Building a Believable and Effective Agent for a 3D Boxing Simulation Game

Maxim Mozgovoy
University of Aizu
Aizu-Wakamatsu, Japan
mozgovoy@u-aizu.ac.jp

Iskander Umarov
TruSoft Int'l Inc.
St. Petersburg, Florida, USA
umarov@trusoft.com

***Abstract*—This paper describes an approach used to build and optimize a practical AI solution for a 3D boxing simulation game. The two main features of the designed AI agent are believability (human-likeness of agent's behavior) and effectiveness (agent's capability to reach own goals). We show how learning by observation and case-based reasoning techniques are used to create believable behavior. Then we employ reinforcement learning to optimize agent's behavior, turning the agent into a strong opponent, acting in a commercial-level game environment. The used knowledge representation scheme supports high maintainability, important for game developers.**

Keywords—believability; behavior capture; learning by observation; reinforcement learning.

1. Introduction

The quality of a virtual agent is usually associated with its effectiveness in reaching own goals. In these terms, an agent that plays chess at grandmaster level is better than an average-skilled AI player. However, in the domain of computer simulation and video games, the factor of *believability* [1] also turns out to be one of the key factors of a successful AI. According to [1], a believable agent "provides the illusion of life, and thus permits the audience's suspension of disbelief". Such an agent normally possesses certain human-like features: it can learn, make mistakes, and adjust own strategy to be effective against a particular opponent. Believability is recognized as an important factor both by researchers and by game developers [2; 3]. A number of recent research projects are devoted to the creation of believable agents for video game environments [4-7].

In our recent work [8], we have demonstrated a possible approach to building of a believable agent for a 3D boxing simulation game using a combination of learning by observation and case-based reasoning. A computer system first learns from a human expert, who demonstrates desired behavior by actually playing the game, then acts according to the formed knowledgebase. This technique (referred to below as *behavior capture*) has a number of attractive features [8]:

- Once the system is set up, no programming is needed to create new boxers.
- By training (playing) different game styles, the game designer creates unique boxer personalities.
- By making weak moves deliberately, the designer can obtain AI's weaker skill levels.

The obvious disadvantage of pure behavior capture is high requirements to a human expert, who is supposed to train the agents. In order to obtain a skillful AI boxer, the human trainer should be a skillful player, too. This can be a problem if behavior capture is implemented as a user-end feature, i.e. when "train your own boxer" mode is a part of the game process. Furthermore, in theory, the human-trained boxer should exhibit roughly the same behavior as its trainer. In practice, AI boxer's skill level turns out to be lower, due to heuristic and imprecise nature of our case-based reasoning algorithms.

Skill level of an AI agent can be raised to some extent by selecting only the best human-supplied behavior fragments as a set of training samples. However, significant improvements in decision making are hard to obtain, since the agent gives no preference to individual learned patterns, and makes the same mistakes as its human trainer.

In the present work, we address this issue by introducing reinforcement learning scheme into behavior capture procedure. On the first stage, the agent learns human-supplied behavior patterns. On the second stage, the agent learns to select only the best patterns, corresponding to effective game strategies. The same algorithm can also work during the game, constantly adjusting to the current opponent's behavior. However, for the sake of simplicity of the experiments, we do not consider such a mode here.

2. Basic Decision Making Scheme

Since the details of our virtual boxer's decision-making scheme are provided in [8], here we will only give a short description of this procedure, crucial for understanding how reinforcement learning is implemented within behavior-capture mechanism.

A. Learning and Acting

A boxing game engine with behavior capture capabilities operates as follows.

In learning mode, a behavior-capture agent (referred to below as *BC Agent*) observes the actions of one of the participating boxers. A boxer can be controlled by either a human or another AI system. Every time the observed player makes an action (including "do nothing" case), BC Agent stores the executed action, paired with the representation of the current state of the game world, into its knowledgebase.

In acting mode, BC Agent uses its knowledgebase to retrieve the most suitable action for a given game world

state upon request from the game engine. Most individual actions last 1-60 frames, while the game runs at a constant speed of 60 frames per second.

We used TruSoft's Artificial Contender AI middleware (www.trusoft.com) to build a customized version of a behavior capture system for the 3D boxing game.

B. Knowledge Representation

Game world states and player actions are stored in plain data structures GameSituation and Action. GameSituation structure stores the values of more than 60 numeric and Boolean attributes for each of the competing players. Each Action is characterized with action type and duration parameters.

During learning phase, an agent stores incoming (GameSituation, Action) pairs into its knowledgebase. Additionally, the agent establishes a link between subsequent records, so the whole knowledgebase can be viewed as a directed graph that contains action chains. Each action chain corresponds to a certain individual game session. Action objects also contain a usage counter that is increased every time when the same (GameSituation, Action) pair arrives at the input. This counter is later used for a weighted action choice (more frequent actions are selected more often, when possible).

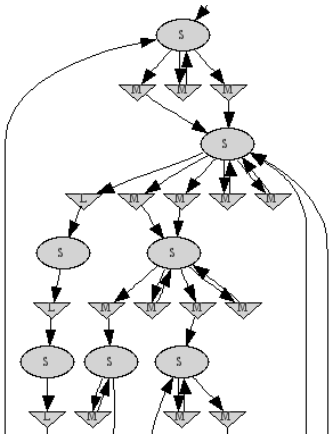

Figure 1. A fragment of a knowledgebase

Figure 1 illustrates a fragment of such an acting graph. Game situations are shown with ellipses, and actions are represented with triangles.

The action selection mechanism is considerably more complex than the learning subsystem, because it contains an additional heuristic search routine. Ideally, when a virtual agent receives the next GameSituation object from the game engine, it should extract the same GameSituation in the knowledgebase. However, in order to obtain reliable decision-making, we cannot assume that this search is always successful. Therefore, the agent should be able to extract the closest (if not perfect) match from the knowledgebase and act in accordance with the found (GameSituation, Action) pair. In our system, approximate matching is achieved through a series of knowledgebase polls with sequentially relaxed search conditions. Thus, BC Agent usually finds several suitable actions, and selects the most reliable one, found under the strictest search criteria. When several actions have the same degree of reliability, the agent selects one of them randomly (we use weighted random choice [9] with action usage counter as weight).

C. Rationale

Acting graph-based decision making scheme, applied in our project, is infrequently used as a basis for intelligent behavior. Normally, a more conventional instrument is chosen. For example, the projects [4] and [6] rely on Bayesian networks, and the agents described in [10] and [11] use standard neural networks.

From the perspective of game developers, acting graph has a number of advantages as a knowledgebase. Probably, the most obvious among them is the possibility to edit it manually. Game developers prefer to keep the control over the agent's behavior, and often consider "black boxes" inacceptable. If a game designer wants to adjust the agent's decisions, straightforward re-training is not the only choice. Instead, the designer can manually remove undesired action chains or combine two acting graphs into a single knowledgebase.

3. Introducing Reinforcement Learning

As noted in the section B, action selection subsystem first extracts a number of actions, having the same (hopefully high) degree of reliability, and then returns an action from this list, according to an action usage counter-based weighted random choice.

We have improved this algorithm by changing the scheme of action weights calculation as follows. After each decision, an aftermath of the applied action is evaluated, and its quality is calculated. This quality affects current action's weight and propagates in the acting graph, also affecting the weights of the previous actions in the chain. Action quality is evaluated on the basis of only one numeric value Q:

$$Q = \Delta HealthOwn - \Delta HealthOpponent$$

Here $\Delta HealthOwn$ is the difference in boxer's health level before and after the action (negative value means that the boxer was punched). The second parameter, $\Delta HealthOpponent$, denotes the difference in opponent's health. The positive value of Q means that BC Agent-controlled boxer got less damage than his opponent as a result of the applied action.

The weight of each action is defined as a sum of direct and indirect rewards (the values of these rewards are stored in the Action object):

$$W = R_d + R_i$$

Initially, all direct and indirect rewards are set to zero. The last applied action gets the following direct reward:

$$R_d = R_{dp} * Cnt / C_{max} + Q * C_R * (1 - Cnt / C_{max})$$

The basic idea is to make frequently used actions more stable, and less affected by casual action quality jumps. In order to increase boxer's adaptivity, the system periodically

examines the acting graph and reduced usage counters of actions, not chosen for a long time.

Parameter	Comment
R_{dp}	Previous direct reward of the action.
Cnt	Usage counter of the action.
C_{max} *[=127]* [1]	The largest possible counter value.
C_R *[=4]*	Actual contribution of action quality (Q). Higher values of C_R make BC Agent less risky.

After calculating the direct reward, we process previous actions in the chain, leading to the last applied action. The indirect rewards of these actions are calculated using the following formula:

$$R_i = R_{df} * C_{IR} * (1 - (frame_{first} - frame_a) * V)$$

Since R_i is stored as an integer, the backpropagation ends when this value becomes close to zero.

Parameter	Comment
R_{df}	Direct reward of the first action in the chain (i.e. of the last applied action).
C_{IR} *[=0.9]*	The contribution of R_d into indirect reward (lower values lead to less "look ahead").
$frame_{first}$	The frame number of the first action in the chain.
$frame_a$	The frame number of the action, being currently processed ($frame_a < frame_{first}$).
V [=0.01]	The speed of reward decrease (lower values lead to less "look ahead")

The resulting action weight is clipped to a range [MinWeight, MaxWeight] (set to [-127, 127] in the current system).

Note that we backpropagate the rewards in the actual chain of actions, occurred in the ongoing game session, rather than in the acting graph knowledgebase. This simple solution turned out to be effective in our case.

4. Experiments

Basic behavior capture-based agent (without reinforcement learning) exhibits believable behavior, as shown in [8]. The believability was demonstrated using two different methods. First, a variation of the Turing test was employed, where people were asked to identify opponents as humans or AI agents in the video recording of the game session. Second, an automatic "behavioral profiles" comparison procedure was run to identify similar boxers (this idea follows the approach earlier suggested in [12]). The comparison was organized as follows. First, we recorded sequences of game actions, performed by different boxers. Next, for each action sequence we computed a vector of probabilities of each possible combination of two successive actions. A cosine of angle between a pair of such vectors was used as a degree of behavioral similarity of the corresponding boxers.

In this section, we will show that reinforcement learning increases effectiveness of BC Agent (i.e. makes it a stronger opponent), still preserving believability of observed behavior.

A. Improving Effectiveness of BC Agent

For this experiment, we organized a short series of three matches between the computer-controlled boxers. The players were operated by the boxing game engine's built-in finite state machine-based AI system[2]. The acting data of one of the boxers was used to train BC Agent. All three matches ended after 14-15 minutes of play with a knockout of one of the opponents. So the total duration of training was about 44 minutes.

As noted before, a pure behavior capture-based boxer is normally less skillful than its trainer. The subsequent game sessions between BC Agent and the built-in AI have shown that BC Agent is able to stand for 8 minutes (on average), before being knocked out.

After turning on reinforcement learning, BC Agent quickly improves its skills, and after 9 matches reaches a stable state, being able to knockout the built-in AI boxer in 14-16 minutes (see Figure 2; white columns denote BC Agent's victories, black columns represent defeats).

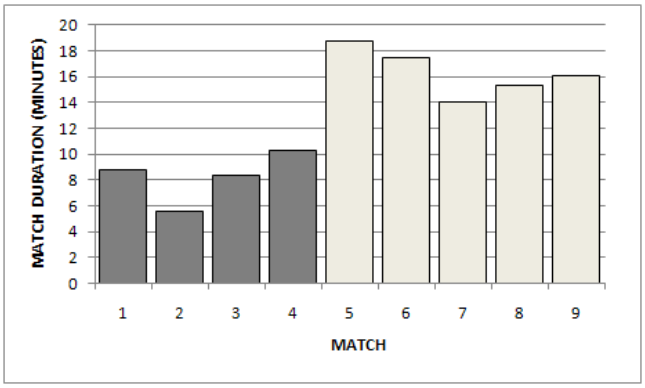

Figure 2. AI-trained BC Agent's progress

It should be emphasized that the agent's improvements are achieved solely by re-evaluating action weights. The agent still selects the actions, demonstrated by its teacher in similar cases, but it now prefers the decisions that are proven to be advantageous.

B. Optimizing Human-Trained BC Agent

To see how a human-trained BC Agent can benefit from reinforcement learning, we have trained an agent of an average strength, able to stand around 5 minutes against a built-in AI, before being knocked out.

[1] We will use square brackets to indicate actual values, used in the experiments.

[2] This AI system was actually used in a commercial boxing game, and it is rather challenging. At least, both authors were able to beat it only occasionally.

Then we have performed a series of matches between our agent and a built-in AI. After four matches, BC Agent won its first game. After ten matches, BC Agent turned into an effective boxer, able to play at the level of a built-in AI (see Figure 3).

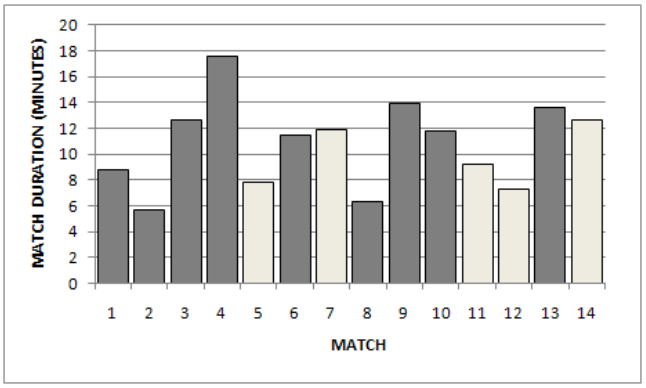

Figure 3. Human-trained BC Agent's progress

C. Testing Believability

Since reinforcement learning changes the behavior of BC Agent, it is not obvious whether the agent remains believable (i.e. human-like in case of human-trained BC Agent) or not.

To test believability, we applied the same behavior profiles comparison method, as described in [8]. We recorded several profiles, corresponding to:

- the built-in AI agent (ai1, ai2, ai3);
- the human player (hum);
- the human-trained BC Agent (bc1, bc2, bc3);
- the same BC Agent with reinforcement learning-modified knowledgebase (rl1, rl2, rl3).

Then we compared the profiles, and visualized the similarity matrix as a plot[3] (see Figure 4; high similarity ratios correspond to shorter distances between the profiles).

As seen in the picture, each group forms a clearly isolated cluster. While rl1 – rl3 profiles show less similarity with the original human trainer than bc1 – bc3 profiles, they still belong to the same family of BC agents. For example, distance between rl1 and bc2 is smaller than between bc3 and bc2. Note also that the built-in AI (ai1-ai3) is far away both from human player, and from human-trained BC agents.

5. Discussion

The introduction of reinforcement learning into behavior capture system raises a number of topics worth a separate discussion.

Perhaps, the most questionable point is our backpropagation scheme. It is based on very simple principles, and it is not designed to be generalizable and/or scalable. Our grading formula is designed specifically for such a fast and reactive game as boxing. Boxing requires neither long-term planning, nor complex strategic decisions. That's why a simple adaptive system, able to adjust rapidly to the opponent's actions, is suitable for our aims.

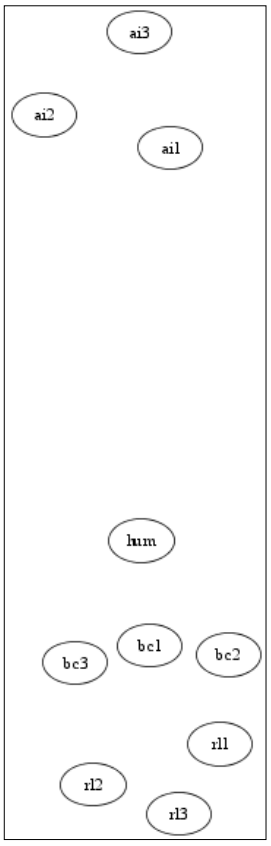

Figure 4. Comparison of behavior profiles

During the first experiment (optimizing the AI-trained BC Agent), we encountered the following problem. When BC Agent becomes too strong, our reasoning system faces difficulties with finding relevant actions in the acting graph. Since in the original training session both players were of the same skill level, certain types of game situations were rare. For example, it is unlikely that either of the opponents was cornered for a considerable time interval.

This problem seems to be fundamental for case-based reasoning systems. While the agent improves own skills, the decision making algorithm has to act in previously unseen game scenarios. As a result, it produces actions of less quality, having not enough relevant cases in the knowledgebase.

The best solution for this problem is to supply the agent with additional training data. However, excessive training has its own drawbacks. Longer training sessions produce more robust agents: a boxer with extensive training history rarely "gets lost", and more often finds actions, having better quality. However, comprehensive knowledgebase dilutes boxer's character, as the boxer receives more and more distinct acting options in the same game situation. Thus, its behavior becomes more "generic", and the difference between characters, trained by different players, becomes less evident.

In order to obtain a boxer that exhibits clear playing style and finds appropriate actions in overwhelming majority of game situations, we designed a two-layered knowledge application scheme: if the next action is not found in a smaller boxer-specific knowledgebase, we continue searching in the extensive generic knowledgebase, shared by all boxers. Thus, short training sessions are used to form distinguishable characters, able to act in most cases; if a boxer is unable to find an action, the subsequent "safety level" supports decision making. This scheme can be generalized into arbitrary number of layers. In addition, safety

[3] This is done by employing *neato* tool from AT&T GraphViz package.

level can be represented with a classic rule-based AI, if case-based reasoning system is still considered not enough reliable.

6. Conclusion

We have extended our earlier behavior capture AI system with optimization algorithm, based on reinforcement learning. While the previous version of the system allowed designing believable game characters, it generally required the trainers to be skillful players. Furthermore, it produced less effective boxers, unable to compete at the level of their trainers. Reinforcement learning adjusts the agent's behavior automatically by cutting off weak actions, thus raising the agent's skill level.

This auto-adjustment preserves believability of game characters. Adjusted agents still exhibit acting patterns, similar to the ones demonstrated by their trainers, as shows the comparison of the agents' behavior profiles.

References

[1] J. Bates, "The role of emotion in believable characters," *Communications of the ACM*, 1994, vol. 37, pp. 122-125.

[2] J. Orkin, "Three states and a plan: the AI of FEAR," *Game Developers Conference*, 2006.

[3] G. Yannakakis and J. Hallam, "Towards optimizing entertainment in computer games," *Applied Artificial Intelligence*, 2007, vol. 21, pp. 933-972.

[4] C. Thurau, T. Paczian and C. Bauckhage, "Is bayesian imitation learning the route to believable gamebots," *Proc. of GAME-ON North America*, 2005, pp. 3-9.

[5] D. Choi, T. Konik, N. Nejati, C. Park and P. Langley, "A believable agent for first-person shooter games," *Proc. of the 3rd Annual AI and Interactive Digital Entertainment Conference*, 2007, pp. 71-73.

[6] R. Le Hy, A. Arrigoni, P. Bessiére and O. Lebeltel, "Teaching bayesian behaviours to video game characters," *Robotics and Autonomous Systems*, 2004, vol. 47, pp. 177-185.

[7] S. Ontanón, K. Mishra, N. Sugandh and A. Ram, "Case-based planning and execution for real-time strategy games," *Lecture Notes in Computer Science*, 2007, vol. 4626, pp. 164-178.

[8] M. Mozgovoy and I. Umarov, "Building a Believable Agent for a 3D Boxing Simulation Game," *To appear in Proc. of the 2nd International Conference on Computer Research and Development*, 2010.

[9] F. Olken and D. Rotem, "Simple random sampling from relational databases," *Proceedings of the 12th International Conference on Very Large Data Bases*, 1986, pp. 160-169.

[10] R. Bonse, W. Kockelkorn, R. Smelik, P. Veelders and W. Moerman. Learning Agents in Quake III. *Technical Report, University of Utrecht*. 2004.

[11] S. Bonacina, P. Lanzi and D. Loiacono, "Evolving Dodging Behavior for OpenArena using Neuroevolution of Augmenting Topologies," *PPSN'08 Workshop (Computational Intelligence and Games)*, 2008.

[12] F. Tencé and C. Buche, "Automatable evaluation method oriented toward behaviour believability for video games," *International Conference on Intelligent Games ans Simulation*, 2008, pp. 39-43.