In: Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), pp. 569–576. ACM, New York (2006)
10. Johansson, S.J., Håård, F.: Tactical coordination in no-press diplomacy. In: Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), pp. 423–430. ACM, New York (2005)
11. Keaveney, D., O'Riordan, C.: Abstract model of a real time strategy game. Technical report, nuig-it-011008, National University of Ireland, Galway (2008)
12. Kemmerling, M., Ackermann, N., Beume, N., Preuss, M., Uellenbeck, S., Walz, W.: Is human-like and well playing contradictory for diplomacy bots? In: Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG 2009), pp. 209–216. IEEE (2009)
13. Kemmerling, M., Ackermann, N., Preuss, M.: Nested look-ahead evolutionary algorithm based planning for a believable diplomacy bot. In: Di Chio, C., et al. (eds.) Applications of Evolutionary Computation—EvoApplicatons 2011: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Torino, 27–29 April 2011, Lecture Notes in Computer Science, vol. 6624, pp. 83–92. Springer (2011)
14. Kraus, S., Lehmann, D.J.: Designing and building a negotiating automated agent. Comput. Intell. **11**, 132–171 (1995)
15. Livingstone, D.: Turing's test and believable AI in games. Comput. Entertain. **4**(1), 6 (2006)
16. Loeb, D.E.: Challenges in multi-player gaming by computers. The Diplomatic Pouch Zine (S1995M) (1995). http://www.diplom.org/Zine/S1995M/Loeb/Project.html
17. Loeb, D.E.: The observation module. The Diplomatic Pouch Zine (S1995R) (1995). http://www.diplom.org/Zine/S1995R/Loeb/Observe.html
18. Luhmann, N.: A Sociological Theory of Law. Taylor & Francis, London (1985)
19. Premack, D.G., Woodruff, G.: Does the chimpanzee have a theory of mind? Behav. Brain Sci. **1**, 515–526 (1978)
20. Ribeiro, J.A., Mariano, P., Seabra Lopes, L.: Darkblade: a program that plays diplomacy. In: Seabra Lopes, L., Lau, N., Mariano, L.M., Rocha, P. (eds.) Proceedings of the 14th Portuguese Conference on Artificial Intelligence: Progress in Artificial Intelligence (EPIA 2009), Lecture Notes in Computer Science, vol. 5816, pp. 485–496. Springer (2009)
21. Shaheed, J.: Creating a diplomat. Master's Thesis, Imperial College, London (2004)
22. Shapiro, A., Fuchs, G., Levinson, R.: Learning a game strategy using patternweights and self-play. In: Schaeffer, J., Müller, M., Björnsson, Y. (eds.) Proceedings of the 3rd International Conference on Computers and Games (CG 2002), Lecture Notes in Computer Science, vol. 2883. Springer (2002)
23. Turing, A.M.: Computing machinery and intelligence. Mind **59**, 433–460 (1950)
24. van Hal, J.: Diplomacy AI—Albert. http://sites.google.com/site/diplomacyai/ (2010)
25. Webb, A., Chin, J., Wilkins, T., Payce, J., Dedoyard, V.: Automated negotiation in the game of diplomacy. http://www.daide.org.uk/external/TheDiplominator.pdf (2008)
26. Windsor, P.D.: What's your point. The Diplomatic Pouch Zine (S1999M). http://www.diplom.org/Zine/S1999M/Windsor/point.html (1999)
27. Yannakakis, G.N., Hallam, J.: A generic approach for generating interesting interactive pac-man opponents. In: Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG 2005), pp. 94–101. IEEE (2005)
28. Yannakakis, G.N., Hallam, J.: Feature selection for capturing the experience of fun. In: Proceedings of the AIIDE'07 Workshop on Optimizing Player Satisfaction, pp. 37–42. AAAI (2007)

Chapter 12
Towards Imitation of Human Driving Style in Car Racing Games

Jorge Muñoz, German Gutierrez and Araceli Sanchis

Abstract In this chapter we focus on creating believable drivers for car racing games. We describe some racing controllers from the commercial games and the academic researchers, their particular features, what are the main problems they want to deal with and how they approach them. Besides, we identify which are the key properties and behaviours required to consider a racing non-player character (NPC) as believable or not, always from the point of view of an external human player that competes against the NPC. Then, we analyze why the current controllers lack what we understand as a believable behaviour and propose a new approach based on imitation learning to create the racing NPCs. We describe this new approach and analyze its advantages and disadvantages compared with other controllers in order to solve the key problems to achieve the desired believability.

12.1 Believability and Racing Games

Game developers are always searching for a way to improve the Non-player characters (NPCs), also known as bots, in order to make them more believable. When the NPC represents a person it should seem, behave and be as intelligent as a human would be [14]. They do so because it is commercially profitable. When the game becomes more believable, closer to reality, the players feel more immersed in the

J. Muñoz (✉), G. Gutierrez · A. Sanchis
University Carlos III of Madrid, Avds. de la Universidad 30, 28911 Leganes, Madrid, Spain
e-mail: jmfuente@inf.uc3m.es

G. Gutierrez
e-mail: ggutierr@inf.uc3m.es

A. Sanchis
e-mail: masm@inf.uc3m.es

P. Hingston (ed.), *Believable Bots*, DOI: 10.1007/978-3-642-32323-2_12,

Fig. 12.1 Screenshot of an open source racing car simulator, TORCS

game, they find it more enjoyable, appealing and engaging. All these features of the game are usually translated into more sales.

In order to achieve a believable NPC it is necessary to create a virtual character that visually looks like it should looks in the real world, but also make it intelligent enough to behave in a realistic manner [1]. In recent years the game industry has made a lot of progress improving the visual appearance of games. NPCs now have higher numbers of polygons and improved textures in order to seem more real. There are NPCs in games on the market today that are almost indistinguishable from real people, and car models that look like photographs of real cars (see Fig. 12.1). But appearance is only part of what's needed in order to create a believable NPC. NPCs need to not only appear real but also behave in a realistic way. NPCs that look like real people are expected to behave as we know they will do in real life and not different. Therefore, the need to improve the realism of NPC behaviour has grown in recent years.

The easiest way to add players with believable behaviour to a game is to include an online mode where human players play against other humans over a network. But this is only useful in some kinds of games and it is not the only mode of gameplay in the games, except in some cases.

In order to create an artificial believable behaviour, the first question we have to answer is "*what is a believable behaviour?*" In racing games, and any other video game, the answer to this question will depend on the person you ask. Believability is a subjective concept, and it is difficult to get a specific answer. In general terms, we can consider that a NPC is believable in a racing game if it drives like a human. But

this leads us to another question: '*how do humans drive in video games?*" or "*what is driving like a human?*" These are complex questions too. The way a human player drives in a game depends on his or her skills, and the driving style of an expert player is very different from that of a novice player.

For example, consider a race where we have a novice and an expert players, where the last is either a human or a NPC. The human player, while driving against this opponent, is going to sense that the opponent perform things that he or she cannot. The human player will conclude that if the competitor is doing something that apparently he or she will never be able to do, then probably the competitor is a NPC. In other scenarios where an experienced human player is racing against a novice competitor, he or she will either think that the opponent is a bad NPC or a very bad player that did not learn to play well. In the opposite, if we have a hardcore human player driving against a bad competitor, he or she is going to think the opponent could be a badly programmed NPC or a very bad player that did not learn to play well before. Thus, it is not clear what a believable behaviour is, it depends on the skill of the person judging it [29] and his or her cultural origins [18]. Despite this subjective answer about believability, we think an average agreement could be reached. Assuming that very good NPCs will not be believable and a very bad NPC will not be believable either, there are two ways to create believable drivers. The first one is to create an average driver with the most common skills of human drivers, so that if it drives like most human players it should be a human. The second one is to create NPCs which drive like the human player they are competing with at that time, that is, NPCs that imitate the driving skills of the current human opponent.

The game industry wants games sufficiently enjoyable and appealing for the player, where the opponents should pose a challenge for players. This means that the opponents are neither impossible to beat nor too easy to win against. An average skilled NPC could be a challenge for some players, but for beginners an average NPC could be impossible to beat, and the expert players will be able to win all the races. The game could be frustrating at the beginning for novices and boring after some time for the expert players. It is possible to tune the average NPC to make it worse or better as needed, but it is difficult to do this and keep the behaviour believable at the same time. That is the reason why ensuring that a NPC that can imitate a human player is a good choice. This NPC can be a bad driver in the beginning for novice drivers and improve itself as the human players improve their skills. Nevertheless, creating a NPC that learns how the player drives and imitates him or her is not an easy task. The games which have so far used this approach have had to include a lot of constraints in order to create a NPC by imitation learning. This is why the companies develop average NPCs that can be tuned. In our work we explore how to create drivers by imitation in order to create believable opponents for players.

A believable driver must show some some basic skills, regardless of the skills of the human player it is competing against or the method used to create the NPC. For a common type of track like the F1-like tracks, asphalt tracks with a good grip and without irregularity, we argue that these skills should be as follows:

- To drive following an optimal (or near-optimal) trajectory that minimizes the time to complete a lap, almost without making any mistakes.
- To avoid collisions with other cars while driving.
- To be able to overtake slower opponents in a straight and to perform a hard brake before a turn to get the inner side of the turn (which is a typical manoeuvre of a professional driver to get an advantage and stay in the optimal trajectory point when the turn starts).
- To prevent an opponent overtaking by means of closing the trajectory and preventing the opponent getting the inner side of the next turn.
- Recovering and continuing racing when something unexpected happens like a collision, failing in overtaking or getting off the road in a turn.

A professional driver would take into account more complex strategies when racing. For example, a long term strategy like fooling the opponent by performing not so well in a specific turn in order to later perform that turn correctly and overtake the opponent. There are also different kinds of racing games like motorbike games or rally games which could require more skills to achieve a believable behaviour, but we will not take them into account in this chapter. We assume that the previous skills are common to all racing games and the most important.

The final objective of our work is to create artificial drivers by imitation of human drivers A NPC able to see what the human is doing while driving, learn what he is doing and then repeat it in order to drive the car. This is difficult and we have decided to focus on imitating the human driving style. We developed and tested a system based on neural networks that learns the drive style of a human player. We consider the drive style as the trajectory followed by the human and the velocity of the car at each point of the trajectory. But we said that more skills—overtaking, recovering, avoiding collisions—are required to create believable drivers, so they are hand coded in the artificial driver. We programmed them in a way to keep the drive style learnt from the player, therefore their execution while driving is only a modification of the trajectory and the speed when the skill is required. The system was tested first without opponents in order to check that similar driving is achieved and then with opponents in a competition to compare our results with other NPCs created by researchers.

In the rest of the chapter, first, we will describe the common techniques used in the game industry to create drivers, some academia related work and why these drivers lack believability (Sect. 12.2). Then, we will summarize the main skills we consider a believable driver must have (Sect. 12.3). After that, we will describe why imitation learning is a good choice to create believable drivers and how it could be applied to a racing game (Sect. 12.4) and an application in the TORCS game (Sect. 12.5). Finally we will show the results of our proposal and argue the advantages and disadvantages of this driver compared with others and the conclusions (Sect. 12.6).

12.2 Game Industry and Academia NPCs

Game industry and academia researchers differ in their goals for developing artificial intelligence (AI) for games. On the one hand, industry focuses on the most profitable AI, creating controllers that maximize the profit (their sales). On the other hand, researchers are seeking to apply new techniques in order to build up the controllers that satisfy some particular properties, or to discover new ways to create controllers. The state of the art in the research is not what the game industry uses in its games, they almost always prefer to use the classical AI techniques that have been proved to work over time and have a deterministic behaviour. The main reason is that if the AI does not work due to an unexpected behaviour when the game is sold, they cannot change it and the clients start to complain about the game, and so the sales decrease. Another difference between the two fields is the way they use the AI. In the case of the game industry they do not worry about cheating on the players by changing the controller features to perform things the human player would never be able to do. This is a way of creating controllers that keep the challenge of the game but these controllers are not believable after some time competing against them.

12.2.1 Game Industry NPCs

A longstanding strategy followed to keep the challenge in racing games is the *dynamic difficult adjustment* or *rubber band rule* [11]. The idea of this rule is very easy: it modulates the in-game parameters, such as the acceleration, maximum speed or grip of the cars, to respond to the player abilities over the course of a race. The parameters are modified to make the opponents run faster when they are behind the player car, and modified to make them run slower when they are ahead of the player (see Fig. 12.2). Furthermore, this modification is in proportion to the distance, when the cars are far away the effects over the velocity are greater and there is no advantage for any car when they are very close. One game where this effect is very noticeable is the famous game *Mario Kart* [24]. The rubber band rule keeps the challenge of the game regardless of the player skills. After some time playing the human players notice that they improve their skills but the opponents keep the challenge of the game. In a simple game such as Mario Kart it is easy for a player to become an expert and perform almost perfect laps, but even in this case the opponents are still difficult to beat and although they make mistakes they are able to challenge the human player. This is frustrating and a signal that the computer is cheating the player. Of course, cheating by modifying the features of the car to make it run faster is not a believable behaviour and should be avoided if our final goal is to achieve a human-like driver. Both the opponents and the human players must compete with the same constraints and aids.

For the game industry the easiest way to create a controller for a racing game is adding some aids to the controller. The aid most often added to the games is a racing

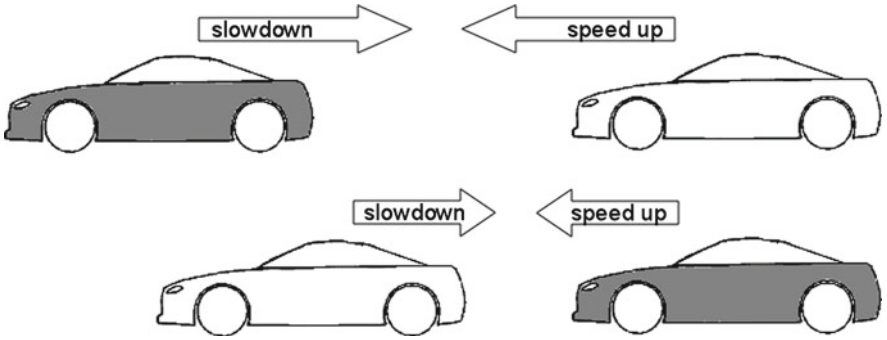


Fig. 12.2 Dynamic difficulty adjustment or rubber band rule. First car built-in parameters are tweaked in order to slow it down and second car is also tweaked to speed it up. Modification is proportional to the distance between the cars

line [2]. This information is not perceived by the human players and it is used by the controller as a reference of the trajectory they should follow. The racing line is the trajectory most experimented human players follow in the track. Therefore, it not only helps the racing controller to perform competitive lap times, it also causes the racing controllers to show a human-like driving style. Since the humans sometimes make mistakes, these are added on purpose in the NPC within the rules to follow the race line to exhibit an even more human-like behaviour. In order to overtake another line is added. This line is the trajectory the humans follow when they want to overtake the opponent ahead. When the rule that decides when an overtaking should be done is fired, the controller starts to follow the line for overtaking instead of the normal racing line until the overtaking is completed, or failed. The problem with this kind of controller is that because one race line is needed per track—two if we also use a racing line for overtaking—the controller is not generalized for new tracks and the race line has to be created for them. Using race lines also means that the game AI has more information about the track than the players, who do not have any knowledge about this line.

When using race lines the code required to follow the racing line is sometimes not as simple as it could appear at first sight. For example, in the game *Colin McRae Rally 2.0* [4, 19], it was necessary to use artificial neural networks [36] for this task. Firstly, they tried to follow the racing line with a rule set to adjust the steering, but the features of the rally tracks made this task impossible to accomplish. The variety of surfaces of the tracks and their sliding properties require a different drive style than in a normal asphalt track: the car has to turn into corners earlier and slide round. The neural networks were used as an interpolation function [4], that is, they learnt how the output is with a given input. They were trained with human data that contains the relation between the elements to control the car as the steering and the pedals (the outputs of the neural networks) and the information about the track, the internal state of the car and the race line (the inputs of the neural networks). The resulting controller learnt the human behaviour to follow the race line, and it was able to drive

almost as fast as the human player. Indeed, the final controller was only a couple of seconds slower to complete a lap in most of the tracks than the expert human player they used to train the neural networks. The resulting driver was competitive enough for most users of the games and shows some human characteristics, as it was created to imitate the human drive style when it follows the race line.

Other games like *Motocross The Force* simplified the problem of creating NPCs by designing a game where all possible controllers developed with the game interface will always be able to complete a lap [6], regardless of how badly the NPC was programmed. From the point of view of the AI, this design is a perfect scenario to create controllers by means of evolutionary algorithms. These algorithms do not have to deal with the problem of creating controllers able to perform a lap in the first steps of the evolution process, they only have to evolve the controllers and improve them. In *Motocross The Force*, the developers used neural networks to create the controllers, and trained these networks with human data. The goal was learning what actions a human does at each point of the track. Therefore, they collected the action a human player does while playing, and then they used them in the controller. The way this was done was dividing the data into sets, one set per track, and training different neural networks with the sets. Once this was done, they joined the results of all neural networks and used the average value of the outputs as the final output of the controller to decide the action to execute.

Both *Colin McRae Rally* and *Motocross The Force* are expected to show a human-like behaviour due to being trained with human data. But this is far from true. They show some features of human driving but they do not show human behaviour. The feed-forward neural networks used in these games learn the output for a given input, but they do not take into account the intentions of the player or the past actions he or she did (recurrent neural networks can handle this issue but they were not used). Neural networks only learn a short period of time, a game step, and are not able to learn anything about the intentions of the player. In two different instants of the game with the same input for the network the human player can have different intentions and perform different actions, and this is translated into the learning algorithm as noise in the training data. With the same input sample the neural network has to learn sometimes two, or more, different outputs and, as this is impossible, it is only able to learn an average value or the most frequent action performed. This is the reason why the created controller shows a behaviour that seems like a human although it is not a completely human-like behaviour.

In *Forza Motor Sport* [20], a racing game for the *Microsoft XBox*, the developers do something new in the industry. They create an in-game mode where the players were able to create an avatar driver of themselves which was able to drive like the human player. That is, a NPC that could replace the human in a race and drive as the player would do if it was playing. For a game company, when they release a game they have to be completely sure everything is going to work as they planned, because if something is wrong the users will not buy another game again, in contrast to researchers where nothing happens if they probe a new technology which obtains unpredictable results sometimes.

The technology included in *Forza Motor Sport* was called *Drivatars* [31]. What the *Drivatars* did to imitate the player behaviour was to include a new game mode where the player can train its avatar. In this mode the player must complete some tasks [32] before getting his own avatar, like driving alone in a track, overtaking some opponents, blocking opponents that attempt to overtake (these tasks are the ones we have identified in the beginning of this chapter as the basic skills of a NPC in order to create a believable behaviour) and other tasks less relevant, such as the pit stop. Once the player completes these tasks the game creates a statistical model of the player behaviour for each driving skill, and when the *Drivatar* replaces the human in a race it shows a behaviour that matches this model. For example, in order to drive alone in a track the *Drivatars* technology creates a statistical model of the race line in each segment of the track. This model corresponds with the trajectory followed by the human player during the training phase. Then, when the *Drivatar* is driving the car, a race line is built up per segment of the track with the statistical models, and then these race lines are joined to create the complete trajectory. This approach to imitating the human drive style has some problems. The first one is that when the avatar drives in a track with unknown segments, segments the human never drove during the training phase, it is not able to create the race line. Another problem with the race line is that, due to the track being split into segments and the model being created per segment, the borders of the race lines in two adjacent segments may not match or the join can be so sharpened that it is physically impossible for the car to follow that trajectory. They solved this problem by increasing the grip of the cars, but not for all the cars in the game, only those driven by the avatars. We can say that the avatars have some advantage with respect to the other players as the car has more grip. The consequence of this cheat is that human players that try to follow the avatar's cars cannot, it is impossible for them. They are not believable NPCs when you drive against them, as it is noticeable they are able to perform the turns in an unrealistic way.

Although we argue that none of the industry NPCs are believable, all the described games made big profits. We are not sure whether the reason was the techniques involved to create more challenge players because some of the games do not use the same techniques in newer versions, but the industry has shown interest in improving their NPCs.

12.2.2 Academia NPCs

The researchers' interests are more focused on new methods to create NPCs automatically with human-level behaviours than on creating engaging and competitive drivers for videogames that increase the sales of the game. New AI techniques such as neuroevolution [17], fuzzy logic [12] or multi-objective evolutionary algorithms (MOEAs) [15] are used to achieve this task.

Neuroevolution is a method to create neural networks through genetic algorithms [28]. Instead of using a mathematical method to train a neural network with a fixed

structure, in neuroevolution a genetic algorithm creates and evolves the structure of the neural network and its internal parameters. In [8] the authors used neuroevolution in order to create neural networks to control the car in the *Screaming Racers* game, a very simple non-commercial game developed for research. Their objective was to probe whether general driver NPCs can be created with neuroevolution. In [5] the authors also use neuroevolution to create a controller for TORCS, a more realistic car racing simulator. The controller was presented to the *Simulated car racing championship* [16] obtaining interesting results compared with other entries, but compared with humans its believability and competitiveness it is far from being a good human-like NPC.

Fuzzy logic is a multi-valued logic based on fuzzy sets, where the logical solutions of the rules are an approximation rather than an accurate value. From the point of view of artificial behaviour, those created with fuzzy logic should be more human-like, due to their uncertainty, than those created with classical logic which gives always a precise value for the rules. Some driver controllers were created with fuzzy logic [10, 16] and the reactions of these controllers seems to be more natural than other kinds of controllers, however the global behaviours are definitely not human-like. In general, the authors of this type of controller search for competitive drivers not human-like behaviours.

Other research works extended the problem of creating drivers for car racing games, and they take into account the believability of the agents in their AI algorithms, furthermore the interest and competitiveness. In [35] the authors used a MOEA along with Genetic Programming [13] to evolve different controllers for a car racing game by imitating human players. The goal of this work was to obtain a set of individuals with different behaviours because, as the authors argue, a car racing game needs NPCs with different behaviours to be believable. If all the NPCs behave in the same way, they would become boring and lacking in believability. The MOEA allows the authors to create the set of NPC but at the same time it was used to create different objectives to measure the competitiveness of the NPC and the similarity of the controller with the human players drive styles. The competitiveness was measured with the lap times (less time means more competitive). The other two objectives were direct measures of how similar are the acceleration and the steering in the evolved controller with respect to the human player.

In general, the video game NPCs are specifically created for the games, for the tracks included in the games, and in some cases these controllers are not able to drive correctly in new tracks. When we test an academic controller in a game competing against the in-built cars, the game NPCs are usually faster than the academic. But the NPCs created by the researchers are more general controllers, they are able to drive in more tracks than those used to train the controller, where the game controllers would not drive correctly. Researchers are focused on creating general drivers and the industry in competitive and enjoyable drivers.

12.3 A Believable Driver

We are not aware of any method to measure the believability of the controllers for car racing games, although some approaches have been used for games (see Chap. 11). The Turing Test [34] could be used as a good approach to measure the human-like behaviour of any given NPC, but it only gives us a binary output: it is believable or it is not; nothing about how believable the NPC is. Following a similar process to the original Turing Test we could define a Turing Test for racing NPC, as has been done in *first person shooting* (FPS) games in the *2K BotPrize* competition [9]. Human players and NPC could compete in a race and after the race the human player could vote on whether the other competitors were human or NPC. These could be used as an indirect measure of the human-like behaviour of the bots. In this chapter we propose the creation of NPC by imitation, therefore the Turing Test to measure the believability is not suitable for us. Instead of this method we use other indirect measures: the lap times, the top speed, the difference between the current speed and the speed of the human at that point, and differences between the human trajectory and the NPC trajectory.

We also argued at the beginning of this chapter that considering a behaviour believable or not in a game depends on the skills of the person judging. A novice player in a game would rarely be able to judge correctly his or her opponents in the game if he or she does not have enough knowledge about the game and the typical behaviours to play it. The complexity of the games also affects the ability to judge. Games with rich worlds with a lot of possible interactions require more time for the player to learn the useful behaviours and it is harder for the programmers to create good NPCs. By contrast, for simple games (such as tic-tac-toe) the NPC players are as good as human players and they are usually indistinguishable from them. Racing games are located in the middle of complex and simple games, their complexity is simple enough to think that believable NPCs can be achieved. From the point of view of a human who judges the believability of a NPC, it has to manifest, at least, some typical behaviours of the drivers in the racing games. We think this minimal set of behaviours is: drive in a near-optimal trajectory, avoid crashes with walls and other cars, overtake opponents, prevent opponent overtaking and recovering.

As a competitive driver it is necessary to drive in an optimal or near-optimal trajectory along the track in order to achieve the best lap times. The optimal trajectory is unknown most of the time and it is almost impossible to follow it perfectly, therefore the human drivers in real races try to follow the best trajectory they can (a near-optimal trajectory) at the top speed the car allows without making mistakes that would lead to losing time in the lap. A human player could not be considered to be believable if it drives without following a good trajectory.

While driving in a track against opponents sometimes these opponents make mistakes and their cars stop in the middle of the suboptimal trajectory. It does not make sense for a car to continue its trajectory and crash into these opponents. A believable behaviour would be to change the trajectory to avoid these obstacles, unless the event has happened recently and there is not enough time to take appropriate

actions to avoid the collision. Also, at the beginning of the races all the cars start very close to each other and it is not possible to follow the trajectory without hitting other cars. Another case is when the car is going to perform a turn and has another car in front of it which is slower and keeping the same velocity will cause the car to hit the opponent. In all of these situations the driver must react and avoid the collision.

Another behaviour the NPC must show is overtaking, because a driver that never overtakes, or never tries to, is not believable. We can consider here two special cases, one for the straights and another for the turns. In the former, if an opponent is slower than the car, it only has to modify its trajectory to avoid hitting the opponent and when it finishes the overtaking returns to the original trajectory. In the latter, the overtaking can be done on the inner side of the turn or on the external side. The most frequent overtaking is done on the inner side, the car uses an advantage to get the inner side of the turn (see Fig. 12.3), either it is faster or brakes later, and as it covers less distance than the opponent it completes the turn first, probably with a higher speed and can return to the near-optimal trajectory after the turn. Overtaking on the external side of a turn is only possible if the opponent is much slower than the car, or when there are two linked turns, one to the left and the next one to the right, and getting the external side of the first turn means gaining the inner side in the next one. These are not the only chances to overtake an opponent, in some special cases the driver can perform a new imaginative move, or it is possible that the car does not have to modify its trajectory to perform the overtaking because the opponent is in another separate path.

While overtaking is a necessary behaviour to consider a driver believable, preventing an opponent overtaking is another one. If the driver of the car notices that the opponent behind him is trying to overtake, the normal behaviour in this case is modifying the current trajectory in order to stay ahead of the opponent all the time. The believable driver would not let an opponent have enough room to get a privileged position, unless the opponent goes so fast that this maneuver becomes dangerous and could produce a collision, or unless the car needs to come back to its trajectory in order to approach the next turn competitively.

The last behaviour a NPC must have to seem like a human driver is to recover from an unexpected situation. An example could be an opponent hitting the car, or the driver performing a turn wrongly and going off the road. When such an unexpected situation happens, the car has to be able to return to normal driving. If the car is still on the track the normal behaviour would be to turn the car until it is aligned with the track or the trajectory. When it is outside the track the car first has to turn enough to go back to the track and then align the car with the track or the trajectory. In both cases, an unexpected situation is often reached because there was, at least, another opponent involved and probably this opponent is also trying to recover. Therefore, while the recovering behaviour is performed it has to take into account the other cars to avoid collisions.

Academic researchers usually split problems into simpler ones, try to solve them and finally integrate all the solutions to solve the original problem. When we talk about games some researchers have focused their work on finding optimal or near-optimal trajectories and creating controllers able to follow these trajectories [5].

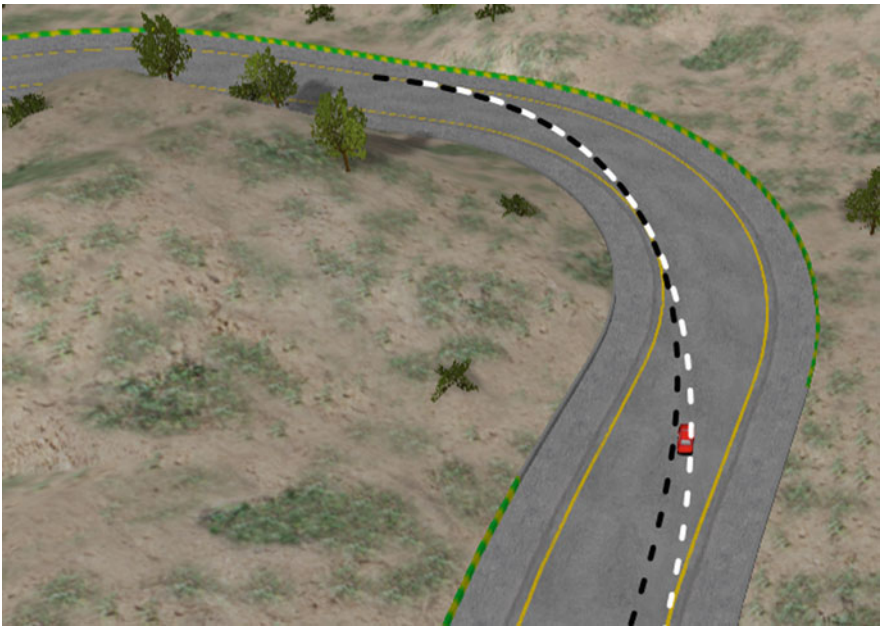


Fig. 12.3 Overtaking example in a turn. *White line* is the normal trajectory, *black line* is the overtaking trajectory, the inner side of the turn

Other researchers try to learn the behaviour to overtake opponents [25], with machine learning or other techniques. These controllers won't present a completely believable behaviour unless all behaviours are integrated effectively in the NPC. One fact that reveals the NPC has been programmed is their predictability, programmed driving is deterministic. Given a set of similar situations the artificial driver will do always the same actions and an experienced player will notice this fact. This happens because of the techniques used to create the behaviours. Game companies want to know how their NPCs are going to behave in all situations and for this reason they use narrow and deterministic AI algorithms like finite state machines, fuzzy logic, behaviour trees, etc. [7, 21]. One solution to the predictability of the NPCs is to include in the game NPCs with different behaviours. With one different behaviour for each car, the game becomes more entertaining and believable. It is more difficult for a human player to know what the opponents are going to do. But this wealth of behaviours requires more time and effort for the game companies.

12.4 Imitation Learning

Considering the definition of believability for computer games as the "*illusion that the agent is controlled by a player*" [29], imitation learning seems one of the best

approaches to create believable NPCs. Imitation is defined in the dictionary as "*copying (or trying to copy) the actions of someone else*". If a NPC could copy the action it sees in human players and could use its learned behaviours for its goals, then the NPC would show the illusion of being controlled by a player [30]. We said before that in order to consider a NPC believable, not all the NPCs in the game have to show the same behaviour, each NPC has to show its own behaviour, although this is very similar to the others. Imitation learning can be used to create different behaviours for a game only using different sets of examples of human driving to train the algorithms. The efforts required to have a varied set of behaviours should be smaller than with other approaches. Imitation learning should solve the problem of believability in games. However, to copy the actions of a human player is a very difficult problem.

Imitation learning has been used actively in robotics, and they have dealt with some problems (like the correspondence problem or goal inferring) that have to been solved in virtual worlds such as games too. When the robot wants to imitate a behaviour, it first looks at what the human teacher is doing, and then it has to transform what it sees into an abstract model and map the object-centered representation into an egocentric representation [27]. For this correspondence we have to solve the problem of the noise of the sensors in the robot, and perform a complex processing to translate what the robot sees into an abstract model useful to work with. Providently, we avoid this problem when we are using video games because we have the whole information of the environment and of all the agents in the game, there is no noise in the virtual environment unless we add it on purpose. The second problem is the meaning of the human behaviour—imitation learning is a good choice when we imitate a behaviour if we know for what it is for. It is not worthwhile imitating a behaviour when we do not know what goal the behaviour aims to achieve, or a useless one that we will never use. The robot or NPC that wants to imitate a behaviour should know the final goal of the behaviour, and this abstract information has to be communicated somehow to the NPC before learning the behaviour. When the communication is not possible, as happens with infants who have not learnt to talk yet, they have to infer the demonstrators' intended goals [27]. These problems, communication and goal inferring, are beyond our interest and we will consider that the goal of the behaviours to imitate are previously programmed in the NPC. And the last problem when we imitate a human, and the most relevant for us when imitating a racing driver in a game, is that humans do not perform the same task in the same way twice. Therefore, when the task is translated into data to learn the behaviour, this data has noise and depending on the amount of noise it could be impossible to learn anything for a machine learning algorithm. It will be important to choose the proper inputs and outputs in order to create the data sets for the training and avoid the noise effect in the learning process.

As well as the problem of learning a behaviour, it is important to bear in mind some problems with the learned behaviours. We are working with believable NPCs and as they are computer programs these always are going to do what they are programmed to. When they have learnt a behaviour, they will repeat exactly the same behaviour over and over. We have to wonder whether "*to repeat perfectly a learned behaviour*

is believable?". If believability means seems to be human, the answer is not. As humans, we are not perfect and we make mistakes, we can do more or less but we end up making a mistake at some point, it is very difficult for a human to repeat exactly the same action twice, there will be some small variations, some of them leading to a mistake. An expert human in a task is not going to make any mistake often, and as we usually would want to imitate experts, the NPC should not usually make mistakes. We can minimize this problem of the perfect NPC by including some noise in the effectors of the NPC in order to not allow the NPC to repeat exactly the same behaviour it has learnt. But, it is not the only problem, we wonder whether "*it is believable to show the same behaviour over and over?*". Well, if the behaviour is the best one for the final goal the answer is true, but if it is not the best one, a believable agent must have the skills to learn a new and better behaviour and repeat it. Humans have the ability to adapt to new situations by changing their behaviours, and so a believable NPC should do the same, adapting its behaviours whether or not they are the best ones or whether the situation changes and a new behaviour is required to accomplish its goals.

Imitation learning can be applied to different levels of abstraction, from a very simple action like teaching a robot to move an arm, to more complex actions like teaching a robot to grab an object over a table, or even teaching a NPC to drive a car. The simplest behaviours, which we can consider as low-level abstract behaviours, are those closer to the sensors and effectors of the robot or NPC, while the complex behaviours are more related with top-level abstract information. It is not the same whether we want to imitate a very simple action or a complex sequence of actions to accomplish a goal—depending on the kind of task we want to imitate the algorithms and the data set should be different.

Focusing on our problem of creating believable drivers with imitation learning, "*what are the kinds of actions we have to imitate to achieve a believable behaviour?*". They could be the low-level actions like how much the car must steer the wheel in a turn, or how much it has to step on the accelerator pedal, or more top-level information like what is the trajectory the car must follow. When we use a low-level representation of the environment and we try to learn the matching between the inputs—sensory information of the environment—and the outputs—the motor skill of the NPC in the environment—the amount of noise in the data, the lack of preprocessing to use more abstract information, and the amount of tasks necessary for driving, make it impossible to learn a useful and admissible behaviour [22]. In environments like racing simulators, the amount of information you can use from the environment is too much, you have to process it somehow in order to get only the useful abstract data for creating the believable behaviour. Also, trying to learn all the required behaviours together to drive a car complicates the task for the learning algorithms. The believable behaviour we want to achieve has to be split into independent smaller behaviours and then we would be able learn them separately. We identified five required behaviours to consider a NPC believable: drive fast, overtake, prevent overtaking, recover and avoid crashing; but more skills could be needed for different games.

12.4.1 Complete a Lap Competitively

The first and probably the main behaviour in order to create a believable NPC is to develop a fast driver, one that can complete a lap in a competitive time. For this goal, the NPC should follow an optimal trajectory with the highest speed it can. The question is how could the NPC create a trajectory good enough and a speed model for any track that leads it to have good lap times. We also have to solve how to make it follow the trajectory at the corresponding speed once it has the trajectory and the speed model.

In order to answer previous questions we need a model of the track, that is, an abstract representation of the track. The one we propose in this chapter is to divide the track into segments of the same length. With the track model and the information about where the car is in the track, we can store information on a human player's driving. Then we can train algorithms with the human data and learn the trajectory followed and the speed model. Once we have the trajectory and speed models we need an algorithm that uses this information and modifies the speed and the steering of the car in order to make it follow the trajectory with the correct speed. The trajectory can be followed taking into account the current error following the trajectory and the error between the car angle and the angle of the trajectory in the next segments. For the speed, another algorithm can use the current speed and the target speed as inputs and set the proper values for the accelerator and brake pedals. In both cases the algorithms could be either scripted or we could use a machine learning algorithm. In the model proposed in this chapter we used a scripted policy, one linear function of the error in the trajectory in order to set the steering and another linear function for brake and accelerator pedals based on the error with respect to the target speed.

12.4.2 Dealing with Opponents

12.4.2.1 Overtaking

Once we include a trajectory and speed model, the car is able to complete a lap but not to deal with opponents. The next step is the overtaking feature; this is a difficult task to learn, it is complex, rarely performed and depends on each situation. Therefore, instead of learning this behaviour we propose an algorithm that modifies the trajectory in order to perform the overtaking.

We use a rule set in our model that changes the trajectory when the car should overtake an opponent. Previously we said that most overtaking is done on the inner side of a turn. Hence, when the car is close enough to an opponent and it is approaching, the overtaking in our model is done by modifying the trajectory and locating the

car on the inner side of a turn before its opponent. And if it does not have enough room on the inner side the trajectory is modified to use the outer side of the turn. When the trajectory is modified we have to bear in mind that big modifications of the trajectory in turns can lead the car to spin or go off the track. Therefore, the trajectory cannot be changed so much in turns. It can only manage greater modifications in a straight when the next turn is still far away and the car has time to come back to its optimal trajectory.

12.4.2.2 Preventing Overtaking

Another behaviour to deal with opponents is to avoid being overtaken. We developed this behaviour in our model in a similar way to overtaking: by modifying the trajectory. If the car is not overtaking any opponent but an opponent is approaching, we only have to modify the trajectory in order to stay ahead of the opponent. As in the overtaking the extent to which the trajectory is modified depends on whether the car is on a straight or in a turn. The modification will be bigger if it is on a straight and smaller if it is on a turn or close to the next turn.

12.4.2.3 Avoiding Collisions

In order to avoid collisions, we used the same approach as with the other behaviours to deal with opponents, the car modifies its trajectory and speed. This behaviour has more priority than the others and inhibits them when fired. We used a rule set for this behaviour. When the car is approaching an opponent very fast (maybe because the car ahead has a problem) it has to brake and to modify its trajectory to avoid the collision. Another situation is when the opponent is alongside the car and the trajectory moves in tht direction, then the car has to modify its trajectory to avoid a collision. We do not take into account any cars behind us, we assume that they will try to avoid a collision, although this does not happen always and leads to unexpected situations. If there is more than one opponent that can crash into the car, then the controller tries to avoid colliding with the car that is closest to it. Usually when trying to avoid a collision with more than one car the situation is too complex even for a human, and the most common response in this case is to only focus our attention on the car which is more likely to crash into us.

12.4.3 Recovering

The recovery behaviour depends on the concrete situation. The most common behaviour is to turn the car back to the road, and then start to follow the trajectory again. It's very difficult to learn this behaviour with an algorithm using human data, it is much easier to use a preprogrammed routine. This behaviour is rarely used in a race, but cars that need to recover from an unexpected situation and do not do so properly are rapidly considered non-human drivers.

12.5 Imitation Learning in TORCS

In order to check our model of believability for racing games we use *TORCS* [16, 33] as a benchmark. *TORCS* is the acronym for *The Open Racing Car Simulator*, an open source realistic racing videogame. We use the application programming interface (API) provided by the *Simulated car racing championship* [16]. The API of *TORCS* provides us with environment information, some of which we select to create our believable NPC:

- Distance from the start line.
- Current gear.
- Revolutions per minute (RPM).
- Position of the car relative to the centre of the track.
- Angle of the car relative to the track.
- Current speed.
- 36 range finders to know the proximity of the opponents (in the 360° around the car, 10° of separation between two adjacent sensors, 100 m of precision).
- 19 range finders to know the distance between the car and the limits of the road (180° only ahead of the car, 10° of separation between two adjacent sensors, 200 m of precision).

It also provides some effectors to control the car:

- Accelerator pedal.
- Brake pedal.
- Steering wheel.
- Gear change.
- Clutch.

The clutch and the gear change are not very relevant in order to create our believable NPC. The clutch is only used at the start of the race, and the gear change is something you cannot observe directly and it is not directly related with the final behaviour. Creating a complex or believable control for the clutch and the gear change is not worthwhile because the final user is not going to notice them. Hence, we created a scripted policy to control the clutch at the start, starting with a high

value and decreasing it over time until the clutch pedal is completely released. For the gear we also used another scripted policy which increases by one gear when the RPM is higher than a value and decreases by a gear when it is lower than a threshold, with some special cases for recovering and for the neutral point. Modifications in the extent to which the scripted policy changes the gear or modifies the clutch are not relevant, as they only affect the performance and not the believability. The scripted policy was also used for the human driver while we were collecting the training data, therefore the gear changes and clutch do not affect the final lap times of the NPC when we compare it with the human driver. A more detailed description of these scripted policies can be found in our previous work [23].

A good trajectory to follow is the key factor of the NPC. We collected human data during driving on different tracks and used it to train neural networks. These networks predict the trajectory followed by the human player (the trajectory model) and the speed at any point in the trajectory (the speed model).

The trajectory is predicted with two neural networks. Their inputs are information about the shape of the track, what we call the track model. This model is a representation of the track in segments of 5 m. It stores the difference of the angles between two consecutive segments. The networks used as inputs the last 15 segments and the next 50 segments of the current position of the car. The output of the first network is the distance between the centre of the track and the trajectory point, the predicted position (p_p), and the output of the second network is the distance between the current trajectory point and the last trajectory point, the predicted difference position (p_{dp}). The result of both networks is combined and the result is a trajectory point (t_p). We used these two neural networks and combined the results in order to decrease the noise during the training and because in our first experiments with one network the trajectory was very sharp and the NPC was not able to follow it correctly. The trajectory points are calculated as in Eq. 12.1, where α is a parameter equal to 0.5 and t_{p-1} is the previous trajectory point.

$$t_p = \alpha \cdot p_p + (1 - \alpha) \cdot (t_{p-1} + p_{dp}) \tag{12.1}$$

For the velocity model, the target speed at each point of the track (t_s), we use a similar process as in the trajectory model with two neural networks. In this case the inputs are the last 15 segments and the next 50 segments from the current position, the trajectory angles of the last 10 segments and the next 40 segments (we need the trajectory model before the speed model), and the current position of the car from the middle of the track. The outputs are the speed at a given trajectory point (p_s) for the first network, and the difference between the current speed and the target speed of the next segment (p_{ds}). The idea is the same we used to predict the trajectory. Equation 12.2 shows how the target speed is calculated. The parameter β is equal to 0.2 when p_d is positive and 0.6 when p_d is negative,

$$t_s = \beta \cdot p_s + (1 - \beta) \cdot (c_s + p_{ds}) \tag{12.2}$$

Once we have the trajectory and the speed models, we have to translate these intentions into actions over the pedals and the steering to control the car. The speed is easily adjustable by the pedals, if the target speed is greater than the current speed the car has to accelerate and to brake otherwise. But a binary model produces fast and abrupt changes in the pedals; therefore, we used a linear model when the current speed is within 10 km/h of the target speed. For example if the current speed is 5 km/h lower than the target speed then the accelerator pedal is stepped up to halfway instead of fully.

In order to follow the trajectory we use a function that steers the wheel proportional to the angle of the car with a point in the trajectory 20 m in front of the car (angle a in Fig. 12.4) plus the error of the current position of the car with the target trajectory (distance d in Fig. 12.4). Equation 12.4 shows how the final steering value (s) is calculated: γ is the maximum turn of the wheels in radians (0.785389); δ, ρ and σ are adjustment parameters set to 0.4, 8 and 5, respectively; and c is a value between -1 and 1 of the distance d in Eq. 12.3 (where λ is the maximum error considered in the distance between the car and the trajectory, equal to 7 m).

$$c = min\left(1, max(-1, \frac{d}{\lambda})\right) \tag{12.3}$$

$$s = a \cdot \gamma + \left(1 - \frac{1}{e^{c \cdot \rho + \sigma}}\right) \cdot \delta \tag{12.4}$$

12.5.1 Experiments

We checked our NPC out in the game using a set of tracks with similar properties, such as similar turns, similar widths and similar surface grips. We let an expert human player drive in the tracks for two laps and stored the data of the second lap with his driving style. If the human made a mistake in a lap we repeated the process until he performed correctly two laps. We used only one lap in order to decrease the probability of making mistakes by the human and to decrease the noise in the data set. Professional drivers are able to perform laps in the same way, but our expert human player was not so good. Table 12.1 shows the tracks used to collect the data, the results of the human player (the best lap time in seconds and the top speed reached in km/h), and the number of patterns for each track.

The neural networks used to predict the models have 2 hidden layers with 20 neurons in the first hidden layer and 10 in the second. The networks were trained using the standard backpropagation algorithm during 20,000 cycles. We used a variable learning rate that decreases linearly with 0.8 as initial value and 0.0001 as final value. Per track we created a set of patterns to compare with the other tracks, and we did not use the same track where we tested the NPC to train the neural networks. This set of patterns is divided into two sets, one for training and the other for testing.

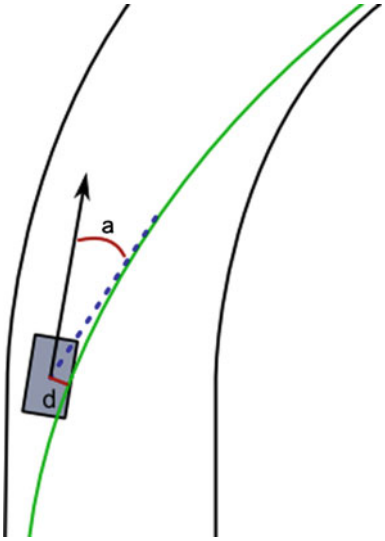


Fig. 12.4 Representation of how the car corrects its position in order to follow the trajectory (*green line*), taking into account the current error in the position with respect to the trajectory (the distance *d*) and the angle (*a*) between the car and the trajectory in the next 20 m

Table 12.1 Human times and number of patterns per track

Track	Best lap	Top speed	Number of patterns
CG speedway 1	41.27	243	1030
Aalborg	78.54	228	1294
E-Road	66.55	265	1632
Alpine 2	100.72	234	1888
Street 1	81.62	275	1912
Ruudskogen	66.60	248	1638
Wheel 1	82.84	267	2130

The training set is four times bigger than the test set. The test set is used to avoid overlearning—we store the neural network that has the lower error on the test set.

Table 12.2 shows the results of the believable NPC created, containing the best lap time, the top speed, a column with the percentage of increment between the best lap time compared with the human player best lap time, and a final column showing the percentage of decrement between the top speeds. Positive values in the last two columns point to a higher lap time or top speed, while negative values point to a lower lap time or top speed.

Table 12.2 with the NPC times shows that the NPC is slower, around 10 % slower, in most of the tracks than the human. The cause of this higher lap time is that the neural networks did not learn the correct speeds. They got stuck at 203 km/h and did not learn higher speeds. The lost time in the laps was because the NPC did not achieve

Table 12.2 NPC times

Track	Best lap	Top speed	Best lap increment (%)	Top speed decrement (%)
CG Speedway 1	45.63	203	10.56	−16.46
Aalborg	94.64	199	20.50	−12.72
E-Road	78.81	201	18.42	−24.15
Alpine 2	96.46	204	−4.23	−12.82
Street 1	91.93	201	12.63	−26.91
Ruudskogen	69.83	201	4.85	−18.95
Wheel 1	91.18	198	10.07	−25.84

the same top speed in the straights as the human player did. The more straights the track had, the longer the lost time, as happened in *E-Road*, the track with more and longer straights. In *Aalborg* the lost time was because the car spun in a turn and went off the track, and it lost time while recovering. The only track where the NPC achieved better times was *Alpine 2*, the reason being that this track has a slippier surface and has fences that prevent cars going off the track. The cars went faster than they should but did not lose time because they crashed against the fences and never went off the track.

Looking at the trajectories predicted by the neural networks, they look like human trajectories. We can say that the proposed method works well for generalizing trajectories for new tracks. Although the speed model is well predicted for low speed, our method is not able to learn the top speeds. This means that the general drive style seems like that of the human player but it has higher lap times.

The behaviours we identified as necessary in order to create a believable NPC—overtaking, preventing overtaking, recovery, and avoiding crashes—were also developed for the NPC as we explained in the previous section (Sect. 12.4). They were not taken into account in Table 12.2 because these experiments run without any opponent, and these behaviours are oriented to deal with other cars. Nevertheless, we entered the NPC created by imitation in the *Simulated car racing championship* in 2010. This competition is divided into three legs and each leg consists of three races on unknown tracks. Each race has a period of learning—the warm-up—then qualifying, and finally the race. We presented in this competition a modified version of the NPC that uses the learning phase to increment the top speed of the speed model, fixing the problem of learning the top speeds. This learning allowed us to increase the speed in the segments where the car was able to follow the trajectory perfectly. For our controller the most interesting leg was the second one because it has the tracks that are most similar to those we used to train the controller. These tracks were the most F1-like in the competition and also the tracks that require more skill in order to be faster than the opponents.

Table 12.3 Simulated car racing championship 2010, second leg: qualifying

NPC	Wild-speed	Petit	Brondehach	Total points
COBOSTAR	10	10	10	30
Autopia	8	5	8	21
Jorge	6	6	3	15
Cardamone	5	4	5	14
Timothy Alford	4	8	2	14
Mr. Racer	3	3	4	10
Joseph Alton	2	2	6	10

Table 12.4 Simulated car racing championship, second leg: race

NPC	Wild-speed	Petit	Brondehach	Total points
Jorge	10	10	8	28
COBOSTAR	8	10	10	28
Autopia	10	8	6	24
Cardamone	5	5	6	16
Joseph Alton	2	4	5	11
Timothy Alford	4	3	3	10
Mr. Racer	4	3	2	9

Table 12.3 shows the results of the qualifying, the score system is that used a few years ago in F1: 10 points for first, 8 for second, 6 for third, 5 for fourth, 4 for fifth, 3 for sixth, 2 for seventh, and 1 for eighth. In the qualifying we were the third fastest car in the first two tracks and sixth in the last one. Table 12.4 shows the results of the race. There were 2 extra points in the race score for the car with the fastest lap and 2 more for the car with the least damage. Only two cars improved their results in the race: our controller and the one that was last during qualifying. We did not get any extra point but our controller was able to win the first two races and came second in the third. Our controller was neither the fastest nor the safest and we did not get any extra points. Although the organizers said that it was very difficult to deal with opponents in this type of track,[1] our controller was able to overtake opponents and win two races and be second in the third one. Our approach of modifying the trajectory in order to deal with opponents and overtake them has very good results compared with the other competitors. We cannot do a deeper analysis as we do not have complete videos of the races, but our hypothesis is that human-like behaviours like the ones in our controller are required in order to overtake in complex tracks against artificial controllers.

[1] Slides with the results and comments about the competition can be found at: http://cig.ws.dei.polimi.it/?p=166

12.6 Discussion

Imitation learning is a good approach to create believable drivers for racing games. Indeed, once the method is developed for one NPC it can be extended to create more NPCs. Imitation learning could be used to create a set of NPCs with different human-like styles, giving the game more realistic opponents.

We argue that five different behaviours are required in order for NPCs to show a believable behaviour: drive fast, overtake, prevent overtaking, avoid collisions and recover. Nevertheless, not all the behaviours of the drivers must be learnt by imitation, most of them are rarely used by a human, they depend on the concrete situations and are difficult to learn. Hence, it is easier to implement them by hand, and as they are not frequently used they will not be a high impact on the believability of the NPC. One future line of research is to analyze whether these behaviours are required in human-like controllers or not, and how it affects believability if one or more behaviours is removed from the NPC. This analysis could be done by performing a Turing Test over NPCs and analyzing the comments of the human judges.

There are some disadvantages of current imitation learning methods. In order to learn a behaviour the machine learning algorithms need enough samples of data from the human and the set of samples have to be consistent. If the data samples are obtained from a human that does not perform two laps in the same way, there will be a lot of noise in the data and the algorithm will not learn the human behaviour. Another problem is that the data have to be preprocessed in order to split the behaviours (drive fast, overtake, prevent overtaking, recovering, etc) and learn them separately. To learn more than one behaviour at a time means noise for the most common machine learning algorithms and they end up without learning any behaviour.

The greatest disadvantage of our method to create believable NPCs is that the less frequent actions are not learnt by the neural networks. It is very noticeable for the top speed, as it is only reached at the end of long straights it is not a frequent action and there are few samples of the top speed. The neural networks are not able to learn it and probably it is considered noise in the training data. For this reason the controller has a lower top speed than the human player and its times are slower in the faster tracks, those with more straights or fast turns.

Despite the problems in the learning algorithm, the presented model is a promising approach for creating NPCs by imitation. The created controller is faster than the human in one of the tracks and performs very well in the others. We think that if we can solve the problem of the top speed the final NPC will be a very competitive controller. The presented method also has the advantage that the NPC can be training while the human improves his or her skills, keeping the competitiveness at the same level as the human player.

Acknowledgments We would like to thank the reviewers and the editor of the book for their constructive comments. We have done our best to include all the comments and to improve the quality of the chapter.

References

1. Bates, J.: SCIENCE., C.M.U.P.P.D.O.C.: The role of emotion in believable agents. Commun. ACM **37**(7), 122–125 (1994)
2. Biasillo, G.: Racing AI Logic. AI Game Programming Wisdom, pp. 444–454. Charles River Media, Hingham (2002)
3. Bishop, C.: Neural networks and their applications. Rev. Sci. Instrum. **65**(6), 1803–1832 (2009)
4. Buckland, M.: Colin McRae Rally 2.0. Interview with Jeff Hannan. http://ai-junkie.com/misc/hannan/hannan.html. Accessed Nov 2010
5. Cardamone, L., Loiacono, D., Lanzi, P.L.: Learning to drive in theopen racing car simulator using online neuroevolution. Computational Intelligence and AI in Games. IEEE Trans. **2**(3), 176–190 (2010). doi:10.1109/TCIAIG.2010.2052102
6. Chaperot, B., Fyfe, C.: Improving artificial intelligence in a motocross game. In: 2006 IEEE Symposium on Computational Intelligence and Games, pp. 181–186 (2006)
7. Cutumisu, M., Szafron, D.: An architecture for game behavior AI: behavior multi-queues. In: Artificial Intelligence for Interactive Digital entertainment Conference (2009)
8. Gallego, F., Llorens, F., Pujol, M., Rizo, R.: Driving-Bots with a Neuroevolved Brain: Screaming Racers. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial **9**(28), 9–16 (2005)
9. Hingston, P.: The 2K BotPrize. In: IEEE Symposium on Computational Intelligence and Games. IEEE (2009). http://www.botprize.org
10. Ho, D., Garibaldi, J.: A fuzzy approach for the 2007 CIG simulated car racing competition. In: IEEE Symposium On Computational Intelligence and Games, 2008. CIG'08, pp. 127–134. IEEE (2009)
11. Hunicke, R.: The case for dynamic difficulty adjustment in games. In: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology, pp. 429–433. ACM (2005)
12. Klir, G., Yuan, B.: Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice Hall PTR Upper Saddle River (1995)
13. Koza, J., Poli, R.: Genetic programming. Search Methodologies, pp. 127–164 (2005)
14. Laird, J., VanLent, M.: Human-level AI's killer application: interactive computer games. AI Mag. **22**(2), 15 (2001)
15. Li, H., Zhang, Q.: Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II. Evolutionary Computation, IEEE Trans. **13**(2), 284–302 (2009)
16. Loiacono, D., Lanzi, P., Togelius, J., Onieva, E., Pelta, D., Butz, M., Lonneker, T., Cardamone, L., Perez, D., Sáez, Y., et al.: The 2009 simulated car racing championship. IEEE Trans. Comput. Intell. AI in Games **2**(2), 131–147 (2010)
17. Lum, R., Meighan, M., Alfaro, H.: Using NEAT to Train a Player for the Iterated Prisoner Dilemma. Machine Learning (2008)
18. Mac Namee, B.: Proactive Persistent Agents-Using Situational Intelligence to Create Support Characters in Character-Centric Computer Games. Ph.D. thesis, Department of Computer Science, University of Dublin, Trinity College (2004)
19. Matthews, J.: Colin McRae Rally 2.0. Interview with Jeff Hannan (2001). http://www.generation5.org/content/2001/hannan.asp. Accessed Nov 2010
20. Microsoft Corporation: Drivatar. http://research.microsoft.com/en-us/projects/drivatar/forza.aspx. Accessed June 2011
21. Millington, I., Funge, J.: Artificial Intelligence for Games. Morgan Kaufmann, San Francisco (2009)
22. Muñoz, J., Gutierrez, G., Sanchis, A.: Controller for TORCS created by imitation. In: Proceedings 2009 IEEE Symposium on Computational Intelligence and Games, pp. 271–278 (2009)
23. Muñoz, J., Gutierrez, G., Sanchis, A.: A human-like TORCS controller for the Simulated Car Racing Championship. In: Proceedings 2010 IEEE Conference on Computational Intelligence and Games, pp. 473–480 (2010)

24. Nintendo: Mario Kart. http://www.mariokart.com/. Accessed Nov 2010
25. Onieva, E., Cardamone, L., Loiacono, D., Lanzi, P.: Overtaking opponents with blocking strategies using fuzzy logic. In: IEEE Symposium on Computational Intelligence and Games, pp. 123–130. IEEE (2010)
26. Polyphony Digital Inc.: Gran Turismo 5. The real driving simulator. http://www.gran-turismo.com/. Accessed Nov 2010
27. Rao, R., Shon, A., Meltzoff, A.: A Bayesian Model of Imitation in Infants and Robots. Imitation and social learning in robots, humans, and, animals (2005)
28. Reeves, C.: Genetic algorithms. Handbook of Metaheuristics, pp.109–139. Springer, New York (2010)
29. Tencé, F., Buche, C., De Loor, P., Marc, O.: The Challenge of Believability in Video Games: Definitions, Agents Models and Imitation Learning. Arxiv, preprint arXiv:1009.0451 (2010)
30. Thurau, C., Paczian, T., Bauckhage, C.: Is Bayesian imitation learning the route to believable gamebots, pp. 3–9. Proc. GAME-ON North, America (2005)
31. Tipping, M., Hatton, M.: Drivatars™ and Forza Motorsport (2006). http://www.vagamelabs.com/drivatars-trade-and-forza-motorsport.htm. Accessed Sept 2010
32. Tipping, M., Hatton, M.: Drivatars™ theory (2006). http://research.microsoft.com/en-us/projects/drivatar/theory.aspx. Accessed Nov 2010
33. TORCS (The Open Racing Car Simulator). http://torcs.sourceforge.net/ (2011). Accessed June 2011
34. Turing, A.: Computing Machinery and Intelligence, pp. 23–65. Epstein et al., Parsing the Turing Test Springer (2009)
35. Van Hoorn, N., Togelius, J., Wierstra, D., Schmidhuber, J.: Robust player imitation using multiobjective evolution. In: Proceedings of the IEEE Congress on Evolutionary Computation (2009)
36. Yang, X., Zheng, J.: Artificial Neural Networks. Handbook of Research on Geoinformatics. p. 122 (2009)

Index

P. Hingston (ed.), *Believable Bots*, DOI: 10.1007/978-3-642-32323-2,