Building a Believable Agent for a 3D Boxing Simulation Game

Maxim Mozgovoy
University of Aizu
Aizu-Wakamatsu, Japan

Iskander Umarov
TruSoft Int'l Inc.
St. Petersburg, Florida, USA

Abstract—This paper describes an approach used to build a practical AI solution for a 3D boxing simulation game. The features of the designed AI agent are based on our deliberate concentration on believability, i.e. human-likeness of agent's behavior. We show how learning by observation and case-based reasoning techniques can be used to create an AI decision-making system for an industrial-level computer game. The chosen AI design principles support high usability and maintainability, which is important for game developers. We prove experimentally that our AI system provides both believable and effective behavior.

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

I. INTRODUCTION

Traditionally, AI methods are aimed at creation of efficient agents, ready to act in complex domains. The quality of an agent can be measured in such terms, as robustness, reliability, effectiveness of decision making (allowing an agent to reach own goals), and algorithmic complexity. However, in the domain of computer simulation and video games, the factor of *believability* [1] turns out to be one of the key factors of successful AI. Defined in [1] as the "one that provides the illusion of life, and thus permits the audience's suspension of disbelief", a believable agent is usually characterized with human-like characteristics, such as capabilities to learn, to "think" for a while between the decisions, to make mistakes, and to adjust own strategy in response to opponent's actions.

The importance of believability factor is emphasized both by researchers and by game developers [8; 12]. Recently, a number of research projects were devoted to the creation of believable agents for various environments, such as Quake 2 [10], Quake 3 [2], Unreal Tournament [5], and WARGUS real-time strategy game [7].

In this work, we demonstrate a possible approach to build a believable agent for a 3D boxing simulation game using a combination of learning by observation with 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 (we will call it *behavior capture*) has a number of attractive features: (1) once the core system is set up, no programming is needed to obtain desired behavior for a particular boxer; (2) by training (playing) different game styles, it is possible to create a wide variety of unique boxer personalities; (3) by making weak moves deliberately, game designers can obtain AI's weaker skill levels.

Furthermore, some of our technical decisions are inspired by wishes of game designers to have tunable and editable AI agents, whose behavior is possible to analyze and adjust according to designer's vision of appropriate game characters.

In order to make experimental environment realistic, we used Xbox 360 game console as hardware. Thus, our algorithms operate under typical game-console speed and memory constraints. As a testbed, we employed an existing boxing game engine.

II. RELATED RESEARCH AND OUR AIMS

Probably, the closest research aims (to build a believable and effective agent for a computer game through learning by observation) are set in works [2], [5], and [10]. All these projects design agents for first-person shooter games, and two of them use Bayesian networks as decision-making mechanisms. The third project relies on a ready-made cognitive architecture.

Our agents operate in a significantly different domain. While first-person shooters require strategic planning and spatial reasoning, boxing game demands human-like short-term awareness and acting. Boxing also needs rapid reaction, so the computational complexity of used algorithms should allow the agent to make 10-30 decisions per second. Furthermore, weak decisions in boxing are immediately utilized by the opponent, as normally boxers keep tight contact.

We also tried to shift from constructing "general" human-like behavior to reproducing idiosyncrasies of particular trainers, as different boxers should be identifiable personalities — not just at the level of preferring to use certain types of punches or being defensive / offensive, but at the level of specific action sequences.

The use of machine learning to build an agent for a fighting game Tao Feng was studied in [14]. The system uses neural network as a basis for decision making. While the obtained agents were able to beat a handcrafted AI, the questions of believability, maintainability and ease of use (for game designers) were not addressed.

III. TESTBED SETUP

The original boxing game engine was modified for separate learning and acting modes as follows.

In learning mode, a behavior-capture agent (let us call it **BC Agent**) observes the actions of one of the 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. Thus, during a typical 15 minute learning session an agent can record around 4000 sample (situation, action) pairs.

The implementations of AI core algorithms and data structures, as well as additional AI designer's tools, were provided by TruSoft's Artificial Contender AI middleware (www.trusoft.com).

IV. LEARNING

Game world states and player actions are described with 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 (in frames)

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

From the perspective of game developers, such acting graph has a number of advantages as a knowledgebase. The most obvious among them is the possibility for a game designer to edit it manually. Game developers are suspicious about "black boxes", and they appreciate the potential chance to remove certain action sequences if they are considered inappropriate or to combine parts of existing graphs into a new knowledgebase.

Since different GameSituation attributes are not equally important, we have selected a set of 28 most valuable attributes for storing in the knowledgebase. Additionally, we have performed discretization to ease further retrieval. For example, we have scaled "distance between the opponents" attribute into a range of seven values only ("very far", "far", "not far", "medium", "almost close", "close", "very close"). By storing a subset of discretized attributes, we preserve acceptable size of the knowledgebases. A typical knowledge file occupies 800-1200 KB of disk space.

Next, we have created six sets of attributes (we call them *static generalizations*), corresponding to representation of an initial GameSituation object with different degrees of precision. The most accurate set contains all 28 values, stored in the knowledgebase, while the least accurate set is represented with 9 attributes only. There are two reasons to have this construction. First, graphs of higher (less precise) abstraction levels are easier to analyze, which is important for game designers. Second, it is used by a decision-making subsystem to extract similar GameSituation objects (see subsection V.A).

Knowledge analysis is supported with TruSoft's AC Knowledge Processor and AC Knowledge Viewer. These tools visualize contents of a knowledgebase using AT&T's GraphViz instrument (www.graphviz.org).

Figure 1 depicts a fragment of the tool's actual output. Game situations are shown with ellipses, and actions are represented with triangles.

V. ACTING

The action selection subsystem is considerably more complex than the learning subsystem. The main difficulty is caused by the necessity of heuristic search. Ideally, when BC Agent receives the currently formed 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, BC Agent should be able to find the closest (if not perfect) match from the knowledgebase and act in accordance with a found (GameSituation, Action) pair.

A. Generalizing GameSituation Objects

Partially, this problem is addressed with aforementioned abstraction levels: we search GameSituation objects using minimal set of attributes and sequentially refine search, while matching graph nodes are still found.

However, in some cases abstraction levels are not enough due to border effects. Consider again "distance between the opponents" attribute. For example, when the distance is "very far", it can be actually on the border with "far", so it is worth analyzing all GameSituation objects with distance equals "far", too. To deal with border effects, we have implemented an optional capability to specify a continuous range for any numerical GameSituation attribute, instead of a single value. Consequently, knowledgebase-searching routine considers all GameSituation objects with a given attribute lying in a specified range as matching. We call this process *dynamic generalization*.

Static generalization of a certain attribute can be viewed as a dynamic generalization of the attribute into the whole range of possible attribute's values. Therefore, static generalization is a particular case of a dynamic generalization.

The advantage of a static generalization is its low cost: by ignoring a value of an attribute, we do not introduce any additional complexity into the search routine, so individual GameSituation matching with generalizations can be performed in $O(\log n)$ time with conventional binary search. Dynamic generalization can be implemented, e.g. with *k*d-trees [13], which means $O(n^{1-1/k} + m)$ time for each range search, where m is a number of reported points, and k is the dimension of *k*d-tree.

B. Extracting Action Chains

In addition to GameSituation attributes, there is another significant factor that should be taken into account when retrieving Action objects from the knowledgebase. The action to be selected should preferably form a sequence with previously applied actions. In other words, the agent should try to reproduce behavior patterns, demonstrated by a

human player, rather than act according to the current GameSituation only, regardless of previous acting history.

Since the knowledgebase is stored in form of a graph, this capability can be incorporated into the searching algorithm. It is enough to pass the previously applied action to the searching routine, and to set an additional constraint: the action to be extracted should be the next in the chain after the previously applied action.

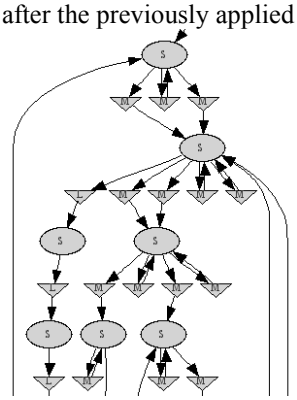

Figure 1. A fragment of a knowledgebase (level 5)

Though it was not necessary for the game of boxing, it could be useful to perform a more in-depth analysis of action sequences, stored in the graph. For example, action chain analysis can predict the outcome of a certain action, helping decision-making subsystem to select better actions.

The rationale for preferring this method to established sequence generation methods (such as n-gram [4]) is again inspired by our initial aim to create clear boxer personalities and to support game designers' requests. By storing all observed action chains (without converting them to more general acting rules) we preserve specific acting idiosyncrasies of a particular trainer. Also, any chain of actions in a game graph represents a target action sequence for a decision-making system, so by analyzing a game graph, a designer can see which specific acting patterns the system tries to follow.

C. Action Selection Procedure

Summing up, the action extraction routine is parameterized with the following elements: a list of actually used GameSituation attributes (static generalization); a list of attributes to be extended with neighboring values (dynamic generalization); a "prefer chain actions" Boolean flag.

We have selected 22 different parameter sets, roughly corresponding to different "confidence levels" of decision making. On the first (most precise) level, no generalizations are used, and "extract chain actions only" flag is set to true. On the 22nd (least precise) level, all generalizations are turned on, and chain factor is not taken into account by the searching routine.

The action selection subsystem searches for suitable actions, sequentially relaxing searching conditions according to confidence levels. When the first acceptable action is found, the system extracts all other applicable actions at the current confidence level, and then returns an action, selected from this list according to a weighted random choice (action counter is used as a weight).

D. Action Matching

Technically, above described means already allow to obtain a believable behavior of a boxer. However, as our experiments showed, the straightforward "extract and apply" approach does not provide effective enough decision making, as boxer often selects weak actions, immediately leading to disadvantaging game situations. In addition to the above described "situation matching" subsystem we have introduced a module for subsequent "action matching" as a system of action filters.

Action filters noticeably increase decision making performance at the cost of giving simple hints on game's basic principles to the boxer. A single action filter analyzes all actions, extracted by action selection subsystem and rejects an action (marks as unacceptable) if it does not satisfy certain criteria. Action filter can also mark it as weak, reducing its priority within a list of extracted actions at the given confidence level. The following description gives examples of typical filters:

"Stumble on ropes". Generally, moving backwards to nearby located ring ropes is a weak action, as it significantly limits subsequent boxer's movements. This filter analyzes backward move actions, leading to stumble-on-ropes state, 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". Similarly, stumbling on opponent (resulting in a clinch state) should not be encouraged. Such actions are allowed only if the human player tried to initiate clinch in the original learning session.

VI. Tuning the BC Agent

A number of experiments with different configurations of confidence levels and action filtering algorithms were performed before we reached BC Agent design, described in the previous sections. The most common issues, discovered during testing, were: (a) agent is unable to find applicable actions or frequently extracts actions at lower levels of confidence; (b) agent finds too many actions on the same confidence level; (c) agent's behavior is weak (i.e. BC Agent is easy to beat); (d) agent frequently fails to continue the action chain.

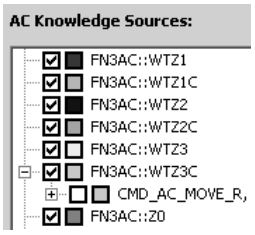

Figure 2. Confidence levels in Game Viewer

We used TruSoft Game Viewer to inspect game recordings and the corresponding knowledgebase files. This tool allows to load a game session and a knowledgebase, and then to examine the actual operation of the decision making system. Figure 2 illustrates an example of a satisfactory case with the action CMD_AC_MOVE_R found at confidence level WTZ3.

If all confidence levels are empty (issue (a)), we should relax action searching conditions. If most levels contain several actions (issue (b)), the conditions have to be strengthened. For each extracted action, Game Viewer highlights the corresponding Action node in the knowledgebase graph, so a game designer can examine game situations, considered as similar to the current one, and fine-tune generalization subsystems if necessary.

Weak behavior (issue (c)) is usually fixed by introducing new action filters. Game Viewer can also show the result of application of each filter to any extracted action, helping the designer to test filter's quality.

The designer can encourage the continuation of action chains (issue (d)) by turning on "extract chain actions" flag for several most precise confidence levels.

VII. Experimental Evaluation

A. Evaluating Effectiveness

To evaluate the effectiveness of AI's decision making (in terms of skill level), we have performed a series of matches between game's built-in industrial-level regular AI (based on a handcrafted finite-state machine) and three human-controlled opponents of different skill levels (let us call them "Weak boxer", "Average boxer", and "Best boxer"). Then the observed human behavior patterns were used to train the corresponding Weak, Average, and Best BC Agents. We also checked the performance of "Null boxer", who never makes any actions.

The experiment was organized as follows. First, each of three human-controlled boxers played a training session (10-20 min) against a built-in regular AI agent, operating in the highest skill mode. Every match was not limited in time and ended with a knockout of either of opponents. Then every trained BC Agent played a series of ten matches against the same built-in regular AI agent. The boxers' results are summarized below.

Null boxer. A built-in regular AI boxer steadily knocks out Null boxer in 50-100 seconds of game time.

Weak boxer. During the training session, built-in AI knocked out Weak boxer in 180-220 seconds of play. All ten matches between Weak BC Agent and built-in AI ended with BC Agent's defeat after 150-200 seconds of play.

Average boxer. This stronger human opponent was able to stand for 713 seconds (first match) and 805 seconds (second match) against a built-in AI. The corresponding BC Agent showed noticeably worse performance, never being able to stand even for 600 seconds. However, its behavior is clearly more effective than Weak boxer.

Best boxer. The human-controlled boxer knocks out the opponent in 85-150 seconds of play. Its BC Agent successor was able to beat built-in AI in seven games out of ten. The shortest match ended in 47 seconds, while the longest took 355 seconds. It worth noting that Best boxer often uses *haymakers* — very powerful slow punches that are hard to perform. If a haymaker is blocked, an opponent can make a fast and effective counter-attack. That's why haymakers make the game less predictable: rapid knockout is a matter of chance, while a series of successful counter-attacks can cause attacker's defeat.

B. Evaluating Believability

The most popular approach to evaluate believability of AI consists in application of the Turing test [11] to the game world. A person is asked to watch video records of matches and guess who the opponents are (people or AI agents). In another test variation, a human plays a match against an anonymous opponent over a network, and tries to guess whether the opponent was a human or a computer [3; 6]. A truly believable agent should be hard to unmask. The work [9] illustrates an automated approach to believability evaluation. A numeric vector-based fingerprint is calculated for each player's action sequence. Then the fingerprints of humans and AI agents are compared by measuring distances between obtained vectors.

We applied both Turing test-based and automated evaluation methods in our experiments.

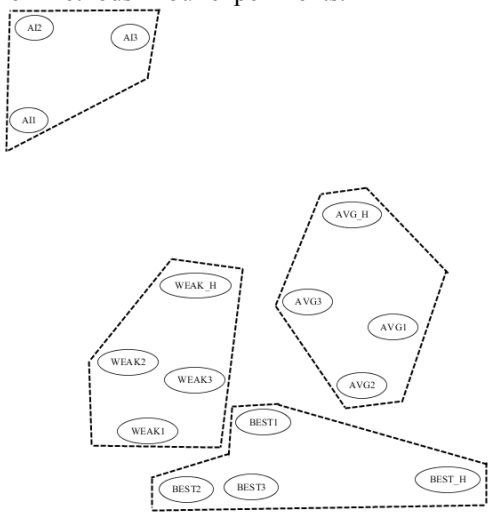

Figure 3. Behavioral similarity of different boxers

To evaluate believability of our agents, we compared behavioral patterns of the following boxers: built-in regular AI; human-controlled Weak boxer; Weak BC Agent; human-controlled Average boxer; Average BC Agent; human-controlled Best boxer; Best BC Agent.

The experiment was organized as follows. We recorded sequences of game actions, performed by each boxer (one sequence for each human-controller boxer and three sequences for each computer-controlled player). Then, 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. The matrix of distances between the players was visualized as a plot using GraphViz *neato* tool. This instrument uses approximate heuristic algorithms to build a picture of weighted graph (in our plot, edges are turned off), so that the visible distance between the nodes is proportional to edge weights. The plot is shown in Figure 3 (The dashed lines, showing the regions of interest, are drawn manually).

Built-in regular AI's behavioral patterns (AI1-AI3) form a clearly isolated cluster, exhibiting style of play that differs from human-controlled boxers and BC Agents significantly.

Weak and Average BC Agents (WEAK1-WEAK3, and AVG1-AVG3) can be also easily identified as members of the corresponding groups. Their human-controlled trainers (WEAK_H and AVG_H) are clearly distinguishable from BC Agents, but still, WEAK_H is close to Weak group, and AVG_H is close to Average group.

While the Best BC Agents form their own cluster, too, this cluster turned out to be close to Weak and Average groups. Furthermore, Best BC Agents' human-controlled trainer was considered as a unique character, hardly associated with any of the clusters. We believe, this happened mostly due to the fact that the playing style of Best BC Agents is indeed close to Weak and Average boxers. The main differences in styles are: (a) Best boxer blocks opponent's punches better; (b) Best boxer is trained to make successful haymakers.

We also tested believability of BC Agent using a technique, described in [3]. We recorded a number of one-minute game fragments between the random opponents. Each opponent could be a human player, a BC Agent or a built-in regular AI.

We asked six people with various gaming experience to watch the fragments and to guess who the opponents are (each boxer could be marked as "human" or "computer"). Each person was asked to analyze nine video clips.

The built-in AI was identified as computer in 85% of cases, while our BC Agent was considered human in 80% of cases.

C. Other Observations

Longer training sessions produce more robust agents: a boxer with extensive training history rarely "gets lost", and more often finds actions on higher confidence levels. 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 human players, becomes less evident.

VIII. Future Perspectives

While our experiments show the feasibility of learning by observation approach to design an AI agent for the game of 3D boxing, several issues need further research.

An optimal duration of a boxer's training session is not easy to determine. A perfect training session should produce a boxer, which (a) is robust (i.e. never "gets lost"); (b) has a clear character; (c) demonstrates sufficient skill level at fight. As we noted, these features can contradict each other.

We evaluated our agents in fights again a built-in "regular" AI agent. Being set to the highest skill level, it is truly challenging to most human players. However, it can be argued that our agents are trained to beat this AI, and their performance might be lower in fights with new, unknown opponents. However, for human players new opponents are more challenging than well-known boxers, too.

Some concerns might be raised by a sufficient number of game-specific algorithms and optimizations in our system. Indeed, the configurations of abstraction levels, action filters, confidence levels and dynamic generalizations are specifically designed for the particular 3D boxing game. Theoretically, some of this work can be automated. However, even in its current form the process of AI design presumably takes less time than manual construction of a rule-based or FSM-based system, implementing comparable features.

IX. Conclusion

We have designed and implemented a behavior capture system for the industrial-level computer game of 3D boxing simulation. Within the system, the characters of AI boxers are created by actually playing the game and, therefore, by demonstrating the desired behavioral patterns. This method helps to build rapidly a variety of player characters. While the system is based on known machine learning methods, it is deliberately tuned to preserve acting style of a player while training agents.

The experiments showed that the resulting agents are *believable*. A simple behavior comparison algorithm groups the agents, trained by the same teacher, into a clearly identifiable cluster. Turing test also demonstrated that in most cases BC Agents are identified as humans by our testers. Furthermore, we managed to train an *effective* agent, being able to beat a skillful built-in AI player.

References

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

[2] Choi, D., Konik, T., Nejati, N., Park, C. and Langley, P. 2007. A Believable Agent for First-Person Perspective Games. In *Proc. of the 3rd Artificial Intelligence and Interactive Digital Entertainment International Conference*.

[3] Gorman, B., Thurau, C., Bauckhage, C. and Humphrys, M. 2006. Believability testing and Bayesian imitation in interactive computer games. *Lecture Notes in Computer Science 4095*, 655.

[4] Damerau, F. 1971. Markov Models and Linguistic Theory. Mouton. The Hague, 196 p.

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

[6] Livingstone, D. 2006. Turing's test and believable AI in games. *Computers in Entertainment (CIE) 4*.

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

[8] Orkin, J. 2006. Three states and a plan: the AI of FEAR. In *Game Developers Conference'06*.

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

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

[11] Turing, A. 1950. Computing machinery and intelligence. *Mind 59*, 433-460.

[12] Yannakakis, G. and Hallam, J. 2007. Towards optimizing entertainment in computer games. *Applied Artificial Intelligence 21*, 933-971.

[13] Cormen, T., Leiserson C., Rivest, R., Stein C. *Introduction to Algorithms (3rd ed)*. MIT Press and McGraw-Hill, 2009.

[14] Graepel, T., Herbrich, R. and Gold, J. 2004. Learning to Fight. *Proc. of the Int'l Conf. on Computer Games: Artificial Intelligence, Design and Education*, 193-200.