Learning Human Behavior From Observation For Gaming Applications

2007

Christopher Moriarty
University of Central Florida

Find similar works at: http://stars.library.ucf.edu/etd

University of Central Florida Libraries http://library.ucf.edu

Part of the Computer Engineering Commons

STARS Citation


This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
LEARNING HUMAN BEHAVIOR FROM OBSERVATION FOR GAMING APPLICATIONS

by

CHRIS MORIARTY
B.S. University of Central Florida, 2005

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in the School of Electrical Engineering and Computer Science in the College of Engineering and Computer Science at the University of Central Florida Orlando, Florida

Summer Term
2007
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. One focus has turned to adding more humanlike characteristics into computer game agents. Machine learning techniques are being used scarcely 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 to create game agents (bots) in. This work attempts to combine neural networks with a modeling paradigm known as context based reasoning (CxBR) to create a contextual game observation (CONGO) system that produces Quake 2 agents that behave as a human player trains them to act. 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 is a technique that offers the gaming community a mode of game play that has promising entertainment value.
Dedicated to my Mom for letting my Dad buy the Apple II GS instead of a car.
ACKNOWLEDGMENTS

I’d like to first acknowledge my friends at the UCF Intelligent Systems Lab. They are some of the smartest and most helpful folks I’ve ever met. In particular, Brian Becker and Gary Stein, whose programming wizardry far exceeds that of which any one person should possess. I can’t thank my family enough for the much needed moral support they always gave me. Last but definitely not least, I also thank my advisor, Dr. Gonzalez, for all his knowledge and guidance.
# TABLE OF CONTENTS

LIST OF FIGURES .................................................................................................................. X

LIST OF TABLES ...................................................................................................................... XI

CHAPTER 1: INTRODUCTION AND BACKGROUND .............................................................. 1

1.1 History of Video Games ................................................................................................. 1
  1.1.2 Consoles vs PC ...................................................................................................... 2
  1.1.3 Graphics Don’t Mean Everything Anymore ......................................................... 3
1.2 Background of Research .............................................................................................. 5
  1.2.1 Next-Generation Game Consoles Adding More Than Graphics ......................... 5
1.3 Game Engine as a Test-Bed for Research ..................................................................... 6
1.4 Game AI Behind the Times .......................................................................................... 8
1.5 Machine Learning for Gaming ................................................................................. 9
1.6 Combining Machine Learning and Knowledge Engineering ...................................... 11
1.7 CxBR and IASKNOT Comparison ............................................................................ 14
1.8 Default Level of Intelligence ..................................................................................... 16

CHAPTER 2: LITERATURE REVIEW ......................................................................................... 17

2.1 First Person Shooter Engines for Research ............................................................... 17
  2.1.1 Systems That Simplify the Use of a Game Environment ..................................... 17
2.2 Complex AI Employed Through Game Engines ......................................................... 19
  2.2.1 SOAR Rule-Based Architecture ......................................................................... 19
  2.2.2 “D’Artagnan Cognitive Architecture” .................................................................. 21
  2.2.3 Machine Learning Study .................................................................................... 23
  2.2.4 Neural Networks applied to a FPS ...................................................................... 25
2.3 Genetic Algorithms for Games .................................................................................. 27
  2.3.1 Neuro-Evolution ................................................................................................. 28
2.4 Learning from Observation ....................................................................................... 29
  2.4.1 “Learning Robotic Primitives” .......................................................................... 29
  2.4.2 Complex Learning from Observation Techniques ............................................ 30
2.5 Summary ...................................................................................................................... 31

CHAPTER 3: PROBLEM DEFINITION AND HYPOTHESIS ...................................................... 34

3.1 General Problem Statements ...................................................................................... 34
3.2 Specific Problem Statements ..................................................................................... 35
  3.2.1 Humanness in Games ......................................................................................... 35
  3.2.2 Validating Humanness and Entertainment Value ................................................. 37
3.3 Hypothesis ................................................................................................................... 37
3.4 Contributions .............................................................................................................. 37

CHAPTER 4: APPROACH ......................................................................................................... 38
CHAPTER 6: TESTING

6.1 Test Subject Selection............................................................... 77
6.2 Test #1: Kill the Running Enemy .................................................. 79
   6.2.1 Test #1 Procedure .................................................................. 79
   6.2.2 Sample Questionnaire for Test #1............................................. 80
6.3 Test #2: Train Bot for a Real Death Match....................................... 81
   6.3.1 Test #2 Procedure .................................................................. 81
   6.3.2 Sample Questionnaire for Test #2............................................. 83
6.4 Test #3: Death Match against Many Human Players.......................... 84
   6.4.1 Test #3 Procedure .................................................................. 84
   6.4.2 Sample Questionnaire for Test #3............................................. 85
6.5 Test Results .................................................................................... 85
   6.5.1 Test #1 Results ....................................................................... 86
      6.5.1.1 Test Subject Alpha.............................................................. 86
      6.5.1.2 Test Subject Bravo.............................................................. 87
      6.5.1.3 Test Subject Charlie............................................................ 87
      6.5.1.4 Test Subject Delta............................................................... 88
      6.5.1.5 Test Subject Echo............................................................... 88
      6.5.1.6 Test #1 Discussion .............................................................. 89
      6.5.1.7 Aiming Discussion .............................................................. 89
   6.5.2 Test #2 Results ....................................................................... 91
      6.5.2.1 Test Subject Alpha.............................................................. 91
      6.5.2.2 Test Subject Bravo.............................................................. 92
      6.5.2.3 Test Subject Charlie............................................................ 93
      6.5.2.4 Test Subject Delta............................................................... 94
      6.5.2.5 Test Subject Echo............................................................... 95
      6.5.2.6 Test #2 Discussion .............................................................. 96
      6.5.2.7 Correlation Between Skill-Level and Humanness-Values ........ 98
      6.5.2.8 Back Propagation .............................................................. 99
   6.5.3 Test #3 Results ....................................................................... 102
      6.5.3.1 Test Subject Alpha.............................................................. 102
      6.5.3.2 Test Subject Bravo.............................................................. 102
      6.5.3.3 Test Subject Charlie............................................................ 103
      6.5.3.4 Test Subject Delta............................................................... 103
      6.5.3.5 Test Subject Echo............................................................... 103
      6.5.3.6 Test #3 Discussion .............................................................. 104
6.6 Testing Summary ........................................................................... 105

CHAPTER 7: SUMMARY, CONCLUSION & FUTURE WORK ................. 107

7.1 Summary ....................................................................................... 107
   7.1.1 Re-usable ................................................................................. 108
   7.1.2 Ensure Playability ................................................................. 108
   7.1.3 Knowledge Acquisition for CxBR ............................................ 108
LIST OF FIGURES

Figure 1: Contextual Observation Module Diagram .......................................................... 41
Figure 2: Input Delay Neural Network Architecture .......................................................... 46
Figure 3: Gaming Performance Diagram ............................................................................ 47
Figure 4: Test 1 MSE Line Graph ....................................................................................... 90
Figure 5: Test 2 MSE Line Graph ....................................................................................... 97
Figure 6: Entertainment-Value Graph .................................................................................. 97
Figure 7: Humanness Summary Graph ................................................................................ 99
Figure 8: Rigorous Training MSE Graph ............................................................................ 100
Figure 9: Training Algorithm Comparison Graph ............................................................. 101
Figure 10: Average Humanness Ratings ........................................................................... 104
LIST OF TABLES

Table 1: Input Feature Set ................................................................. 23
Table 2: Class Diagram for the base Context ......................................... 57
Table 3: Class Diagram for CxBR engine .............................................. 58
Table 4: FANN Constant Parameters .................................................. 60
Table 5: FANN Variable Parameters ..................................................... 61
Table 6: Summary of Contexts Descriptions ........................................... 75
Table 7: Test Subject's Quake 2 Skill Levels .......................................... 77
Table 8: Test Subject Skill Level Descriptions ....................................... 78
Table 9: Skill Levels for Extra Players in Test #3 .................................... 84
Table 10: Test #1 Questionnaire - Alpha ............................................. 86
Table 11: Test 1 MSE - Alpha .............................................................. 86
Table 12: Test #1 Questionnaire - Bravo ............................................. 87
Table 13: Test #1 MSE - Subject 2 ....................................................... 87
Table 14: Test #1 Questionnaire - Charlie ........................................... 87
Table 15: Test #1 MSE - Charlie ........................................................ 87
Table 16: Test #1 Questionnaire - Delta .............................................. 88
Table 17: Test #1 MSE - Delta ............................................................ 88
Table 18: Test #1 Questionnaire - Echo ............................................... 88
Table 19: Test #1 MSE - Echo ............................................................. 88
Table 20: Test #2 Questionnaire - Alpha ............................................. 91
Table 21: Test #2 MSE - Alpha ............................................................ 92
Table 22: Test #2 Questionnaire - Bravo ............................................. 92
Table 23: Test #2 MSE - Bravo .......................................................... 93
Table 24: Test #2 Questionnaire - Charlie .......................................... 93
Table 25: Test #2 MSE - Charlie ........................................................ 94
Table 26: Test #2 Questionnaire - Delta .............................................. 94
Table 27: Test 2 MSE - Delta ............................................................. 95
Table 28: Test #2 Questionnaire - Echo .............................................. 95
Table 29: Test 2 MSE - Echo .............................................................. 96
Table 30: Test #3 Questionnaire – Alpha ............................................ 102
Table 31: Test #3 Questionnaire - Bravo ............................................ 102
Table 32: Test #3 Questionnaire - Charlie .......................................... 103
Table 33: Test #3 Questionnaire - Delta .............................................. 103
Table 34: Test #3 Questionnaire - Echo .............................................. 103
Table 35: Summary of Questionnaire Data .......................................... 105
CHAPTER 1: INTRODUCTION AND BACKGROUND

The gaming industry has proven itself to be a serious competitor in the entertainment business sector. This $10 billion industry produces more revenue than even the Hollywood movie industry [1]. Video games descend from the original arcade games, such as pinball and slot machines. Pinball and gambling machines are still associated with video games and can be found at most present-day arcades. One of the first video games created was a table-tennis derivative displayed on an oscilloscope, created in 1958 by an admitted pinball player, William A. Higinbotham. Higinbotham used the game to entertain visitors coming to the Brookhaven National Laboratory. Soon after, the video game company “Atari” was started and made the game “Pong” as their first major project. Pong was similar in many ways to Higinbotham’s table tennis, except it used a television and custom arcade controls instead of an oscilloscope.

1.1 History of Video Games

In the 70’s and early 80’s, Atari became the driving force in the video-game industry. Other companies such as “Midway” and “Bally” who were known for making pinball machines also entered the market. In the late 70’s, arcades contained all-time classic games such as “Space Invaders” and “Asteroid.” Before long, Atari created another Pong system that could be played on a regular television in the comfort of people’s homes. This was a monumental landmark and soon after led to the Atari 2600 home gaming console being created. Atari dominated the 8-bit gaming industry with the 2600, even
with other competitors such as “Colecovision” and “Intellivision”. It wasn’t until 1985 that the Nintendo Entertainment System (NES) became the next home console to control the industry. Future generations of gaming consoles would boast about improvements made to graphics and sound quality. This created a trend where the newer system with superior graphics would phase out the older system from the market. More companies progressively entered into the home gaming market, such as SEGA, NEC, PANASONIC and more recently Sony and Microsoft, creating a world-wide competitive money making industry.

1.1.2 Consoles vs PC

The PC also became a platform for video games and was the major influence of the popularity of online play. Attempts were made early on to have online play with console gaming systems such as Super Nintendo and Genesis, although it wasn’t until Microsoft’s online service, “XBOX Live!” that console network play became comparable to a PC’s. Games made for a PC are backwards compatible dating back until the days of DOS, while consoles are limited to the games made specifically for it. The new consoles offer compatibility for their company’s previous model, although the PC still has the upper hand in the number of games available. Although, PCs require an installation of the game and also may require updates to hardware and/or drivers to properly play the game. Consoles have one specific purpose, which is obviously to play games. Therefore, buying and playing games on a console system is significantly easier and more reliable than on a PC. Overall, Consoles offer lower cost, ease of use, and a more comfortable playing
experience (couch and TV). PCs have the upper hand with the number of games and the fact that their hardware is upgradeable, making the PC always on the cutting edge of technology.

1.1.3 Graphics Don’t Mean Everything Anymore

The gaming industry currently faces a new barrier. The addition of better graphics is becoming less important than adding entertainment value to a game. What happens when the game’s graphics are realistic and the only differences in graphics from one game to the next are the quality of shading techniques used? Will this higher quality shading technique add enough to the game to consider it to be better? This time is approaching rapidly, although some argue it is already here [2]. Game play has become the more important factor in the design of a game, while impressive graphics are expected as the norm [3]. With playability becoming a more important factor, doors are opening for artificial intelligence (AI) techniques along with other playability enhancements to be applied.

One relatively new genre which makes use of network connectivity to allow players to enter an online multi-player gaming experience is called “massively multi-player online” (MMO) games. MMO games have proven their popularity among the gaming community with games like “World of Warcraft,” which currently has over five million subscribers. The success of the 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 humanness in non-player characters
(NPC), which is any character in a game that is not controlled by the 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 thesis describes a system that allows a game player to create customized intelligent game agents (bots) in the Quake II environment. When the player chooses to create a new bot, he/she will then perform as he/she wants the bot to perform. “Learning from observation” techniques are used to capture the knowledge of the player. Connectionist and symbolic artificial intelligence practices are combined to apply the captured knowledge into a fully-functional bot. The game environment outputs a high amount of sensor data, which can be difficult for a single machine learning algorithm, such as neural networks, to use in raw form [4]. A contextual engine was created that used 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.
1.2 Background of Research

Interest in video games has increased steadily over the years. Every generation of game consoles has pushed the limits to what computers of the time can process. Each console has had to prove itself by posting the best hardware specifications. Because of this trend, the games that were made for these systems were often evaluated exclusively by their quality of graphics. This was a constant theme in the gaming industry in the early 90’s with the exception of over-hyped cinematic graphics that were prevalent on systems such as Panasonic’s 3DO [5].

1.2.1 Next-Generation Game Consoles Adding More Than Graphics

Three new next-generation consoles were released in 2006: Microsoft Xbox 360, Sony Playstation 3 and the Nintendo Wii. Each company has realized that simply adding processing power was not going to be the only selling point of their game systems. Besides creating cutting-edge hardware components, Microsoft has added many features to the Xbox 360 allowing it to have many media playing attributes, along with improving their internet-based game play (Xbox LIVE!). Similarly, Sony has created a state of the art machine with network game play, but has also added gyro-sensors to the controller to add features such as detecting the movement of the player. Nintendo stands out from the pack and has stated that they are changing the face of gaming. They did not enter into the hardware specs war with Sony and Microsoft; rather they have re-designed their control input to be completely innovative. Their games are more physically interactive through
the use of this new input controller, which intends to attract audiences other than the average gamer.

While the graphics of the next-generation systems have improvements, game development companies are also turning to modern artificial intelligence techniques to improve the entertainment value of their games [1]. This concept is not exclusive to the new consoles being released, but has been steadily becoming more popular as the improvements to the aesthetic features of games have begun to plateau and is now more than ever playing an important role in the success or failure of a game [6].

1.3 Game Engine as a Test-Bed for Research

Academic artificial intelligence research can benefit significantly from utilizing game environment tools to simulate synthetic agents. In the past, game programmers have often had to compromise their artificial intelligence (AI) algorithms to run effectively on the hardware resources available. This struggle has been considerably eased by the exponential increase of CPU power and through the use of a separate Graphics Processing Unit (GPU). The GPU relieves most rendering and animation processing from the main CPU which leaves more resources for implementing complex AI techniques [1, 3]. These game development tools present a valuable test-bed for research by offering an environment in which physics and graphics are already accounted for.

Laird has shown that using a publicly released game engine such as Quake II is indeed a practical solution for academic AI research [2]. Creating a test-bed can often take time away from the actual research being done. The newer engines such as Quake
III Arena could offer nicer special effects, but have posed technical problems with some implementations because of only having partially released source code [7]. While most of the newer engines only allow users limited modification privileges, there are engines such as Unreal Tournament and Quake II that have completely released source code. The internet currently hosts an array of websites that are dedicated to user created modifications to these games. These websites also serve as a reliable reference for programmers creating their own modifications. A large part of this online community is dedicated to the modifications of the FPS genre, under which the Quake and Unreal series fall. These modifications range from custom graphic models to fully functional AI enemies. Such AI enemies are more commonly known as “Bots,” and offer the player opponents to play against, besides the ones that may have come packaged with the game.

AI can add entertainment value to a game, although this is difficult to guarantee. A player needs to take easily to understanding the game, and must be convinced that the model is accurate. When a model is trained poorly, it should still be able to function at a minimal level. This is to prevent the model from degrading the entertainment value of the game. One could argue that maybe the player intended to create a poorly trained bot, which is one reason it is difficult to guarantee that a user will be satisfied with the model. The training process should be seamless, with a minimal amount of loading times to prevent the player from getting bored or confused. A review of the literature shows that other related works have experienced such problems in their research.
1.4 Game AI Behind the Times

Game developers have a tendency to create AI that gives more of an illusion of being intelligent, rather than authentically thinking intelligently [6, 8]. More specifically, the AI agent will make intelligent decisions, but they will be the same intelligent decisions every time. It is true that there are situations that happen once and would only require a single reaction. However, these methods are still being applied into situations that are seen many times. One main reason for this is that it is quicker and more reliable for game developers to stick with what they already know. This is because game developers are often burdened with difficult deadlines, which frequently lead to using practices previously known to work. It is also beneficial for game developers to build on previous code, which is a probable cause for the great number of sequels being produced.

Marketing a game is often easier to exhibit to customers when its visual elements are displayed, which is why they are often created sooner in the development cycle. A game’s AI is heavily dependent on the environment. If the game environment is still under major construction, it may be difficult to develop AI while managing the changes being made to the environment [6]. Because of these issues, AI is often left for the end of the development cycle, which takes away from the amount of time the developers have to implement possible new techniques. A new technique could be more practically applied if a fully functional agent was presented along with the work.

There are certain disadvantages with the “known to work” AI techniques upon which game developers often fall back to. The two most popular “good old fashioned AI” paradigms used in FPS agents are finite state machines (FSM) and rule-based
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 [1]:

- 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 bugged rule that causes it to be vulnerable to a certain attack, a player will exploit this. An exploitation can cause the bot to seem scripted or unnatural. An exploration of modern AI techniques will show that these problems can be addressed.

### 1.5 Machine Learning for Gaming

The use of machine learning (ML) techniques such as Neural Networks (NN) and Genetic Algorithms (GA) has shown their usefulness in some commercial games [9], but are still used scarcely in the industry. There are a few significant reasons for this:

- Machine learning techniques can sometimes lead to unpredictable local maxima [10]
- Game Developers often stick with what they know [6]
- Feature vectors are often too complex to control the agent with machine learning alone [11]
- Academia often creates new paradigms without creating fully functioning agents.

An example of local maxima in a trained game agent could be [10]:

9
The human observation data shows that shooting only occurs 5% of the time, the ML algorithm could find a local maximum to never shoot.

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. Game developers sticking to what they know was discussed previously, but applies more directly now. Using an unsupervised ML technique requires more time in the development cycle for validation to be sure that the network or algorithm performs well under all conditions. This procedure would be time-consuming and risky if there wasn’t a previous model showing how to implement the ML structure. As game environments become more complex, the number of features that can affect an agent in the game make it near 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 [10] [11].

Extensible AI has been offered in game engines since the release of the original Quake engine. Extensible AI allows users to make modifications to the non-player character’s (NPC) attributes and intelligence inside of the game. The makers of Quake, “Id”, wanted their users to be able to customize the AI of their enemies to fit their playing style. Adding these features has proven to be popular among game players [3]. This thesis explores a technique that would allow users to create NPCs modeled after themselves by learning the player’s behaviors from observation. Falling under the category of extensible AI, this technique should bring more entertainment value to the
game play. The goal is for the user to be able create the bot by displaying the playing style that they want the bot to learn.

Current and past research on using game engines as a test bed hasn’t been as beneficial to the gaming industry as to the research community. The reasons for this are often because the goal of the research is not to improve game play, but to implement a specific AI paradigm. When in the context of FPS’s, research in this field often ends up with agents that can accomplish a few tasks, while lacking significantly in others. Left alone, the research cannot be directly applied to a game to increase entertainment value. It may be possible to creatively implement the new technique into a complete agent, but as mentioned earlier, unless this method has been “tried and true,” it is unlikely that it will be used in a production game. This research seeks to create a fully-functional agent that provides the opportunity to test whether the new technique developed here indeed adds significant entertainment value to the game.

1.6 Combining Machine Learning and Knowledge Engineering

A well-known phrase in the field of ML is that there is “no free lunch.” The No Free Lunch theorem states that it is essential to optimize the chosen ML algorithm for the problem at hand. Otherwise, a random search will be just as reasonable of a solution [12]. This becomes especially true when modeling human behavior. For this domain, it is essential to gather domain knowledge to incorporate expert behavior, and then use ML techniques to build upon that knowledge. For this thesis, behaviors of the bot need to be customized based on a player’s actions. Therefore, not all behaviors need to be
implemented with domain knowledge; it should be used to assure that the agent will perform as desired in certain crucial situations. “Implicit behavior” incorporates the reactive actions humans make, which they did not necessarily think about prior to acting. These implicit behaviors are not easy to extract from an expert through traditional knowledge engineering techniques [13]. ML techniques can be used to learn from observation and capture and represent these actions.

Sidani created a ML system that learned how humans implicitly react at traffic signals by observing a human expert perform in a simulation [14]. Sidani created a knowledge representation element called a “Situational Awareness Module” (SAM) for the different simulated situations. The main role of the SAM was to modularize implicit knowledge into a generalized finite group of skills. The motivation for simplifying the situational knowledge was to allow a Neural Network (NN) to learn the observational data more effectively. The entire framework for the system was named “IASKNOT” and has three modes:

1. **Data-Collection** – Collects information about the current situation as well as the observational data coming in from the human expert

2. **Neural Network Training** – Encapsulates the data for each situation into separate neural networks (Neural Network Knowledge Unit) and inserts them into a SAM

3. **Performance** – Assembles a trained neural network with the explicit knowledge that was gathered by using traditional knowledge engineering techniques
Sidani also employed knowledge engineering techniques to extract explicit knowledge from a human expert. This knowledge was imbedded into a SAM through the use of rules. Therefore, this hybrid system’s SAM contained a Neural Network Knowledge Unit (NNKU) and a set of rules that represent domain knowledge. The main function of these rules was to determine when a NNKU should be activated. This methodology bears a significant resemblance to the more modern paradigm known as Context-Based Reasoning (CxBR).

Gonzalez and Ahlers [15] present CxBR as a modeling method that can efficiently represent tactical human behaviors in a simulated agent. The ideas from which CxBR derives its methodology are:

- A given situation requires a set of specific actions and procedures that properly address the current state of affairs.
- As a situation develops into another situation, a different set of actions and procedures may be required to attend to the new situation.
- Events that are feasible under the current situation are constrained by the current situation itself.

CxBR divides a knowledge base into smaller modules known as “contexts,” which group the knowledge needed to represent any given context. Humans often reason in such a manner in that for any given situation, they will only exercise the knowledge needed to deal with the current situation or “context”. An example applied to a video game could be:
A player is crucially low on health and has no chance of defeating all enemies that are attacking. Therefore, the player needs to escape to safety or to find more health. Avoiding combat at all costs would cause the player to disregard close combat fighting skills and opt for the current “escape” situation.

Thus, captured knowledge for this context excludes the use of features such as tactical fighting, which in turn decreases the size of its feature vector. The main difference between CxBR and IASKNOT is in the hierarchical flow of the algorithm. A brief explanation of both techniques will be presented for comparative purposes, after which the differences will be discussed.

1.7 CxBR and IASKNOT Comparison

At the highest level, CxBR has a mission context which defines the goals, plans and constraints. The mission context indirectly controls the agent and is designed to be a high level description of the task at hand. The mission context also contains what is known as the universal sentinel rules. These rules supersede all other transitional rules and are used for situations that must be addressed under all situations. The next level is that of the major contexts. Major Contexts are the main control element for the agent. Major contexts contain the action knowledge, transitional knowledge as well as the list of possible next major contexts to transition to. Transitional knowledge is more formally represented as sentinel rules. Sub-contexts are the lowest level, and they represent actions that are performed in a certain major context that may be too complex for a single
function. This can promote reuse if more than one major context can make use of the same sub-context.

The IASKNOT framework has three main components that contribute to the flow of the reasoner. The Event Recognizer monitors the situations and records what events are occurring. The Event Activator reads the input from the Event Recognizer and searches the knowledge base for a NNKU to activate. When more than one NNKU is activated at once, a situation could arise where they contribute conflicted outputs. An Action Resolver is then used to examine the entire situation and the correlation between objects to decide on a correct output.

The important differences between CxBR and the IASKNOT framework start with the output collision resolution. First, IASKNOT allows for more than one NNKU to be active at one time where CxBR only allows one context to be active. In the event that a context needs to be prioritized, universal sentinel rules give the ability to place more important contexts above others. CxBR requires each context to have a list of possible transitions. Transitions are more important with CxBR because only one context is active at any one time. If the context states that it is still active, then the CxBR engine will check only its universal rules, and if none fire, it will then return control to the currently-active context.

This thesis is essentially an extension of Sidani’s work in that it has improved the techniques used and applied them to a much more complex environment. CxBR is used instead of IASKNOT, but has preserved the idea of combining symbolic and
connectionist knowledge into the contexts. Lastly, the simple traffic light simulation has been replaced with the Quake II environment.

1.8 Default Level of Intelligence

In a perfect world, the observation system would capture enough situations for an agent to be able to generalize in all new situations. However, this is not practical in the real world. Therefore, in order to prevent the bot from looking “foolish”, some scripted AI may be necessary.

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. To be sure that a player doesn’t create agents that aren’t functional or significantly hinder the game’s fun-factor, it is desirable to incorporate default knowledge into the bot. This will require an extraction 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 CxCB-based bot also required human intervention. 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.
CHAPTER 2: LITERATURE REVIEW

The academic research community has made use of game engines as a test bed for artificial intelligence research dating back to the 1950’s in Samuel’s machine learning research with checkers [16]. In more recent years, three-dimensional game engines have begun to offer the researcher a simulation environment that already accounts for the graphics and physics. Publicly released engines such as Quake II, Unreal Tournament offer inexpensive, flexible simulation environments [2]. This allows the programmer to focus more on the actual AI research rather than the preparatory work needed to set up experiments.

2.1 First Person Shooter Engines for Research

Although there are some publicly released FPS engines, there is a learning curve associated with using their source code. Attempts have been made to create a layer on top of the source code that contains useful functions and variables. This allows researchers to decrease the amount of time used to understand the source code of their game environment, and concentrate more on their research.

2.1.1 Systems That Simplify the Use of a Game Environment

Brown and et al have created a Quake II modification that communicates via socket I/O to an externally produced program to act as a stand-in simulation environment [17]. This opens the door for the use of any programming languages with socket I/O libraries
available. The overall purpose for “Quagents” has a more pedagogical goal. The idea was that including video games in homework assignments would spark more of an interest in the classroom. This could logically lead to students being more motivated to do their assignments and possibly to learn more than they would have previously. Therefore, the student’s assignments included programming their routines with their language of choice into a Quake II agent, rather than the previously used basic 2-D Matlab simulator.

The system was tested with a graduate level AI course taught at the University of Rochester [7]. A wide range of projects were assigned such as: state-space problem solving, learning algorithms, production systems (using Jess), natural language understanding, and computer vision. Validation of their project wasn’t a complete success because to their method of using the standard university “end of semester” evaluations. They were able to receive some written comments that the projects were interesting, although they didn’t get any tangible evidence that the projects helped the students learn more. Overall, the modification that they have created will allow future AI developers to test their new algorithms and techniques on a fully functional game agent.

Adobbati’s efforts with Gamebots were to take the Unreal Tournament engine and create a system that could be used for a multi-agent research test bed [18]. This system was a modification to Unreal Tournament that allowed a bot to be controlled by network sockets. This leaves room for large internet-based testing to be conducted, although the authors do not express this as their intent. Users could simply host a game locally, and
have a Gamebots agent connect and play against them. The Gamebots agent could then send data gathered from the match back to the server that was controlling it.

Gamebots is now an open source project hosted by SourceForge. This makes finding and using the source code and program much more convenient. Other small projects that are considered add-ons to Gamebots are also available. These tools include: “Data Logs”, and “VizClient”. “Data Logs”, as the name implies, is a server side utility that saves the output of all internal events into a log file. “VizClient” gives the user a top-down view of the map which shows the current location of the agents. The animations of the agents moving around on VizClient can be saved and loaded in order to compare to another agent. Gamebots offers a framework that has useful development tools and the possibility for large network based testing.

2.2 Complex AI Employed Through Game Engines

In the past, processing power was often a bottleneck for implementing AI techniques. Graphics are now processed on a separate graphics processing unit (GPU), which alleviates much stress from the main CPU. Complex game AI techniques that require real-time performance are feasible, with the help of the GPU and the current state of the art CPU.

2.2.1 SOAR Rule-Based Architecture

Laird used the SOAR rule-based architecture to create intelligent agents within Quake II [2]. The Quake II agent’s AI was based upon a previous SOAR framework made for
autonomous military pilots. All of the knowledge needed for the control of the Quake agent was encoded into approximately 700 rules. The first of two goals of the project was to give their “Quakebot” human-like behavior with a varying level of skill. The second goal was to develop and test a method for evaluating the humanness of the Quakebot.

To fulfill the first goal, the agent’s perceptual information and motor commands were modified to be the same as what a human player would have to use. This was done to avoid the obvious pitfall of having an omniscient enemy that can’t be surprised in an attack, or always shoots perfectly, etc. Four variable parameters were chosen to give the agent a varying level of skill: Decision Time, Aggressiveness, Number of Tactics, and Aiming Skill. An evaluation was done with these parameters to decide which was most effective in varying the bot’s skill and which was most effective in varying the bot’s humanness.

The first experiment was an individual match between an expert Quake II player versus a single Quakebot. The amount of kills and outcome of the match were recorded. Videos of this match from the agent’s perspective were captured for later evaluation. The next experiment was to have three humans of varying skill play against the same bots that the expert played against. The match was also recorded this time but from the human player’s perspective. The final and most effective tool of evaluation in this project was to have human judges watch a mixture of the recorded matches and answer two questions after each one. The first question was to rate the humanness of the player on a scale from
“one to ten” and the second question was to answer “yes or no” if the player was a human or a computer.

The last method of evaluation, of course, drew themes from the classic Turing Test. One difference from the Turing Test however, was that the observer viewed the behavior from the agent’s perspective as it played the game. This was done because a human’s ability to evaluate the agent would be hindered if it were also trying to play against it; also the human is limited to only being able to see a small part of the bot’s behavior while playing against it. Observing from the agent’s perspective will allow the human to devote their full attention to the evaluation and to be able to see every move that the bot makes.

The conclusions of this work remark about the large deviations in the humanness ratings from one evaluator to the next. Regardless, the data gathered showed that the parameter that was most influential on the average humanness rating was “decision time”, with “aiming skill” as a close second. If this was to be done more correctly, judging the humanness over a set of contexts could be more effective. Doing this would require dedicating more time to the validation of a system, and under certain restraints may not always be an option.

2.2.2 “D’Artagnan Cognitive Architecture”

Youngblood investigated alternative ways to continue his AI research in robotics because the costs of having a rich environment for the robot along with the hardware requirements became excessive [19]. Consequently, employing his AI research through the use of a
free or inexpensive game engine became an attractive alternative. The research being conducted is focused on a human behavior architecture named D’Artagnan Cognitive Architecture (DCA). DCA uses multiple agents together to perform tasks as a single cognitive structure, based on the Gestalt psychology, “the whole is greater than the sum of its parts” [19]. The Cognitive-based Agent Management System (CAMS) was chosen to implement the DCA and the Quake II environment was the chosen simulator for CAMS to interface to.

Unlike the Turing-Test styles of validation that Laird employed in [2], Youngblood uses a clustering technique to determine the humanness of his DCA agent. The recorded data are a set of interaction points in a given Quake II level, as well as how often and when the player uses these interaction points. Interaction points are places in the environment were used because movement data is noisy and would not be useful in its raw form. Human players of all skill levels were then asked to play the same quake levels that the agent played. While they were playing, their data was collected. Once all the data were collected, the K-Means clustering algorithm was used to find any clusters that human players’ data formed. The goal was to also capture the agent’s data and cluster it in with the human data set to see if it fit into any of the human clusters that have formed. If the agent was found to be associated with a human cluster, it was then said to have human-like characteristics.

The clustering validation technique used here poses some subjective problems, such as, if the agent is found to be in or close to a human cluster, how human is it?
2.2.3 Machine Learning Study

Geisler performed a basic study on the performance differences of three well-known machine learning algorithms applied to a “First-Person Shooter” game [10]. Naïve Bayes, NN’s and ID3 Decision Trees were trained with observational data collected from an expert player. The underlying motivation for this work was to find a way to capture the decisions made by humans that are often unexplainable. Geisler makes the assumption that this observational data represent examples of correct or incorrect decisions.

Table 1: Input Feature Set

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Closest Enemy Health</td>
</tr>
<tr>
<td>2</td>
<td>Number of Enemies in Sector 1</td>
</tr>
<tr>
<td>3</td>
<td>Number of Enemies in Sector 2</td>
</tr>
<tr>
<td>4</td>
<td>Number of Enemies in Sector 3</td>
</tr>
<tr>
<td>5</td>
<td>Number of Enemies in Sector 4</td>
</tr>
<tr>
<td>6</td>
<td>Player Health Discretized to 0-10</td>
</tr>
<tr>
<td>7</td>
<td>Closest Goal Distance</td>
</tr>
<tr>
<td>8</td>
<td>Closest Goal Sector</td>
</tr>
<tr>
<td>9</td>
<td>Closest Enemy Sector</td>
</tr>
<tr>
<td>10</td>
<td>Distance to Closest Enemy</td>
</tr>
<tr>
<td>11</td>
<td>Current Move Direction</td>
</tr>
<tr>
<td>12</td>
<td>Current Face Direction</td>
</tr>
</tbody>
</table>

The observational data collected are narrowed down by selecting features that were found to be most important for the control of the agent. A list of the features chosen is compiled as Table 1 above. The basic output that results from the aforementioned input features are: accelerate, move direction, facing direction, and jumping. Note that
this is not the complete output needed for a fully functional game agent. The author mentions that there is a high quantity of spatial data about the enemy location and has a very high level of noise. To alleviate this problem, the enemy location data is determined by breaking the world up into four sectors around the player. Rather than capturing the precise position of other objects in the environment, only the one of four quadrants in which they reside will be recorded. Another attempt to simplify the training data was to discard any patterns that are the same five times in a row, only keeping one copy of it. A possible side effect of throwing repetitive patterns away is that time delays may not be fully realized in the trained neural network. In situations where the agent would benefit by pausing for a moment, the agent may be forced to move.

The ID3 algorithm was implemented by pairing it with “rule post-pruning”, which used 10% of the training data as a “tune-set”. To implement the Naïve Bayes algorithm, the author had to state the assumption: “the features in a First Person Shooter are independent enough of each other to allow accurate classification.” For this basic application, the assumption holds true, although it would be questionable for a fully functional game agent. The NN implementation used the standard multi-layer perceptron (MLP) network with back-propagation and 10% of the training data was used to determine the end of training.

The experimental results showed that the NN always had the lowest error rate if given that it was sufficiently trained. It is a limiting factor for a NN if training has strict time contraints or if real-time processing is needed. Modifications such as boosting and bagging were applied to improve the error rates of the NN’s, although the training still
requires the same amount of time. Naïve Bayes has higher error rates than NN although it has a negligible training time, and could be more easily applied to real-time situations. On smaller data sets, the ID3 and Naïve Bayes algorithms actually achieved lower error rates than NN, making them both more attractive for real-time learning game agents.

The “future work” section of Geisler’s research applies almost directly to the goals of this body of work. Geisler suggests that some manual control is needed to completely realize a functional game agent. To do this, he proposes using hand-coded rules in the form of finite state machines (FSM). His example suggests having the NN control the movement of the agent but when the bot reaches the desired location, the FSM takes control.

2.2.4 Neural Networks applied to a FPS

Zanetti completed a body of research containing themes similar to this thesis. His high-level goal was to capture implicit behaviors of human beings through the use of machine learning techniques. The test-bed for this research was the First Person Shooter Quake III Arena, which has its source code only partially released to the public [11].

The first issue that Zanetti addresses is that the behaviors that a human uses can often be too complex to be completely realized. In an effort to simplify the behavioral data, the behaviors were split into simpler sub-behaviors. Now, when the application is gathering data, it will divide the data vectors into three separate sub-behaviors which will be used later to train NN’s. This idea is similar to the logic behind CxBR, in that it is an attempt to simplify human behavior representation by dividing behaviors into smaller
sub-contexts. While the “sub-behavior” idea was used for the right reasons, it was not implemented thoroughly enough. The author only uses three sub-behaviors: “Fight Movement”, “Aim & Shoot”, and “Routing.” Each of these behaviors still contains complex behaviors, which should be further sub-divided in order to more effectively capture individual implicit behaviors.

Zanetti implemented his system using a multi-layer perceptron neural network for each of the three sub-behaviors mentioned above. Genetic algorithms were chosen for the training algorithm on the grounds that the training would be off-line and have no time constraint. For the “routing” sub-context, the map of the environment was segregated into individual interaction points similar to Youngblood’s implementation in [19]. The other sub-behaviors didn’t have any pre-processing done to their data. The author had hoped to create a bot that could function and be used as a worthy opponent, although the results show that the implementation fell short of that goal. The first reason for this was that the NN’s were unable to learn all the data properly in a given sub-behavior. For example, in the “Aim & Shoot” sub-behavior, the bot didn’t learn to actually fire at the enemy. The cause of this was attributed to the fact that the human player only shoots for 5% of the match and this caused the network to settle on a local maximum of never shooting. This is a strong reason to further contextualize observation data before training. The author further concluded that to have a fully functional bot, other AI techniques would be needed to be employed in conjunction with the current machine learning techniques.
2.3 Genetic Algorithms for Games

Genetic algorithms (GA) are stochastic methods based on Darwin’s theory of natural selection. These originally concentrated mostly on mathematical optimization problems. It is also used in projects that promote artificial life (A-life), and can also be applied to human behavior modeling. Cole applied GA to tune the parameters of a FPS bot [20].

The bot was implemented as an expert system with parameterized behavior control. Cole argues that tuning these parameters by hand can be a time-consuming task and that a GA could decrease the amount of time used tweaking. The game engine used was a popular modification to Half-Life called Counterstrike. While the engine does not allow for source code modifications, it does allow extensible AI in the form of parameter libraries. The two parameters that were selected to tune are: weapon-selection and aggressiveness.

Evolved models were created by having the bots in training to fight each other. The fitness function not only quantified the bot’s winning ratio, but also the bot’s skills. The total time for the evolution process to complete was said to take over two hours. This is very reasonable for an off-line optimization, although not very practical for an in-game implantation. The author argues that this automates the process of creating bots, although the expert system has to already be in place for this to work. Creating the expert system is not a negligible task, and it can also be argued that by the time a system is made, the programmer will already have a good idea of what parameter values should be used.
2.3.1 Neuro-Evolution

Neuro-Evolution is the process of training NN’s with genetic algorithms (GA). This is a useful solution for situations where the neural network’s performance can be easily measured, but it’s nearly impossible to establish a list of input-output pairs. For example, collecting observational data from a game can often contain noisy data from environment sensors. Noisy data can be filtered, although it is difficult to establish a list of perfect input output pairs.

Stanley completed a real-time implementation of neuro-evolution making it suitable for a game environment [21]. His modified neuro-evolution algorithm, real-time NeuroEvolving of Augmenting Topologies (rt-NEAT) was used in the academic game project “NERO.” Evolving agents with rt-NEAT starts with a simple NN that has no hidden nodes and is incrementally “complexified” as the evolution process needs to realize more complex behaviors. The game puts the player in the role of a trainer for a group of robot soldiers. The player has no direct control over the soldiers, instead is given tools that reward and discipline the soldier’s actions. Based on the values of these tools, the game will choose the best performing soldiers, remove the worst, and evolve more in real-time. Once the player is satisfied with his trained forces, a battle against another player’s force or a computer generated force will take place.

The rt-NEAT method works in real-time for NERO’s game style, although real-time evolution would not work for a single agent environment such as an FPS. This is because an individual soldier in NERO is not really evolving itself, it is being evaluated
over a given time interval and then being replaced with the next evolved agent. Using rt-NEAT for an FPS would extend training times because each agent would be evaluated individually. If a test bed could be run at an increased speed while evolving agents, this solution could be more viable. If an evolved agent is to perform based on a human’s performance, it would use the observational data to determine its fitness. Therefore, if the simulator can only run at its game play speed, it would take just as long for each genetic agent to be evaluated as it took to create the observation data.

2.4 Learning from Observation

The majority of the literature that describes observational learning techniques applies to robotics. These techniques are often based upon performing simple actions, and do not constitute controlling a complete game agent. Other methods have been used for military simulations, which offer some insight for creating game agents.

2.4.1 “Learning Robotic Primitives”

The main theme for learning in robotics is the use of primitives. Primitives are behaviors that have been broken down into simple units of actions. Bentivegna made use of primitives to allow a robot to learn how to play air hockey and marble mazes from observing a human play [22]. In his air hockey implementation the primitives represented shot types, such as: bank shot, straight shot, and defensive block. The system would observe the state of the game (puck location, velocity, etc) and would then detect the human’s primitive actions.
In an industrial robotics implementation, primitives allowed the system to develop task models from observation to promote re-use [23]. The task models were represented through symbolic primitives as well as stochastic trajectories. Overall, the use of primitives is a symbolic method used to ease the learning process. For example, without primitives, the robots mentioned would have learned in terms of complex motor and gear features. In this thesis, primitives aren’t considered a perfect solution because of the complexity of a game environment. There are too many primitives to be able to use them efficiently in every situation. Regardless, using symbolic representations of behaviors to assist the learning process is a central theme of this thesis.

2.4.2 Complex Learning from Observation Techniques

In a common observational learning system, the data is collected and represented in a static reduced form. When such observational learning systems are found to have a flaw in a single area, it could require for the entire system to be re-trained with updated data. This isn’t the case for an interactive learning system developed by Könik [24]. The training process is done by having a human and an agent visibly performing in a graphical user interface (GUI) concurrently. This allows the human expert to see the flaws in the agent’s performance and address the problem areas.

Applying this technique to this body of work would be difficult for a human to carry out. The human would be required to play the game while also observing the agent’s actions. This would hinder the human’s playing experience, potentially
decreasing the entertainment factor. Regardless, this idea can be used in this thesis for debugging purposes when combined with a visual plug-in such as Gamebots [18].

Another common problem when training observational data into a ML system is explanation of actions. In other words, when systems represent the data in a compressed abstract manner, the explanations of the output cannot be easily understood. Fernlund argues that by evolving models using Genetic Programming, a human could understand the output by reading the source code of the agent [25]. Fernlund’s system, GenCL, combined genetic programming (GP) with context-based reasoning (CxBR) to create models of human driving data gathered from simulator. These models were able to generalize new situations, although within the same simulator and with the same general constraints that were set.

Fernlund states that the use of GP leads to long evolution times. In this thesis, the goal is to produce a working model with minimal waiting time. Loading times in games are considered negative, and can lead a player to becoming bored. Therefore, GP is not a reasonable candidate for the current research. Perhaps after this research is complete there will be room to implement certain contexts with GP and others with NN. This could instill a component into a model that would allow it to update itself with the next evolved generation of training data after each use.

2.5 Summary

Gamebots and Quagents are both similar tools for assisting bot development using game engines. Both projects show ease the use of game engines for academic research.
Obtaining the source for Quagents from the author may be possible, although Gamebots is now an open source project hosted on Source Forge. Therefore, using Gamebots is more convenient and will have more information about it.

Laird’s SOAR-based Quakebot was one of the first attempts to bring academia’s attention to the use of game engines for research. Machine learning was not used in this implementation, although creating a human-like bot was a goal. Laird developed evaluation techniques were very similar to the traditional Turing test. This required extensive cooperation from human volunteers, which wasn’t quite met. The results showed large deviations in evaluations, which means that more human test subjects were needed to smooth out the distribution. Youngblood’s DCA bot was advanced, although it did not include the use of machine learning. His clustering validation technique was novel, although it does not have any way to gauge how human the bot is.

Geisler’s work presented the comparison of multiple ML algorithms. The functionality of the bot was simplified to movements and shooting. NN’s were among the algorithms presented, and it was stated that the bot was unable to realize all behaviors exclusively using NN. Zanetti’s work used multiple NN’s on a set of sub-behaviors. The sub-behaviors that were created and used still contained functionality that was too complex for a single NN to realize.

Learning from observation is used primarily in the field of robotics, making use of basic behaviors known as primitives. Simplifying complex behaviors is indeed a useful tool for reducing the search space for algorithms but, primitives would be over-bearing to apply to create a fully functional bot. Fernlund’s system made use of genetic
programming and context-based reasoning to train computer generated forces based on data collected from a human’s performance. Genetic programming isn’t a viable tool for use in a situation where minimizing training time is important. Konik created a system where the agent and the human perform in the same environment so that the human can observe exactly when and where the agent doesn’t perform as expected. This does not transfer well into a game environment, because it would be difficult for a human to perform and observe the agent concurrently. Overall, this review of the literature showed that game engines are useful for AI research but that machine learning techniques have trouble modeling complex behaviors.
CHAPTER 3: PROBLEM DEFINITION AND HYPOTHESIS

3.1 General Problem Statements

Throughout the history of video games, aesthetic features such as graphics and sound have determined the quality and on some level, the success of a game. With the continuing improvements to microprocessors, excellent graphics and sound have become a standard [3]. The recent game consoles appearing on the market this year have pushed the level of graphics quality once again. However, this time the graphics do not add much more than texturing to what was already considered to be realistic. A limit has been reached where adding higher resolutions and more detailed shadowing techniques have a negligible effect on the entertainment value of the game. These new video game consoles being released have also added extra features to their systems, such as the Nintendo Wii’s innovative remote-control-like input controller. Playability has become more important in the game development process, leaving room for game AI to be improved. This leads to the first problem addressed, which is:

- To create a game AI technique that can be implemented with re-usable characteristics.

Modern AI techniques are being used very scarcely in the video game industry. Arguments have been made that game developers are limited in the freedom they are given to experiment with new techniques because of marketing and project deadlines [6]. Academia also plays a role, although not specifically intentional. Most academic research is to prove scientifically that some proposed research does indeed contribute to
the state of the art. For this reason, their research rarely considers the immediate entertainment value that their research can offer the gaming community. When a new paradigm is introduced, game developers are hard pressed to develop their own research and testing process to decide to whether they can make use of the new technique. The idea is that there have been games produced using advanced AI techniques and proved to be successful from a marketing and entertainment standpoint [9]. The next problem being addressed is:

- How can an AI paradigm be created to ensure that it preserves or improves the playability of the game

### 3.2 Specific Problem Statements

The specific problems addressed in this research lie in creating a more human-like game agent. Attempts have been made to construct bots using NN, but they all report that the system was unable to learn all of the necessary behaviors for a fully functional bot.

#### 3.2.1 Humanness in Games

With the world-wide success of multi-player online games such as World of Warcraft, it’s clear that gamers enjoy playing with other human players. With that in mind, game developers have been adding human-like behavior into their non-player characters (NPC) in hopes of increasing the realistic features of the game. The common techniques used to incorporate humanness into a game agent is often a rule-based system [8]. Systems
constructed in this manner can often become predictable, which in turn could allow a human gamer to exploit certain behaviors that the computer agents make.

Machine learning techniques have been used to capture human behaviors into a model. More specifically, attempts have been made to use NN’s to create human-like bots in first person shooter (FPS) games [4, 10, 11]. 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 environment observation data must be less complex. Dividing the data into contexts would then reduce the necessary behaviors for any one NN to realize. Although, 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.
3.2.2 Validating Humanness and Entertainment Value

The first area of research 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 need to be developed and used*

3.3 Hypothesis

Generating player models from observation using a combination of context based reasoning and neural networks will produce human-like behavior and can enhance the entertainment value of a game.

3.4 Contributions

1. A contextual game observation paradigm used to automatically create first-person shooter agents using neural networks.

2. An implementation of the system for Quake II

3. Validation procedures that assess humanness and entertainment value of the models.
CHAPTER 4: APPROACH

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

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

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

4.1 Introduction to Contextual Game Observation (CONGO)

CONGO is a contribution of this thesis which extends Sidani’s work in many ways. The first of these extensions is the more complex environment in which the system observes a human perform. This environment (Quake 2) requires more complex 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. Lastly, in order to offer the community valuable advances, the system must offer entertainment significance.

CONGO delivers the ability to train a fully functional human-like game agent that can serve as a teammate or enemy in a game environment.
4.1.1 Learning from Observation using CxBR and NN

Learning from observation is implemented using two AI paradigms. Each of these could offer improvements to a game agent or non-player character (NPC) independently. For instance, CxBR could replace the classical finite state machine and offer a simple transition to using hierarchical contexts and sentinel rules, allowing for the modeling of an agent to be more intuitive. Machine learning, specifically NN’s, 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.

4.1.2 Synergistic Combination

The AI paradigms just mentioned can combine to synergistically create 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.
4.2 Overview of the System

When a person uses CONGO, they interface with three main modules. These modules are:

1. Contextual Observation Module
2. Network Training Module
3. Game Performance Module

The first module is where the players train their bot by playing game as they would want the bot to perform. The network training module is clearly where the data gathered in module 1 are used to train the NN that the bot will use. The game performance module combines the newly-trained NN’s and the bot’s default AI to create a fully functional game agent.

4.2.1 Contextual Observation Module

The contextual observation module passively collects input-output patterns based on a human player’s actions. The system will be active 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. A basic CxBR engine is used that only has the ability 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 the rules that decide when they are active.
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. 60 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 NN’s to take longer to train. Therefore, the redundant patterns are filtered out for contexts such as these.

4.2.1.1 Context Inheritance

The contexts that are used in the CxBR engine contain key features that are needed in each one. Inheritance is used to create contexts from an abstract “base context” class. Therefore, every instance of a context that has been inherited contains the bare necessities to function. Those basic properties consist of:

- Sentinel Rules
4.2.1.2 Modified CxBR Engine Algorithm

CONGO uses a slightly modified CxBR algorithm to ensure that contexts finish all of their desired actions. When a context is activated, its active flag is set and the context’s actions are executed. At the end of the actions, the context decides whether it needs to still be active. If the context decides that it needs to be active, the CxBR engine honors the active flag above other contexts, except for the contexts with universal sentinel rules. Even with an active flag up, a universal sentinel rule will have higher priority. The capability for a sub-context to also have a sub-context (sub-sub-context) is implemented, although it was not used in this implementation of CONGO. This was done through a recursive check of the context’s sub-context lists as will be seen in Chapter 5.

4.2.1.3 Pseudo Code for Modified CxBR Algorithm

ENGINE:

// Check for Active Flags in the Universal Sentinel Rules

FOR All Universal Major Contexts
    IF (checkActiveFlag( ))
FOR Subcontexts of this Context
    IF(checkSentinelRule( ))
        activateSubContext();
        GOTO ENGINE;  //return when context finishes
    ENDIF
ENDFOR
activateContext( );
GOTO ENGINE;
ENDIF
ENDFOR

// Check Universal Sentinel Rules

FOR All Universal Major Contexts
    IF (checkSentinelRule( ))
        FOR Subcontexts of this Context
            IF(checkSentinelRule( ))
                activateSubContext();
                GOTO ENGINE;
            ENDIF
        ENDFOR
        activateContext( );
        GOTO ENGINE;
    ENDIF
ENDFOR

// Check for Active Flags in the Sentinel Rules

FOR All Major Contexts
    IF (checkActiveFlag( ))
        FOR Subcontexts of this Context
            IF(checkSentinelRule( ))
                activateSubContext();
                GOTO ENGINE;
            ENDIF
        ENDFOR
        activateContext( );
        GOTO ENGINE;
    ENDIF
ENDFOR
// Check Sentinel Rules

FOR All Major Contexts
  IF (checkSentinelRule( ))
    FOR Subcontexts of this Context
      IF(checkSentinelRule( ))
        activateSubContext();
        GOTO ENGINE;
      ENDIF
    ENDFOR
    activateContext( );
    GOTO ENGINE;
  ENDFOR
ENDIF
ENDFOR

4.2.2 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 [10, 26]. The RPROP training algorithm was chosen along with the use of Time-Delay, which are both explained in the next sections.

4.2.2.1 RPROP – Neural Network Training Algorithm

RPROP, known as resilient propagation, is a modification of back propagation learning algorithm devised by Reidmiller and Braun [27]. 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.

4.2.2.2 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 [10]. 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 will make use of 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 [28].
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.

4.2.3 Gaming Performance Module

The same engine is re-used from the observation module, with the addition of NN’s and scripted AI. 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 NN’s or scripts that use keyboard and mouse commands to control the bot. 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. This inhibits the system from having the ability to execute movements that a human player could not perform.

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, which is more or less there to give 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.

4.3 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 II, although it is possible to add or remove contexts if necessary. The nature of human behavior is often unpredictable. This is something that CONGO accounts for, although, the naming of the contexts may be somewhat misleading. This is because even though the environment and player variables show that a player should be in a certain context, the player may be in fact not be so. For example, in the event that a player’s health is dangerously low and the player’s weapon is insignificant compared to that of the enemy, one would hope that a player would retreat to find health. However, after monitoring various players’ tactics, it was observed that they do not always retreat. Therefore, a trained bot’s actions may not
reflect the name given to the context, although it was the wish of the player for it to perform in this manner.

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.

- Item-Hunting Context
  - Wander Sub-Context
  - Stuck Sub-Context
- Attack Context
  - Stuck Sub-Context
- Retreat 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
4.3.1 Reaction Contexts

There are two contexts listed that have a slightly modified sentinel rule structure; “just-saw-an-enemy” and “counter-attack.” When monitoring human players in either of these contexts, it can be seen that their behavior changes drastically from entering the context to only a short time after. The problem is that there is no clear-cut way to define a rule structure to determine when the player is done with the reaction. An example is shown below to help clarify this idea:

The counter-attack context becomes active when a player is shot at but does not have an enemy in sight. This is a crucial reaction for a bot to understand because it represents a decision that a human made. The human could have decided to get cover from the fire, or on the hand could have pursued the direction of the gun fire. This reaction is complex and does not get learned well if it is all just shoved in the attack context.

What is known is that after a short amount of time, the player has transitioned into another context. Therefore, observations were gathered for the approximate time (two seconds) that a player is in each of the contexts. This will attempt to exclusively capture the reaction that a player displays upon entering the context. The timing observed for these contexts is used to determine when to un-set the active flag.

4.4 Summary

This thesis uses two AI paradigms to implement learning from observation: Context Based Reasoning and Neural Networks. Each of these can offer some improvements to
creating a more human-like NPC, although when combined, they are even more effective. CONGO is the system that does just this. CONGO is comprised of three main modules: Contextual Observation, Training, and Gaming Performance.

The observation module will contextually divide observation patterns in order to minimize the number of them, based on the functionality only needed for a certain context. The training module is comprised of input delay neural networks using the RPROP training algorithm. Lastly, the game performance module is where the NN’s are inserted into their contexts and the bot is able to independently perform in the Quake 2 environment.

There are three context schemes used are:

1. NN Controlled
2. Scripted
3. Combination of NN’s and Scripting

The scripted contexts are implanted as re-useable sub-contexts and represent the bot’s main default knowledge. “Reactive” contexts employ a temporal feature which allows them to capture the single reaction a player has. The two reactive contexts in this implementation of CONGO, just-saw-an-enemy and counter-attack, are both NN controlled.
CHAPTER 5: IMPLEMENTATION

This chapter deals with the implementation of CONGO for Quake 2. An overview of Quake 2 is given, which provides game-play information as well as a description of the source code. After that, the modifications that were made to the source and other tools that were used are explained. The CxBR engine has been discussed at a high-level thus far and will be discussed further in terms of programming.

5.1 Quake 2 Introduction

The first person shooter (FPS) genre, under which Quake 2 falls, gained its popularity from the addition of multiplayer modes of game-play. The need for AI enemies arose quickly when there wasn’t someone to play against online. When the source code for quake was released, the door opened for programmers to attempt to create the next bot that everyone bragged about beating.

5.1.1 Game Play

The focus of the game is from the perspective of a human carrying a weapon. The environment contains many items which can be picked up by a player. A list of item types is as follows:

- Health
- Weapon
- Ammo
The most common multi-player mode in Quake 2, or any FPS for that matter, is the *death match*.

The match starts with all players using the default weapon known as the blaster. Players can use this weapon to attempt to kill other players, although there are many more effective weapons that can be found. Therefore, an experienced player will navigate the map in search of a weapon of choice as well as ammo, armor and any power-ups that can be found. When players confront each other, they can engage in battle where a power meter gauges how much life you have left. Once a player gets killed, a point is then awarded to the killer. The killed player is then respawned at a random point on the map and can resume game play. All weapons and items fall out of the player when they get killed and are then usually picked up by the killer. The winner is declared by reaching the max number of kills first, or is the one with most number of kills when time runs out.

### 5.1.2 ACEbot

This open source project, *Artificial Control Experiment*, is one of the most popular bots for Quake 2 programmers [29]. It became popular because the authors they commented their code very well. Many other bot projects have based their code from the ACEbot. The ACEbot is a competitive bot which offers high entertainment value. The code structure is based on fuzzy logic and only uses functions that represent what a human
player would have to work with. Examples for this are the viewing functions, which can only see in front of them, and the input functions are actually key commands that are sent to the Quake server. The path algorithms and techniques are also impressive. The bot can read node maps (map layouts supplied by online community) or make its own. When the bot is placed in a new map, it creates a memory of where it has been and can begin to use search algorithms to find the shortest path (through the nodes) to the enemies or items that it sees. This thesis makes use of the following functionality from the ACEbot:

- Bot spawning
- Map node structure
- Path Finding Functions
- Used as a comparison in testing

5.2 Quake 2 Source Code

The Quake 2 source is written using C and comes with a Visual Studio 6 project file. An initial CxBR engine was made inside of this C project file, although it was not intuitive and made use of function pointers to implement contexts. In an attempt to make a more intuitive CxBR engine, another version was created using C++ outside of the Quake 2 project using a more heavily object-oriented approach. It was also desired to use a newer version of Visual Studio. Vertigo Software created a port of the source code dubbed “Quake 2.net”, to compile and run under Visual Studio.Net 2003 and also made it freely available. Using the Quake 2.net version of the source code under the Visual Studio.Net
2003 development environment allowed for a much more organized and intuitive CxBR engine to be created.

5.2.1 Quake Structure

The Quake 2 source is structured using a client-server hierarchy. This is done by using a dynamic link library (DLL) as the client which connects to the main server executable. The server can run in two modes, the obvious mode is as a dedicated server. The second is as a “listen server”, which is a client and server together. The listen server is used when a player wants to host a server and play in it as well, all under one instance of Quake 2. The DLL is where the majority of code changes were made, although some important changes involving bot control were made to the executable as well.

5.2.1.1 Sever Executable Modifications

As mentioned in Chapter 4, Quake 2 offers internal control of agents (entities) within the environment. The problem with using Quake’s internal control scheme is that it allows the bot to perform movements that a human could not, or would not do. Therefore, a more fair and natural control scheme was created by allowing actual keys and mouse commands to be sent to the game’s window. Now the bot is given the same control scheme that a human has. Modifications were needed because the game’s main window naturally listens to the actual keyboard and mouse. This caused any mouse commands that were sent to the window to become negated because the mouse isn’t actually
moving. Therefore, the server was augmented to ignore actual mouse and keyboard commands when a “bot control” flag is set, rather than accepting both sets of inputs.

5.2.1.2 Client DLL Modifications

The DLL imports pointers to all the functions needed to communicate to the server what the bot is doing. It also exports pointers to functions that are used by the client for the server to actually execute. The ACEbot spawning functionality makes use of the Quake 2 function “ClientConnect.” This function has been modified to also be able to point to alternate locations (bot code), instead of another human player in the game. This functionality is induced by typing “sv addbot” in the Quake 2 console. Another way to spawn a bot is by making it take over the first person view. This can be done by typing “sv botbrain” in the Quake 2 console. Other pointers are set up as well, inside of the client connect function. One particularly helpful function is “ClientThink,” which tabulates all movements that the client wished to perform and sends a user command structure “ucmd” to the server for processing. This function is called once for every client frame, making it the perfect place to put the CxBR engine. This is entry point for CONGO into the Quake 2 code.

5.3 CxBR Engine Implementation

The CxBR engine was designed so that every context would inherit their functions and attributes from a base class. Polymorphism was also used to implement different functionality into the extended classes while preserving the same function names.
5.3.1 Base Context Class

The base context class contains the functions and variables needed for a basic context to be extended. Polymorphism was implemented into the functions responsible for checking sentinel rules, and firing the context’s actions. This was done because each context has a different set of sentinel rules and actions, and it is more intuitive for the engine to be able to call polymorphic functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sentinelRules(edict_t * self,</td>
<td>virtual bool</td>
</tr>
<tr>
<td>string name)</td>
<td></td>
</tr>
<tr>
<td>fireContext(edict_t * self,</td>
<td>virtual void</td>
</tr>
<tr>
<td>string name)</td>
<td></td>
</tr>
</tbody>
</table>

The first three attributes are self explanatory, although “subContexts” needs a brief explanation. Using the std::vector data structure in C++, subContexts is a list of pointers to all of the sub-contexts the context associated with it. Actual sub-contexts also extend from this class; this makes it possible for even sub-contexts to have sub-contexts, which is permissible in CxBR. The two functions in the class also require an explanation as a result of their use of a Quake 2 structure: “edict_t.” Each player in the game has its own edict_t, which contains all the necessary Quake 2 environment variables related to bot. A pointer to this structure is passed to these two functions to allow the context to have access to the current state of the player and environment.
5.3.2 CxBR Engine Class

The CxBR engine was implemented as its own class to allow the Quake 2 server to have the ability to run more than one CONGO bot at once. The class contains an intuitive set of functions and attributes. The boolean “start” flag is used mostly for debugging, it gets set by a specified keystroke. If the specific keystroke is detected it will start or stop the engine when the bot is loaded into the game. When the engine is stopped, the bot no longer functions and can be manually controlled. This can be used to manually place the bot in a situation where the user wishes to observe it’s behavior.

<table>
<thead>
<tr>
<th>CXBR_Engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>majorContexts : vector&lt;context *&gt;</td>
</tr>
<tr>
<td>universalContexts : vector&lt;context *&gt;</td>
</tr>
<tr>
<td>lastContext : string</td>
</tr>
<tr>
<td>start : bool</td>
</tr>
<tr>
<td>initializeContexts(void) : void</td>
</tr>
<tr>
<td>checkAllSentinelRules(edict_t*) : void</td>
</tr>
<tr>
<td>private loadContext(context *) : void</td>
</tr>
</tbody>
</table>

The vector data structure is used again to hold lists of contexts for the engine to process. Initialize contexts is used once when the bot is loaded into the game, and populates the “allContexts” and “universalContexts” vectors. The one private function “loadContext” is by the “initializeContexts” function to simply push the context into either the majorContexts or universalContexts vector.
5.4 Fast Artificial Neural Network Library (FANN)

The FANN library is an open source neural network toolkit for many different languages, including C++. The project implements RPROP, which is the chosen training algorithm for CONGO. Other features include a graphical interface for choosing parameters and making graphs.

5.4.1 Parameter Choosing

Some parameters change between networks ( ) while others stay constant for each network ( ). Sarle [30] states that in the event that early stopping is used, a higher number of hidden nodes is essential. Another rule of thumb says that the number of hidden nodes should not exceed twice the number of inputs. Therefore, because early stopping is indeed being used, the networks number of hidden nodes is set to be equal to twice the number of inputs. Furthermore, the early stopping criterion used is a minimum mean squared error (MSE).

5.4.1.1 Mean Squared Error

The MSE is computed by summing the squared differences between what the neural network predicted versus the actual value, and then normalize the quantity by dividing by the number of components that went into the sum. This quantity represents how accurately the network performs on a given dataset. The MSE can also be used as an early stopping criterion for training algorithms.
For this research, the MSE is computed in the training module, and represents how well the neural network performed on the same observed data it was trained on. Also, goal MSE values are set for each NN’s training, and if this goal value is reached the network will stop training early. Shown below are the formula and parameters for computing MSE.

\[
MSE = \frac{1}{n} \sum_{j=0}^{n} (O_{j_{NN}} - O_{j_{actual}})^2
\]

- **MSE** - Mean Squared Error
- **n** - Number of output nodes in the neural network
- **O_{j_{NN}}** - The value that the neural network has outputted at node \( j \)
- **O_{j_{actual}}** - The correct value for node \( j \) specified by the provided dataset

### 5.4.2 FANN Implementation

The FANN library comes with sample project files for an array of programming languages, including Visual Studio .NET 2003. From the example, it is easy to implement FANN using different parameters and input patterns. To train all NN’s in one function, a C struct was created hold the parameters for the NN’s that the CxBR engine will be using. A series of functions are then called for each struct:

- **fann_create_standard( )** - Creates a standard fully connected multi-layer perceptron neural network.
- **fann_set_activation_function_hidden( )** – Sets the activation function for the hidden layers
• fann_set_activation_function_output( ) – Sets the activation function for the output layer

• fann_train_on_file( ) - Trains on an entire dataset, for a period of time, by reading from the training data directly from a file.

• fann_save( ) - Save the entire neural network to a configuration file.

Finally, inside the game performance module, the NN’s are used by calling the following function: fann_run( ).

5.5 Context Descriptions

This sections contains the descriptions of each context and its functionality, along with the criteria for it to be active (sentinel rules). A table of the descriptions is provided at the end of the section. The template for each explanation is:

• **General Description:** Explains the role of the context

• **Sentinel Rule:** Pseudo code for the sentinel rule

• **Sub-Contexts:** The names of any sub-contexts attached

• **Control Scheme:** One of the three context control schemes
  
  o [NN only, Scripted, Combination]

• **Universal:** Whether or not the context is universal

• **Observation Description:** Describes how the observation module gathers the necessary variables

• **Scripted Actions:** Any functionality that is hard-coded into the context

• **Number of NN:** Number of neural networks used to control the context’s actions
5.5.1 Item-Hunting Context

**General Description:** This context is active when there aren’t enemies around of which the player is aware. This gives it ample time to gather as many items, weapons, ammo and power-ups as possible.

**Sentinel Rule:** IF no enemy in sight

**Sub-Contexts:** *Wander* and *Stuck*

**Control Scheme:** Combination

**Universal:** No

**Observation Description:** When the context is entered, a snapshot of all item-types in view is taken. When an item is picked up, the system writes an I/O pattern describing which item was obtained, after which a new snapshot is taken and the process repeats.

**Scripted Actions:** Outputs above a certain value trigger the system to look for that item type. If one is found, the bot is then guided to pick up the item using a computed shortest path.

**Number of NN:** 1

**Time Delay:** None

**Inputs to NN:**
● Player Health
● Item categories in sight
● Current Gun Inventory

Outputs from NN:
● Category of the item that was picked up

5.5.2 Attack Context

General Description: This context is active when a player has an enemy clear in view and the enemy is within the close “MELEE” range. Players usually exhibit certain sequences of maneuvers to try to make the enemy’s fire miss them. These maneuvers can be realized as long as they are the prominent behavior performed under a given set of inputs.

Sentinel Rule: IF enemy is in sight AND is in close range

Sub-Contexts: Stuck

Control Scheme: NN only

Universal: No

Observation Description: The keys pressed on the keyboard and the mouse movements are monitored.

Scripted Actions: None

Number of NN: 4

Network 1: WEAPON AIMING

Time Delay: 4 previous inputs
Inputs:

- Pitch, Yaw, and Distance to enemy

Outputs:

- Mouse Movements (left, right, up, down)

Network 2: *GUN PREFERENCE*

Time Delay: No

Inputs:

- Current Gun Inventory

Outputs:

- Currently Equipped Gun

Network 3: *FIRING*

Time Delay: 4 previous inputs

Inputs:

- Pitch and Yaw of enemy

Outputs:

- Fire Weapon

Network 4: *MOVEMENT*

Time Delay: 4 previous inputs

Inputs:

- Pitch, Yaw, and Distance to enemy

Outputs:

- Keyboard commands for:
- Strafe Left or Right
- Move Forward or Back
- Jump
- Crouch

5.5.3 Retreat Context

General Description: This context is controlled by its sub-contexts for the most part. A NN is used to decide which sub-context to activate, and then the sub-context takes over control until the context exits. If the context is re-entered, the NN again analyzes which sub-context to activate based on the new set of inputs.

Sentinel Rule: IF player’s health dangerously low

Sub-Contexts: Last Stand, Run Away

Control Scheme: NN only

Observation Description: System records how close the player is to the enemy and whether or not the player has run away from the enemy or remained to fight.

Scripted Actions: None

Number of NN: 1

Time Delay: No

Universal: YES

Input to NN:
- Most recent pitch, yaw and distance to enemy

Output from NN:
Stay and fight

Run away

5.5.4 Counter-Attack Context

General Description: This context is designed to capture the reaction that a player has to being fired at without seeing an enemy. The context is active for two seconds, and that is enforced through a timer function. Once the two seconds are up, the context is no longer active unless the player is still being shot at by an unseen enemy.

Sentinel Rule: IF player was fired at AND no enemy is in sight

Sub-Contexts: None

Control Scheme: NN only

Universal: YES

Observation Description: The system monitors if a player is shot at, and records the general direction from which the shot originated.

Scripted Actions: None

Number of NN: 1

Time Delay: 2 previous inputs

Input to NN:

• Discretized direction from which enemy fire come

Output from NN:

• Keys being pressed

• Mouse Movements
5.5.5 Enemy in Sight Context

**General Description:** This context allows the player to teach the bot different behaviors for when an enemy is in sight, although it is far away. Under these circumstances it is not always beneficial to approach the enemy. Although, sometimes if the player has a weapon they prefer, it may be beneficial to attack from long distance. The context has multiple NN’s used for attacking and one used to decide whether or not to approach the enemy. The attacking NN’s are identical to the ones found in the *attack* context. Essentially, this context allows the system to create bots that could be considered “campers” or “snipers,” - a strategy that some players use to passively kill enemies.

“Camping” in computer games is the practice of a player hanging out in one part of the game world waiting for enemies to come to the player rather than actively searching for them. A “Sniper” is similar to a camper although it is usually done by finding a long range weapon and staying in areas that permit long range attacks.

**Sentinel Rule:** IF enemy is in sight AND is in far range

**Sub-Contexts:** *Stuck*

**Control Scheme:** Combination

**Universal:** No

**Observation Description:** The distance to the enemy is recorded and used to determine whether the player is pursuing the enemy or not. The keys pressed on the keyboard and the movements made by the mouse are monitored.

**Scripted Actions:** If the first neural network decides that it wants to approach the enemy, a script is used to navigate towards the enemy using a calculated shortest path.
Number of NN: 5

Network 1: **APPROACH ENEMY OR NOT**

**Time Delay:** No

**Inputs:**

- Player’s Health
- Pitch, Yaw, and Distance to enemy

**Outputs:**

- Firing Weapon?
- Moving Toward Enemy?

Network 2: **AIMING**

**Time Delay:** 4 previous inputs

**Inputs:**

- Pitch, Yaw, and Distance to enemy

**Outputs:**

- Mouse Movements (left, right, up, down)

Network 3: **GUN PREFERENCE**

**Time Delay:** No

**Inputs:**

- Current Gun Inventory

**Outputs:**

- Currently Equipped Gun

Network 4: **FIRING**
**Time Delay:** 4 previous inputs

**Inputs:**
- Pitch and Yaw of enemy

**Outputs:**
- Fire Weapon

**Network 5: MOVEMENT**

**Time Delay:** 4 previous inputs

**Inputs:**
- Pitch, Yaw, and Distance to enemy

**Outputs:**
- Keyboard commands for:
  - Strafe Left or Right
  - Move Forward or Back
  - Jump
  - Crouch

---

### 5.5.6 Just-Saw-an-Enemy

**General Description:** This is the second of the aforementioned “reaction contexts.” *Just Saw an Enemy* is designed to capture the reaction a player has when an enemy was seen but is suddenly lost. Based on a number of variables, including which context the player transitioned from, health and current gun, a NN then decides whether to navigate to the last place that the enemy was seen.
Sentinel Rule: IF no enemy in sight AND the last context did have an enemy in sight

Sub-Contexts: Stuck

Control Scheme: Combination

Universal: No

Observation Description: The system records the last context, as well as if the player has moved closer to the enemy.

Scripted Actions: If the NN decides that it wants to pursue the enemy, a path is calculated to the last place the enemy was seen. Then the bot navigates towards it, allowing the NN to jump, crouch or fire.

Number of NN: 1

Time Delay: 2 previous inputs

Input to NN:

- Last Context
- Last distance the enemy was seen at
- Current Gun
- Enemy Gun
- Player’s Health

Output from NN:

- Pursue enemy?
- Limited keys (jump, crouch, fire)
5.5.7 Wander Sub-Context

**General Description:** This context is part of the default knowledge of the bot. The main duty of wander is to guide the bot around when it is not occupied by any other context. This most often happens when there are no items that the bot desires in the *item hunting* context. The wandering is done by a static path around the map that goes between the two main rooms without going out into the wide open. If the bot is in a place that isn’t on the path, a new path is made to get the bot to the wander path.

**Sentinel Rule:** IF no enemy in sight AND no items are desired

**Sub-Contexts:** None

**Control Scheme:** Scripted

5.5.8 Stuck Sub-Context

**General Description:** This context is also part of the bot’s default knowledge. It is used on many other major contexts to help the bot out of hopeless situations. It detects two indications of being stuck. The first is the more obvious situation when the bot is stuck on an object or wall. The second is when the bot is oscillating between going two different directions. To guide the bot out of the place in which it is stuck, the stuck context remembers the last few steps it took the bot to get into the position it is in. It backtracks to the location before it was stuck, and then adds the destination where the bot was headed to a temporary ignore list. Things added to the temporary ignore list become active again after ten seconds.
**Sentinel Rule:**

- IF bot has not moved and has an object in front of it OR
- The bot is oscillating between two points

**Sub-Contexts:** None

**Control Scheme:** Scripted

---

5.5.9 *Last-Stand Sub-Context*

**General Description:** *Last-stand* and *run-away* are both exclusive sub-contexts to the retreat major context. As explained previously, the retreat context uses a NN to merely make a decision as to which sub-context to activate. When the *last stand* context becomes active, it uses the NN’s trained for the *attack* context.

**Sentinel Rule:** IF the neural network in the retreat context outputs a value passed the threshold AND it is higher than the *run-away* output node

**Sub-Contexts:** None

**Control Scheme:** Attack networks are re-used

---

5.5.10 *Run-Away Sub-Context*

**General Description:** *Run Away* is used when a player trained a bot to actually retreat when it is in the retreat context. This is a scripted action that first determines which path to take to get furthest away from the enemy. Then the script navigates away so long as the retreat context is active.
Sentinel Rule: IF the neural network in the retreat context outputs a value passed the threshold AND it is higher than the last-stand output node

Sub-Contexts: None

Control Scheme: Scripted

5.6 Summary

The Quake 2 death-match mode is used in this research, which is the most popular mode in multi-player FPS games. Some functionality was used from the ACEbot project, which was introduced in Chapter 4. This functionality includes: path-finding and spawning bots into the game. The Quake 2 source code is written in C, and provides a Visual Studio 6 project file. Quake.NET is a project which ported this project to be compiled under Visual Studio 2003. The structure of the Quake 2 source code uses a client DLL which communicates to the server executable. Modifications were made to the server to allow for keyboard and mouse commands to be sent to the screen. The client DLL is where the CxBR engine is implemented, and modifications were made in the code to be able to spawn bots as players. The CxBR engine inherits its contexts from a base context. The base context uses polymorphism for context specific functions. Functionality is also added for sub-contexts to have their own sub-contexts (sub-sub-context). FANN is a neural network library which implements many algorithms, one of course being RPROP, which is used in this thesis. Table 4 summarizes the provided descriptions of each context.
### Table 4: Summary of Contexts Descriptions

<table>
<thead>
<tr>
<th>Contexts</th>
<th>Sub-Contexts</th>
<th>Control Scheme</th>
<th>Universal</th>
<th># of NN’s</th>
<th>Time Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item-Hunting</td>
<td>• Wander</td>
<td>Combination</td>
<td>No</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>• Stuck</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attack</td>
<td>• Stuck</td>
<td>NN only</td>
<td>No</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Aim: 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Gun Pref: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Fire: 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Movement: 4</td>
</tr>
<tr>
<td>Retreat</td>
<td>• Last-Stand</td>
<td>NN only</td>
<td>Yes</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>• Run-Away</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Counter-Attack</td>
<td>• Last-Stand</td>
<td>NN only</td>
<td>Yes</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>• Run-Away</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enemy-in-Sight</td>
<td>• Stuck</td>
<td>Combination</td>
<td>No</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Approach: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Aim: 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Gun Pref: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Fire: 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Move: 4</td>
</tr>
<tr>
<td>Just-Saw-an-Enemy</td>
<td>• Stuck</td>
<td>Combination</td>
<td>No</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Wander</td>
<td>None</td>
<td>Scripted</td>
<td>No</td>
<td>0</td>
<td>n/a</td>
</tr>
<tr>
<td>Stuck</td>
<td>None</td>
<td>Scripted</td>
<td>No</td>
<td>0</td>
<td>n/a</td>
</tr>
<tr>
<td>Last-Stand</td>
<td>None</td>
<td>NN only</td>
<td>No</td>
<td>(re-use Attack)</td>
<td></td>
</tr>
<tr>
<td>Run-Away</td>
<td>None</td>
<td>Scripted</td>
<td>No</td>
<td>0</td>
<td>n/a</td>
</tr>
</tbody>
</table>
CHAPTER 6: TESTING

This chapter describes a series of experiments designed to prove the hypothesis stated in Chapter 3. There will be set of three tests given to each volunteer tester. The main factors that need to be validated are:

1. The entertainment value of CONGO
2. The humanness as compared to other bots
3. 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.

Each test covers more of the functionality of the system. The first test is designed as a proof of concept, and simply tests a small subset of features from the system that represent the core of the CONGO approach. The second test is designed to prove the complete hypothesis, therefore validating the research. Lastly, the third test is designed to
explore the full potential of the system by testing beyond what it is designed for.

Questionnaires are issued for each test subject to gather information such as:

- Behaviors that the test subject successfully trained the bot to perform
- The entertainment value as compared to the ACEbot
- How human the bot acted in each context

6.1 Test Subject Selection

The main goal for selecting test subjects was to gather as many different skill levels as possible. This is important to justify the system’s entertainment value for many types of gamers. Five test subjects were selected in which all have different skill levels as shown in Table 5. The players assigned their own skills levels (0-10) following the descriptions in Table 6.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Skill Level (0-10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – Alpha</td>
<td>5</td>
</tr>
<tr>
<td>2 – Bravo</td>
<td>0</td>
</tr>
<tr>
<td>3 – Charlie</td>
<td>3</td>
</tr>
<tr>
<td>4 – Delta</td>
<td>7</td>
</tr>
<tr>
<td>5 – Echo</td>
<td>10</td>
</tr>
</tbody>
</table>
Table 6: Test Subject Skill Level Descriptions

<table>
<thead>
<tr>
<th>Skill Level</th>
<th>Skill Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Never played video games</td>
</tr>
<tr>
<td>1</td>
<td>Have tried video games</td>
</tr>
<tr>
<td>2</td>
<td>Play video games rarely (no FPS)</td>
</tr>
<tr>
<td>3</td>
<td>Play video games occasionally (no FPS)</td>
</tr>
<tr>
<td>4</td>
<td>Play FPS’s occasionally</td>
</tr>
<tr>
<td>5</td>
<td>Have played many video games (including FPS but not Quake 2)</td>
</tr>
<tr>
<td>6</td>
<td>Experienced at many games but not FPS</td>
</tr>
<tr>
<td>7</td>
<td>Experienced at FPS (not Quake 2)</td>
</tr>
<tr>
<td>8</td>
<td>Experienced at Quake 2</td>
</tr>
<tr>
<td>9</td>
<td>Expert at Quake 2</td>
</tr>
<tr>
<td>10</td>
<td>Expert in many FPS games including Quake 2</td>
</tr>
</tbody>
</table>

The first test subject, labeled Alpha, is an experienced gamer, although not specifically in Quake 2. The second test subject, Bravo, rarely plays video games and has never played Quake 2 before. The third test subject, Charlie, plays video games on an infrequent basis and has never played a FPS. The fourth test subject, Delta, is an avid video game player although is not an expert at Quake 2. Finally, the fifth test subject, Echo, is an expert Quake 2 player as well as many other FPS games. Due to the length and complexity of the tests, takes about 1 hour to complete the entire set of tests and questionnaires, it was difficult to find more test subjects.
6.2 Test #1: Kill the Running Enemy

In Test #1 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 [26]. As stated in Chapter 5, 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.

6.2.1 Test #1 Procedure

Before the test begins, the player completes the first questionnaire that asks which behaviors the player intends to teach the bot. A novice player may not know what behaviors that they will use, other than simply killing the enemy. However, an expert player will be able to train the bot with examples such as combat maneuvers, and gun preferences. Later in the test, these taught behaviors will be compared to actual behaviors.

The test begins by opening the observation module. This module starts a listen server, which is a client and a server both open under one instance of Quake 2. The test subject is the only player in the game at first, but then an enemy joins the game from another instance of Quake 2. The enemy will run around the player following the same pattern without leaving the contained area. The player will have the opportunity to kill the enemy until they feel that they have properly trained the bot.
The training module trains the NN’s and inserts them into the gaming performance module. Now the performance module opens up another listen server with only the newly created bot inside of it. The enemy should now be inserted and placed in the same room as the bot. This is the final phase of the test where the player watches his/her bot try to kill the enemy as it was taught. The bot will be allowed to try to kill the static enemy in five different locations. Each location will require the bot to aim differently (i.e. above, below, far away, near, very near). Concluding the final phase, the player will complete rest of the questionnaire for Test #1.

6.2.2 Sample Questionnaire for Test #1

You will be training your bot to shoot an enemy that runs around you. The enemy will not fire at you at all, this is simply to teach your bot the attack context (aiming, shooting, and movement).

1. Rate your experience as a Quake 2 player
   >______ (0 – 10)

2. Circle or describe the behaviors you wish to instill into the bot, then at the end of the match, rate from 0 – 10 as to how well you feel the bot learned the behavior.

   Approach Enemy ____
   Aiming ____
   Constant Firing ____
   Precise Firing ____
   Others: ______________________________________________________
   ____________________________________________________________
   ____________________________________________________________

   After your bot has been trained, watch it try to kill the running enemy again.

3. How many times did the bot kill the enemy?
   >______ (0-5)

4. Rate the humanness of the bot’s attacking
   >______ (0 – 10)
6.3 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. The player is not be asked to specify behavior for the reaction contexts, because they are there to capture the implicit reaction a player has, therefore it would be difficult to specify the actions the player intends to do in that situation. Lastly, to have another experience which to relate, the player will play a match against the ACEbot.

6.3.1 Test #2 Procedure

The observation module is opened again, and this time another human player joins the game (rather than only a running enemy). The test subject proceeds to fight the human opponent until the match is over. The end of the match is decided when either a player gets ten kills or a time limit of 10 minutes expires. Note that the same player, who has a skill level of 5, is used to play against every test subject in Test #2. At the conclusion of the match the data will be sent again to the training module.

When the training module finishes, the NN’s are inserted into the game performance module. Now the performance module opens with the newly-trained bot inside. Each test subject then plays against his/her own creation in a one vs. one death match. Because it would be difficult for the player to observe the bot’s behaviors while
they are playing against it, a video of the bot is recorded. Then after the match, the
player is able to watch the recording and observe the bot’s behavior more closely. Lastly,
a match against the ACEbot is conducted in the same manner as the previous match.
### 6.3.2 Sample Questionnaire for Test #2

1. **Circle or specify behaviors for each context that you wish to instill into the bot. Rate the behaviors you circled from 0 – 10 after watching the bot.**

<table>
<thead>
<tr>
<th><strong>Item Hunting Context</strong></th>
<th><strong>Retreat Context</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pick everything up ____</td>
<td>Never Retreat ____</td>
</tr>
<tr>
<td>Prefer guns and ammo ___</td>
<td>Always Retreat ____</td>
</tr>
<tr>
<td>Don’t pick anything up ___</td>
<td>Retreat if gun is blaster ___</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Humanness: ___ (0-10)</td>
<td>Humanness: ___ (0-10)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Attack Context</strong></th>
<th><strong>Enemy in Sight Context</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Favorite Gun: ______</td>
<td>Favorite Gun: ______</td>
</tr>
<tr>
<td>Approach Enemy ____</td>
<td>Approach Enemy ____</td>
</tr>
<tr>
<td>Aiming ____</td>
<td>Fight from a distance ____</td>
</tr>
<tr>
<td>Constant Firing ___</td>
<td>Aiming ____</td>
</tr>
<tr>
<td>Jumping ____</td>
<td>Jumping ____</td>
</tr>
<tr>
<td>Crouching ____</td>
<td>Crouching ____</td>
</tr>
<tr>
<td>Precise Firing ___</td>
<td>Precise Firing ___</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Humanness: ___ (0-10)</td>
<td>Humanness: ___ (0-10)</td>
</tr>
</tbody>
</table>

2. **What was the outcome of the training?**
   - Your Kills: ____
   - Enemy Kills: ____

3. **What was the outcome of the match?**
   - Bot Kills: ____
   - Your Kills: ____

4. **Rate ACEbot’s entertainment value >____ (0-10)**

5. **Rate ACEbot’s humanness >____ (0-10)**

6. **Rate CONGO’s entertainment value >____ (0-10)**

83
6.4 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. Some examples illustrating potential problems with having multiple enemies are:

- When an enemy is found, the CxCB 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.

- The bot could oscillate aiming between enemies. This could cause the bot to never attack either enemy.

6.4.1 Test #3 Procedure

Open the game performance module from Test #2. Next, three human players join the match. These three players are constant in all Test #3 matches, and are also of varying skill levels:

<table>
<thead>
<tr>
<th>Code Name</th>
<th>Skill Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foxtrot</td>
<td>9</td>
</tr>
<tr>
<td>Golf</td>
<td>6</td>
</tr>
<tr>
<td>Hotel</td>
<td>3</td>
</tr>
</tbody>
</table>
There are no teams established; the bot is forced to fight against all enemies encountered. The match ends when the first player or bot has reached ten kills, or five minutes runs out. During the match, the test subjects watch their bot from a first person perspective. This will allow them to answer the remaining questions on the questionnaire.

6.4.2 Sample Questionnaire for Test #3

1. Rate the humanness of each context

   Item Hunting Context
   Humanness: ___ (0-10)

   Retreat Context
   Humanness: ___ (0-10)

   Attack Context
   Humanness: ___ (0-10)

   Enemy in Sight Context
   Humanness: ___ (0-10)

2. What was the outcome of the match?
   Human Player Kills: ____
   Bot Kills: ____

6.5 Test Results

The data gathered from testing consists of a questionnaire, and a mean squared error (MSE) table from the NN. Only a subset of the contexts is used for Test #1, which are those needed for attacking. This subset includes the attack context, and just-saw-an-enemy context. The charts and graphs are accompanied by a discussion that explains some trends that were discovered. The questionnaire data was mainly used to collect information about the humanness and entertainment values of the system. Although,
other data on how well the bot learned certain behaviors was also collected to better explain what happened in the tests.

6.5.1 Test #1 Results

This section displays and discusses the results of Test #1.

6.5.1.1 Test Subject Alpha

Table 8: Test #1 Questionnaire - Alpha

<table>
<thead>
<tr>
<th>Player's Quake 2 Skill Level</th>
<th>5/10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desired behaviors and score</td>
<td></td>
</tr>
<tr>
<td>• Aiming – 5/10</td>
<td></td>
</tr>
<tr>
<td>• Approach Enemy – 7/10</td>
<td></td>
</tr>
<tr>
<td>• Precise Firing – 7/10</td>
<td></td>
</tr>
<tr>
<td>Rate the humanness of the bot's attacking</td>
<td>6/10</td>
</tr>
<tr>
<td>Results of killing tests</td>
<td>5/5</td>
</tr>
</tbody>
</table>

Table 9: Test 1 MSE - Alpha

<table>
<thead>
<tr>
<th>Attack 1: Aiming</th>
<th>0.44422</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack 2: Gun preference</td>
<td>0.06940</td>
</tr>
<tr>
<td>Attack 3: Firing</td>
<td>0.22480</td>
</tr>
<tr>
<td>Attack 4: Movement</td>
<td>0.05606</td>
</tr>
<tr>
<td>Just Saw an Enemy</td>
<td>0.22775</td>
</tr>
</tbody>
</table>
### 6.5.1.2 Test Subject Bravo

#### Table 10: Test #1 Questionnaire - Bravo

<table>
<thead>
<tr>
<th>Player's Quake 2 Skill Level</th>
<th>0/10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desired behaviors and score from testing</td>
<td>Aiming – 3/10, Stay far away – 8/10, Precise Firing – 5/10</td>
</tr>
<tr>
<td>Rate the humanness of the bot's attacking</td>
<td>6/10</td>
</tr>
<tr>
<td>Results of killing tests</td>
<td>4/5</td>
</tr>
</tbody>
</table>

#### Table 11: Test #1 MSE - Subject 2

| Attack 1: Aiming | 0.40967 |
| Attack 2: Gun preference | 0.07574 |
| Attack 3: Firing | 0.19324 |
| Attack 4: Movement | 0.09366 |
| Just Saw an Enemy | 0.24392 |

### 6.5.1.3 Test Subject Charlie

#### Table 12: Test #1 Questionnaire - Charlie

<table>
<thead>
<tr>
<th>Player's Quake 2 Skill Level</th>
<th>3/10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desired behaviors and score from testing</td>
<td>Aiming – 4/10, Constant Firing – 10/10</td>
</tr>
<tr>
<td>Rate the humanness of the bot's attacking</td>
<td>7/10</td>
</tr>
<tr>
<td>Results of killing tests</td>
<td>4/5</td>
</tr>
</tbody>
</table>

#### Table 13: Test #1 MSE - Charlie

| Attack 1: Aiming | 0.41839 |
| Attack 2: Gun preference | 0.01215 |
| Attack 3: Firing | 0.22829 |
| Attack 4: Movement | 0.00825 |
| Just Saw an Enemy | 0.13495 |
6.5.1.4 Test Subject Delta

Table 14: Test #1 Questionnaire - Delta

<table>
<thead>
<tr>
<th>Player's Quake 2 Skill Level</th>
<th>7/10</th>
</tr>
</thead>
</table>
| Desired behaviors and score from testing | ● Aiming – 3/10  
● Precise Firing – 8/10 |
| Rate the humanness of the bot’s attacking | 3/10 |
| Results of killing tests | 4/5 |

Table 15: Test #1 MSE - Delta

| Attack 1: Aiming | 0.57831 |
| Attack 2: Gun preference | 0.01387 |
| Attack 3: Firing | 0.31561 |
| Attack 4: Movement | 0.59871 |
| Just Saw an Enemy | 0.31982 |

6.5.1.5 Test Subject Echo

Table 16: Test #1 Questionnaire - Echo

<table>
<thead>
<tr>
<th>Player's Quake 2 Skill Level</th>
<th>10/10</th>
</tr>
</thead>
</table>
| Desired behaviors and score from testing | ● Aiming – 3/10  
● Precise Firing – 8/10 |
| Rate the humanness of the bot’s attacking | 6.5/10 |
| Results of killing tests | 4/5 |

Table 17: Test #1 MSE - Echo

| Attack 1: Aiming | 0.47264 |
| Attack 2: Gun preference | 0.03934 |
| Attack 3: Firing | 0.24896 |
| Attack 4: Movement | 0.73124 |
| Just Saw an Enemy | 0.22714 |
6.5.1.6 Test #1 Discussion

This test is a good evaluation of what the attack context is capable of in a non-hostile situation. The enemy is simply running a pattern around a contained area. The test subject is not under the stress of being attacked, which allows him to act completely offensive. By acting only offensively, the player eliminates any different behaviors that could be caused by having to dodge enemy fire. The results showed that all subjects were able to create bots that were successful in killing the “running enemy” most or all of the time. The situations where the bots were unable to kill the enemy were a result of aiming problems, that even if there were an unlimited amount of time given to the bot, the enemy would never get killed. For example, the bot never appropriately learned to aim at an enemy above it; therefore in a situation when the enemy is above the bot, it might always aims too low. This would cause the bot to never be able to kill the enemy, regardless of the game duration. This points out a possible flaw in the training process. Overall, the five CONGO bots created did demonstrate enough learned behavior to kill the enemy at least four out of five times.

6.5.1.7 Aiming Discussion

The NN used for aiming in the attack context is by far the most crucial network with respect to creating a successful bot. This is particularly true in Test #1, where the bot must be trained to kill enemies placed in multiple situations. It was found that even if the aiming was only slightly off, that was enough for the bot to consistently miss the enemy.
It can also be seen in Figure 4 that the aiming network consistently had a relatively high MSE regardless of the skill level of the player. This implies some constant level of noise that must be present within the training data.

Players that chose weapons that require high accuracy often found that their bot would have more trouble killing enemies. The aiming network would have a general trend to keep the shots around the enemy, although not always directly on it. With this in mind, players that used weapons have a larger area of effect were more successful in hitting their enemies. This implies some inaccuracies on the part of the network.

![Figure 4: Test #1 MSE Line Graph](image-url)
6.5.2 Test #2 Results

Similarly to 6.5.1 Test #1 Results, this section displays and discusses the results of Test #3.

6.5.2.1 Test Subject Alpha

Table 18: Test #2 Questionnaire - Alpha

<table>
<thead>
<tr>
<th>Desired Item-Hunting Behaviors and score</th>
<th>● Pick everything up (8/10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item Hunting Humanness</td>
<td>5/10</td>
</tr>
<tr>
<td>Desired Retreat behavior and score</td>
<td>● Retreat if gun is blaster (5/10)</td>
</tr>
<tr>
<td>Retreat Humanness</td>
<td>5/10</td>
</tr>
</tbody>
</table>
| Desired Attack behaviors and score     | ● Approach enemy (5/10)  
||● Aiming (4/10) 
||● Precise Firing (5/10) 
||● Favorite Gun: Super Shotgun (10/10) |
| Attack Humanness                       | 7/10                        |
| Desired Enemy In Sight Behaviors and score | ● Approach Enemy (7/10) |
| Enemy in Sight Humanness               | 7/10                        |
| Results of training match              | ● Test Subject kills: 8  
||● Enemy kills: 10 |
| Results of testing match               | ● CONGO bot kills: 3  
||● Enemy kills: 10 |
| ACEbot match outcome                   | ● Player kills: 1  
||● ACEbot kills: 10 |
| ACEbot humanness                       | 1/10                        |
| ACEbot entertainment value             | 4/10                        |
| CONGO entertainment value              | 6/10                        |
Table 19: Test #2 MSE - Alpha

<table>
<thead>
<tr>
<th>Attack/Event</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack 1: Aiming</td>
<td>0.44421</td>
</tr>
<tr>
<td>Attack 2: Gun preference</td>
<td>0.01917</td>
</tr>
<tr>
<td>Attack 3: Firing</td>
<td>0.21063</td>
</tr>
<tr>
<td>Attack 4: Movement</td>
<td>0.44176</td>
</tr>
<tr>
<td>Counter Attack</td>
<td>0.54099</td>
</tr>
<tr>
<td>Just Saw an Enemy</td>
<td>0.15374</td>
</tr>
<tr>
<td>Item Hunting</td>
<td>0.05315</td>
</tr>
<tr>
<td>Enemy In Sight 1: Approach or Not</td>
<td>0.37471</td>
</tr>
<tr>
<td>Enemy In Sight 2: Gun Preference</td>
<td>0.07143</td>
</tr>
<tr>
<td>Enemy In Sight 3: Firing</td>
<td>0.19275</td>
</tr>
<tr>
<td>Enemy In Sight 4: Movement</td>
<td>0.44469</td>
</tr>
</tbody>
</table>

6.5.2.2 Test Subject Bravo

Table 20: Test #2 Questionnaire - Bravo

<table>
<thead>
<tr>
<th>Desired Item-Hunting Behaviors and score</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Item Hunting Humanness</td>
<td>7/10</td>
</tr>
<tr>
<td>Desired Retreat behavior and score</td>
<td>Always Retreat (7/10)</td>
</tr>
<tr>
<td>Retreat Humanness</td>
<td>8/10</td>
</tr>
<tr>
<td>Desired Attack behaviors and score</td>
<td>Aiming (4/10)</td>
</tr>
<tr>
<td></td>
<td>Constant Firing (8/10)</td>
</tr>
<tr>
<td></td>
<td>Favorite Gun: Hyper-Blaster (10/10)</td>
</tr>
<tr>
<td>Attack Humanness</td>
<td>6/10</td>
</tr>
<tr>
<td>Desired Enemy In Sight Behaviors and score</td>
<td>Fight from a distance (7/10)</td>
</tr>
<tr>
<td>Enemy in Sight Humanness</td>
<td>7/10</td>
</tr>
<tr>
<td>Results of training match</td>
<td>Test Subject kills: 2</td>
</tr>
<tr>
<td></td>
<td>Enemy kills: 10</td>
</tr>
<tr>
<td>Results of testing match</td>
<td>CONGO bot kills: 2</td>
</tr>
<tr>
<td></td>
<td>Enemy kills: 10</td>
</tr>
<tr>
<td>ACEbot match outcome</td>
<td>Player kills: 0</td>
</tr>
<tr>
<td></td>
<td>ACEbot kills: 10</td>
</tr>
<tr>
<td>ACEbot humanness</td>
<td>4/10</td>
</tr>
<tr>
<td>ACEbot entertainment value</td>
<td>5/10</td>
</tr>
<tr>
<td>CONGO entertainment value</td>
<td>6/10</td>
</tr>
</tbody>
</table>
Table 21: Test #2 MSE - Bravo

<table>
<thead>
<tr>
<th>Action</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack 1: Aiming</td>
<td>0.44923</td>
</tr>
<tr>
<td>Attack 2: Gun preference</td>
<td>0.01378</td>
</tr>
<tr>
<td>Attack 3: Firing</td>
<td>0.31053</td>
</tr>
<tr>
<td>Attack 4: Movement</td>
<td>0.49112</td>
</tr>
<tr>
<td>Counter Attack</td>
<td>0.68124</td>
</tr>
<tr>
<td>Just Saw an Enemy</td>
<td>0.21978</td>
</tr>
<tr>
<td>Item Hunting</td>
<td>0.06224</td>
</tr>
<tr>
<td>Enemy In Sight 1: Approach or Not</td>
<td>0.25976</td>
</tr>
<tr>
<td>Enemy In Sight 2: Gun Preference</td>
<td>0.05326</td>
</tr>
<tr>
<td>Enemy In Sight 3: Firing</td>
<td>0.27641</td>
</tr>
<tr>
<td>Enemy In Sight 4: Movement</td>
<td>0.57254</td>
</tr>
</tbody>
</table>

6.5.2.3 Test Subject Charlie

Table 22: Test #2 Questionnaire - Charlie

<table>
<thead>
<tr>
<th>Desired Item-Hunting Behaviors and score</th>
<th>Pick up everything (8/10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item Hunting Humanness</td>
<td>8/10</td>
</tr>
<tr>
<td>Desired Retreat behavior and score</td>
<td>Never Retreat (9/10)</td>
</tr>
<tr>
<td>Retreat Humanness</td>
<td>8/10</td>
</tr>
<tr>
<td>Desired Attack behaviors and score</td>
<td>Aiming (4/10)</td>
</tr>
<tr>
<td></td>
<td>Constant Firing (8/10)</td>
</tr>
<tr>
<td></td>
<td>Favorite Gun: Shotgun (9/10)</td>
</tr>
<tr>
<td>Attack Humanness</td>
<td>6/10</td>
</tr>
<tr>
<td>Desired Enemy In Sight Behaviors and score</td>
<td>Approach Enemy (10/10)</td>
</tr>
<tr>
<td>Enemy in Sight Humanness</td>
<td>10/10</td>
</tr>
<tr>
<td>Results of training match</td>
<td>Test Subject kills: 5</td>
</tr>
<tr>
<td></td>
<td>Enemy kills: 10</td>
</tr>
<tr>
<td>Results of testing match</td>
<td>CONGO bot kills: 2</td>
</tr>
<tr>
<td></td>
<td>Enemy kills: 10</td>
</tr>
<tr>
<td>ACEbot match outcome</td>
<td>Player kills: 0</td>
</tr>
<tr>
<td></td>
<td>ACEbot kills: 10</td>
</tr>
<tr>
<td>ACEbot humanness</td>
<td>2/10</td>
</tr>
<tr>
<td>ACEbot entertainment value</td>
<td>5/10</td>
</tr>
<tr>
<td>CONGO entertainment value</td>
<td>8/10</td>
</tr>
</tbody>
</table>
### Table 23: Test #2 MSE - Charlie

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack 1: Aiming</td>
<td>0.40390</td>
</tr>
<tr>
<td>Attack 2: Gun preference</td>
<td>0.01761</td>
</tr>
<tr>
<td>Attack 3: Firing</td>
<td>0.18382</td>
</tr>
<tr>
<td>Attack 4: Movement</td>
<td>0.23157</td>
</tr>
<tr>
<td>Counter Attack</td>
<td>0.77831</td>
</tr>
<tr>
<td>Just Saw an Enemy</td>
<td>0.18628</td>
</tr>
<tr>
<td>Item Hunting</td>
<td>0.04349</td>
</tr>
<tr>
<td>Enemy In Sight 1: Approach or Not</td>
<td>0.33061</td>
</tr>
<tr>
<td>Enemy In Sight 2: Gun Preference</td>
<td>0.19558</td>
</tr>
<tr>
<td>Enemy In Sight 3: Firing</td>
<td>0.27641</td>
</tr>
<tr>
<td>Enemy In Sight 4: Movement</td>
<td>0.21142</td>
</tr>
</tbody>
</table>

### 6.5.2.4 Test Subject Delta

### Table 24: Test #2 Questionnaire - Delta

<table>
<thead>
<tr>
<th>Desired Item-Hunting Behaviors and score</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefer guns and ammo (8/10)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item Hunting Humanness</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3/10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Desired Retreat behavior and score</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never Retreat (10/10)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Retreat Humanness</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Desired Attack behaviors and score</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aiming (4/10)</td>
<td></td>
</tr>
<tr>
<td>Precise Firing (8/10)</td>
<td></td>
</tr>
<tr>
<td>Favorite Gun: Rockets (9/10)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attack Humanness</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Desired Enemy In Sight Behaviors and score</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approach Enemy (5/10)</td>
<td></td>
</tr>
<tr>
<td>Favorite Gun: Rockets (9/10)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enemy in Sight Humanness</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5/10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Results of training match</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Subject kills: 10</td>
<td></td>
</tr>
<tr>
<td>Enemy kills: 2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Results of testing match</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONGO bot kills: 2</td>
<td></td>
</tr>
<tr>
<td>Enemy kills: 10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACEbot match outcome</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player kills: 6</td>
<td></td>
</tr>
<tr>
<td>ACEbot kills: 10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACEbot humanness</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACEbot entertainment value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3/10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONGO entertainment value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5/10</td>
<td></td>
</tr>
</tbody>
</table>
Table 25: Test 2 MSE - Delta

<table>
<thead>
<tr>
<th>Attack/Counter Attack</th>
<th>Delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack 1: Aiming</td>
<td>0.56432</td>
</tr>
<tr>
<td>Attack 2: Gun preference</td>
<td>0.05124</td>
</tr>
<tr>
<td>Attack 3: Firing</td>
<td>0.39812</td>
</tr>
<tr>
<td>Attack 4: Movement</td>
<td>0.49821</td>
</tr>
<tr>
<td>Counter Attack</td>
<td>0.67521</td>
</tr>
<tr>
<td>Just Saw an Enemy</td>
<td>0.31281</td>
</tr>
<tr>
<td>Item Hunting</td>
<td>0.03412</td>
</tr>
<tr>
<td>Enemy In Sight 1: Approach or Not</td>
<td>0.41872</td>
</tr>
<tr>
<td>Enemy In Sight 2: Gun Preference</td>
<td>0.19872</td>
</tr>
<tr>
<td>Enemy In Sight 3: Firing</td>
<td>0.43521</td>
</tr>
<tr>
<td>Enemy In Sight 4: Movement</td>
<td>0.41527</td>
</tr>
</tbody>
</table>

6.5.2.5 Test Subject Echo

Table 26: Test #2 Questionnaire - Echo

<table>
<thead>
<tr>
<th>Desired Item-Hunting Behaviors and score</th>
<th>Pick everything up (9/10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item Hunting Humanness</td>
<td>6/10</td>
</tr>
<tr>
<td>Desired Retreat behavior and score</td>
<td>Retreat if gun is blaster (4/10)</td>
</tr>
<tr>
<td>Retreat Humanness</td>
<td>3/10</td>
</tr>
<tr>
<td>Desired Attack behaviors and score</td>
<td>Aiming (2/10)</td>
</tr>
<tr>
<td></td>
<td>Precise Firing (8/10)</td>
</tr>
<tr>
<td></td>
<td>Favorite Gun: Railgun (9/10)</td>
</tr>
<tr>
<td>Attack Humanness</td>
<td>2/10</td>
</tr>
<tr>
<td>Desired Enemy In Sight Behaviors and score</td>
<td>Fight from a distance (5/10)</td>
</tr>
<tr>
<td></td>
<td>Favorite Gun: Railgun (9/10)</td>
</tr>
<tr>
<td>Enemy In Sight Humanness</td>
<td>3/10</td>
</tr>
<tr>
<td>Results of training match</td>
<td>Test Subject kills: 10</td>
</tr>
<tr>
<td></td>
<td>Enemy kills: 1</td>
</tr>
<tr>
<td>Results of testing match</td>
<td>CONGO bot kills: 1</td>
</tr>
<tr>
<td></td>
<td>Enemy kills: 10</td>
</tr>
<tr>
<td>ACEbot match outcome</td>
<td>Player kills: 10</td>
</tr>
<tr>
<td></td>
<td>ACEbot kills: 6</td>
</tr>
<tr>
<td>ACEbot humanness</td>
<td>3/10</td>
</tr>
<tr>
<td>ACEbot entertainment value</td>
<td>7/10</td>
</tr>
<tr>
<td>CONGO entertainment value</td>
<td>7/10</td>
</tr>
</tbody>
</table>
### Table 27: Test 2 MSE - Echo

<table>
<thead>
<tr>
<th>Attack/Context</th>
<th>MSE Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack 1: Aiming</td>
<td>0.48454</td>
</tr>
<tr>
<td>Attack 2: Gun preference</td>
<td>0.24318</td>
</tr>
<tr>
<td>Attack 3: Firing</td>
<td>0.23174</td>
</tr>
<tr>
<td>Attack 4: Movement</td>
<td>0.79676</td>
</tr>
<tr>
<td>Counter Attack</td>
<td>1.46609</td>
</tr>
<tr>
<td>Just Saw an Enemy</td>
<td>0.21847</td>
</tr>
<tr>
<td>Item Hunting</td>
<td>0.01999</td>
</tr>
<tr>
<td>Enemy In Sight 1: Approach or Not</td>
<td>0.40982</td>
</tr>
<tr>
<td>Enemy In Sight 2: Gun Preference</td>
<td>0.26692</td>
</tr>
<tr>
<td>Enemy In Sight 3: Firing</td>
<td>0.19292</td>
</tr>
<tr>
<td>Enemy In Sight 4: Movement</td>
<td>0.66059</td>
</tr>
</tbody>
</table>

#### 6.5.2.6 Test #2 Discussion

The test subjects were unable to create a bot through the CONGO system that was truly competitive. The main symptom plaguing the CONGO bots was its inability to aim accurately. Addressing the expert test subject again, Figure 5 shows a very high MSE value for the attack context’s movement network. This shows how variable Echo’s movements really were. Despite this, the test results still show that the entertainment value and humanness of the bot were at acceptable values. This was because all bots were able to obtain one or more kills and also displayed many humanlike behaviors. While this was not always competitive for the players, it seems that CONGO system gains its entertainment value through its process as well as the bot it creates. Shown in Figure 7, all players reported a higher entertainment value as compared to the well-respected ACEbot, with the exception of Echo, who rated them equally. This shows that creating a bot using CONGO, and then watching it display actions taught by the player him/herself is significantly entertaining for players.
Figure 5: Test 2 MSE Line Graph

Figure 6: Entertainment-Value Graph
There are some interesting correlations between the humanness values reported and the player’s skill levels. Figure 7 shows that Echo (the expert player) consistently reported the humanness of his bot lower than the other subjects. This is also seen in Delta’s (avid game player) results, although not to the same extent. During the tests, Echo exhibited complex maneuvers that the other test subjects did not. Echo’s maneuvers were also different given the same situation, which is to be expected from an expert player. Moving in such a manner allows them to be less predictable to other human players. For further analysis, it can be seen from the MSE plot in Figure 4 that Echo’s “Attack Movement” neural network had the highest value. This is a clear indication that the results for those NN’s contained more inconsistent patterns than any of the others. Delta also exhibited similar complex movements in Test #1, but from the results of that test, he adjusted to train the bot in Test #2 more reliably.
6.5.2.7 More rigorous NN training

A goal of this research was to keep the time it takes to train a CONGO bot to minimum. However, the weak performance of the trained CONGO bots in Test #2 suggest that better NN training may be called for. To ensure that accuracy was not sacrificed for speed, the number of hidden nodes was increased by 50%, and the maximum number of epochs was doubled. The new NN’s were trained on the same data collected from Test #2 for Charlie. Figure 8: Rigorous Training MSE Graph shows the mean squared error differences between the more rigorous from the original (fast) training. The “Attack1:
Aim” MSE wasn’t improved by rigorous training, although small improvements were made on other NN’s. When this new bot was inserted into the game performance module, there weren’t any noticeable behavior changes made. The next step was to try another training algorithm on this network.

![Figure 8: Rigorous Training MSE Graph](image)

6.5.2.8 Back Propagation

RPROP was used in this thesis and it was successful at speeding up the training process and creating reliable NN’s, with the exception of the critical aiming network. Therefore, the traditional back-propagation training procedure [31] was used on the same data from the previous section. The number of epochs and amount of hidden nodes were kept the same as the “fast” RPROP implementation. Some MSE values decreased, although others were significantly worse. Also, the training time now takes over 30 minutes to
train the same bot that took only two minutes to perform RPROP. Figure 9 shows the difference between all three training scenarios; fast RPROP, rigorous RPROP, and back-prop. Overall, back-prop does not yield results that will improve the overall system.

![Training Algorithm Comparison Graph](image-url)

**Figure 9: Training Algorithm Comparison Graph**
6.5.3 Test #3 Results

This section displays and discusses the results obtained from Test #3.

6.5.3.1 Test Subject Alpha

Table 28: Test #3 Questionnaire – Alpha

<table>
<thead>
<tr>
<th>Item Hunting Humanness Rating</th>
<th>5/10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retreat Humanness Rating</td>
<td>5/10</td>
</tr>
<tr>
<td>Attack Humanness Rating</td>
<td>7/10</td>
</tr>
<tr>
<td>Enemy In Sight Rating</td>
<td>7/10</td>
</tr>
</tbody>
</table>

Results of match
- Test Subject - 4
- Expert Player – 10
- Experienced Player – 5
- Novice Player – 2
- CONGO Bot - 2

6.5.3.2 Test Subject Bravo

Table 29: Test #3 Questionnaire - Bravo

<table>
<thead>
<tr>
<th>Item Hunting Humanness Rating</th>
<th>6/10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retreat Humanness Rating</td>
<td>6/10</td>
</tr>
<tr>
<td>Attack Humanness Rating</td>
<td>5/10</td>
</tr>
<tr>
<td>Enemy In Sight Humanness Rating</td>
<td>7/10</td>
</tr>
</tbody>
</table>

Results of match
- Test Subject - 1
- Expert Player – 10
- Experienced Player – 6
- Novice Player – 1
- CONGO Bot - 2
### 6.5.3.3 Test Subject Charlie

**Table 30: Test #3 Questionnaire - Charlie**

<table>
<thead>
<tr>
<th>Item Hunting Humanness Rating</th>
<th>6/10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retreat Humanness Rating</td>
<td>9/10</td>
</tr>
<tr>
<td>Attack Humanness Rating</td>
<td>4/10</td>
</tr>
<tr>
<td>Enemy In Sight Humanness Rating</td>
<td>7/10</td>
</tr>
</tbody>
</table>

**Results of match**
- Test Subject - 3
- Expert Player – 10
- Experienced Player – 6
- Novice Player – 1
- CONGO Bot - 3

### 6.5.3.4 Test Subject Delta

**Table 31: Test #3 Questionnaire - Delta**

<table>
<thead>
<tr>
<th>Item Hunting Humanness Rating</th>
<th>4/10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retreat Humanness Rating</td>
<td>6/10</td>
</tr>
<tr>
<td>Attack Humanness Rating</td>
<td>4/10</td>
</tr>
<tr>
<td>Enemy In Sight Humanness Rating</td>
<td>5/10</td>
</tr>
</tbody>
</table>

**Results of match**
- Test Subject - 7
- Expert Player – 10
- Experienced Player – 4
- Novice Player – 0
- CONGO Bot - 2

### 6.5.3.5 Test Subject Echo

**Table 32: Test #3 Questionnaire - Echo**

<table>
<thead>
<tr>
<th>Item Hunting Humanness Rating</th>
<th>6/10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retreat Humanness Rating</td>
<td>3/10</td>
</tr>
<tr>
<td>Attack Humanness Rating</td>
<td>2/10</td>
</tr>
<tr>
<td>Enemy In Sight Humanness Rating</td>
<td>2/10</td>
</tr>
</tbody>
</table>

**Results of match**
- Test Subject - 10
- Experienced Player – 4
- Novice Player – 1
- CONGO Bot - 2
6.5.3.6 Test #3 Discussion

It was originally hypothesized that bots created using the CONGO implementation would perform poorly in a multiplayer environment. The data collected and displayed in Figure 8 refutes this idea to some extent. The graph shows that the bot’s humanness was only marginally lowered in Test #3. This seems to be attributed to the fast pace of game play with Quake 2. The bot was put in situations where multiple people were attacking it, and it reacted and attacked one of the enemies. Situations such as these are resolved very quickly by either having all players but one get killed, or by having some players escape. This means that if CONGO were to display poor behavior, the situation doesn’t last long enough to be noticeable. One last trend is that the data consistently showed the CONGO bot’s to be slightly above a novice player. This is able to be determined because the same three players; Foxtrot, Golf and Hotel, joined all five Test #3 matches. Therefore, it was easy to see where the bot’s skill ranked among them.

Figure 10: Average Humanness Ratings
### Table 33: Summary of Questionnaire Data

<table>
<thead>
<tr>
<th>Questionnaire Category</th>
<th>Alpha</th>
<th>Beta</th>
<th>Charlie</th>
<th>Delta</th>
<th>Echo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skill Level</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>Test 1: Humanness</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>3</td>
<td>6.5</td>
</tr>
<tr>
<td>Test 2: Item Hunting Humanness</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Test 2: Retreat Humanness</td>
<td>5</td>
<td>8</td>
<td>8</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>Test 2: Attack Humanness</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Test 2: Enemy-in-sight Humanness</td>
<td>7</td>
<td>7</td>
<td>10</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Test 3: Item Hunting Humanness</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Test 3: Retreat Humanness</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Test 3: Attack Humanness</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Test 3: Enemy In Sight Humanness</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>ACEbot Humanness</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>ACEbot entertainment value</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>CONGO entertainment value</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

### 6.6 Testing Summary

The results gathered proves the hypothesis, although interesting insights were obtained regarding the entertainment value of the CONGO system. First, Test #1 showed the potential of the attack context in a non-hostile situation. The trained bots 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, the process of making a bot and watching it perform the way you intended is entertaining it itself.
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 NN’s. Using RPROP keeps the training time low which is important to preserve the entertainment value.

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.
CHAPTER 7: SUMMARY, CONCLUSION & FUTURE WORK

This chapter summarizes the research by readdressing the problem statements made in Chapter 3. The solutions to the problems are listed along with their effectiveness. Conclusions are then drawn from the testing done in Chapter 6. The last section proposes future work that could extend the work described in this thesis.

7.1 Summary

Overall, this work extended the research done by Sidani [32] in these ways:

- More complex simulation environment
- More complex human behaviors to observe
- Offered value to the gaming industry

Sidani’s IASKNOT system had success in capturing implicit human behaviors by modularizing observation data and applying neural networks to each module. A simple traffic light simulation was used with IASKNOT to provide a controlled environment for the observation system. CONGO uses a more advanced modeling paradigm, CxBR, to contextualize observation data from the FPS game Quake II.

In Chapter 3, the problems that this body of work addressed were stated. This section reviews these statements and discusses how they were solved.
7.1.1 Re-usable

In order for the game industry to benefit from this research, a system with re-usable characteristics must be contributed. This was addressed by creating a general paradigm that can be implemented across multiple games. A CONtextual Game Observation system (CONGO) was designed and implemented. FPS games have a similar structure, as such, their contexts are similar to the implementation in this thesis.

7.1.2 Ensure Playability

Playability is ensured by using methods that increase the humanness of the NPC. In hopes to preserve or increase entertainment value, CONGO uses a technique based on the idea of extensible AI, which is when a game allows players to modify the behaviors of the NPC. With CONGO, players are given the ability to train the AI of their bot by acting out how their bot should act. With minimal training times, a player can have a fully trained bot in a matter of minutes.

To create a more humanlike NPC, two AI methodologies are synergistically combined to create a learning from observation system:

- Context Based Reasoning (CxBR)
- Neural Networks

7.1.3 Knowledge Acquisition for CxBR

The base set of contexts was established by observing Quake 2 expert players play the game and break down the actions they performed into contexts. This set consists of:
• Item Hunting Context
• Attack Context
• Retreat Context
• Counter-Attack Context
• Enemy in Sight Context
• Just Saw an Enemy Context

There were some reactive behaviors that did not fit the normal context specifications. These are behaviors that span for a only brief period of time. For these behaviors, there are two contexts that were made to capture a single reaction that a player has. Those two “reactive contexts” are: Just Saw an Enemy, and Counter-Attack. These contexts forced to be active for two seconds, they return control to the engine.

7.1.4 Choose Appropriate Neural Network

Previous research in this domain has shown that multi-layer perceptron performed the best given that they were trained sufficiently [10]. A requirement for this implementation called for a speedy training process. This was because training times are directly equated to loading times in games, which carries a negative connotation. The RPROP training algorithm is a modified back propagation technique that significantly reduces the time it takes to train a network, without sacrificing much accuracy.

It was shown in previous works that humans do not make decisions based on the current situation alone [26]. Time delay neural networks are used to supply CONGO
with the ability to reason based on previous states. Overall the system takes under two minutes to train all NN’s required for a complete bot.

7.1.5 Developing a Means to Test Humanness and Entertainment Value

The testing procedure was composed of three tests that grow incrementally harder for the bot to perform well. The first evaluates the attack context in controlled situations. For this, the player is not concerned about the enemy returning fire. The player only needs to teach the bot the necessary behavior to kill a “running” enemy in different locations.

The second test is designed to prove the entire hypothesis of this thesis. The player must train the bot while playing against another human player. Once the bot is trained, the test subject then plays against it. 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. A well-known scripted bot known as the ACEbot is set up to play against the test subjects in order provide a reference of which to compare the CONGO system.

The third test was thought to be the test that would show the bot’s limitations. This was set up by having more than one enemy in the game against the bot as well as each other. The data gathered from this test shows that not much humanness was lost in this test. These results are thought to stem from the fast-paced game play, in that a situation where the bot would perform badly only happens for a very short while.
7.2 Conclusion

The hypothesis of this work states:

*Generating player models from observation using a combination of context based reasoning and neural networks will produce human-like behavior and can enhance the entertainment value of a game.*

The experiments performed support the hypothesis completely. There were, however, results that were not foreseen. It was shown that the test subjects all created bots that were at a skill of novice, or slightly better. Despite this, the entertainment value of the system was consistently rated higher than that of the ACEbot. This rating takes into account that CONGO is not just a bot, but it is an extensible AI technique that allows players to creatively make bots to their liking.

A notable observation from the testing is that most test subjects would lose sight of the fact that they were training a bot. For example, the player would do senseless funny things to show off their skill, or something to that effect. The observation system never turns off, therefore it is always watching. These actions are recorded and potentially become patterns that confuse the neural network training algorithm which increases the mean-squared error. Since this type of game play (training your own bot) is still new, it is likely that players need to learn proper tactics to better train the CONGO bots.

An interesting discovery was found regarding a tight relationship between the player’s weapon and aiming. The aiming network does not take into account what weapon the player is holding. This will increase the number of inputs of the network by a
factor of the number of possible weapons and the number of time delays used. This would increase the time for the training process to complete, although it may make new distinctions on how to aim more effectively for different weapons.

7.2.1 Implementing CONGO for another Game

This section will help guide the reader with helpful advice for implementing CONGO into another game. The most convenient situation is to use another FPS game, although other game genres can also make use of CONGO’s style of game play.

7.2.1.1 FPS Implementation

The largest implementation issue encountered was the re-use of ACEbot’s path-finding functionality. When CONGO is to be implemented into another FPS game, it should be accompanied by a reliable set of path-finding functions. Even if a context’s neural network makes an intelligent decision, the path-finding scripts can make it seem un-intelligent by carrying out the decision out poorly.

A benefit to using an FPS is that you can keep a similar context structure as presented in this thesis. Although, a good idea would be to an expert at the game prepare a list specific to the new game at hand. Another game may present new contexts to be added.
7.2.1.2 Teamwork

One implementation for teamwork wouldn’t require any modifications to the CxBR engine. First of all, the same contexts needed for an individual bot would also be necessary. Next, it should be decided which contexts should make use of teamwork. These contexts should then have a sub-context that inherits any functionality from its major context but it will collect its own set of observation data. The last task is to create a sentinel rule to activate the teamwork sub-context.

7.2.1.3 Other Game Genres

Implementing CONGO into another game genre would require acquisition of the contexts and transitions required for the game. Once this has been completed, the next important task will be to contextualize the output patterns for each context. These tasks should make use of domain experts for reference.

7.3 Future Work

Adding humanness into games is still a relatively new idea, and is being adopted slowly by the gaming industry. The most obvious reason for this is the difficulty to assure that a trained agent will not perform in a manner that makes the game less fun. Here are some ideas that could be used to extend this work.
7.3.1 Realistic Pathfinding

The path finding technique used in this thesis does not have any basis for making an agent appear human-like. It is simply an algorithm that solves shortest path problems. Graham [33] attempts to create realistic movement in a gaming environment by using NN’s and GA’s to enhance traditional techniques. Because path finding was scripted in this implementation of CONGO, it means that it would have to be re-scripted for other implementations. If Graham’s work were to be integrated into CONGO, it would allow implementations proceeding to not require the use of domain specific scripted path finding techniques.

7.3.2 Online Learning

There are online learning networks that could be implemented using feedback from the environment to improve the bot as it plays. This would be helpful to aid the difficult process of training an accurate aiming network. Also, it would add another layer of unpredictability to the bot, which would make it seem more human-like in its decisions.

7.3.3 Implement CONGO into a completely client-side bot

The current implementation requires that the bot be running as the server, because modifications were made to both the server and the client. It would be beneficial to implement the bot completely into a client for creating new modes of game play. One such mode could be to have two player’s trained bots fight each other. Another mode could be to construct a team of bots, and play against another human and their team of...
bots. In this mode, players could get creative and make a team of bots that all contribute to the team in different ways.

7.3.4 Clustering Validation

Youngblood used a clustering technique to determine to an extent if a bot was acting human [34]. Another way of doing this experiment would be to have a number of other traditional Quake II bots also get clustered to see if any bot clusters form. If indeed that is successful then the agent in question could then be clustered. Now with a bot cluster(s) and a human cluster(s), it can be determined with more accuracy how human an agent is, relative bot and human clusters.
REFERENCES


[9] R. Evans, "AI in Games: A personal View,


