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,

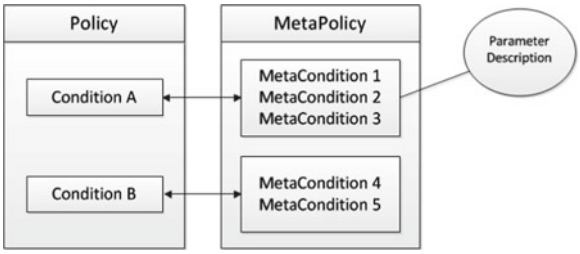

Fig. 10.1 A relation between a policy and a corresponding metapolicy. For each policy's parameter there is a corresponding parameter description. Parameter descriptions consist of a number of metaconditions

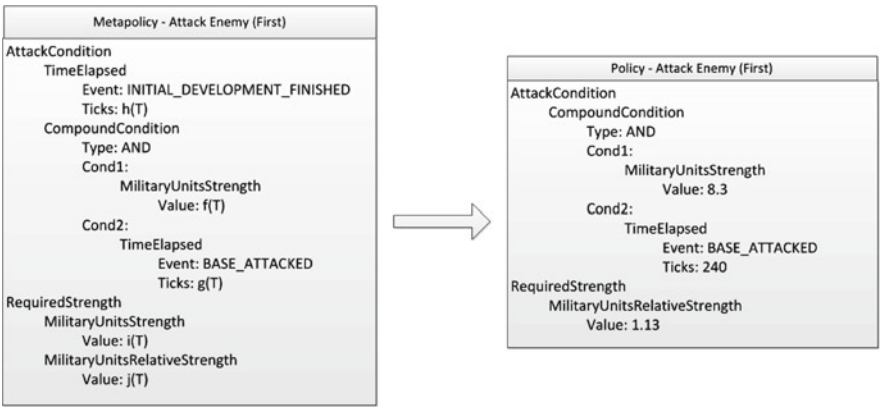

Fig. 10.2 An example of a metapolicy and policy generated from it. In the former, *AttackCondition* and *RequiredStrength* are parameter conditions and *TimeElapsed*, *CompoundCondition*, *Military UnitsStrength* and *MilitaryUnitsRelativeStrength* are metaconditions. f, g, h, i and j are functions taking 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

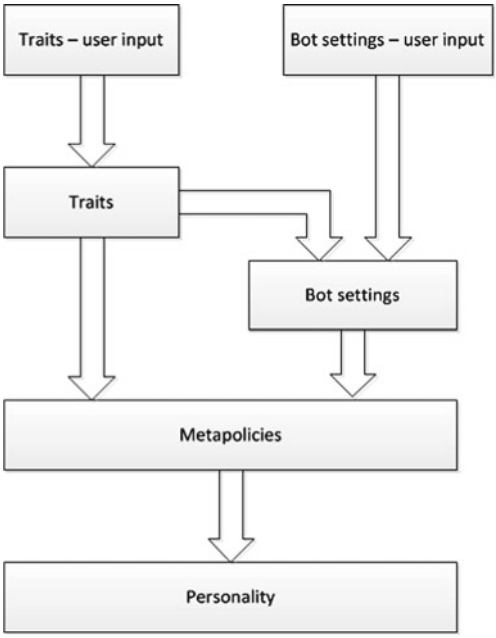

Fig. 10.3 Scheme of a personality creation process

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.

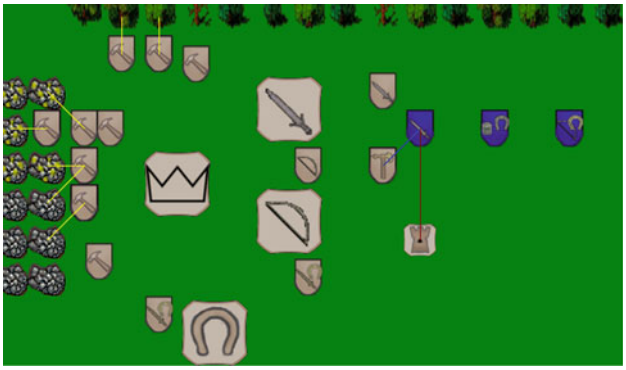

Fig. 10.4 First test environment. A *white* player base is attacked by a *blue* player

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.

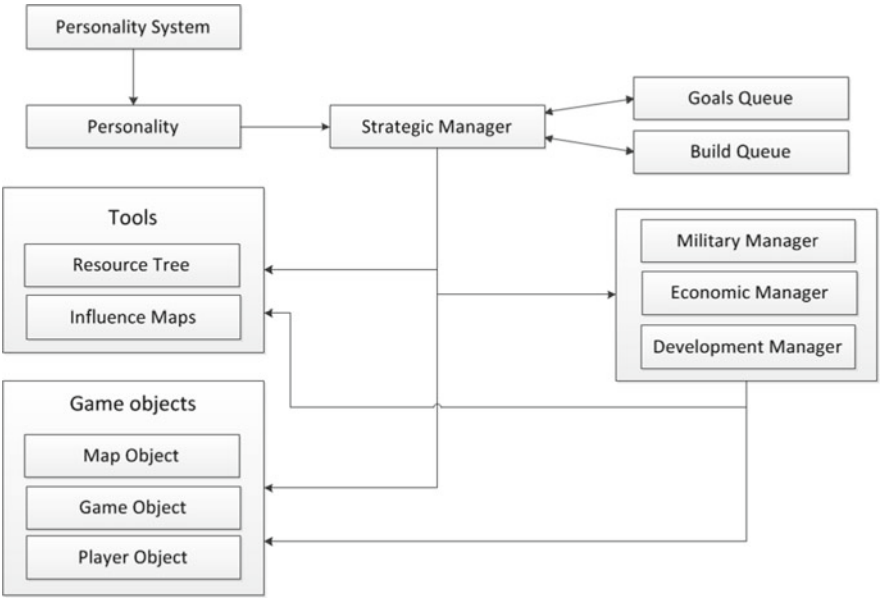

Fig. 10.6 Bot structure

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)