Agent-based Players for a First-person Entertainment-based Real-time Artificial Environment

G. Michael Youngblood and Lawrence B. Holder

Department of Computer Science & Engineering
The University of Texas at Arlington
Arlington, Texas 76019-0015
{youngbld, holder}@cse.uta.edu

Abstract

The necessity for improved players and opponents in first-person entertainment-based real-time artificial environments has inspired our research into artificial game players. We employed the approach of creating agent-based players, one using simple reflex-action based components and another based on a back propagation neural network, to interact in a modified Quake II environment. Our evaluation of these agents utilizes two metrics, a statistical measure and graph similarity based on clustering from player performance traces against interactive feature points in the environments—both comparing the agents to a sample of 20 human players. The reflex-based agent was able to complete 29 of 100 test levels with 73.1% within statistical human-performance levels and 15.4% similar to the human players, while the BPNN-based agent was unable to complete any levels. These results, including some additional findings that reveal the potential for a combinational approach, are inspiring an architectural vision of a multi-agent system to provide an advanced artificial intelligence engine for the embodiment of advanced artificial characters in gaming and simulation domains.

Introduction

Computer-based interactive entertainment programs are growing in use and popularity. The interactive entertainment industry currently grosses $9.4 billion dollars a year (E3 2002). Despite its size as an industry, there are still many growth areas. In particular, the community of users has identified a few emerging needs: 1) the need for improved intelligence in opposing and supporting characters and 2) the desire for better single player experiences (Cass 2002). Current mechanisms for imitating intelligence involve the use of finite state machines, path planning, and other simple algorithms. Advanced character intelligence is becoming a much sought after feature and selling point for games (e.g., Black & White) (Cass 2002). There is an inherent need for advanced artificial intelligence (AI) engines in electronic entertainment.

Artificial intelligence engines, providing compellingly intelligent opponents and fertile grounds for the learning single player, can leverage advanced AI techniques and systems to play in these environments.

This paper will examine a reflex-agent based system and a back propagation neural network (BPNN) agent based system playing in a first-person entertainment-based real-time artificial environment—Id Software's Quake II game environment. Related work will be briefly presented and followed by a discussion of the design of the two systems under research. Section 4 will discuss the setup of the player environment and the evaluation metrics followed by a description of the testing of the two systems. The findings are then presented from this testing and conclusions are drawn. The paper ends with the future direction of this work.

Related Work

The Soarbot Project also uses the Quake 2 engine, as well as the Unreal Tournament and Descent 3 game engines, and couples it to the Soar inference engine for creating an Autonomous Intelligent Platform (AIP) of Soar agents that will compete against and cooperate with humans and other Soar agents in the game environment. The Soarbot project has been prominent in many recent publications (Laird 2000, 2001).

Gal A. Kaminka *et al.* (2002) have developed a virtual reality platform based on Unreal Tournament (UT). GameBots is composed of a socket-based API communication extension to UT, that allows agents to participate in the environment, and a development tool set that includes replacement graphics and starter sample code. GameBots replaces the gore of UT with male and female wizards that utilize a wide range of magical wands that shoot bubbles or green slime, but the goal is still to inflict this magic on other players, preferably of the other team. The project hosts tournaments for agents and posts the results that can be viewed on their website.

Other work in this area includes many of the developed game bots for first-person shooter (FPS) games. Neural networks (NN) are very widely used and research involving their use in robotic and classic games (e.g., robotic soccer, chess) are publicly available. The gaming AI community also has online tutorials specifically for using NNs in FPS games (Champandard 2003).

Multi-agent System Players

Interfacing with the Quake II engine proved to be a challenge due to intolerance of the game engine for time lags. A design decision was made to make the AI engine be completely separate from the game. Having enough processor power to not interfere with the demands of the game created a problem that was solved by distributing the system between two machines. Furthermore, there was a desire to make every component of a system be independent and swappable with different components (agents) that may provide improved and different capabilities. It is envisioned that these agents, each performing a specific task, would create a connectionist network. Thus, a multi-agent system (MAS) approach was taken.

CAMS

The Cognitive-based Agent Management System (CAMS) architecture was designed to meet the requirements of a multi-agent system (MAS) for creating a game-playing AI engine. Due to moving towards a cognitive basis for this work, the real-time requirements for interaction with Quake II, and the speed and object-oriented methodology of C++, there was a need for a custom MAS since no current systems offered a usable solution.

The CAMS infrastructure provides an agent framework for agent implementation that focuses on the management interface to an agent, an attention agent that acts as the local agent society supervisor, and a bootstrap agent that facilitates the movement (starting and restarting of agents) on different machines.

The CAMS attention agent has the ability to control the agents within its agent society, change their priority levels, control their communication links to other agents, and establish anytime processing constraints.

DCA Reflex-based Agent

The invention and development of CAMS set the stage for the creation of agents to populate a system for playing Quake II. Using the human player as a model we intend to employ work from human cognitive research to form an agent society into the D'Artagnan Cognitive Architecture (DCA, named after *Dumas'* hero in *The Three Musketeers*). The DCA consists of distinct areas of focus or models of cognition centered on a specific task that when connected form an architectural system for interacting in an environment. Each model is independent and should work on its own, but may require help from the other models in order to perform adequately or as designed (e.g., the memory model is just a storage model, but it may not be useful without a perceptual model to supply information to store or a learning model to form higher-level knowledge for storage).

The DCA is an architecture consisting of basic model types that may exist in multiple forms (with the exception of the attention model provided by CAMS) that comprise the nodes of a connected, directed graph.

In this phase of our research, we have focused on creating a base reflex agent utilizing the subsumption architecture (Brooks 1986). To facilitate a basic agent system, the basic nodes that exist in the current DCA are the action, attention (provided by CAMS), effector, memory, percept, and reflex. Figure 1 shows the configuration of the DCA reflex-based agent—the arrows in the figure represent lines of communication.

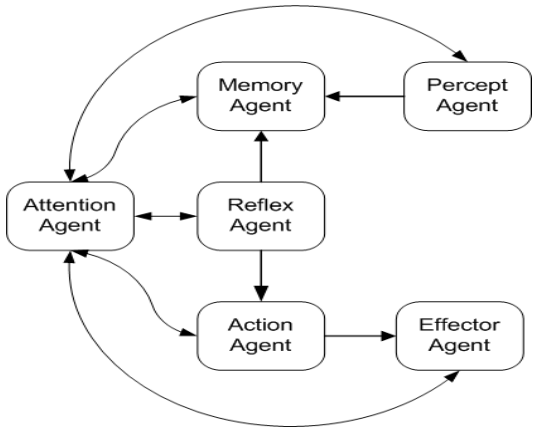

Figure 1. DCA Reflex-based Agent Configuration

The perceptual input node brings in all information from single or various sources into the architecture. Any preprocessing of information is done in the perceptual input (e.g., data conversions, data expansions, spatial representation). In the Quake II environment, percepts include all of the information coming from the game engine for the player (e.g., position, health, inventory, etc.).

The memory model node receives the perceptual information and stores it into short-term memory (STM) for access by all nodes. The memory model may then commit some information to long-term memory or perform other actions. We currently have only established the use of STM which currently acts only as percept data storage for agent access.

Upon the receipt of percepts and storage into memory for availability to all nodes, the functional nodes that watch and wait for new information in memory start processing. The reflex agent, which uses subsumption architecture, provides a reactive action to the action agent. The action agent determines the best action to take at any given moment based on the input. In our reflex-based model this will merely echo the reflex action.

The reflex agent's subsumption architecture contained several behavior layers. This agent did maintain a small amount of state information, so was not purely reactive, but generated some internal sensory information such as whether or not it had been stuck in the same position for a while. At the lowest level the agent would perform a random motion movement with a preference to move forward and rotate left on its axis. If the agent was close to a goal it was provided positional data and could then sense it as a percept, moving towards the goal. These goals included opposing forces, and when in range, the agent would fire its stun gun. If the agent was stuck, then it would perform a maneuver to become unstuck. At the highest level, if the agent was dead, it would not perform any action. Through these layers actions were generated.

The effectors node takes the ultimate action produced from the action model (reflex agent) and performs that action. In the Quake II environment, this is seen as movement, firing, or other actions allowed by the game engine.

Information comes into the DCA, is stored, processed, and resolved into an action, which is acted upon. This cycle repeats until the agent dies or is stopped either by game completion or operator intervention.

BPNN Agent

In contrast to the CAMS-DCA system, we also developed a CAMS-BPNN (back propagation neural network) agent as a comparison approach. This system utilized the same architecture as described in the previous section with the exception that the BPNN agent replaces the reflex agent.

The BPNN agent was implemented in two stages. The first stage was to develop a training program that would read in the training data set. The network would then be trained, setting the weights of the connections until convergence below the specified mean-square error. The second stage was an execution engine that could load the weights and run the network based on an input vector, producing an output vector. The second stage is wrapped in the CAMS-DCA shell to allow it to receive percepts which are then encoded in binary as an input vector to the network which would produce an output vector that is then deciphered into (a) corresponding action(s).

The project has collected twenty data sets of humans playing each of the designed scenarios. This information was converted into training data for the neural network. The core perceptual information needed to make a movement decision is the player's current x and y position in the scenario and their current heading or yaw, h, giving a 29-bit input for any given position in the environment.

The target output is the action taken, which is represented in a 17-bit classification scheme of which each bit represents a specific action. A rule-based program was developed to take the raw player data and generate this encoding scheme. All of the player data was then concatenated together into a single training set of 3639 examples.

The BPNN implementation specifically employs the use of a *logistic function* as the node transfer function between both hidden and output layers. For this work, the bias term is set to zero, and the momentum term is set to one. Due to well-documented reasons of the feasibility of the network to approach the limit of a one or zero value in application (Levine 2000; Rumelhart, Hinton, and Williams 1986), we consider anything over 0.9 to be 1 and anything less than 0.1 to be 0.

The network follows a standard BPNN algorithm during training, and then uses the calculations from input through the hidden to the output layer based on the trained weights to convert network input into network output (i.e., position information into action information).

The BPNN was trained on the training data set of 3639 examples (it should be noted that this represents actions in only 0.00044375% of the possible 144,000,000 positions, many duplicated), which were collected from human players as described in the next section. The number of hidden nodes was set to values between 1 and 30 and run to convergence. The effect of more nodes increases the rate of convergence but causes the network to be less general (Levine, 2000). In application, networks with more than 3 hidden units often could not generalize in the applied environment, so a 29-3-17 network was used for all further evaluations. Minimum mean-square error was set at 0.001 for all training runs. Evaluation of the network training was compared to the training data set to determine if the network learned the training data set. The training set was reused due to a lack of additional data and the desire to put as much known data into the training set.

Results indicate that the networks could learn the training set with a consistent 88% accuracy. The learning rate was varied from 0.1 to 0.3 in a 29-3-17 network. Contrary to literature (Levine 2000) indicating that oscillations usually occur with high learning rates, we did not observe this phenomenon. Regardless, we chose a 29-3-17 network with a learning rate of 0.2 and a minimum mean-square error of 0.001 for our agent network. The full test would come from the integration and interaction with the Quake II environment.

System Evaluation Methods

The two metrics we employ for evaluation become the basis for our claims on performance and consistency. The Time-Score metric taken over a trial set of evaluated human players will create a data set that will allow for statistical analysis. For each level we compute the maximum and minimum values of the combined Time-Score metric over the trial set of human players. If another player falls within that range for a specific level we shall say that they were within the range of human performance.

We define *human-similarity* through the utilization of our clustering metric to be the property of being clustered in the same group as another human player. This means that there exists a strong enough similarity in time and

traversal through a specific Quake II trial level that players are clustered together in the evaluation environment.

Evaluation Environment

A test set of one hundred Quake II levels was created starting with simple one-room environments and culminating in five levels that represent the typical difficulty of the original game. Each level was created to slowly introduce new concepts starting with very simple levels and gradually working up to the last five challenge levels. The goal of each level was to find a compact disc (CD). All players were told to maximize their health by not receiving damage and collecting health points, and to move through the levels as quickly as possible. Each level is independent, and the player state is reset upon entry in the next level. An example test level is shown in figure 2.

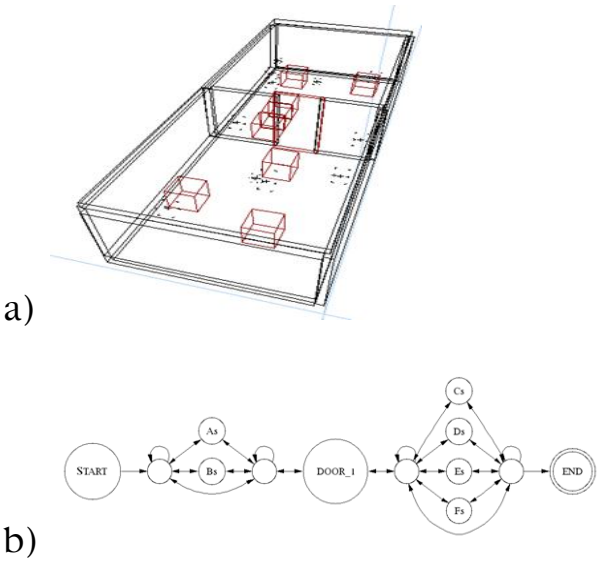

Figure 2. Scenario a) 3D view b) valid state transitions.

Human Trial Data

We found twenty people willing to participate in our Quake II human trials. A large amount of data was collected which consisted of the name, position in three-dimensional space, and state information for every active object in the Quake II environment updated at a rate of 10 Hz. We needed to determine a way to represent a player's interaction in the Quake II environment for each level in a way that made it easy to compare players to other players. Observing the test set of levels we created, we noted that we could break down each level into a set of interaction feature points based on attaining or doing certain things in each level (e.g., pushing a button, picking up a health item, and so forth). These interaction feature points were all of the objects in each level that a player could interact with and that were tracked as individual objects by the game engine. Using this we could generate a graph composed of each level's interaction feature points as nodes, and the ability to move from one interaction feature point to another would create the edges as was physically possible in each level. For each level, the interaction feature points were identified and a valid state transition diagram (e.g., figure 2.b) was generated for additional validations.

One problem with the graph representation is that there is a definite issue of timing in games such as Quake II, and often the real difference between performances in a level is the time it takes different players to accomplish the same task. Gonzalez (1999) and Knauf *et al.* (2001) also note the importance of time in validation of human models. We capture time by weighting the edges a player traverses between interaction feature points with the time it had taken the player to travel that. Now we can abstract a player's performance in a level, removing the noise of motion through the environment, and capturing their approach of interaction—moving from one interaction feature point to another in a level while also capturing the time aspect of their performance.

Data Evaluation Metrics

We developed two metrics for the evaluation of the data. The first metric produces a gross indicator of relative performance on each level based on the overall goals of maximizing the player's health and minimizing the time spent in each level. The second evaluation metric uses the graph data generated from each level for each player and generates a matrix of edit distances between all players to serve as the distance function for use in *k*-means clustering (Friedman and Kandel 1999). By clustering player performance, we hoped to see clusters of players grouped by their relative skill level. If we evaluate agent trial data with human trial data and the agents cluster into groups with human players, then we can assert that the agent played similar to that group of humans. If a player or agent constantly appears in the same cluster in a clustering over several levels, then we could classify the player or agent by its group associated performance.

We generated a matrix of edit distances between player graphs over all players for each level. *K*-means clustering was performed for each level. We seeded our initial cluster centers with random members of our data set, and we iterated our clusters over all possible initial seed values.

We utilized the normalized mean intra-cluster distance (Zhao and Karypis 2002) as the clustering quality measure and exercised k from 2 to (n-2), where n is the number of trials by humans and/or agents, over the data set to ensure we find the best clustering. The resultant clusters are used to group players by the similarity of their performance in the game.

Further details of this evaluation methodology can be found in Youngblood and Holder (2003).

Agent Trials

Trials were run on the CAMS-DCA reflex-based agent over the 100 test levels of the evaluation environment and

data collected in the same manner as the human trials. The CAMS-BPNN agent was first tried on a single scenario to test for fitness to compete in the rest of the levels. Only completed levels were subject to the metric evaluations described in section 4.

Performance

Understanding the data is key to understanding the results of clustering. We have taken data with a very high degree of dimensionality in the form of a graph and generated a distance metric between these graphs in this high-dimensional space.

Generating a data set for comparison when there are an infinite number of combinations of choices in each level is difficult. Our human trial data set suffers from its small size, but is sufficient for clustering other similarly behaving humans. However, there may exist human players that may not be classified with the current set of players. What we present is an objective form of evaluating performance against a particular set of humans and the means to test human-similarity within the boundaries of the known human models captured in the system.

The Time-Score metric is presented as a means for identifying if a new player falls within the band of human performance. The clustering metric is for determining human-similarity. Both metrics are based on a set of fixed procedures and the use of a trial human data set.

Using the presented Time-Score and Clustering metrics we have been able to evaluate the agent systems presented here.

DCA Reflex-based Agent

The CAMS-DCA reflex-based system was at its best able to complete 27 of the 100 test levels. The reflex-based agent completed 73.1% of its completed levels within human performance levels, and 15.4% were performed in a human-similar manner. In a side experiment we also introduced a concurrent random action generating agent that intermixed actions with the reflex agent for some additional interesting results. The combination reflex-random agent completed 69.0% of its completed levels within human performance levels, and 3.4% were performed in a human-consistent manner, but as a combined system was able to complete 29 test levels.

BPNN Agent

Interacting with the Quake II environment seemed to be a challenge for the CAMS-BPNN agent. There were no problems in receiving input or providing output, but the type of output the BPNN produced did not allow the agent to perform well in the environment. Using the base 29-3-17(0.2:0.001) network, the agent had a tendency to turn left. Concerned, we examined the training data to discover that the majority of moves were indeed left turns. In the absence of a known position to elicit a learned action, the network generalized the unknowns to left turns—a logical conclusion. To reduce this effect, we also modified the agent to perform a random action when it had an activation value of less than 0.90 over all outputs (since there was never a zero output case). This caused the agent to progress a little and the random action was usually only momentary. Under this progression it was observed that the network did actually generate actions when it came across a state that it had learned and was not generalizing—the random acts tended to move the agent out of local movement minima (i.e., stuck turning left forever). Unfortunately, this is just not enough to successfully play a Quake II level—especially, when there are elements of danger in the environment

Another issue arises in that the training data is an accumulation over 20 human players' actions and may contain inconsistencies or contradictions for a given state and the action to take. The majority should rule, but since the training set was extremely small in comparison to state space size, they most likely caused states with no clear action to take. We evaluated training on just the examples from a single player, but there was little difference—most likely a result of such a small training set in comparison to the size of the state space. What was interesting is that different individual players tended to have different common moves, so generalization on a single player basis yielded different general movements for the network.

In general, the agent did learn the training set to 88% accuracy, was integrated into the Quake II environment, generated actions based on input in real-time, and survived for a while in the test environment but was unable to complete any of the levels. Due to the inability of the agent to complete the scenario, the advanced evaluation techniques were not utilized.

Conclusions

Through the production of a Time-Score metric as an overall performance evaluation and a clustering metric for determination of human-similarity, we provided a means to evaluate players and agents in the Quake II environment. We first showed a general classification of system performance as compared to a defined set of human acceptable performance ranges and the ability to cluster a player's performance with those that played in a highly similar manner. This includes the evaluation of temporal performance as well as a spatial approach to sub-goal problem solving in those environments. The resulting clusters of players have a clear connection to performance in such environments and provide a means of classification based strictly on perfomance groups. These evaluations are conducted solely through numerical analysis in a defined process and do not contain any steps of subjective evaluation.

Evaluation of our CAMS-DCA system employing a reflex agent, and in parallel a combination random-reflex agent, showed that this system was at its best able to complete 29% of the levels. The reflex agent completed 73.1% of its completed levels within human performance

levels, and 15.4% were performed in a human-similar manner. The combination agent completed 69.0% of its completed levels within human performance levels, and 3.4% were performed in a human-similar manner. We observed an 11.5% increase in the ability to complete levels through the use of the combined agent over the best performance of the reflex agent alone. The ability of this system to improve performance through a combination of agents hints at a Gestalt factor to this approach.

A 29 input, 3 hidden, and 17 output node back propagation neural network with a learning rate of 0.2 and minimum mean-square error of 0.001 was developed and trained using the 3639 examples from 20 human players' actions. The CAMS-BPNN system did provide valid movement actions, but often required some additional help to produce output. The actions generated, however, did not seem to be goal progressing. The major problem seems to stem from a lack of training examples to sufficiently train the network given that the existing training set, at most, only covers a small fraction of the state space. Overall, the results are promising for the future use of neural networks in electronic entertainment artificial intelligence as the system was very fast and performed well with what knowledge it had learned. In the end, a neural network is only as good as its training data and this type of problem would appear to require a large amount of consistent data to be effective.

Utilizing a two-part evaluation metric and a multi-agent based simple reactive system yielded promising results and an evaluation mechanism toward generating better computer generated forces for electronic entertainment and simulation.

Future Work

Our future work involves completing an expanded full implementation and testing of the DCA; growing the maturity of CAMS through further study, development, testing, utilization, and research; working to optimize the clustering criterion function for our player clustering; streamlining and automating the trial and evaluation process; and improving the reference test level set. Some of the problems we observed are due to our small sample size—we need to bring in a much larger set of human trial data.

However, our future work is not utilizing the Quake II environment, but instead is using a popular community developed game called Urban Terror by Silicon Ice Development based on the Quake III game by Id. This new environment provides richer and more real-world scenarios with a terrorist/counter-terrorist theme that is currently a popular genre of games and may also tie into better simulations for investigating current security problems in today's society. With support from the Urban Terror online community we are also working to build a human-player database from player volunteers from around the world. We hope that this research will lead to better artificial players to enhance online and offline game play.

References

Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2:4-23.

Cass, S. 2002. Mind games. *IEEE Spectrum* (39:12). Piscataway, NJ: IEEE.

Champandard, Alex. J. 2003. Bot Navigation: Neural Networks for Obstacle Avoidance. Online at http://ai-depot.com/BotNavigation/Obstacle-Introduction.html.

E3 Expo. 2002. Electronic Entertainment Expo. Framingham, MA: E3 Expo.

Friedman, Menahem and Abraham Kandel. 1999. *Introduction to Pattern Recognition: Statistical, Structural, Neural, and Fuzzy Logic Approaches*. London: Imperial College Press.

Gonzalez, A. 1999. Validation of Human Behavioral Models. In Proceedings of the 12th International Florida AI Research Society, 489-493. Menlo Park, CA: AAAI Press.

Kaminka, G. A., M. M. Veloso, S. Schaffer, C. Sollitto, R. Adobbati, A. N. Marshall, A. Scholer, and S. Tejada. 2002. GameBots: A Flexible Test Bed for Multiagent Team Research. *Communications of the ACM* 45 (1): 43-45.

Knauf, Rainer, Ilka Philippow, Avelino Gonzalez, and Klaus Jantke. 2001. The Character of Human Behavioral Representation and Its Impact on the Validation Issue. In The Proceedings of the 14th International Florida AI Research Society, 635-639. Menlo Park, CA: AAAI Press.

Laird, John. 2002. Research in Human-Level AI Using Computer Games. *Communications of the ACM* 45 (1): 32-35.

Laird. J. E. 2001. Using a Computer Game to Develop Advanced AI. *Computer* 34(7):70-75.

Levine, D. S. 2000. *Introduction to Neural and Cognitive Modeling*. Mahwah, NJ: Lawrence Erlbaum Associates.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. 1986. Learning internal representations by error propagation. *Parallel Distributed Processing* (Vol. I, pp. 318-362). Cambridge, MA: MIT Press.

Youngblood, G. M. and Holder, L. B. 2003. Evaluating Human-Consistent Behavior in a Real-time First-person Entertainment-based Artificial Environment. In The Proceedings of the 16th International Florida AI Research Society, 32-36. Menlo Park: AAAI Press.

Zhao, Ying and George Karypis. 2002. Evaluation of Hierarchical Clustering Algorithms for Document Datasets. Technical Report 02-022. Minneapolis: University of Minnesota.