Behavior Capture with Acting Graph: A Knowledgebase for a Game AI System

Maxim Mozgovoy[1,*] and Iskander Umarov[2]

[1] University of Aizu, Tsuruga, Ikki-machi, Aizu-Wakamatsu, Fukushima, 965-8580 Japan
mozgovoy@u-aizu.ac.jp
[2] TruSoft Int'l Inc., 204 37th Ave. N #133, St. Petersburg, FL 33704 USA
umarov@trusoft.com

Abstract. Behavior capture is a popular experimental approach used to obtain human-like AI-controlled game characters through learning by observation and case-based reasoning. One of the challenges related to the development of behavior capture-based AI is the choice of appropriate data structure for agents' memory. In this paper, we consider the advantages of *acting graph* as a memory model and discuss related techniques, successfully applied in several experimental projects, dedicated to the creation of human-like behavior.

Keywords: Behavior capture, learning by observation, case-based reasoning, knowledge representation.

1 Building Believable Game Characters with Behavior Capture

1.1 Believable Behavior: A Key Feature of Game AI

Modern computer games and simulation-and-training applications are often characterized as "virtual worlds". This name emphasizes the growing complexity of game/simulation environments that are able to create higher sense of immersion than ever. This is done not only through hi-quality audiovisual technologies and detailed interactive physical models, but also with the help of modern AI methods.

Many virtual worlds are inhabited both by human-controlled characters and AI agents that serve as world's neutral "native population", allies or enemies. For example, in Unreal Tournament game (Deathmatch mode), independent players try to kill each other in a 3D map, and each player can be controlled either by human or by a computer (in this case it is usually called "a bot"). In general, computer-controlled characters are found in a variety of video games and training simulators. A good example of such simulator (or a "serious game") that involves computer-controlled opponents is Virtual Battle Space 2. This software is a variation of 3D world, specially designed for initial training of soldiers, and includes numerous training scenarios, ranging from vehicle driving in dangerous conditions and team combat to cultural-aware interaction with local population [1, 2].

[*] Supported in part by the Fukushima Prefectural Foundation, Project F-23-1, FY2011.

S. Kikuchi et al. (Eds.): DNIS 2011, LNCS 7108, pp. 68–77, 2011.
© Springer-Verlag Berlin Heidelberg 2011

Detailed and realistic virtual worlds set high demands on the quality of AI-controlled characters. Relatively simple game environments provide limited acting options for an AI engine, so handcrafted finite-state machine-based scripted decision making systems usually work well. Complex virtual worlds allow computer-controlled agents to exhibit complex behavior patterns, thus making the design of realistic human-like AI behavior an increasingly difficult task.

This trend is well known to both academic researchers and game creators. First, it is widely emphasized that today's AI-controlled game characters should be *believable*, i.e. human-like and virtually indistinguishable from human-controlled characters, in order to increase the overall enjoyability of a game [3, 4, 5]. Second, it is admitted that handcrafted AI systems are hardly able to provide believable behavior: scripted AI is easily recognized by experienced players, especially in complex virtual worlds. For example, even the best systems, participated in 2K BotPrize believability competition among Unreal Tournament bots were unable to deceive human judges [6].

1.2 Behavior Capture

In today's research projects human-like believable behavior is typically constructed by means of analyzing actual human behavior patterns and subsequently implementing them in AI system. Among them, most interest is evoked by the methods that can automatically construct agents' knowledge by observing behavior of human players. This process is known as *behavior capture* [7]. Behavior capture was used, for example, to build Unreal Tournament bots [8, 9], computer-controlled boxers [10, 11], and an AI system for a real-time strategy game [12].

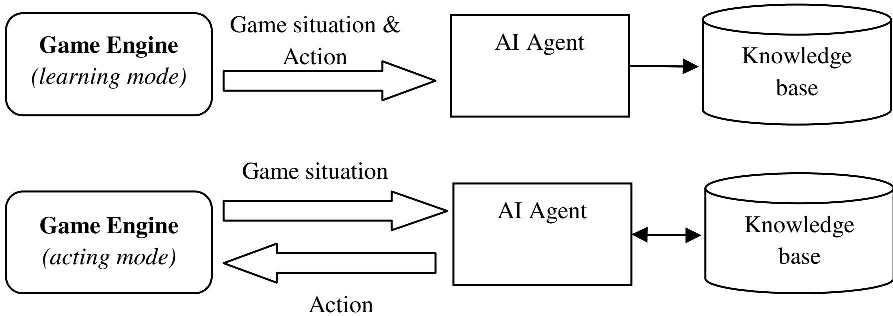

Fig. 1. Learning and acting of a behavior capture-based AI character

While general principles of behavior capture can be described as simply as "watch what the user does and try to reproduce the same patterns" (see Fig. 1), every particular game world sets own challenges. In our works [10, 11, 13] we identified several difficulties, related to practical implementation of behavior capture, common to a wide variety of computer games, and tried to address them in our AI architecture. Currently, our implementation is distributed as a set of tools and libraries under the

name of Artificial Contender [14]. Below we will introduce the method of representing agents' knowledge in Artificial Contender.

2 Knowledge Representation with Acting Graph

2.1 Addressing Challenges and Requirements

Our system was designed with the following goals in mind [15]:

- complex, non-repetitive behavior of AI agents;
- distinct personalities of AI characters, exhibiting a variety of skill levels and playing styles;
- the capability to design, edit and adjust AI's behavior (for a game designer).

These requirements served as a basis for our decision to use a variation of finite-state machine that we call *acting graph* as a primary data structure of an AI agent's knowledgebase (see Fig. 2; a similar solution was used in [9]).

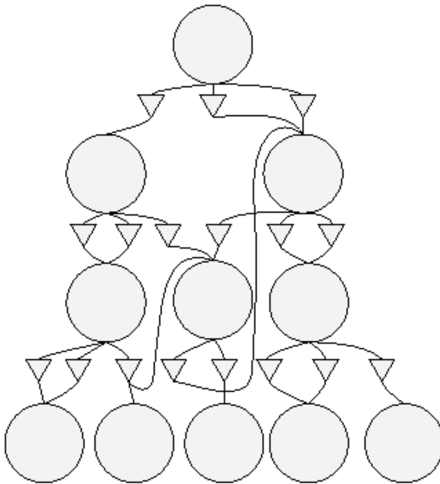

Fig. 2. Acting graph

The nodes of this graph correspond to game situations. Game situation is a unique description of the current state of the game world, represented with a set of numerical attributes, defined by the game designer. For example, for the game of boxing such attributes may include the coordinates of both opponents, their directions (where opponents look), body position (standing, leaning, blocking, etc.), health state of each player, and so on.

The edges of the graph correspond to the observed character's actions that introduce changes into the game states. For example, a simple action "move left" connects two game situations that have a difference in character's horizontal coordinate. There are no restrictions on incoming and outgoing connections: (a) one action may lead to several new game states (e.g., due to random factors involved in a game, the same action may yield different results); (b) different actions may lead to the same game state; and (c) distinct actions may connect the same pair of game situations (if a character is blocked between two walls, both "move forward" and "move backward" actions yield the same result). Each edge also has an associated probability: while a certain game situation may have numerous outgoing actions, not all of them may be equally preferable.

The ready acting graph represents a complete knowledgebase of a computer-controlled character. Normally it is being constructed automatically during learning by observation phase. A human expert plays the game, and the computer system builds the acting graph on the fly according to the following procedure:

```
wait for the next user action (A)
S = (current game situation)
WHILE game is not finished
   wait for the next user action (A')
   S' = (current game situation)
   find graph nodes for S and S'
   (if a node does not exist, create it)
   establish a link between S and S', and label it with A
   (if this link exists already, increase action probability)
   A = A'; S = S'
END LOOP
```

Let us now consider how the selected data structure helps to achieve the stated goals. The acting graph stores all behavioral patterns, demonstrated by human players. Unlike many knowledge representation mechanisms, such as neural networks, it does not eliminate the noise: even if a certain sequence of actions occurred only once during the training session, it will be still preserved in the graph. Thus an AI agent acquires all idiosyncratic elements of its trainer's style. By asking different human experts to train individual game characters, we obtain separate AI agents with different styles of acting [10].

Another significant advantage of acting graph is the possibility of manual modification. Acting graph can be visualized (we do it with AT&T's GraphViz tool [16]) and edited by the game designer. It is possible to remove unwanted or unintentional sections, to create artificial acting sequences, and to join separate graphs into a single knowledgebase.

Acting graph also lets the AI system to analyze the consequences of applied actions. The game designers might want to increase AI agent's skill level by means of automatic reward-and-punishment schemes (the use of reinforcement learning in behavior capture-based AI is discussed in [11]) or with the help of a heuristic action evaluation function. Such a function can traverse a graph, discover that a certain action is always weak (e.g., it always leads to game states with lower health level of the character), and discard it.

In general, clear and understandable structure of acting graph leaves enough room for new experiments. For example, in one of our research projects we tried to improve adaptivity of AI agents as follows. The agent is programmed to constantly learn new acting sequences from its current opponent. Each action is marked with a timestamp (when it was learned by the system). After certain time interval, old actions are removed from the graph. With this technique, we were able to obtain highly adaptive behavior: an agent tries to learn its opponent's tactics, and quickly changes behavioral patterns when the opponent decides to try another style.

2.2 Decision Making System

While automatic building of a knowledgebase is a rather straightforward process, the use of agent's knowledge for decision making involves more complicated techniques. In order to follow human player's style of behavior, the AI system has to perform case-based reasoning: it needs to identify a node in the acting graph that matches the current game situation, and to apply one of the actions, found in outgoing edges. The complications are caused by heuristic nature of matching algorithm: perfect matches are rare, so the system needs to be able to relax matching conditions gradually until an approximate match is found.

Our system allows the game designer to specify the sequence of search operations and their types, used to find an approximate match. There are two basic options: exact search with attribute exclusion (*static generalization*) and search with attribute variations (*dynamic generalization*).

Exact search finds a node that perfectly matches the given game situation. Since game situations are coded with numbers, this is done in O(log *n*) time for a graph, stored as a binary search tree. Attribute exclusions add more flexibility: the game designer can specify game situation attributes that are not taken into account while matching. So if the exact match is not found, we can repeat the search with relaxed conditions. In order to implement this feature, we require the game designer to define all searchable combinations of attributes in compile time. During learning by observation, the system builds additional acting graphs with reduced nodes, and stores them in separate binary search trees (see Fig. 3).

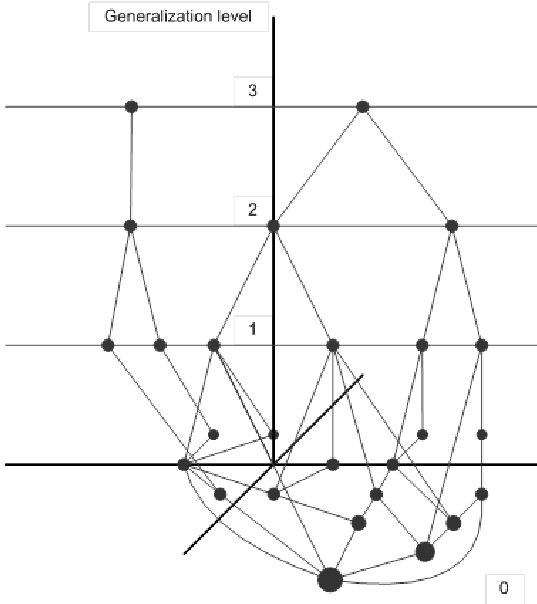

Fig. 3. Static generalization levels

Dynamic generalization is a wrapper around basic search routine. It allows the designer to specify an admissible matching range for each attribute instead of its exact value. For example, if the current game situation is represented with a tuple of three attributes (a_0, b_0, c_0), the use of dynamic generalization on two first attributes with a range [-1…1] will match the following nine tuples:

$(a_0 - 1, b_0 - 1, c_0)$ $(a_0, b_0 - 1, c_0)$ $(a_0 + 1, b_0 - 1, c_0)$
$(a_0 - 1, b_0, c_0)$ (a_0, b_0, c_0) $(a_0 + 1, b_0, c_0)$
$(a_0 - 1, b_0 + 1, c_0)$ $(a_0, b_0 + 1, c_0)$ $(a_0 + 1, b_0 + 1, c_0)$

This technique is useful when a certain attribute is important and thus cannot be excluded, but its exact value may slightly vary (as in case of game characters' coordinates). Currently, dynamic generalizations indeed rely on multiple calls to the basic search routine, thus potentially leading to combinatorial explosion of searches. However, in our practical experiments we were able to obtain satisfactory results with minimal use of dynamic generalizations. As a future work, we plan to implement dynamic generalizations with *k*d-trees, which should result in much lower $O(n^{1-1/k} + m)$ time for each range search, where m is the number of reported points, and k is the dimension of *k*d-tree [17].

The resulting set of actions, associated with the matching graph nodes, can be further re-ranked or filtered by additional heuristic functions. We use many such functions, both universal and game-dependent. The most important universal ranking function extracts the actions that continue the currently executed acting chain (i.e. the actions outgoing from the target graph node of the last used action). As a rule, such actions should be preferred by the AI. Also, we use weighted random choice in order to take into account action probability, stored in the graph.

3 3D Boxing: An Example Architecture

Our experiments with behavior capture-based AI for a 3D boxing game are described in the papers [10] and [11]. Here we will only discuss basic knowledge configuration for the 3D boxing AI, in order to provide a practical example of a graph-based decision-making system.

3.1 Game State Attributes

Original game states of the boxing game[2] are represented with a set of more than 60 numeric and Boolean attributes for each of the competing players. The most important attributes include:

- the identifier of a boxer's current animation sequence (this attribute describes an actual pose of a boxer);
- distance between the opponents;
- is-player-close-to-knockout-state Boolean flag;
- is-player-on-ropes Boolean flag;
- the direction to nearest ropes (boxing ring edge);
- health and energy values of a player;
- the identifier of a current boxer's animation sequence on the previous frame.

Each action is characterized with the following elements:

- action identifier (a type of an action) — one of 50 built-in action types, such as "move left", "move right", "right jab" or "right-hand high block";
- action duration (in frames).

[2] We used a full-fledged commercial boxing game engine.

Not all of game state attributes were considered important, so we have selected a set of 28 most valuable attributes to be stored in the knowledgebase. Additionally, we have performed necessary discretization to ease further retrieval. For example, "distance between the opponents" is measured in pixels, and thus can have hundreds of distinct values. We have scaled this attribute into a range of seven values only ("very far", "far", "not far", "medium", "almost close", "close", "very close"). The same operation was performed with other continuous attributes, such as boxer's health and energy levels.

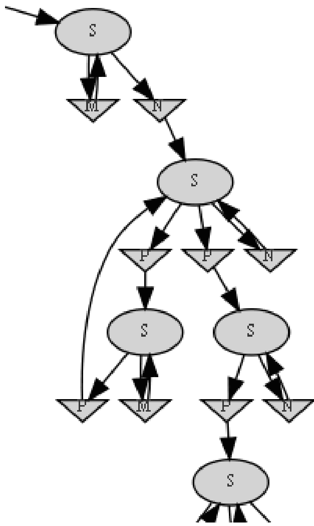

Fig. 4. Acting graph of 3D boxing game (actual fragment of level 2 graph, visualized with GraphViz)

3.2 Generalizations

For the system of static generalizations, we have selected six different sets of attributes. The most accurate set contains all 28 values, while the least accurate set is represented with 9 attributes only (see Table 1 and Fig. 4). So the AI system can find a match for the current game situation on any of these six levels of abstraction.

For the system of dynamic generalizations, the following attributes were chosen:

- distance between the opponents;
- identifier of a boxer's current animation sequence (it can be generalized to possible "neighboring" sequences — e.g., a boxer can be in lean state, then in stand state, or in stand state, then in make-punch state, but it cannot move to the make-punch state directly from lean state);
- (same as above) animation identifier, belonging to the opponent.

Table 1. Configuration of abstraction levels

Level	Attributes
0	28
1	22
2	17

Level	Attributes
3	15
4	12
5	9

3.3 Decision Making

As mentioned above, the game designer can specify any sequence of calls to graph search function in order to achieve desired AI performance. In general, actions found with fewer generalizations, and actions that continue the current acting chain are more preferable.

In our case, the system uses at most 22 invocations of the graph search function. Each invocation is parameterized with: (a) level (numerical identifier) of chosen static generalization; (b) Boolean flag indicating whether dynamic generalizations are used; (c) Boolean flag indicating whether the system should extract actions of the current acting chain only (see Table 2).

These 22 parameterizations roughly correspond to different "confidence levels" of case-based reasoning decision maker. The system searches for suitable actions, sequentially relaxing searching conditions according to confidence levels. The first acceptable action is returned as a result.

The 20th confidence level is reserved for a special heuristics: if no actions were found on levels 1-19, the system generates "do nothing" action. The rationale for this decision is simple: if no highly confident actions are available, it might be better just to do nothing and to give the agent the second chance to find a better action on the next request than to proceed directly to less confident "safety levels" 21 and 22.

To make AI less predictable, we also experimented with a slightly modified version of this algorithm. In this version, when the action selection subsystem finds an applicable action, it first extracts all other applicable actions at the current confidence level, and then returns a random action from this actions list.

Table 2. Confidence levels[3]

L	S	D	C
1	1	off	True
2	1	on	True
3	2	off	True
4	2	on	True
5	3	off	True
6	3	on	True
7	0	off	false
8	1	off	false
9	1	on	false
10	2	off	false
11	2	on	false

L	S	D	C
12	3	off	false
13	3	on	false
14	4	off	true
15	4	on	true
16	4	off	false
17	4	on	false
18	5	off	true
19	5	on	true
20	WAIT		
21	5	off	false
22	5	on	false

3.4 Heuristic Filters

As noted earlier, before an action is considered acceptable, it is analyzed with a set of ranking/filtering functions. In our system, we used only four such filters:

[3] **L** = confidence level, **S** = static generalization's abstraction level, **D** = dynamic generalizations, **C** = "extract chain actions only" flag.

- "Stumble on ropes". This filter analyzes backward move actions, leading to stumble-on-ropes state (normally they are considered weak), and marks an action as acceptable only if the original move action in the knowledgebase resulted in a similar stumble-on-ropes state in the human-played game (i.e. it really was a human player's intention).
- "Stumble on opponent". Analogously, stumbling on opponent (cinch) is usually a disadvantaged situation, and should not be encouraged. Actions, leading to clinch, are allowed only if the human player tried to initiate clinch in the original learning session.
- "Repeating actions". An action is ranked as weak, if it matches one of the last N (in our experiments, N = 8) used actions. This filter makes boxer's behavior less predictable and less repetitive. Note though, that "same action" means "same action object in the knowledgebase". The boxer can make two identical actions in a row, but they should correspond to distinct objects in the acting graph.
- "Defer non-punches". Punch actions are considered stronger than non-punches. This filter marks all non-punch actions as weak, so punch actions will always be preferred to alternative actions at the same confidence level.

4 Conclusion

The feasibility of our approach has been evaluated and proven in a series of experiments, involving the games of 3D boxing and soccer. We obtained believable and effective characters, able to exhibit human-like behavior style (almost indistinguishable from human actions) and to beat human-controlled opponents.

Our method does not implement reasoning capabilities and long-term planning, so its applicability to virtual worlds that demand these features is still an open question. We believe that our system can be used, at least, as a tactical AI decision maker, while high-level strategic reasoning can be supplied by another AI solution.

The representation of AI agent's knowledge as a game graph provides us with two major advantages: the agent keeps track of all behavioral patterns of its human trainer, and the obtained knowledge is easy to visualize and edit. While the latter point might not seem major from the theoretical point of view, it is an important factor for game developers, who are responsible for AI quality and prefer to have more control over system configuration.

In addition, our case-based reasoning algorithm is fast. We keep a minimal set of expensive operations and achieve our goals with fast search routines. Since game AI systems have to work in realtime conditions, speed and robustness of decision making algorithms are usually among key requirements, set by the game designers.

References

1. VBS Worlds: Cultural Awareness Training Simulation, `http://www.vbsworlds.com/?page_id=72`
2. Hughes, S.: Real Lessons from Virtual Battle. BBC News (August 29, 2008)

3. Taatgen, N.A., van Opplo, M., Braaksma, J., Niemantsverdriet, J.: How to Construct a Believable Opponent using Cognitive Modeling in the Game of Set. In: 5th International Conference on Cognitive Modeling, pp. 201–206 (2003)
4. Choi, D., Konik, T., Nejati, N., Park, C., Langley, P.: A Believable Agent for First-Person Perspective Games. In: 3rd Artificial Intelligence and Interactive Digital Entertainment International Conference (2007)
5. Glende, A.: Agent Design to Pass Computer Games. In: 42nd Annual ACM Southeast Regional Conference, pp. 414–415 (2004)
6. Hingston, P.: A Turing Test for Computer Game Bots. IEEE Transactions on Computational Intelligence and AI in Games 1(3), 169–186 (2009)
7. Funge, J.: Cognitive Modeling for Games and Animation. Communications of the ACM 43(7), 40–48 (2000)
8. Schrum, J., Karpov, I.V., Miikkulainen, R.: UT^2: Human-like Behavior via Neuroevolution of Combat Behaviorand Replay of Human Traces. The University of Texas at Austin (2011)
9. Le Hy, R., Arrigoni, A., Bessiere, P., Lebeltel, O.: Teaching Bayesian Behaviours to Video Game Characters. Robotics and Autonomous Systems 47, 177–185 (2004)
10. Mozgovoy, M., Umarov, I.: Building a Believable Agent for a 3D Boxing Simulation Game. In: 2nd International Conference on Computer Research and Development, pp. 46–50 (2010)
11. Mozgovoy, M., Umarov, I.: Building a Believable and Effective Agent for a 3D Boxing Simulation Game. In: 3rd IEEE International Conference on Computer Science and Information Technology, vol. 3, pp. 14–18 (2010)
12. Ontanon, S., Mishra, K., Sugandh, N., Ram, A.: Case-Based Planning and Execution for Real-Time Strategy Games. In: Weber, R.O., Richter, M.M. (eds.) ICCBR 2007. LNCS (LNAI), vol. 4626, pp. 164–178. Springer, Heidelberg (2007)
13. Mozgovoy, M., Umarov, I.: Believable Team Behavior: Towards Behavior Capture AI for the Game of Soccer. In: 8th International Conference on Complex Systems, pp. 1554–1564 (2011)
14. TruSoft Int'l, Inc., `http://www.trusoft.com`
15. Mozgovoy, M., Umarov, I.: Behavior Capture: Building Believable and Effective AI Agents for Video Games. International Journal of Arts and Sciences (to appear, 2011)
16. AT&T GraphViz, `http://www.graphviz.org`
17. Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms, ch. 10. MIT Press and McGraw-Hill (2001)