Proceedings of the 2007 IEEE Symposium on Artificial Life (CI-ALife 2007)

Evolving a Better Adversary: A Case Study in a German Castle

Daniel S. McFarlin (*Member, IEEE)* and Peter M. Todd

***Abstract*– Mainstream video games are increasingly benefiting from more sophisticated adversarial artificial intelligences. The quality of these synthetic opponents is becoming a significant competitive advantage that was once exclusively reserved for graphics. This new generation of synthetic opponent relies on dynamic planning systems such as STRIPS to realize realistic and challenging adversarial behavior. Such systems have been embraced by game developers as they provide for transparent representation of agent state and behavior, have low CPU utilization and are available in toolkit form. Concurrently, the dramatic proliferation of parallel computational units in modern hardware architectures is also facilitating the use of connectionist models of artificial intelligence in gaming. However, significant barriers such as neural network training set generation and turnaround time for neural network development have inhibited widespread adoption of such techniques. To overcome these barriers, we present an infrastructure that automates neural network development through the use of a genetic algorithm to evolve the behavioral training set of an adversarial artificial intelligence. The infrastructure uses an existing game, Wolfenstein 3D, as a simulation environment. We compare the effectiveness of the neural network generated by this system against a manually constructed neural network and the original game AI. All three models are pitted against human players.**

***Keywords*: artificial intelligence, games, genetic algorithms, neural network applications.**

I. Introduction

Adversarial Artificial Intelligence (AAI), the simulation of human-like opponents in a domain specific context, has been a cornerstone of AI research since the inception of the field. One of the original aspirations of the field was to create a chess playing program that could routinely trounce grandmasters. The original "Proposal For the Dartmouth Summer Research Project On Artificial Intelligence" contained such a goal [1]. Algorithmically, devising such a chess playing AI proved to be tractable. The initial component, the minimax algorithm, was developed and proven by John Von Neumann in the mid-40's [2]. Combined with alpha-beta pruning, developed by Claude Shannon, chess playing programs have gone on to challenge and best the world's finest human player [3]. Achieving that goal in practice, however, had to wait nearly 50 years for computational resources to mature to the point where the hardware could sustain and process the massive search spaces required to win this two-player, zerosum, transparent game [3].

Modern AAI theoreticians and implementers long for the days of minimax, alpha-beta pruning and (relatively) straightforward games like chess. The modern electronic adversary now inhabits a rich, 3-D world where the gamestate space dwarfs that of the "humble" chessboard. Where there were once two players (one electronic, one human) there are now an arbitrary number of human and electronic opponents. Added to this is increasing demand for realistic, innovative and doctrinally-sound adversaries. Note that realism in modern AAI may include "artificial stupidity" as one of its distinguishing characteristics [4]. This is a far cry from the near-perfect chess-playing automata of old. However, just as the original, algorithmically sound chess playing programs were hamstrung by a lack of computational resources (both processing speed and memory storage capacity), so too is modern AAI in its most bleeding-edge incarnations: military simulators and first-person-shooter (FPS) video games. These time-constrained applications must (by virtue of their design goals and intended purposes) feature realistic AAI that now must compete with a host of other processes and functions for vital CPU time and memory storage [5].

The challenge for the modern AAI in military simulators/FPS games can be summed up from the perspective that academic AI models are too general to be specifically useful whereas game implementer models are too specific to be generally useful [6]. There is even some debate in the upper echelons of the industry whether substantial investment in next-generation (read: academically developed model-based) AAI is worth the effort. John Carmack, the legendary game designer who developed the engines for Wolfenstein 3D, the Doom franchise and the Quake franchise, has recently stated his view that AAI is mostly "a matter of scripting" and that further investment in more advanced forms of AAI is generally a "waste of time." It should be noted that the majority of top-tier AAI implementers vehemently disagree; Carmack's latest effort, Doom 3, was much lampooned for its laughably bad AAI [7]. Despite Carmack's statements to the contrary, the significant utility of advanced AAI models is generally agreed upon.

Manuscript received October 31, 2006.

This work was supported in part by the Indiana University-Bloomington Cognitive Science Department's Summer Research Grant and the Hutton Honors College Summer Research Grant.

Daniel S. McFarlin (dmcfarli@cs.indiana.edu) is an undergraduate pursuing a dual BS in Computer and Cognitive Science at Indiana University, Bloomington, IN 47405 USA

Peter M. Todd (pmtodd@indiana.edu) is a professor of Informatics, Cognitive Science, and Psychology at Indiana University, Bloomington.

Connectionist and production-rule models developed in academia are of enormous theoretical and practical importance in many areas of cognitive modeling and simulation. Within the context of the academy, such models are sufficiently fast (though not generally run in real-time) and often constrained to very specific subdomains to ensure that they are computationally feasible and sufficiently easy to analyze and explain [8]. The explanatory powers of these models and their connection to human cognition would seem to suggest them heavily for roles as AAI in games and military simulations. The challenged is to find ways of constructing such systems so that they reliably produce the desired adversarial behaviors. Neural networks (NN's), the pre-eminent connectionist model, provide for a robust, generalized and adaptive mechanism upon which to realize AAI behaviors. Additionally, there is a substantial precedent for the use of NN's in agent-based behavioral models of artificial life [9].

Rather than implement an effort-intensive Good-Old-Fashioned AI method of programming AAI behaviors, we turn here to the use of artificial life methods for discovering adaptive behavior in a more automated manner. In particular, we employ genetic algorithms (GA's) to evolve training patterns for neural networks that will help control the AAI behavior. GA's have long been applied to the evolutionary discovery of particular effective NN's in various domains [10]. More recently, they have also been applied to the discovery of creative behaviors for characters in games [11]. In the rest of this paper, we demonstrate an empirical comparison of two different such methods of automating AAI development, and show that they can improve game-play relative to the AAI built into one existing popular game.

II. The Problem of Training Adversaries

Modern video games are benefiting from Moore's Law in a number of ways. Video game graphics are enjoying radical improvements in resolution, realism and availability courtesy of ever improving graphics processing units. Modern CPUs are thus relieved of the burden of performing both graphics and game engine calculations. As a consequence, game engines are increasingly incorporating new features that were previously computationally unattainable. Physics has emerged as a new feature of modern games thanks to a multitude of commercial and academic toolkits. Dedicated physics processors contained on daughterboards are now available to average consumers. Game designers are also integrating highly sophisticated sound engines that employ a variety of psycho-acoustic properties to create immersive and realistic sound environments.

Finally, the artificial intelligence in games has gained greater prominence as a competitive feature. F.E.A.R., the 2005 Game of the Year, won in no small part due to its formidable adversarial artificial intelligence (AAI) [12] – [15]. In order to satisfy performance requirements and other constraints, F.E.A.R. relied on a version of the STRIPS planner to enable the AAI entities to engage in individual and collective dynamic planning. Using A*, the planner formulated a sequence of behaviors that satisfied a least cost plan. The formulation of this sequence was amortized over several game engine timesteps to ensure consistent game performance. Despite these constraints, the AAIs exhibited a remarkable degree of improvisation, adaptation and coordination.

A planning system, rather than some other AI approach, was employed for several reasons. The most compelling reason was ease of modification and debugging. A level designer could readily inspect the formulated plan, modify game environment elements to influence an alternative plan, and verify that a subsequently generated plan conformed to the level design requirements. Decreasing the computational expense was another reason. The planner could still function while only inflicting a 1% load on the CPU. Nevertheless, the latency requirements were such that only six game entities at a time could engage in concurrent planning.

Modern and emerging hardware architectures offer solutions to the computational expense challenge. The Xbox 360 contains a triple-core CPU in which each core is dedicated to either graphics, game engine, or a miscellaneous operation such as AI, physics or sound [16]. The forthcoming Cell architecture from IBM has even greater computational power and a higher degree of parallelism [17]. Once computational overhead is no longer a significant constraint, the remaining barriers to potentially more effective connectionist AAI models are those that interact with the design, implementation and maintenance of such models.

A. The Need for Adaptive Automated Network Training

One of the most significant barriers to connectionist models in game AAI is the initial configuration of the training set. Formulating the input tuples requires a deep understanding of the representative input space, while generating corresponding output values requires a full appreciation for desired agent behavior. This combined task consumes a considerable amount of time in terms of initial formulation, validation and implementation. Additionally, the desired behavior for a given input tuple may evolve to meet new performance and difficulty requirements. Environmental constraints, such as level size, may further affect such behavior.

Modern game development is a dynamic process in which such requirements are in a constant state of flux up until the final ship date. Constant tweaking is an integral part of this process. Consequently, quick turnaround, maintainability and predictability are the most desirable characteristics for AAI development. Unfortunately, these characteristics stand in general opposition to connectionist and other sub-symbolic methods. Encouraging adoption of connectionist models entails building infrastructure to automate the delicate and time consuming connectionist development process while preserving the benefits inherent to such models.

B. Two Connectionist Models

The following sections describe two different approaches to designing connectionist models of AAI. In the first section, we describe the development of a neural network in which the entire training set is developed by a human domain expert. We refer to this neural network as the human-guided neural network (HGNN). This neural network architecture and its supporting infrastructure form the basis for the subsequent development of a semi-automated neural network evolution system using a genetic algorithm-based simulation. We refer to this model as the genetic algorithm-guided neural network (GAGNN). These models are then compared, via play-testing, to the original finite-state-machine (OFSM) model that shipped with the original Wolfenstein 3D game.

III. INFRASTRUCTURE

A. Wolfenstein 3D

The initial connectionist approach (HGNN) combined a rule system in the form of a probability-based finite-state-machine with a neural network that determines state transitions The source code is available on request. Understanding the architecture first requires some understanding of the game. Wolfenstein 3D (Wolf3D) is a 2D first person shooter (FPS) released in 1992 for MS-DOS [18]. (The illusion of 3D objects is provided by aspect permutations of 2D images called sprites in comparison to true 3D models in modern games.) It is written primarily in C and x86 assembly language. A revolutionary game, it effectively started the FPS craze that persists to this day. In the game, the player moves through a fairly uniform level of rooms and corridors reach an elevator that ascends to the next level. Various types of human and non-human adversaries contest the player's progress. The player is armed with four possible weapons: a knife, a pistol, a sub-machinegun and a machinegun. The player's health ranges from 0-100 as does the player's ammunition level (the ammunition is interchangeable among all firearms).

The most common enemy in the game is a lowly human guard (see **Figure 1)** armed with a pistol containing an infinite amount of ammunition. All of the AAIs in Wolf3D rely on deterministic finite-state-machines (FSM) for their behavior [19]. The transitions from one state to another are deterministic. Some randomness is incorporated in the AAI targeting accuracy and enemy detection. In some cases, the AAI will not detect the player until the player moves to within a certain proximity. In other cases, the AAI will actively look for the player and detection will be quite rapid.

Figure 1
In-game screenshot of the test-level and AAI

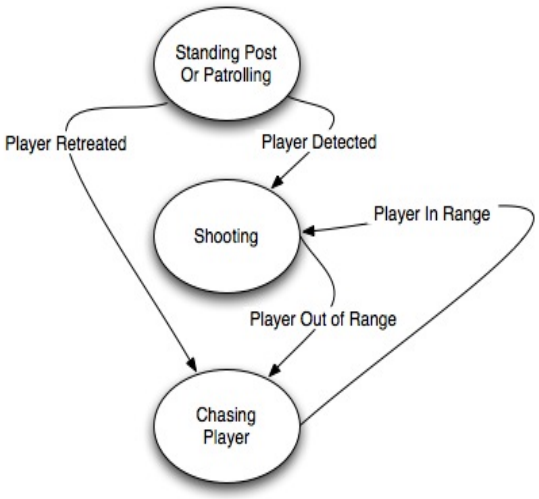

Figure 2
Original State Diagram for Guard Unit

From a behavioral standpoint, the guard in the original game could only be described as pitiful, though the supposedly more advanced enemies are hardly any better. The state diagram for the FSM used to control the guard is shown in **Figure 2**. These AAIs generally charge or chase the player, only stopping at predictable points along the pursuit route to fire. There is little notion of self-preservation or tactics. Our enhancements to the AAI involved three stages: doctrinal changes, probability influences, and statistically influenced state transitions. Finally, the FSM transitions were changed to rely on a neural network.

The new behaviors added during doctrinal enhancement consisted of evasive shooting behaviors called "strafe-and-shoot" and tactical retreating behaviors. The former behavior entails the AAI constantly and randomly moving in directions that are perpendicular to the player. The AAI pauses briefly to fire at the player. At no time during the maneuver does the

AAI reduce the distance to the player. Strafe-and-shoot is effectively defensive in nature as it combines evasion with a slightly impaired offensive capability as accuracy is penalized by the constant movement. The tactic of greatest defensive value is the tactical retreat. This retreat involves moving away from the player in a staggered manner. Movement away from the player is interrupted by firing. The AAI is effectively covering its own retreat and hopefully forcing the player to remain at a distance that prevents a close pursuit.

The next type of AAI enhancement, probability influences, were incorporated primarily into the visual system of the AAI. The AAI now detects the player with a probability that is influenced by the AAI's position, the lighting level and the acoustics of the room and the AAI's proximity to disturbances such as the sound of gunshots. The modified state diagram incorporating the NN is shown in **Figure 3**. The transitions in the figure are influenced by the relative strength of the player compared with the relative strength of the guard.

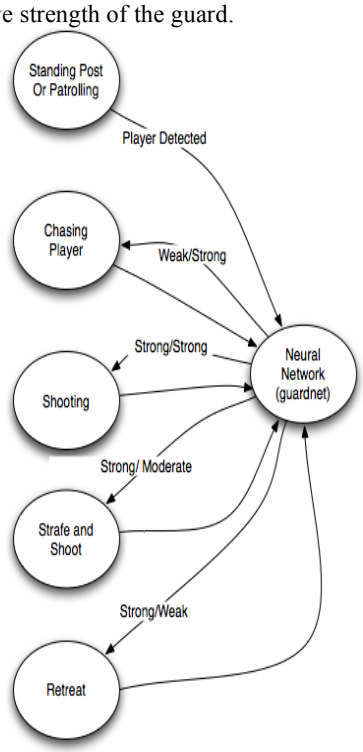

Figure 3
Modified State Diagram For Guard Unit
(Player/Guard)

Lighting and acoustic properties are fixed at compile time. The original Wolf3D code had a probabilistic detection model that influences the accuracy of the AAI's observations of the player such as the player's health, weapons choice and ammunition levels. These observations along with the current health level of the AAI are collectively "fed" into the adversary's neural network. This network is a standalone process that communicates with the Wolf3D executable via the file system. Though not the most efficient inter-process communication mechanism, it is one of the few available in MS-DOS (which Wolf3D must be run in). The neural network itself is written in C++ and compiled on Windows using the MinGW C++ compiler and IDE.

The beginning of every behavior contains a function, `agent_think`, that extracts the gamestate information (subject to the probability influences described above) and writes it to a file. The neural network detects the new file, reads the data and writes a new file containing the behavior that the AAI should next engage in. The neural network contains four input nodes, 10 hidden layer nodes and one output node. The network was trained on forty gamestate-decision pairs. As described earlier, each gamestate is represented by a 4-tuple containing the AAI's current health, the current health of the player, the player's current weapon of choice and the player's ammunition level. The output decision values range from 0 – 3: 0 corresponds to a decision to retreat; 1 is strafe-and-shoot; 2 is stationary shoot (where the AAI comes to a complete halt and makes an aimed shot at the player); and finally, 3 represents advance, in which the AAI charges the player's position. The decisions are arranged in a spectrum from defensive to offensive. The training set was intended to be representative of the gamestate space.

The network was a standard feed-forward network and utilized back-propagation for weight-learning. It required about three hours of training to reach an aggregate global error of less than 10^{-5}. Output from the network was categorized based on a deterministic model that favored aggression in cases where the AAI had previously performed an offensive action and favored defensive actions when the last behavior was defensive. The neural network did not posses any recurrent properties and the rounding behavior occurred in post-processing. Even with optimizations for the x86 CPU architecture, the neural network was still quite processor intensive and slow relative to the timeslice allocated for each AAI's thinking process. As a result, in some cases the neural network's decision would not be calculated in time, forcing the AAI to continue with its current action until the AAI again invoked the `agent_think` function. The delay, however, in player action and AAI reaction is barely noticeable.

Though this model is a significant improvement over the OFSM, it entails a considerable degree of human involvement during development. Reducing human involvement in this construction process is crucial. The next section examines this problem and an implemented solution, after which we compare the performance of the different approaches.

IV. Evolution of An Infrastructure

One important step in automating the connectionist model development process is automatically determining the outputs for the training set input tuples. In standard NN training, these values are supplied explicitly by a developer [20]. The developer bases the outputs on the desired agent behavior. In the context of an FPS, such behavior generally includes offensive and defensive maneuvers in conjunction with weapons employment. Developing effective AAI behavior is

generally a trial-and-error process. This process is influenced by the developer's biases and many of the first iterations may be undermined by preconceived notions of output behaviors that, while sound in theory, fail to translate to effective behaviors in the game environment. The ultimate effectiveness of AAI behavior depends in large part on human player behavior within the confines of the game environment.

This naturally leads to the insight that a neural network's output set can be derived from simulation in the game environment with an opponent. Commercial game development benefits from extensive human play testing. Consequently, human opponent behavior can be drawn on from recordings of play test sessions. These recordings generally reveal common tactics and strategies employed by human players. Traditionally, such recordings are generalized into heuristics that guide refinement of the game environment and AAI [21]. Such heuristics can be as simple as "enter an unexplored room backwards," or complex maneuvers such as firing from cover. These heuristics can be further incorporated into simulated human opponents and set against AAIs in the embryonic state of development. In our approach, AAIs can then be evolved over a series of simulated combat epochs. This approach employs a genetic algorithm to refine the behavior of an AAI set against a simulated human opponent. We first evolve a sequence of behaviors that works well against the simulated human for each possible input situation, and then condense the evolved sequence down to a single behavior to be performed in that situation. The details are as follows.

A. GAGNN Implementation

The genetic algorithm operates on an initial randomly generated population of chromosomes that each encode a sequence of possible AAI behaviors [22]. For each input tuple, specifying the initial state of both combatants (ammunition and health for the human and the AAI), a different initial population of AAI behaviors was generated and the genetic algorithm executed to evolve better ones. The initial population contained forty chromosomes of AAI behavior sequences, with each of the 20 behaviors in the sequence selected at random from a set of predefined primitive AAI behaviors. The behaviors were represented using direct integer encoding. One-hundred evolutionary epochs were run for each separate population, which was sufficient to find the effective behavior sequences. On average, a genetic algorithm run (100 epochs of 40 chromosomes) took 1.36 minutes.

During each epoch for each input tuple, a simulation was conducted in which the AAI responded to the heuristic-based simulated-human opponent by following the sequence of behaviors specified by each chromosome. In other words, the AAI's behavior was "scripted" by the actions specified in each chromosome being evaluated. Each behavior occurred in one timestep of the simulation. The game environment was restricted to a 10x10 unit empty room with the combatants spawning at opposite corners. The simulation ran until the AAI or the "heuristic human" opponent died or the behaviors encoded in the chromosome were exhausted. The heuristic human opponent employed a constant-distance shoot-and-strafe tactic that maximized distance to the AAI. This distance was augmented by firing the heuristic human's weapon after every perpendicular strafing movement. A record was kept of the amount of damage inflicted to the heuristic human opponent by a particular chromosome behavior sequence.This This record was later used in the fitness function. simulation was consistent with the actual game environment in that it restricted the AAI to one behavior per timestep while the heuristic human opponent was allowed two moves per timestep. In other words, the simulated human is allowed to fire and shoot in the same timestep compared to one action in the same timestep performed by the AAI.

After all chromosomes had been executed, the genetic algorithm selected the top twenty most lethal (in terms of damage inflicted) for reproduction. The lethality fitness function has the nice side-effect of encouraging the AAI to balance the need to inflict maximum damage with the need to survive long enough to inflict a significant amount of damage. The monogamous couplings of the top twenty population members were randomly selected and two offspring were produced per coupling. Offspring were produced by a probabilistic combination of simple crossover and mutation. Crossover occurred with a probability of .80 whereas mutation occurred with an independent probability of .05. In the event of mutation, four elements in the chromosome were randomly replaced with a separate, randomly generated AAI behavior. The generated offspring together with their parents formed the new population for the subsequent epoch.

Once all epochs had been executed, the most lethal chromosome was selected from the final population. The arithmetic mean of the direct integer encoded behaviors was computed with rounding dictated by the mode (most frequent value): if the mode was lower than the mean, the mean was rounded down, otherwise up. The mean was truncated in the event of ties in value frequency. In all cases, the final mean value was used as the output value for the supplied input tuple in the neural network training set.

V. METHODS FOR TESTING AAI PERFORMANCE

Evaluating and comparing the performance of the original FSM (OFSM), human-guided neural network (HGNN) and genetic-algorithm-guided neural network (GAGNN) requires a qualitative and quantitative analysis of game play testing that pits a real human opponent against the AAI in the context of the game environment. Wolf3D provides two fairly revealing in-game metrics that are presented to the player at the completion of a level. The metrics are level completion time and current ammunition quantity. Though indirect, these metrics provide a good insight into the quality of the AAI and will be the basis for our comparison.

A. Experimental Setup

The opening level of Wolfenstein 3D was selected as the test level due to the homogeneity of the AAIs present and the ease of level navigation. The level contains ten rooms and seventeen guard units. The medium difficulty settings (which dictates AAI movement speed) was selected. Game play occurred on a 1.8 Ghz Pentium M, 1 GB RAM Dell 800 running Windows XP. An FPS-veteran human player, who was restricted to using the entry-level pistol weapon was required to destroy all enemy units, minimize level completion time, follow the same path for each trial and maximize health and ammunition stores prior to level completion. These requirements are consistent with the reward system in the game which awards points in direct proportion to the amount of health and ammunition present at end-level and the number of enemies destroyed. Points are also awarded based on the proximity of the player's completion time to the "par time" for the level which is generally a nearly unattainably short period of time. High scores translate into additional game lives and access to hidden features, which do not concern us here. The aforementioned level-end metrics were recorded for the human player after the completion of each trial, and 25 non-interleaved trials were conducted for each of the three types of AAI. The subject was able to discern which model he was playing against as each model exhibited an obvious qualitative difference.

B. Results

Table 1 contains the arithmetic means and standard deviations for level-completion time and ammunition present for the human player competing against each model. These data properties were computed using SPSS.

Model	Time (s)	Std. Dev.	Ammo	Std. Dev.
OFSM	117.20	13.28	52.52	5.85
HGNN	169.72	35.10	50.84	5.02
GAGNN	141.04	23.68	48.80	6.49

Table 1: Arithmetic Mean and Standard Deviation for level-completion time and ammunition for human playing against 3 model types.

C. Analysis

The OFSM model AAI is firmly the "easiest" challenge to the human player based on the collected metrics and player interviews. The "par time" for the first level is 90 seconds. In comparison, the mean level-completion time for playing against the OFSM was only 27 seconds greater. Additionally, the player completed the level with the greatest amount of ammunition in reserve. In general, higher level-completion times signify a need for backtracking in order to shore up depleted health by obtaining health packs seen earlier in the level. Smaller ammunition reserves signify a tendency to engage in firefights with the AAI at longer range as lethality is inversely proportional to distance from the target. A human player can exploit his advantage in firing and moving in a single timestep (compared to the two timesteps needed by the AAI) by moving perpendicular to the target AAI (i.e., strafing). This forces the AAI to expend a timestep to orient itself to the human player and then another timestep to fire.

This strafing strategy is generally regarded as the most effective strategy in Wolfenstein 3D which is why it served as the basis for the heuristic human adversary in the GAGNN model. Consequently, the genetic algorithm produced an AAI whose behavior was marked by a bias to defensive maneuvers. In other words, the GAGNN AAI has a tendency to maintain its distance from the player and only typically only fires using a defensive strafe-and-shoot maneuver. Combined with the observed AAI's tendency to retreat to more defensible positions, the lowest mean ammunition reserve value agains the GAGNN AAI indicates that the AAI's defensive posture forced the human player to expend more ammunition to compensate for the long range lethality penalty. Additionally, the GAGNN's long average level-time completion and its middle-rank standard deviation is indicative of the enhanced difficulty relative to OFSM and the greater consistency of opposition quality relative to HGNN. Finally, the arithmetic mean of the genetic-algorithm generated output values for the NN training set in this case was 1.3. This corresponds roughly to a strafe-and-shoot defensive posture.

In contrast, play against the HGNN exhibited the highest standard deviation for level-completion time. This arose from the greater aggression exhibited by AAI in the game. The average human-guided output value (the output value associated with a particular input tuple by the human domain expert) for the neural network was 2.5 which corresponds roughly to stationary shoot in which the AAI remains stationary to improve shooting accuracy at the expense of defensive protection. The highest standard deviation is also attributable to the tendency of the AAI to engage in kamikaze tactics after sustaining near fatal levels of damage. As a last resort, the AAI charges the player in the hopes of inflicting the greatest amount of damage before being destroyed. This tactic is not consistently effective as it depends on the AAI's initial distance to the human player. The initial distance varies across AAI's in the level depending on a random position that is determined every time the level is loaded. The kamikaze behavior influences the ammunition usage as well. By charging, the AAI allows the human opponent to expend less ammunition as the lethality increases when the AAI closes the gap. Conversely, the AAI may be in a position to inflict some grievous damage before being destroyed leading to a high degree of variability in game play. While a high degree of variability may be desirable to human players in the final product, consistency in AAI behavior is a valuable asset to level designers and game developers who must reconcile the maintainability issues of variable AAI behavior with the need to integrate such behavior into a cohesive game atmosphere and environment. Thus overall, for effective game adversary development, the more consistent and easier-to-produce (due to automation) GAGNN approach may be preferable.

VI. Conclusions and Future Work

In this paper, we have demonstrated an infrastructure that is capable of automating a significant and resource intensive stage in the development of connectionist models of AAI. By utilizing a tool from artificial life, the genetic algorithm, to explore a behavior space in the simulated game environment, developers are able to utilize the game itself as a testbed for evolving challenging AAIs. The AAIs exhibit behavior that is intelligent, effective and engaging. The GAGNN model performs comparably to domain expert guided neural networks (HGNN) while incurring only a fraction of the development cost thanks to the use of an automated "heuristic human" opponent for evaluating AAI strategies in the actual gaming environment.

Future developments of this approach should embrace more sophisticated fitness measures, such as difficulty metrics that measure whether an AAI's play will be "good for beginners." More advanced gaming engines should be adopted as they become publicly available. The Unreal engine has been heavily adopted for AAI research as it offers true 3D environments. Massively-Multiplayer Online Games (MMOG) such as "World of Warcraft" ,which contain hundreds of thousands of players participating simultaneously, can provide ample opportunities for fitness evaluation against a staggering variety of human opponents. MMOGs also provide a fertile testing ground for the underdeveloped area of cooperative artificial intelligence. It is hoped that the further development of these approaches will facilitate wider adoption of connectionist models in mainstream gaming, which will in turn provide for more accessible and robust AAI research environments based on these games.

Acknowledgments

We are indebted to the advice and insight of Rob Goldstone, Drew Hendrickson and the anonymous reviewers.

References

[1] J. McCarthy, J. Minsky, Rochester, Shannon, "A Proposal For the Dartmouth Summer Research Project on Artificial Intelligence," 1955.
[2] J. Von Neumann, "Theory of Games and Economic Behavior," Princeton, NJ: Princeton University Press, 1944.
[3] J. Schaeffer, "Gamut of Games," in *AI Magazine*, fall issue, 2001
[4] L. Linden, "The Use of Artificial Intelligence in the Computer Game Industry," in *AI Game Programming Wisdom*, fall issue, 2001.
[5] J. Carmack, "Opening Statements," Quakecon, www.gamespy.com/articles/641/641662p3.html, 2005.
[6] D. Isla, "Dude, Where's My Warthog? From Pathfinding to General Spatial Competence," in proceedings of AIIDE Conference, 2005.
[7] P. Tozour, "Spot the Irony," blog entry, www.ai-blog.net, 2005.
[8] A. Hoffman, "On the Computational Limitations of Neural Network Architectures," Technical Report, University of New South Wales Computer Science Dept, 2005.
[9] R. Wray and J. Laird, "Synthetic Adversaries for Urban Combat Training," www.soartech.com, 2004.
[10] G.F. Miller, P.M. Todd, and S.U. Hedge, "Designing Neural Networks Using Genetic Algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms,* San Mateo, CA: Morgan Kaufmann, (pp. 379-384), 1989.
[11] J. Hong and S. Cho, "Evolution of Emergent Behaviors for Shooting Game Characters in Robocode," in *Proceedings of Evolutionary Computation*, vol. 1, 2004, pp. 634–638.
[12] J. Orkin, "3 States And A Plan: The AI of F.E.A.R.," in proceedings of Game Developers Conference, 2006.
[13] J. Orkin, "Agent Architecture Considerations for Real-Time Planning in Games," in proceedings of Artificial Intelligence and Interactive Entertainment Conference, 2005.
[14] J. Orkin, " Towards Real-Time Planning in Games," in proceedings of AAAI Game AI Workshop, 2005.
[15] J. Orkin, "Applying Blackboard Systems to FPS," in proceedings of Digital Media Collaboratory, UT Austin: 2003.
[16] J. Stokes, "Inside the Xbox 360, part I: procedural synthesis and dynamic worlds," in *Ars Technica*, 2005.
[17] J. Stokes, "Introducing the IBM/Sony/Toshiba Cell Processor," in *Ars Technica,* 2005.
[18] J. Romero and T. Hall, "Wolfenstein 3D," Dallas, Texas: id Software, 1992.
[19] J. Carmack, "Wolfenstein 3D: Source Code," Dallas, Texas: id Software, 1995.
[20] M. H. Cheong, "Functional Programming and 3D Games," technical report, University of New South Wales Computer Science Dept., 2005.
[21] P. Spronk and E. Postma, "Online Adaptation of Game Opponent AI In Simulation and Practice," *Game-On*, 2005.
[22] M. Mitchell, "An Introduction to Genetic Algorithms," Cambridge, MA: MIT Press, 1998.