Learning Human Behavior from Observation for Gaming Applications

Christopher Moriarty and Avelino J. Gonzalez
Intelligent Systems Laboratory
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816-2450
USA
cmoriarty@isl.ucf.edu

Abstract

The gaming industry has reached a point where improving graphics has only a small effect on how much a player will enjoy a game. The focus has turned to adding more humanlike characteristics into computer game agents. Machine learning techniques are scarcely being used in games, although they do offer powerful means for creating humanlike behaviors in agents. The first person shooter (FPS), Quake 2, is an open source game that offers a multi-agent environment in which to create game agents (*bots*). The work described in this paper seeks to combine neural networks with a modeling paradigm known as context based reasoning (CxBR) to create a contextual game observation (CONGO) system that produces humanlike Quake 2 bots. A default level of intelligence is instilled into the bots through contextual scripts to prevent the bot from being trained to be completely useless. The results show that the humanness and entertainment value as compared to a traditional scripted bot have improved, although, CONGO bots usually ranked only slightly above a novice skill level. Overall, CONGO offers the gaming community a mode of game play that has promising entertainment value.

Introduction

Game play has become the more important factor in the design of a game, while impressive graphics are expected as the norm (Johnson, 2001). With playability becoming a more important factor, doors are opening to apply artificial intelligence (AI) techniques along with other playability enhancements.

The success of the *massively multi-player online* (MMO) genre shows that gamers do enjoy playing with other human players even if the other human players aren't in the same physical place. This could extend for players appreciating realism in non-player characters (NPC), which is any character in a game that is not controlled by a player. Another playability enhancement is extensible AI, which has been implanted in certain popular games to allow the player to customize the AI of their enemies or teammates. In popular *first-person shooter* (FPS) games such as Half-life and Unreal, users are allowed to use a scripting language to implement their own modifications into the game. The FPS genre is characterized by the first-person view in a three dimensional environment focused on a handheld weapon. While an FPS will have sufficient game play for a single player, many also have online multiplayer modes in which players can compete against other human players.

This research describes a system that allows a game player to create customized intelligent game agents in the Quake II environment. Such AI agents are more commonly known as "bots" among the gaming community. A player then starts the match and plays in a manner that they desire their bot to perform. "Learning from observation" techniques are used to capture the knowledge of the player. Connectionist and symbolic AI practices are combined to apply the captured knowledge into a fully-functional bot. The game environment outputs a large amount of sensor data, which can be difficult for a single machine learning algorithm, to use in raw form (Chapman, 1999). We created a contextual engine that uses knowledge engineering techniques to divide and manage the captured knowledge. Multiple neural networks are then trained for the separate contexts to make use of the contextualized environment data. This system is the contribution of this research and represents a novel approach to extensible AI in video games.

Game Engine as a Test-Bed for Research

Academic artificial intelligence research can benefit significantly from utilizing game environment tools to simulate synthetic agents. Laird has shown that using a publicly released game engine such as Quake II is indeed a practical solution for academic AI research (Laird, Assanie

and Bachelor 1999). Creating a test-bed can often take time away from the actual research being done.

The two most popular "good old fashioned AI" paradigms used in FPS agents are finite state machines (FSM) and rule-based systems (Thomas 2004). These two techniques have proven to be effective to completely control game agents such as bots in many games. However, there are certain negative characteristics in bots that make use of FSM and/or a rule-based architecture (Rabin 2004):

- Predictability becomes apparent
- Bot AI can become too perfect
- Non-human behavior is noticeable

These three issues are directly linked to the replay value of a game. In particular, when a bot has a rule that causes it to be vulnerable to a certain attack, a player will exploit this. Such exploitation can cause the bot to seem scripted or unnatural. An exploration of modern AI techniques will show that these problems can be solved.

Machine Learning for Gaming

The use of Neural Networks (NN) and Genetic Algorithms (GA) has shown their usefulness in some commercial games (Evans 2002), but are still scarcely used in the industry. There are a several reasons for this:

- Machine learning techniques can sometimes lead to unpredictable local maxima (Geisler 2002)
- Game Developers often stick with what they know (Nareyek 2004)
- Feature vectors are often too complex to control the agent with machine learning alone (Zanetti 2003)

One example of local maxima in a trained game agent could be when observation data shows that shooting only occurs 5% of the time, and in turn learns never to shoot (Geisler 2002). Machine learning can add a level of unpredictability into a game. This unpredictability is desirable for making the agent more human-like, although it is not desirable if it leads to unpleasant user experiences. As game environments become more complex, the number of features that can affect an agent in the game make it nearly impossible to be used in raw form. Therefore, pre-processing of data or other forms of minimizing the amount of noise and maximizing the amount of useful data becomes very important, but can be a formidable task (Geisler 2002) (Zanetti 2003).

Learning by Observation

The phrase "Learning by Observation" has its roots in biology. Studies have shown that humans fully develop observational learning by the age of 24 months (Abravanel 1998). By that age, children can easily learn a simple task by observing another person performing the task. Inspired by how humans and other mammals learn by observation, the machine learning community has developed a number of theories on learning by observation, applied in different areas.

Sidani created a ML system (IASKNOT) that learned how humans implicitly react at traffic signals by observing a human expert perform in a simulation (Sidani 1994). IASKNOT combined neural networks and symbolic knowledge to optimize the observation data being used to train the networks. This paper describes an extension of Sidani's work that improves on the techniques he used and applies them to a much more complex environment.

Determining when a machine learning algorithm completes its training is a heavily investigated problem. There have been many attempts to develop validation techniques and stopping of training criteria. In spite of this, there is always a chance that the network is poorly trained in new situations. In this work, a default level of knowledge was investigated and applied to the bot in the event that the bot's poor performance would hinder the human player's gaming experience. Accomplishing this required elicitation of domain knowledge from an expert player. However, this isn't the only topic that requires domain knowledge. Dividing the game into contexts and establishing the rules that govern the CxBR-based bot also required human intervention.

Problem Statement

The first problem addressed was: *To create a game AI technique that can be implemented with re-usable characteristics.* This leads to the follow-up problem being addressed: *How can an AI paradigm be created to ensure that it preserves or improves the playability of the game.*

More specifically, the problems addressed in this paper lie in creating a more human-like game agent. Attempts have been made to use NN's to create human-like bots in FPS games (Geisler 2002) (Chapman 1999). These attempts report that the NN was unable to realize all behaviors because of the complexity of the environment. To use NN's to realize a bot's behavior, the observed environment data must be less complex. Dividing the data into contexts would then simplify the necessary behaviors for any one NN to realize. This process must be done automatically in the background while a player is demonstrating the behaviors they would like to see their custom bot reflect. A reasoning paradigm known as context-based reasoning (CxBR) is the solution to the contextualization process, although knowledge engineering techniques must be employed to construct the contexts and transitions between them. This poses the next two problem statements:

- *Need to obtain knowledge from a game expert to establish all contexts that a player can be in, as well as the variables needed for the transitions to and from these contexts.*

- *Must choose and implement the appropriate neural networks to use for representing the observational data of the contexts.*

The first area of research that requires validation is the humanness of the bots. Humanness can often be a subjective matter to assess, therefore, this research will need to determine the means to validate the humanness factor of the created bots.

Another important factor to validate is the entertainment value of the system. This should not only be tested by volunteers, but should also be compared against other games that are similar in nature. *Methods to evaluate entertainment value will be developed and used.*

Approach

This section explains the approach taken to expand the foundation of ideas that Sidani presented in his work. Sidani's system was applied to a basic traffic light situation and was able to show that:

- Learning from observation is a valuable way to capture implicit human behaviors
- Input data can be simplified by only using it when needed per situation
- Neural Networks trained on situation-specific data can be more effective than training on an entire data set

This research was inspired by these accomplishments, and has implemented an extension of this work into a gaming application.

Introduction to CONtextual Game Observation (CONGO)

The first extension of Sidani's work is the more complex environment in which the system observes a human perform. The Quake 2 environment requires more intricate human behaviors to be learned for an agent to be functional. The most important behaviors are strategic and tactical, which allow the trained agent to act more human-like.

This research implements Learning from observation using CxBR and Neural Networks. Each of these offers improvements to a game agent or non-player character independently. For instance, CxBR could replace the classical finite state machine and offer a simple transition among hierarchically organized contexts and sentinel rules, allowing for the modeling of an agent to be more intuitive. Machine learning in the form of NNs, can be used to generalize a NPC's action to new situations. This could allow a programmer to reduce the amount of hard-coded actions required for the NPC.

More interestingly, the AI paradigms just mentioned combine to synergistically create a single new method for creating agents from observed human behavior. At the core of CONGO, there is a CxBR engine for determining which context a player is in by monitoring environment variables. For example, if a player has an enemy in close range and is firing at it, the system would gather data for the *attack context*. Since the system knows that the data are only for a specific situation, the input-output patterns are minimized to only what is necessary to function in the current context. When the human player is finished playing, one or more NN's are trained with each context's data. The system then uses the same CxBR engine with the newly-trained NN's combined with default domain knowledge to create a fully functioning game agent.

The following sections will explain CONGO's three main modules: Contextual Observation Module, Network Training Module and the Game Performance Module.

Contextual Observation Module

The contextual observation module passively collects input-output patterns based on a human player's actions for the entire duration of a match. Figure 1 shows the basic flow of the system. The Quake 2 environment passes variables to the CxBR engine, which then outputs the data into the currently-active context's input/output file. The CxBR engine's duty for this module is only to switch in and out of contexts and write patterns out to data files. The engine is comprised of a set of hierarchical contexts along with fixed rules that decide when they are active.

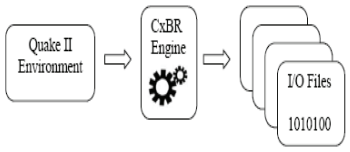

Figure 1 - Contextual Observation Module Diagram

The input/output (I/O) patterns are recorded for each frame that the game server graphically produces. The exact number of frames per second depends on the hardware being used. Quake 2 is an older game and most current systems can push the frame rate to 60 frames per second with ease. Sixty frames per second also produces 60 I/O patterns per second, which contain a large amount of redundant patterns. These redundant data are useful in situations where the data represent a temporal time line of a human's behavior. Some behaviors, such as gun preference, are simple binary decisions carried out instantaneously. Redundant data are not useful in such cases and will only cause the NN's to take longer to train. Therefore, the redundant patterns are filtered out for contexts such as these.

Training module

This module is used after all of the data from the observation module are collected. The files are then formatted in preparation to be passed to the neural network training algorithm. Previous research using NN's have shown that back propagation learning algorithm was capable of learning behaviors such as aiming, or paths around a map [7, 12]. The *RPROP* training algorithm was

chosen along with the use of *Time-Delay*. These are both explained in the next sections.

RPROP – Neural Network Training Algorithm

Resilient Propagation is a modification of back propagation learning algorithm devised by Reidmiller and Braun (Riedmiller and Braun 2003). The modification is a local-learning scheme that uses an update value for each weight to change the weight only when the sign of the partial derivative changes. Tests show that RPROP reduces the chance that a weight update will oscillate, allowing it to converge more often. Furthermore, the number of steps in the training procedure is significantly reduced from traditional gradient descent procedure, thus making RPROP a faster and computationally more efficient learning algorithm.

Time Delay Neural Networks

It has been shown that creating NN's that are purely reactive was not effective in capturing human behaviors in a game simulation (Geisler 2002). Through the use of *time delay neural networks* (TDNN), a network can make decisions based on more than just the current situation. TDNN's have a standard feed-forward structure with the addition of memory nodes. This allows for temporal learning, meaning that the network makes decisions not only based on the present state, but also upon previous ones. A sub-class of the TDNN is the *input-delay neural network.*

Input-delay neural networks (IDNN) concentrate only on the input to the network, while time-delay networks require internal delays at every neuron. This implementation of CONGO uses IDNN. As shown in Figure 2, along with present input pattern, the desired amount of previous input patterns are fed into the IDNN at the input layer. An advantage of the IDNN is having a less complex network than the original TDNN, but preserves the same temporal processing capability (Clouse 2003).

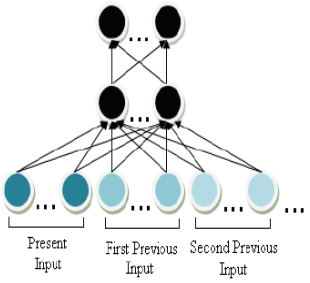

Figure 2 - Input Delay Neural Network Architecture

After the training module creates and trains the NN's, they are exported into the gaming performance module. This is where the agent is placed into the Quake II environment to perform autonomously. The NN's are combined with a default level of intelligence realized through scripted, rule-based AI, all of which is then inserted into the CxBR engine's contexts.

Performance CxBR engine

The same engine is re-used from the observation module, with the addition of NN's and the previously mentioned scripted default knowledge. As seen in Figure 3, the CxBR Engine receives input from the Quake Environment and processes a context to choose just as the observation module did. Then, the context executes actions based upon outputs from a NN or from a script. One concern that arose was how to output functional commands to the Quake engine. Quake 2 has built-in functionality to control agents inside of the environment. The use of this functionality can give the bot abilities to move in ways that a human player cannot. With NN's controlling the output, it can become quite easy for a bot to move and turn quite unnaturally. More importantly, the bot could appear to have an unfair speed or precision advantage. This is why it was decided that the contexts should send keyboard and mouse commands to Quake II, instead of using internal variables.

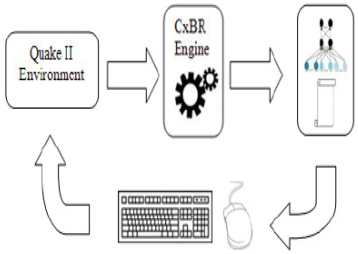

Figure 3: Gaming Performance Diagram

There are three types of control schemes for the contexts. The first scheme is designed for complete control from trained NN's. These are the contexts that require tactical or strategic movements and may contain multiple NN's running concurrently. The second type of control scheme uses a NN to make a decision and once a decision is made, a script carries out the action. This is done to keep the NN's small, but preserve the humanness of the decisions being made. For example, in the *item-hunting* context, the NN decides whether or not to get a certain item when one of its output nodes reaches a certain threshold. After which, a script is called to navigate and pick up the item. The third type is a completely scripted context that gives the bot some minimal level of intelligence. One example to illustrate this is if the bot becomes completely stuck in a corner, a *stuck* context will see that the bot hasn't moved and is surrounded by walls. This will cause the context to become active, and its functions will help the bot maneuver out of the corner.

Context Breakdown

The contexts in this implementation of CONGO are tailored for the FPS genre of game play. They are general enough to apply to games other than Quake 2, although it is possible to add or remove contexts if necessary. The core CxBR engine used by the observation module only contains the contexts that are in need of data to train the

NN's. Others, such as scripted contexts, are implemented as sub-contexts and no data need to be gathered for them. To be clear, there are also sub-contexts that are not scripted. The complete list of contexts and sub-contexts is shown below. The scripted sub-contexts are italicized, and the contexts using both a NN and a script have an asterisk.

- Item-Hunting Context*
  - *Wander Sub-Context*
  - *Stuck Sub-Context*
- Attack Context
  - *Stuck Sub-Context*
  - Retreat Sub-Context*
  - Last Stand Sub-Context
- Run Away Sub-Context*
  - *Stuck Sub-Context*
- Counter-Attack Context
- Enemy-In-Sight Context*
  - Approach Sub-Context
  - Attack Sub-Context
- Just-Saw-An-Enemy Context

Testing

This section describes a series of experiments designed to prove the hypothesis stated earlier. There will be set of three tests given to each volunteer tester. The main factors that need to be validated are:

- The entertainment value of CONGO
- The humanness as compared to other bots
- The accuracy of the learned behaviors

Entertainment value is a scale that quantifies the fun-factor of the game experience. This is a familiar parameter for most gamers because it is how magazines and websites gauge the quality of a game. The second factor, humanness, can be described as how realistically the bot reacts in situations. The most obvious situation is when gamers are usually able detect that an agent isn't human is when the agent moves in ways that a human isn't able to do. This usually results in the player thinking that the match is unfair, and the majority of gamers do not approve of this in an FPS multi-player environment. The accuracy of the learned behaviors will show how well CONGO was able to learn behaviors that players tried to instill into the bot.

Test subjects were assigned skill levels based on their familiarity with the game. A skill level of ten represents an expert Quake 2 player, while a zero represents someone who has never played Quake 2 or any other games like it. Questionnaires were issued for each test subject to gather the following information:

- Behaviors that the test subject successfully trained the bot to perform
- The entertainment value as compared to a well known traditional scripted ACEbot (Yeager 1998)
- How human the bot acted in each context

Detailed descriptions of each test are below:

Test #1: Kill the Running Enemy: The test subjects were asked to train a bot to kill an enemy that was running around in a contained area. The enemy does not fire at the player, and will not leave the area that the player is in. This represents testing only the attack context that from previous research, has proven to be a formidable task (Zanetti 2003). Moreover, the attack context is controlled completely by NN's. Therefore, this test will be able to show the captured behaviors without using any scripted actions.

Test #2: Train Bot for a Real Death Match: This test involves training a complete bot. To do this, the observation module observes the test subject play in a death match against another human player. Similar to the last test, the player is asked the behaviors they intend to instill into the bot beforehand. This time there are context-specific behaviors that they must specify. Lastly, to have another experience to which to relate, the player will play a match against the ACEbot.

Test #3: Death Match against Many Human Players: This test uses the same bot that was trained in the previous test in a match with multiple other human players. This test pushes the bounds of what CONGO bots were designed to do. This implementation of CONGO was not designed to play against multiple enemies. When an enemy is found, the CxBR engine transfers to *enemy in sight* context. This context keeps a single pointer at that enemy, therefore when another enemy comes into sight; there is possibility that it would be ignored. In that same situation, the bot could also oscillate aiming between enemies. This could cause the bot to never attack either enemy.

Results

A summary of these results are shown below in Table 1. In this table, the five test subjects rated the questions asked between 1 and 10, 10 being the best and 1 being the worst. CONGO made it possible to rate individual context's humanness, although the ACEbot had to be rated as a whole.

The results confirm the hypothesis, although interesting insights were obtained regarding the entertainment value of the CONGO system. First, Test #1 demonstrated the utility of the attack context in a non-hostile situation. The trained bots for all subjects were all able to kill the enemy at least four out of the five different situations. This confirms at the very least a basic level of competence for the CONGO bots.

Secondly, Test #2 showed that in most cases the humanness and entertainment value of a bot created with CONGO was better than that of the ACEbot's. The interesting trend found in this test was that because of slight aiming problems the bots created weren't truly competitive for the players, although the entertainment

value was still increased. This shows that even though the bot only posed a small threat, watching it perform the way you intended is entertaining it itself.

Questionnaire Category	Beta	Charlie	Alpha	Delta	Echo
Skill Level	0	3	5	7	10
Test 1: Humanness	6	7	6	3	6.5
Test 2: Item Hunting Humanness	7	8	5	3	6
Test 2: Retreat Humanness	8	8	5	10	3
Test 2: Attack Humanness	6	6	7	6	2
Test 2: Enemy-in-sight Humanness	7	10	7	5	3
Test 3: Item Hunting Humanness	6	6	5	4	6
Test 3: Retreat Humanness	6	9	5	6	3
Test 3: Attack Humanness	5	4	7	4	2
Test 3: Enemy-in-sight Humanness	7	7	7	5	2
Acebot Humanness	4	2	1	0	3
Acebot Enetertainment value	5	5	4	3	7
CONGO Entertainment value	6	8	6	5	7

Table 1 – Summary of Results

Third, Test #3 refuted the notion that a bot made using the CONGO implementation would perform in-humanly in a death match with more than one enemy. The results showed that the humanness only suffered a marginal decrease, and was able to perform just as well as it did in a one-on-one match. See (Moriarty 2007) for complete data and analysis.

Further tests were conducted by sacrificing speed to try to improve the accuracy of the NN's. The results of these tests showed that other approaches didn't seem to improve the accuracy of the bots being trained. Using RPROP keeps the training time low, which is important to preserve the entertainment value.

Summary

With CONGO, players are given the ability to train the AI of their bot by playing how they expect it to act. To ensure playability and create a more humanlike NPC, Context Based Reasoning and Neural Networks are synergistically combined to create a *learning from observation* system. The base set of contexts was established by observing Quake 2 expert players play the game. There were some reactive behaviors that did not fit the normal context specifications. These are behaviors that span for only brief period of time. These behaviors were made to capture a single reaction that a player has. These contexts are forced to be active for two seconds, then they return control to the engine.

The RPROP training algorithm was used and is a modified back propagation technique that significantly reduces the time it takes to train a network, without sacrificing much accuracy. It was also shown in previous work that humans do not make decisions based on the current situation alone (Riedmiller and Braun). For this, time delay neural networks are used to supply CONGO with the ability to reason based on previous states.

The testing procedure was composed of three tests that grow incrementally harder for the bot to perform well. Because it is difficult to observe the bot's behavior while playing against it, a video is recorded from the bot's perspective for the test subject to watch after the match. The tests showed that although the bots produced by CONGO were ranked only slightly above novice skill level, they improved the humanness and entertainment value over a more traditionally scripted bot.

References

Johnson, J. W. D., "Computer Games with Intelligence," IEEE Int. Fuzzy Systems Conf., pp. 1355-1358, 2001.

Chapman N., "NeuralBot," http://homepages.paradise .net.nz/nickamy/neuralbot/index.html, 1999.

Laird, J. E., Assanie, M., and Bachelor, B., "A Test Bed for Developing Intelligent Synthetic Characters," pp. AAAI 2002 Spring Symposium Series: Artificial Intelligence and Interactive Entertainment., 2002.

Thomas, D., "New Paradigms in Artificial Intelligence," AI Game Programming Wisdom 2, pp. 29-39, 2004.

Rabin, S., AI Game Programming Wisdom 2. Hingham, Mass: Charles River Media, 2004.

Evans, R., "AI in Games: A personal View," http://www.gameai.com/blackandwhite.html, 2002.

Geisler, B., "An Empirical Study of Machine Learning Algorithms Applied to Modeling Player Behavior in a "First Person Shooter" Video Game." M.S. thesis, Dept. Computer Science, University of Wisconsin-Madison, WI, 2002.

Nareyek, A., "AI in Computer Games," Queue, vol. 1, pp. 58-65, 2004.

Zanetti, S., "Application of Machine Learning AI to First Person Shooter Games. MSc Dissertation," in Computer Games Technology: Liverpool John Moores University, 2003.

Sidani, T., "Learning situational knowledge through observation of expert performance in a simulation-based environment," vol. Ph.D.: University of Central Florida, 1994.

Riedmiller, M. and Braun, H., "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," Proceedings of the IEEE International Conference on Neural Networks, pp. 586-591, 1993.

Clouse, D. S., Giles, C. L., Horne, B. G., and Cottrell, G. W., "Time-delay neural networks: representation and induction offinite-state machines," IEEE Transactions on Neural Networks, vol. 8, pp. 1065-1070, 1997.

Moriarty, C., "Learning Human Behavior from Observation for Gaming Applications ," M.S. thesis, University of Central Florida, 2007.

Abravanel E., S. Ferguson "Observational learning and the use of retrieval information during the second and third years" Journal of Genetic Psychology; Dec 1998, v.159, 4, 455(2), 1998

Yeager S., "ACEbot: Artificial Control Experiment," http://www.axionfx.com/ace, 1998.