AI in the Game: Feasibility of Artificial Neural Networks in Real-Time Strategy Game Development

Matthew Paul
paulmr@goldmail.etsu.edu
East Tennessee State University

Abstract

Adding challenging artificial intelligence to real-time strategy games can increase revenue by extending the game's life span. An artificial neural network, or artificial brain, demonstrates human-like ability to react and optimize strategy, increasing challenge. While the AI increases challenge, neural nets are not cost effective for game studios to use.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. Introduction

Real-Time Strategy (RTS) games are an evolution of entertainment derived from board games and strategy games in general. The game industry had a total revenue of 20.77 billion USD in 2012[1], almost double the 10.83 billion USD generated by the movie industry[2]. Many newer video games have used micro-transactions, small in-game purchases, to increase the longevity of the games being produced. According to Jorgensen, micro-transactions are a valuable source of revenue because of availability and high profit margin[3]. Increasing the time a player spends playing such a game increases the revenue generated without requiring the increased costs of game development. Graphics have been used to increase a game's desirability in the past, but have reached a period of diminishing returns. Other techniques such as advanced Artificial Intelligence (AI) are now used to provide the complexity in games[4].

II. Literature Review

The complexity of RTS AI has fueled research into the field[5]. Being partially observable with unknown opponent strategy makes decent AI construction challenging. The enormous action space further complicates AI, as some AI techniques can only work across a small action space.

A. Evolving neural networks through augmenting topologies

Coevolutionary algorithms, applying basic genetic principles to computing, was used provide online adaptability[6]. Markov chains [7], plan recognition [8], dynamic scripting [9], and offline strategy generation [10] have all been applied to game AI successfully.

These studies demonstrate differing techniques for RTS strategy. Artificial Neural Networks (ANNs) have also been used to develop strategy for video games. One example is the rtNEAT system, which developed AI for a first-person shooter game [11].

ANNs solve problems by mimicking structures found in biological brains, mainly neurons and synapses. Stanley demonstrates that Neuroevolutions (NEs), using genetic algorithms to evolve neural networks, perform well in complex environments[12]. A system for evolving topologies, the layout of the neurons in the agent, is shown to be effective in NeuroEvolution of Augmenting Topologies (NEAT). Various problems significant to NE are examined, namely meaningful crossover, speciation, and minimal topology creation.

1) Meaningful Crossover: One technique used to provide genetic diversity is crossover. When two agents are selected to reproduce, the genes of each "parent" have a chance of breaking off of one set of genes and attaching to the opposite set. A successor's genes are then chosen at random from the resulting sets. The chosen genes are used to populate the next generation.

Traditionally, enabling meaningful crossover in ANNs that evolve topologies is difficult. Most Genetic Algorithms (GAs) do not evolve ANN with variable topologies because the crossing over of genes often destroys the functionality provided by the topology. NEAT addresses this by examining Nature's solution, which is *homology*. Homology uses alleles, which are one half of a gene that determines traits in the ANN. Two genes show homology if they are alleles of the same trait[12]. NEAT implements homology through historical markings. Historical markers are unique to each gene and are built up from the historical markers of every genes ancestors. NEAT implements meaningful crossover in a variable topology ANN through these historical markers.

2) Speciation: Speciation is the protection of populations from being replaced by the fittest members of society. Getting stuck on local maxima is a problem that must be solved by well-implemented GAs. When an agent is evolved to include new neurons, the fitness characteristic will likely rate that agent lower than others who have had time to optimize their topologies. Without protecting that agent through speciation, the GA will cull the agent without giving it enough time to optimize its newly evolved neurons.

Speciation accomplishes this protection by grouping agents into niches of similar characteristics. Each agent competes with only the other agents contained within its "species." If speciation does not account for dominance of agents, it will prevent optimization by restricting the dominant species from thriving.

NEAT handles speciation by examining the historical markers and grouping agents whose topologies are similar. The number of agents in each species is determined by the average fitness of the species' population. If the species is less fit, less space is given to the species for reproduction. This grouping protects agents who have a more optimal topology that is unoptimized but allows species who have had enough time to optimize and continue to be less fit to be weeded out.

3) Minimal Topology Creation: Traditionally, GA concerned with variable topologies introduce variability into the topology by randomly permuting nodes and connections between these nodes. The issue with approach is that complexity is created in the topology and rarely weeded out by the GA. Stanley argues that while such systems find solutions to the tasks, they are rarely minimal topologies[12]. Such a system evolves more slowly due to the extraneous nodes being selected for mutation during evolution. NEAT, rather than using random permutations, starts with the least possible configuration for the task. The GA instead adds additional neurons and connections only when necessary. NEAT also removes species who have additional connections and neurons that did not contribute to the solution.

Stanley states that NEAT is much more responsive to change due to it only adding complexity when necessary. Tests performed using the double pole balancing with velocity method demonstrate that NEAT outperforms conventional ANN by using only 3% of the number of generations required to solve the problem.

Method	Evaluations	Generations	No. Nets
Ev. Programming	307,200	150	2048
Conventional NE	80,000	800	100
SANE	12,600	63	200
ESP	3,800	19	200
NEAT	3,600	24	150

TABLE I
COMPARISON OF PERFORMANCES ON THE DOUBLE POLE BALANCING WITH VELOCITY INFORMATION. THE PROBLEM IS NEITHER TOO HARD OR TOO EASY TO SOLVE WITH ANN. NEAT OUTPERFORMS CONVENTIONAL ANN BY USING 3% OF THE NUMBER OF GENERATIONS TO SOLVE THE PROBLEM.

Stanley demonstrates that NEAT is a technique that solves the issues of effective evolution better than conventional ANNs. Applying the NEAT system and its descendant, rtNEAT, to the RTS system has potential. NEAT handles larger action spaces well and generates the minimal structure needed. These requirements lend themselves well to the RTS environment. By definition, RTS games have large action spaces. The minimal structure would allow the ANN to respond to an opponent's strategy more rapidly than if an unnecessary structure had to be re-trained.

B. Real-time challenge balance in an RTS game using rtNEAT

In the paper "Real-time challenge balance in an RTS game using rtNEAT," the author's Jacob Kaae Olesen, Georgios N. Yannakakis, and John Hallam explore using rtNEAT in a RTS environment. The game *Globulation 2* is the framework for the evolved agents. Issues with evolving agents for RTSs are examined and several are solved. While some issues with the paper are present, the relevance to topic and the supporting authors' presences in the field outweigh the issues.

1) Context: The need for this type of research is due to a lack of agents that "adjust game difficulty according to player skills.[13]" Maintaining difficulty, or *challenge*, is important to the enjoyability of a game. Malone's three factors are *challenge*, *curiosity*, and *fantasy*[14], as cited by Olesen. This paper attempts to resolve one of the factors posited by Malone, thereby increasing enjoyability. There are several other papers referenced by Olesen that affirm appropriate challenge as key to game enjoyability. Although one of the references used was self-referential, five outside sources were used as well.

For this research, *Challenge* was assumed to be the chief factor of playability of a game. An ANN is chosen from machine learning techniques due to the authors' earlier publications demonstrating ANNs effectiveness at challenging the player at various levels[15]. NEAT and rtNEAT is used as the ANN because they have been proven to solve complex problems and have been demonstrated useful in the video game environment.

2) Globulation 2: Globulation 2 (G2) is used for the research due to its focus on macro-management. Like most RTS games, G2 utilizes resources to construct workers, warriors, and buildings. The paper highlights the differences, however, that lend G2 to a viable framework in which to utilize ANNs. Rather than focus on micro-management, such as precise positioning of units, the game operates by using objectives. An objective might tell the workers where to build or the warriors which place to defend or attack.

This focus on macro-management lends itself well to ANNs because of the reduced action-space that is handled by an agent. The paper extends this idea further by saying that an agent closely mimics player's interaction with G2.

3) Challenge Rating for Globulation 2: The paper demonstrates a metric that determines the Challenge Rating (CR) of a certain opponent. Several aspects that play a key role in the *challenge* were determined from the author's RTS knowledge, forum posts, and interviews of "experienced computer gamers." The author states that factors that are

- Number of warriors
- Aggressiveness
- Infrastructure
- Defense infrastructure
- Average training level of warriors
- Number of workers

not related to the opponent, such as map layout, are excluded. The author made these exclusions in order to follow closely with previous research completed([15].

Two of the aspects determined to be challenging were implemented in NEAT, number of warriors and aggressiveness. The agents based on the first two aspects were trained against a standard game agent found in G2 called *Nicowar.* The paper did not state whether the evolved agents were then pitted against human players to determine if the constructed metric correlated with the players' notions of *challenge.*

The difference between the challenge of the ANN and the *Nicowar* agent was used as the fitness metric for NEAT. The CR was calculated using all of the factors mentioned in II-B3. Individual fitness functions were not included for each characteristic.

4) Off-line Learning: The paper uses NEAT to provide the off-line learning used in testing the ANN against *Nicowar*. One issue the author faced was that the original implementation did not minimize the difference between the agent's and *Nicowar*'s CR until the fitness was determined. The author corrected this oversight by randomly sampling the fitness at points in the simulation. This correction resulted in the evolved agents keeping the CR difference much smaller during the simulations.

One problem that exists with evolving the agents offline is that, as the player learns the game, the agent cannot change in real-time to keep the CR difference small.

5) Real-Time Learning: The author uses rtNEAT to evolve the ANN agents in real-time as the game is played. The scenario is set up to avoid elimination of any participants. This change means that no opposition is actually achieved in the simulation.

The paper presents that using rtNEAT to evolve agents for G2 did marginally better than off-line learning. 50 generations were used to evolve the ANN agent. This small number of generations and the lack of interactions provided by only 7 agents are cited as obstacles to improving the Real-Time Learning system.

6) Analysis: The paper seems to involve several critical assumptions. One of the assumptions that was appropriate was the importance of "challenge" in the video game environment. This assumption allowed the author to narrow the focus of the paper and provide depth to the research.

Other assumptions were made that weaken the article. The author assumes that the aspects of challenge chosen for G2 represent adequate challenge to human players. This assumption is based on examination of his own knowledge, reading of forum posts, and interviewing with experienced computer gamers. It is not apparent in the paper whether or not the sources were vetted against the community-at-large or some well-designed objective metric.

The biggest flaw in the article is the choice of a metric measuring CR. While earlier research may have used the same technique, this article fails to cross-verify the algorithm with human players. Human interaction is necessary for this research due to the nature of the topic. An algorithm was designed to represent a person's perception of *challenge* without verifying that the output correlates with reality.

The number of warriors and the aggressiveness were calculated according to fitness characteristics. Specific agents were evolved using NEAT to maximize both aspects. The other four aspects' fitness characteristics are then assumed to be involved in the fitness characteristic. For the final experiment, aggressiveness would have to be narrowly defined to an artificial measure because of the lack of opposition.

Lastly, the author uses GA with a population of 7 and only 50 generations. Most literature, including NEAT's and rtNEAT's specify that many more are necessary in order to begin fitting the problem description. The research had a decent core, but did not handle the execution well enough to merit a continuation of research in this area. Before continuing, this research must be examined with more rigor.

7) Relevance: Initially, the relevance of this paper seemed high. However, through a close reading, the flaws in the approach weaken the relevance to the paper.

The paper demonstrates that NEAT and rtNEAT can be applied the RTS genre. The hypothesis that ANNs are useful in well-defined, limited action spaces is confirmed here as well as in other sources. Any implementation that involves GAs will need to have the correct parameters such as population and number of generations before it can be considered well-implemented.

Any function that attempts to correlate highly with human perception must be cross-verified with actual human interaction. Also, any fitness function must be constructed carefully so as to not allow the fitness characteristic to be a wholly artificial construct devoid of any connections to reality. If a fitness characteristic is constructed with a certain goal in mind, that characteristic must be allowed to perform in a realistic setting.

III. Methodology

The experiment consists of construction of the simulation, integration with the sharpNEAT library, and evaluation of the data generated. The simulation will be constructed as an abstraction to the typical RTS environment. The reduced action space will allow for simplicity in the ANN. The simulation will have an opponent controller that will oppose the sharpNEAT controller. The use of an electronic opponent allows for the simulation to be conducted without interaction with the system and at a speed not possible with human interaction.

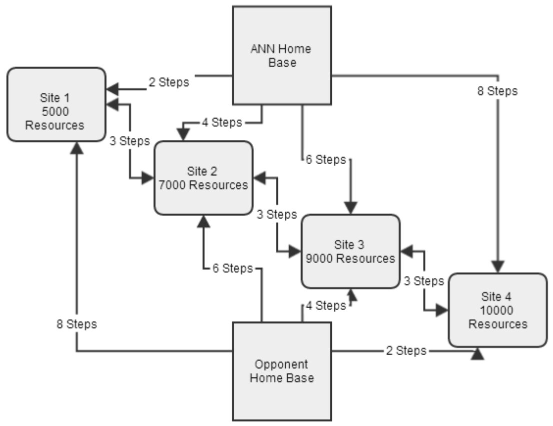
The integration with the sharpNEAT involves creating a translation between the simulation and the system. A "black box" will be created that will handle inputs and outputs. The parameters will be constructed as to best demonstrate the system.

The data generated with the simulation interacting with the test controller and the sharpNEAT controller will be logged for offline examination.

A. Simulation Environment

The simulation represents an abstraction of the standard RTS environment. A simple abstraction enables the ANN to operate more efficiently with the reduced action space. Determining the reasoning behind actions taken by the ANN is easier because patterns created by the ANN do not obscure the outputs. Although the implementation of the simulation does not occur in real-time, the simulation requires decision-making to be made in tandem with the opponent in a fixed amount of time. This abstraction allows the results to be stored as ticks, or discrete spans of time during which actions can occur.

Fig. 1. Representation of simulation.



The simulation models the action space using a "home base" for each player and 4 sites for resources. The player can create units and a resource building and send units from their base to explore the environment. The bases are invulnerable and are not able to be attacked. This restriction allows the ANN to focus on maximizing the amount of resources netted without concern for base defenses.

The order, as shown in figure 2, was chosen in order to better approximate the RTS game. The first step in a simulation tick is to let both the player and the enemy think. This thinking involves determining which units to build, which units to move, and whether or not to stop the simulation. The enemy AI is not allowed to quit. This option is reserved for the NEAT player, allowing them to signal that they wish to end the simulation. Allowing this gives the possibility of an ANN quickly gathering resources and then ending. For the fitness function described

```
while simulation is running do
    Let NEAT player think
    Let Enemy player think
    Update the movement queue
    for all sites do
        Battle forces in site
        Collect resources in site
    end for
    Enact NEAT player's intention
    Enact enemy player's intention
    ticksElapsed++
end while
```

Fig. 2. Pseudocode of one tick in a simulation

in equation 1, the score increases the faster the resources are gathered.

Updating the movement queue is placed before both players acting and the battling and resource collection. This order is important to keeping both the ANN and the enemy AI from moving units before the units had a chance to fight. If the movement queue was updated after battle or player actions, a unit who has dropped out of the movement queue could then be placed immediately back into the queue, thereby skipping any fight that was in that site.

Enacting each player's intention needed to be robust. Each tick, a player could attempt to create a unit as well as order any number of units to move. If the player has the resources available in the build pool, the unit is created.

As for unit movement, care was taken in order to not allow units being ordered while traveling between the sites as well as ordering dead units. When selecting units to order to a location, only units that are not in the movement queue are allowed to be ordered.

During battle, some units that were alive when the players were given time to think would be killed before the intentions of the players were enacted. In this case, the input is ignored.

The 4 resource sites are situated at different distances from each base. From the ANN's base, sites 1, 2, 3, and 4 are 2, 4, 6, and 8 ticks away, respectively. This distance means that a unit sent at tick 0 will arrive at site 3 on tick 6. The distances from each site to the opponent's base is 8, 6, 4, and 2 ticks for sites 1, 2, 3, and 4.

Once having reached any site, units are able to travel to other sites adjacent to that site. The time taken is 3 ticks for travel between 2 sites. This time is 1 tick more than if the unit had traveled directly to that site from the base. This restriction encourages the ANN to determine the effective location of a unit when that unit leaves the base. Without this specification, a common strategy would be to send all units through the closest site until they reach the final position.

Each site has a pre-determined number of resources to be gathered. The number ranges from 5,000 in site 1 and increases by 2,000 each successive site until 11,000 resources in site 4. This variance encourages the ANN to determine which site is optimal for the strategy being implemented rather than always choosing site 1. Each site can hold up to 5 workers, a resource building, and any number of military units of each player.

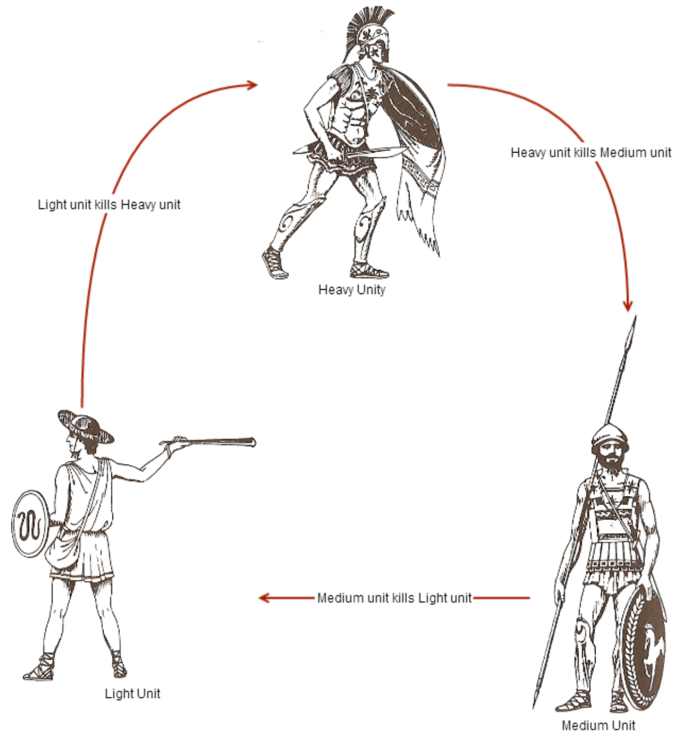
When the simulation begins, each player has 1 action that it can use per tick. This action can be used to create workers, a resource building, or military units. All units are able to be given a new order each step. This restriction forces the ANN to prioritize the order in which units are produced. The ANN could focus on rushing workers out to the site first and then send defensive units. Conversely, defensive units could be sent to secure the site, after which workers and a resource building could be tasked to the site.

1) Unit Types: 5 unit types exist in the simulation. 2 economic units exist, the worker and the resource building. The worker is the only unit in the simulation that can gather resources. The resources that a worker gathered can only be brought to a resource building in the same site that the resources were gathered. This requirement is necessary to preclude the strategy of building and defending one resource building while gathering resources from other sites. The simulation's method of combat would give an advantage to that strategy which would not allow the ANN to evolve correctly.

The resource building is a unit in the fact that it can be placed in sites and that it can be destroyed. Once placed, the resource building cannot be moved. The simulation restricts the ANN to only 1 resource building in existence at one time to prevent the ANN from building one in each. The resource building takes several units to destroy but is sufficiently expensive that the ANN will not automatically build a large number of resource buildings.

The other 3 types of units in the simulation are military units. The simulation follows a *rock, paper, scissors* paradigm found in popular RTS games such as StarCraft (see fig. 3). The essence of this approach is that each type of unit will have 1 weakness and 1 strength. In order to keep the simulation as simple as possible, every unit will cost 100 resources and take 1 tick to build.

Fig. 3. Rock, paper, scissors relationship between the three units


Light unit kills Heavy unit
Heavy unit kills Medium unit
Heavy Unity
Medium unit kills Light unit
Light Unit
Medium Unit

Against a unit's prey, 1 unit can destroy 2 units before being destroyed. Destroying the second unit will also cause the destroyer to cease to exist. This restriction means that a prey will be destroyed by one-half a predator.

The travel time for each unit will be the same regardless of the unit type. Each site lies a certain distance from each other, as noted in fig. 1. While moving from site to site, the units will not be able to attack or receive new instructions. This restriction is necessary to keep combat inside the sites rather than between them. Furthermore, the order of destruction for the types of units starts with prey military units, predator military units, workers, and then resource gathering building. This order simulates a strategy that the decently skilled RTS player would choose in maximizing unit effectiveness.

```
Require: Simulation for battle, Site of battle, Player 1, Player 2
Ensure: The units have fought, and the dead units removed

$playerList \leftarrow$ list of player 1 units in site
$enemyList \leftarrow$ list of player 2 units in site
if units from both players are in site then
    $playerLookup \leftarrow$ $playerList$ grouped by type
    $enemyLookup \leftarrow$ $enemyList$ grouped by type
    for all units in playerList do
        FINDTARGETUNIT(enemyLookup, unit)
    end for
    for all units in enemyList do
        FINDTARGETUNIT(playerLookup, unit)
    end for
    $deadList \leftarrow$ All dead units in site
    $Simulation$REMOVEDEAD($deadList$)
end if
```

Fig. 4. Pseudocode of site battle code.

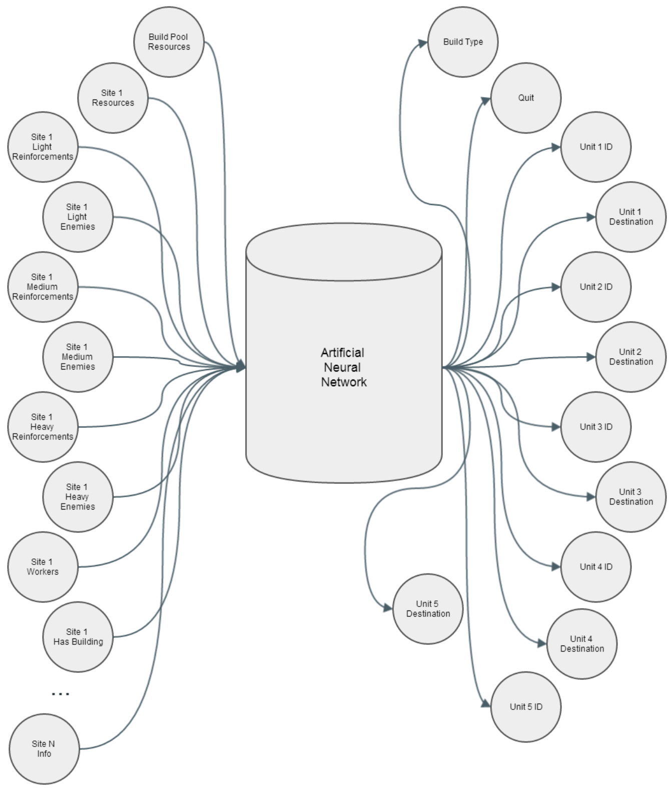
```
Require: Enemy unit lookup, Attacking unit
Ensure: The most effective attack performed on a live enemy unit

$hasAttacked \leftarrow false$
$attackOrder \leftarrow$ order of effective attack
while things to attack and!hasAttacked do
    $targetType \leftarrow attackOrder.$DEQUEUE
    $targetUnit \leftarrow$ first living unit of type $targetType$
    if $targetUnit! = null$ then
        $unit.$ATTACK$(targetUnit)$
        $hasAttacked \leftarrow true$
    end if
end while
```

Fig. 5. Pseudocode of site battle code.

2) Encoding and Decoding for the Artificial Neural Network: Choosing the inputs and outputs are important to the performance of the ANN. If a bad input or output is used, the ANN must work to overcome the myopia created in order to improve the fitness.

The inputs chosen are the resources available to build and the site specific information for each site. The inputs for each site are the available resources and both the reinforcement count and enemy count of all 3 warrior types. In addition, the worker count is input as well as whether or not a building is available in the site. The building input is 0 if there is no building or 1 if there is a building.

Fig. 6. Diagram of the inputs and outputs to the ANN



The outputs chosen are the type of unit to build, if any, whether or not to quit, and 10 outputs representing the unit index in the player's available units list as well as the destination index to the movable sites list.

3) Data Logged: Consideration was given to how much data was to be logged during evolution. For each generation, both max and mean were recorded for fitness and complexity. This data was retrievable from NEAT as the evolution took place. Lean data logging during the process ensured that the simulation would complete quickly. Approximately 200 generations could be completed in 5 minutes, which is 6,000 simulations every minute.

This speed can be compared to the code for the visualization software created. The software examines the state of the simulation after every tick and generates an image. The software writes out the images for 1 simulation in 30 seconds. To write 1 generation of the evolution would require 75 minutes.

IV. Observations

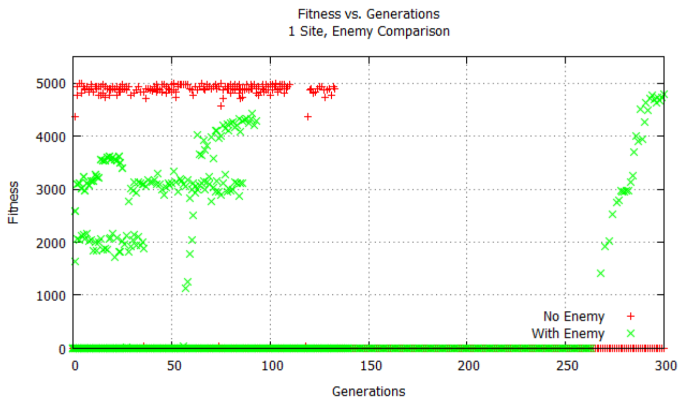
The observations performed included a number of changes to observe the ANN responding to different levels of complexity. Altering the number of sites available in the simulation from 1 site to 4 sites produces different gains in fitness. Two different fitness functions were used. The first determined the fitness based on the number of resources collected. The second function used a decay function in order to prioritize collecting resources quickly. Finally, the reaction of the ANN to the enemy was observed.

In all cases, there is a wide variance between the generations producing ANN with 0 fitness. This is due to the fact that ANN begin by randomly permuting weights between the nodes.

A. 1 Site

The ANN began by training in a simulation containing only 1 site from which to collect resources. To successfully collect the resources, the ANN must create a building and at least 1 worker. After creating the necessary units, the ANN then must move them to the site. The experiment was performed both without and with an enemy to determine what effect, if any, the enemy would have on the ANN.

Fig. 7. Fitness vs. Time in Ticks for 1 site, with enemy comparison.



In figure 7, it can be seen that some runs of the ANN never learned anything. The ANN evolved without an enemy, shown in red, learned very quickly how to collect all of the resources in the 1 site. The ANN that dealt with an enemy had a harder time collecting all of the resources, as shown by the green ‘x’s. After learning how to collect, there is a trend upwards as the ANN optimizes its algorithm.

To examine how the ANN responded to increased complexity in the requirements, a rate of decay was introduced into the fitness algorithm.

$$[h]\text{fitness} = R_{\text{initial}} \times e^{\frac{tick}{-5}} \qquad (1)$$

Equation 1 awards progressively smaller fitness for the same amount of resources gathered over longer periods of time. For example, 1,000 resources gathered over 3 ticks produces 548.81 fitness, while the same amount gathered over 6 ticks produces only 301.19 fitness. Some of the ANN returned max fitness which, while theoretically possible, is not supposed to be obtainable due to the requirement of a building and workers being produced. Further examination is required to determine what is occurring in these situations.

Fig. 8. Fitness (Resources Gathered Quickly) vs. Generations for 1 site, with enemy comparison.

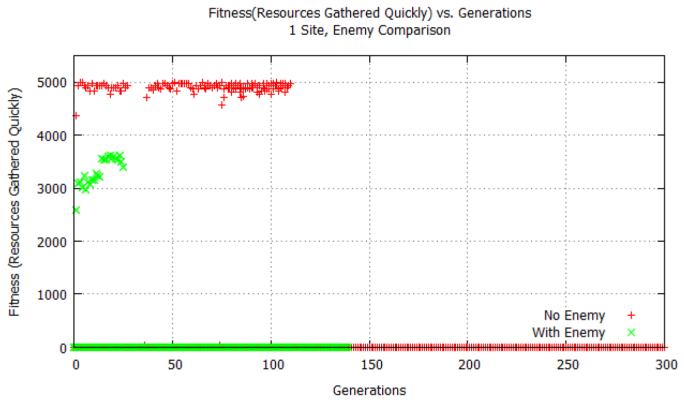


figure 8 shows a similar trend. Without the enemy, the ANN is able to efficiently collect many of the resources. Introducing an enemy, however, speeds up the the rate at which the ANN optimizes the fitness characteristic. This same trend of faster learning in the presence of the enemy is present in the 4 sites experiments as well.

B. 4 Sites

The complexity of the problem space was increased from 1 site to 4 sites. This increase allows the ANN more time before an enemy attacks, but forces the ANN to learn to group a building and workers together to gather resources.

Fig. 9. Fitness vs. Generations for 4 sites, with enemy comparison.

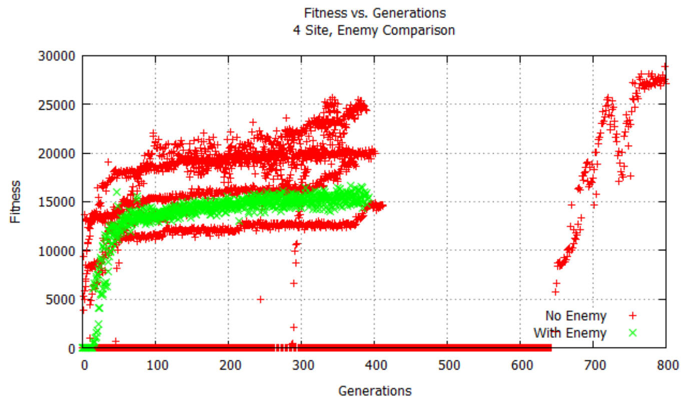


Figure 9 demonstrates more fully that the ANN increase its fitness in a rough logistic curve. The different “lines” that appear in the “No Enemy” data demonstrate that the ANN is evolving different structures to approach the same task. The ANN optimizes along a single strategy, which may have a lower maximal fitness than another possible strategy.

As with the 1 site experiment, the complexity was increased by awarding faster collection using equation 1. This experiment with the addition of the enemy represented the most complexity presented to the ANN in the study.

Fig. 10. Fitness vs. Generations for 4 sites, with enemy comparison.

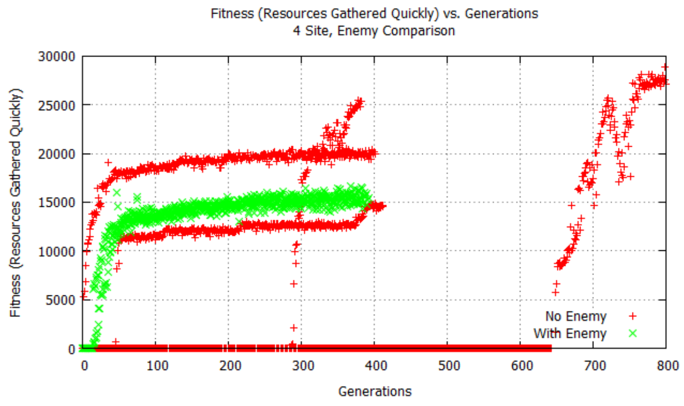


Figure 10 shows that the enemies presence increased the gain in fitness over no enemy.

C. Neural Networks Evolved: a Comparison

TODO: Must find a way to examine the evolved ANN.

V. Conclusions

A. Neural Network Performance

The performance of the ANN was varied. On some of the runs of the experiments, the ANN quickly achieved a very high fitness. Other runs saw no improvement of the performance above the starting 0 mark. This variance is due to the learning nature of the ANN. Until the ANN finds a strategy that works, it must randomly combine its nodes to produce output. Once a strategy has been found that increases the fitness, however, the ANN is able to improve and begin searching through the space of all possible beneficial strategies.

In almost all cases, the ANN learned more quickly with the presence of an enemy.

Increasing the complexity of the fitness characteristic by rewarding faster collection rates slowed the fitness gain. However, the ANN adapts to this constraint well and performance increases are seen.

A startling observation that was made during the study was that the presence of an enemy generally caused the ANN to learn much more quickly. Two probable explanations exist for this behavior. First, and more likely, is that the simulation itself changes based on the presence of the enemy. The ANN takes advantage of an unseen bug and exploits it to increase its gathering efficiency.

Second is that the ANN is demonstrating a defense reaction to the enemy presence. This behavior, being unaccounted for in the design, would be an example of emergent behavior.

B. Neural Network Feasibility

The original question that was studied was whether or not an advanced AI technique such as an ANN could be used in a commercial video game environment. Several obstacles exist that must be accounted for if an ANN is to be used.

First, the time constraints on the project mean that the development team has little time to work on a project with no clear return of investment. The ANN could increase the marketability of the project, but it may also use resources without producing any benefit to the project. This uncertainty in return is shunned by the video game industry in order to more likely return a profit.

Second, an ANN has little guarantee of performance. Before the project is finished, there is little indication of how well the ANN will respond to the action space presented. Much time would be spent implementing it to find that the benefit it brings is too small. This time could have been spent in adding a feature that promised some return.

Third, ANN are non-deterministic. As the ANN maximizes its fitness, it may find ways to circumvent or exploit the intended action space. This behavior may involve anything from determining a powerful style of play to gaming the fitness function. When examining the actions of an ANN, there is no easy way to determine why the ANN is doing any one thing. This non-determinism runs counter to the development work flow used in the video game industry. When a video game presents a bug, it is isolated, reproduced, examined, and then handled. With an ANN, the process is much harder because of small changes in the inputs leading to drastically different outputs.

Lastly, the largest obstacle in using an ANN in commercial video games is that of selecting the correct inputs, outputs and fitness functions. Research has determined that the choice of inputs and outputs is an important factor in the ANN's performance. One improvement to the study would be to develop a better method for determining which units to move. This improvement could factor in the location of these movements as well as the health and type.

The fitness function is the defining characteristic of an ANN. This function is the measurement by which the ANN evolves. If the function was chosen poorly, the result would be an ANN that was "fit," but that did nothing to increase the playability of the video game. One common mistake is to determine a fitness function that appears related to the intended behavior. The ANN learns using that function. When the output is examined in the video game context, it is found that the ANN has either exploited a flaw in the video game or that it is pursuing the fitness function at the expense of engaging game play.

These obstacles represent significant investments in order to make ANNs viable in commercial video games. Unless an experienced AI engineer is available to develop the ANN, it is unlikely that ANNs will find widespread usage in the commercial video game industry.

C. Possible Usage in Video Games

One possible strategy to include ANN into video games is through pre-evolving the ANN before packaging it into the video game. This process would allow the developers more control over the ANNs rather than it developing unwanted behavior when in the hands of players. Pre-evolving would allow for ANN to be used either offline, being evolved only in the hands of the developers, or online, learning from the players as the game is played. This pre-evolution would give a greater guarantee that the ANNs would function as desired, while still allowing the ANN to increase playability.

Another strategy to include ANNs would be to narrow the action space over which the ANN must function. Reducing the inputs from an entire player down to an individual unit would reduce the action space to only that which is around the individual unit. The ANN could focus on keeping that unit alive through movement and attacking. This focus would allow the ANN to evolve to perform well against a specific unit/opponent. Allowing for the AI to fall back on basic unit function should the ANN cease to perform as well would improve the guarantee of performance.

Another area that ANNs would perform well would be in prediction. In the RTS genre, the ANN could predict the composition and arrival of enemy waves before they come. This prediction would give an advantage to the player or the AI, depending on whether or not they used the ANN. The prediction would also allow the player to ignore the predictions if they became erratic. Avoiding the playability drop by enabling bypassing the ANN gives a better guarantee of performance.

Another predictive use of ANNs in video games would be winner prediction. Many games released require the use of statistics such as score and unit composition as well as a human interpreting the actual game play to predict who is winning. Using a ANN, this prediction could be based off of the stats as well as an examination of the action space. While it would not supplant a human interpreter, it could provide a means to improve the meaning of the scores presented in each game.

VI. Future Work

Several areas of future work are available. First, providing harder test AI would demonstrate how well the ANN performs against a more realistic opponent. Moving to a commercial RTS would be a better indicator of both the performance as well as the feasibility of implementing an ANN in a commercial video game. Last, the response of the ANN to enemy presence could be explored to determine whether or not the response is due to a design flaw or emergent behavior.

A. Increased Complexity Through Harder Enemy AI

The enemy test AI in the study builds one wave, which it then sends to a semi-random site. this wave is the only interaction the enemy AI has with the ANN. If the ANN can withstand the initial wave, it will be able to collect resources without worrying about future attacks. Due to time limitations on the study, multiple waves were not implemented.

Adding the ability for the test AI to send multiple waves would increase the complexity that the ANN would have to account for. The ANN would be required to respond in a more tactical manner. Sustained unit selection and movement on the ANN's part would be needed in order to better protect against the enemy. This change could demonstrate that, while collecting resources is the fitness characteristic, other actions not directly related to resources will also evolve.

B. Move to Commercial RTS

The study implemented an abstraction of a commercial RTS video game. This allowed for the time allotted for the study, but does not translate as easily to commercial video games. Expanding the study to include a commercial RTS platform would be a better indicator of whether ANNs would perform well in that environment. The feasibility of implementing an ANN could be better evaluated in a commercial game environment. The speed of the ANN could be examined, as the video game would not wait for the ANN to think for long periods of time.

The action space of the simulation represented a small portion of a commercial video game's action space. While the ANN performed well in the simulation, such performance may degrade over such a large action space. One behavior that might be better demonstrated is that of input atrophy. Input atrophy would be demonstrated as the ANN began to ignore inputs it deemed unimportant to achieving its goal.

C. Response to Enemy Presence

Further study would be performed to determine the cause of the change in performance based on the presence of an enemy. The probable cause is a design flaw in the simulation. The ANN could be exploiting a flaw that presents itself with an enemy. If this bug is the case, the study must be repeated after the flaw is accounted for.

Another possibility is that the ANN could be demonstrating emergent behavior. If this possibility is the reason for the change, more research must be done in order to better explain the cause. The ANN might be responding to competition from the enemy AI. The ANN could be demonstrating the trait of self-preservation. As the ANN is seeking to maximize resource collection, it could be determining that the best way to do so in the presence of the enemy is to defend itself. More study would be done in this area to determine whether or not the response is due to a design flaw or if the ANN is demonstrating emergent behavior.

VII. Appendices

References

[1] "Industry facts," http://www.theesa.com/facts/, accessed: 2013-08-14.

[2] "Domestic movie theatrical market summary 1995 to 2013," http://www.the-numbers.com/market/, accessed: 2013-10-06.

[3] "Electronic arts' management presents at morgan stanley technology, media & telecom conference (transcript)," http://seekingalpha.com/article/1227161-electronic-arts-management-presents-at-morgan-stanley-technology-media-telecom-conference-transcript?page=5, accessed: 2013-11-24.

[4] S. Rabin, *AI Game Programming Wisdom*. Charles River Media, 2002.

[5] M. Ponsen, H. Muñoz-Avila, P. Spronck, and D. W. Aha, "Automatically generating game tactics through evolutionary learning," *AI Magazine*, vol. 27, no. 3, p. 75, 2006.

[6] P. Demasi and J. d. O. Adriano, "On-line coevolution for action games," *International Journal of Intelligent Games & Simulation*, vol. 2, no. 2, 2003.

[7] C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia, "Generalizing plans to new environments in relational mdps," in *In International Joint Conference on Artificial Intelligence (IJCAI-03*. Citeseer, 2003.

[8] D. C. Cheng and R. Thawonmas, "Case-based plan recognition for real-time strategy games," in *Proceedings of the Fifth Game-On International Conference*, 2004, pp. 36–40.

[9] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma, "On-line adaptation of game opponent ai with dynamic scripting," *International Journal of Intelligent Games & Simulation*, vol. 3, no. 1, 2004.

[10] M. Ponsen, "Improving adaptive game ai with evolutionary learning," Ph.D. dissertation, Citeseer, 2004.

[11] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Evolving neural network agents in the nero video game," *Proceedings of the IEEE*, pp. 182–189, 2005.

[12] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[13] J. K. Olesen, G. N. Yannakakis, and J. Hallam, "Real-time challenge balance in an rts game using rtneat," in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE, 2008, pp. 87–94.

[14] T. Malone, *What makes computer games fun?* ACM, 1981, vol. 13, no. 2-3.

[15] G. N. Yannakakis and J. Hallam, "Towards optimizing entertainment in computer games," *Applied Artificial Intelligence*, vol. 21, no. 10, pp. 933–971, 2007.