Evolving Neural Networks in NPCs in Video Games

Katie Reddemann
Division of Computer Science
University of Minnesota, Morris
Morris, Minnesota, USA 56267
redde019@morris.umn.edu

ABSTRACT

This paper will look at how evolving artificial neural networks can be used to advance and enhance game play. We will look at neuroevolution in recent video games and how it can be applied to video games from the past. Also, we discuss expanded concepts of Neuroevolution of Augmenting Topologies (NEAT) and how evolving neural networks are applied to video games.

Keywords

Neural Networks, Evolving Game Play, Evolving NPC, Neuroevolution

1. INTRODUCTION

The video game industry has grown exponentially over the years [11]. Millions of video games are sold every year [4]. But just having the most recent, life-like graphics can not make a game sell. Artificial neural networks and neuroevolution can make a game more interesting.

Many video games feature dull, repetitive non-player characters (NPCs) and many video games have very little to no artificial intelligence (AI) techniques [5]. A challenge for AI is the ability to react to the environment around them, do tasks more efficiently, and react to new situations. Neuroevolution can be used in video games to improve NPCs. These techniques would benefit not only video games but other parts of computer science, such as robotics, because of adaptability [5].

Many NPC scenarios are scripted by developers and never change. The repetitive nature of NPCs may cause a player to play the game once and never play through it again. Giving NPCs new behaviors within video games create new and interesting scenarios through each play through. Giving NPCs a more human-like behavior would make players feel like they are playing with a fellow human instead of a mindless machine [2]. Older games had restrictions with resources so NPCs did not take priority, but with today's technology, there are very few restrictions.

UMM CSci Senior Seminar Conference, May 2015 Morris, MN.

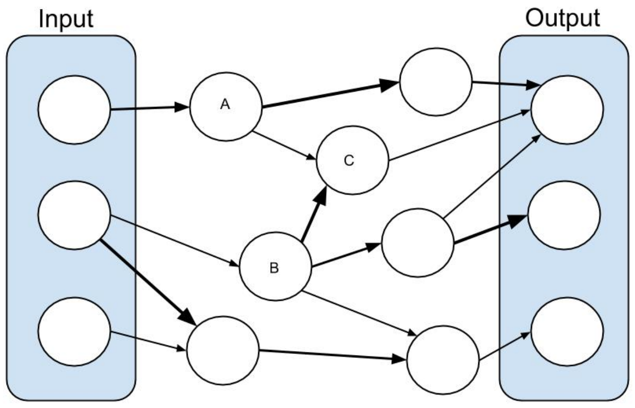

Figure 1: An example of a neural network [8].

Next, we will talk in depth about neural networks, neuroevolution and NEAT. We will look at the video games Fight and Flight and NERO that implement neuroevolution. We discuss applying neural networks and neural evolution to older games like Super Mario. Then, look at the issues with neural networks and neuroevolution.

2. BACKGROUND

2.1 Neural Networks

Artificial Neural networks are powerful tools because they can learn new behaviors and even solve problems that designers had not anticipated [3]. They do this using statistical pattern transformation and generalization [5]. Neural networks work very well with outside input and they don't always need specific outputs.

Neural networks are based on of the complex connections of neural networks in the biological brain[3]. Each node within the network represents units and the nodes are connected with links. The links have weights attached to them. In the example shown in Figure 1, the links are represented with arrows. The weights attached to them are shown through the thickness of the arrows. The input nodes send data through the links to the center nodes. The data comes from the user. The center nodes are called the hidden nodes and do all the work behind the scenes [8]. The output nodes outputs the new data. The more complex the neural network is, the more nodes it will have.

In Figure 1, when data from node A and node B move through the link to node C, node C has to decide which one takes precedence. The link between A and C is a thinner line, so C could absorb B's input. Node C could also take both inputs in but may favor the data from node B [8].

2.2 Evolutionary Computation

Evolutionary algorithms are inspired by biological evolution that is an iterative process that uses crossover and mutation to create new children. Crossover takes two parents then takes traits from each of them and creates a new child. Mutate is when a parent traits is changed to create a new child. Fitness is a function how well the traits of the child perform in the context of the problem.

2.3 Neuroevolution

Neuroevolution involves the evolution of neural networks through evolutionary algorithms. The algorithms evolve the weights and structures of the networks [5]. The inputs, outputs and the hidden nodes are set, then the links between each node go through the evolution. The simplest way is to genetically evolve the values of the weights then crossover and/or mutate a population.

When a neural network is evolved, it uses backpropagation to calculate the errors and go back through the neural network. The errors are compared to each other and the weights of the links are changed. The process could change multiple links or just one with the biggest error [8]. The value of the fitness is based on how well the network performs. The fitness is by determined how close the neural network gets to the desired output.

A method of developing complexity in networks is Neuroevolution of Augmenting Topologies (NEAT). The population to begin with is small networks that, over time, become more complex with evolution [5]. NEAT can find the right complexity for the problem with minimal number evolutions.

Neuroevolution works well for video games. Many populations can be managed and the neural networks are constant. They can adapt in real time and because of recurrences, memory can be implemented [5].

3. VIDEO GAMES THAT LEARN

3.1 Fight or Flight

Fight or Flight is a game that was designed to evolve multi-modal behavior. Multi-modal behavior is when evolving agents exhibit distinctly different modes of behavior under different circumstances [6]. The game has two different modes. Fight involves the player in the center of a ring surrounded by four NPCs. The player can hit the NPCs with a bat and the NPCs can ram into the player. The player can only take five hits before dying. Flight is very similar to Fight. The player is surrounded by four NPCs but the player is defenseless. The player must run from the NPCs and get to safety. The NPCs are impervious to damage and the player will die in five hits [6].

The NPCs are controlled using neural networks and can tell where the player is and where the other NPCs are. The NPCs must be able to operate differently between the Fight and Flight, game modes so they have different objectives. In Fight, the NPC's behavior is based on damage dealt, damage received, and time alive. In Flight, NPC's behavior is based on is how much damage is dealt [6]. These objective are the fitness score. The better the score, the better the NPC.

3.1.1 Algorithm

A modified evolutionary algorithm, NSGA-II (Non-Dominated Sorting Genetic Algorithm) is used to evolve the NPCs. It works well because the NPCs have multiple objectives and the algorithm helps optimize decisions. NSGA-II sorts the population of neural networks into non-dominated Pareto fronts by fitness score. The Pareto method will make sure that the NPCs will not become better if it costs other NPCs getting worse [10]. The NPC's neural networks are cloned and then mutated. This is when the neuroevolution takes place [6].

The neuroevolution is similar to NEAT. The evolution can change the weights of links, add new links between nodes or clone nodes and put them elsewhere in the neural network. When mutation takes place, the links can have different weights and connections, and there could be some disarray. So there is a final mutation called Merge Mutation. This reduces network structure to prevent bloat while preserving fitness.

In Figure 3, Merge Mutation has determined that merging the gray nodes in A will preserve fitness while shrinking the network. The dashed lines are links between the nodes that will be deleted. Merge Mutation merges the two nodes then deletes the unnecessary connections.

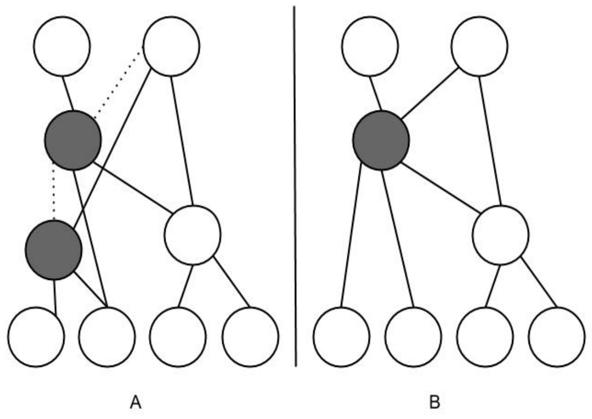

Figure 3: Merge Mutation on a tree.

To encourage multi-modal behavior, the output nodes define the agents behavior, but it is difficult to determine which mode the output came from. The preference node's value is attributed to different behavior outputs whether it is damage dealt, damage received or time alive for Fight or damage dealt for Flight.

There is a potential problem with the algorithm: how to divide the objective output. There could be two modes (one for Fight and one for Flight) or there could be four modes (one for each objective). A new method is devised to figure out the optimal number of output modes. This New Mode Mutation adds nodes until the desired output is reached. The new networks need to be monitored just in case the changes are too drastic.

3.1.2 Experiment

The NPCs networks were evolved then the final network was copied to four NPCs that would become a team. Because the neural networks are the same, the NPCs could expect the same results from each member from the team.

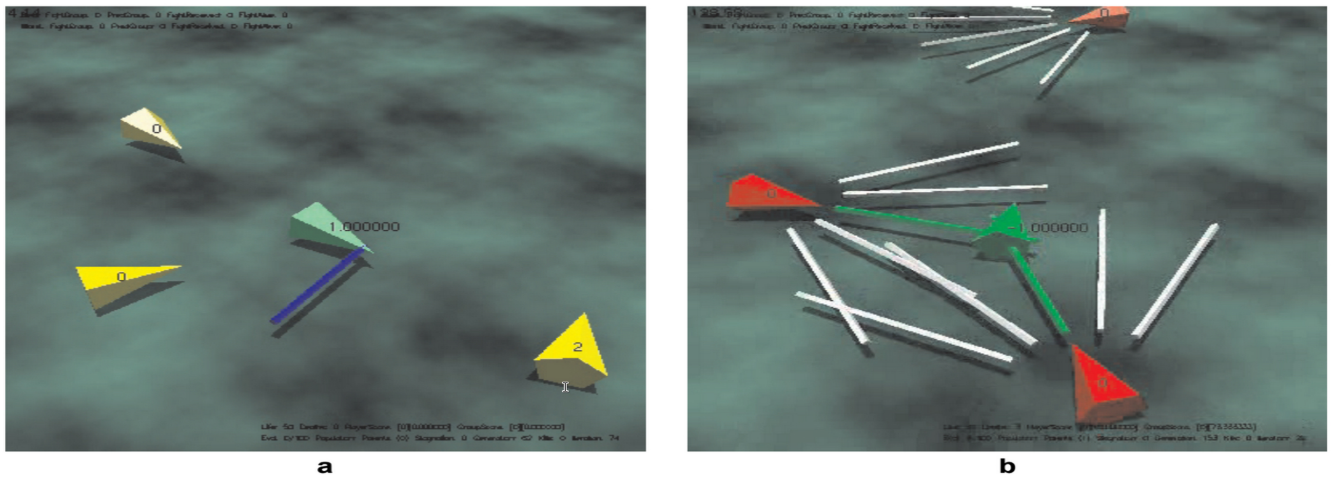

Figure 2: (a) Fight features player swinging a bat around to damage attacking NPCs. The NPCs are surrounding the player to ram and deal damage. (b) Flight has the player running away from the NPCs. The bars on the darker NPCs represent sensors from detecting the player.

The fitness score is calculated for the whole team, so teamwork is a necessity. The calculated score makes the team more valuable than the individual NPC.

Because humans will not remain interested for the amount of evolutions it would take to evolve the NPCs, a bot was created to be the opponent of the NPCs. The bot and the evolved NPCs will act differently for each type of game. In Fight, it will chase the nearest NPC and swing its bat while moving. In Flight, the bot will move backwards away from the nearest NPC. This is to keep the NPCs in its line of sight. This made it harder for the NPCs because they move at a constant speed.

The first few generations had trouble evolving any new methods. The bot was too fast so it was slowed. When the NPCs evolve enough to beat the slowed bot, the speed was increased little by little. Goals are the way the NPCs progress, one for each objective. Once the goals are met the difficulty is raised.

1Mode and ModeMutation were the experimental conditions. 1Mode use neural networks with one output mode which has two nodes (one for Flight and one for Fight). ModeMutation would begin with neural networks that have one output mode but will gain more as mutation takes place. The output mode will need three output nodes (two nodes from objectives and a preference node) [6].

3.1.3 Results

The NPCs under the ModeMutation condition were twice as successful as the NPCs under 1Mode. The ModeMutation NPCs survived four of ten trials when the bot was full speed while the 1Mode NPCs only survived two out of ten. The ModeMutation NPCs were well rounded in all objectives and 1Mode's NPCs focus more on individual objectives.

The ModeMutation NPCs developed a multi-modal behavior through a corralling tactic in Fight. One NPC would become bait and the bot would chase after it. The other NPCs would follow behind the bot using a diagonal tactic to cover more distance and catch up with the bot. The NPCs would circle the bot and attack. The NPCs would make sure to hit the bot so it would be pushed back into the center of the corral.

The 1Mode NPCs lacked teamwork. Individuals were unbalanced in there objectives. In once instance of the trails, the NPCs in Fight would avoid taking damage. That is all they focused on. The NPCs did no damage only avoided being hit and managed to live throughout the whole trial. The 1Mode NPCs had extreme scores but rarely had good scores across all objectives.

The ModeMutation NPCs met the goals better than the 1Mode NPCs. The evolved team had better scores across the board and had better attack strategies. The ModeMutation NPCs developed the desired multi-modal behavior.

The use of neural networks and neuroevolution created a smarter NPC. The NPCs learned over time using the bot and could possibly do the same against a player. There could be more strategies that could evolve from playing against a player and more unexpected outcomes.

3.2 NERO Video Game

NERO (NeuroEvolving Robotic Operatives) is a video game that was created by a team of over 30 students for a two year period. Players would act as a drill instructor or a trainer [5]. They would teach the team of NPCs called agents through training skills. The game starts with the agents having no skills, but they can learn. The player must design exercises and goals that, ideally, will increase in difficulty. This allows the agents to learn the basic skills then become more advanced. The agents have sensors that include enemy radar, an "on target" sensor, an object rangefinder, and line-of-fire sensors.

The player controls ideal behavior of the NPCs through sliders. The sliders specify the reward and punishment, for example, when their agents attack enemies, follow teammates, get hit, etc. Each slider is a fitness component. The components are normalized to a Z-score, and then the fitness is computed as the sum of the normalized components multiplied by their slider level [5]. The agents learn through their evolving neural networks that are evolved though rtNEAT [7].

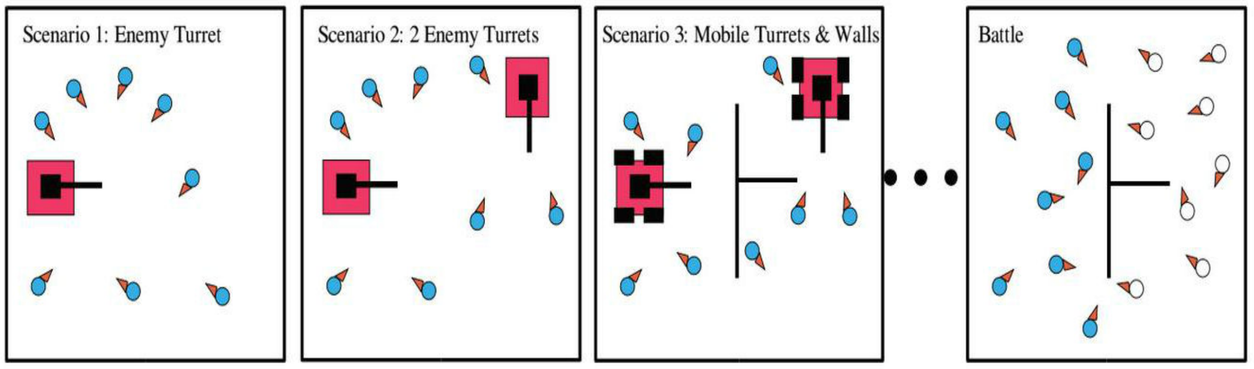
Scenario 1: Enemy Turret
Scenario 2: 2 Enemy Turrets
Scenario 3: Mobile Turrets & Walls
Battle

Figure 5: A training sequence in NERO [7].

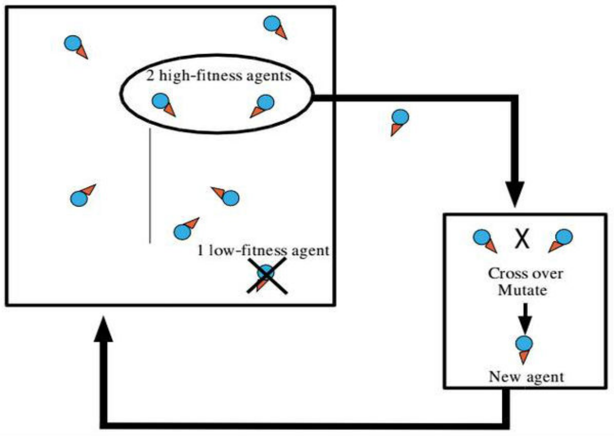

Figure 4: Replacement Cycle in rtNEAT [7].

3.2.1 *Real-Time Evolution*

Real-Time Evolution (rtNEAT) is when neural networks evolve during game play. rtNEAT uses NEAT as a base. In NEAT, each generation of neural networks is mutated and replaced, which makes it difficult to use in real time. In rt-NEAT, the worst agent in the generation is removed every few game ticks (a set time within the game) and replaced with a child of the two best agents. In Figure 4, the fitness is calculated for each NPC. The neural networks of the two agents with the best score (circled) crossover and mutate to create a child. The child replaces the lowest scoring agent. This takes place continuously over the course of a game. This allows for game play to flow smoothly. rtNEAT is flexible enough to revert an entire population of agents to be less complex then build a new behavior. The player can interact with agents and have the complex neural networks react fast enough.

3.2.2 *NERO Game Play*

There are two game modes in NERO: training and battle. The training begins with fifty agents on a field. As rtNEAT replaces agents and adds complexity to neural networks, behavior changes and the agents learn how to complete goals set up by the player [5]. The player creates a specific field to complete a goal. The player can add turrets and walls, and even create a maze to teach the agents. Once the particular goal is completed, the player creates a harder field to train in. In Figure 5, the player starts out with a single turret on the field. The goal is to defeat the turret and to learn how to avoid getting hit. The player decides if the agents have completed the goal in a satisfactory matter. Then the agents move on to a more difficult field. The progression of the fields adds turrets and walls to change the behavior of the agents. A strategy that could evolve is using walls as a shield agents enemy bullets and trying to avoid the enemy as much as possible. Another strategy is to aggressively seek the enemy and fire [5]. After these training levels, the player can save their team for use in the battle mode and for later training.

In battle mode, the player chooses twenty of their best agents. They can pick from different teams they have saved. The opponent the player faces is another player. The opponent has a team that they have trained. The victor is the one with the last agent alive or the most agents left alive if time runs out which happened frequently with teams that avoided enemies [5]. Depending on the strategies each player used, one may have an advantage over the other. The agents that tend to avoid the enemy did poorly against teams that were aggressive and ran towards the enemy.

Having a player see their progress in real time could make a video game more engaging. The player must spend time and effort building their team and interacting with the NPCs to create their ideal team. In general, the players agreed that the game play was interesting and entertaining [5].

4. APPLYING NEUROEVOLUTION TO OLDER GAMES

Neuroevolution could be applied to existing games to make their replay value higher. By making game modifications (mods), extra features that are added to a game, neural networks and neuroevolution can be added. The mods are easy to implement and safe to use because they do not change the original structure of the game [5]. Many games allow mods already and are even open sourced. Open sourced games are a good opportunity for adding neural networks and neuroevolution. People do not necessarily need the permission of the creator to change the game.

4.1 Super Mario Evolution

Infinite Mario Bros is a public domain clone of Super Mario Bros made by Nintendo. It is playable on the web, and the Java source code is also available. The game features Mario, who is player-controlled. Mario can walk and run left and right, jump, and shoot fireballs. There are three states that Mario can be in: small, large, or fire.

The goal of each level is to get to the end and lower a flag. Sub-goals include collecting coins, reaching the end of the level quickly, and getting a high score by killing enemies [9]. The enemies damage Mario; if he is large, the enemies turn him small and renders him unable to kill them. If he is small, the enemies kill him and he loses a life. If he is a fire state, the enemies turn him to the larger form. Mario will lose a life if he falls into a hole regardless of what state he is in. There are items within the game. These items can be hidden or out in the open and have different effects on Mario.

4.1.1 *Adding Evolution*

The mod will create infinite number of neural networks to complete the level. To add neural networks and neuroevolution, timer had to be removed from the game because it would be obsolete [9]. The dependency on graphics was removed and there was some refactoring done on the original code to make it more usable and readable. The new, modified software was a single-threaded Java application. There was also a TCP interface for controllers.

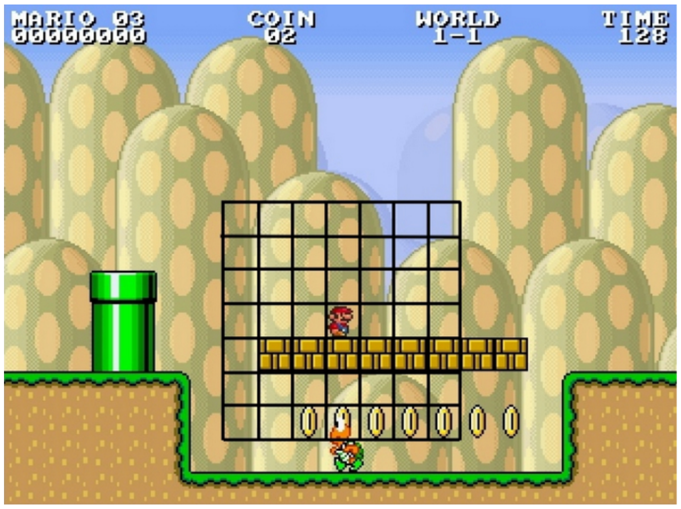

Figure 6: The sensor grid that surrounds Mario [9].

The nodes in the neural networks would have input values of 0 (on) or 1 (off). The constant value would be one. The inputs would be: if Mario was on the ground, if Mario could jump, how many environmental obstacles there were or how many enemies are present. Mario would have a grid of sensors put around him (Figure 6). The grid will not tell where coins or items are and Mario cannot "see" past the grid, like the enemy that is below. The inputs fed into two different neural networks, either a Multi-Layer Perception (MLP) or Simple Recurrent Network (SRN) [9]. The networks would have 10 hidden nodes and three different sizes of links between them: small, medium and large.

HyperGP is a hybrid neuroevolution/genetic programming algorithm. HyperGP evolves the link weights of neural networks as a function of their coordinates (x, y) in a grid called a substrate using genetic programming. HyperGP is used to evolve SRN to become HyperGP SRN. HyperGP SRN was used to evolve 100 generations of 100 individuals.

4.1.2 *Testing*

The fitness was based on how far Mario could progress through levels. The levels were between 4000 and 4500 units long, and were randomly generated. After the evolution was run, the generalization capacity for best network in the population was tested. The tests would run each size of SRN, MLP, and HyperGP SRN. Each evolved Mario was tested one level at at time starting at a difficulty of level 0, the easiest level. Success was reaching a fitness score of 4000 or above. This would be interpreted as clearing a level or being close to it [9]. The difficulty was raised after the level was cleared.

Many of the evolved Marios had problems clearing different levels of the same difficulty. There were problems with jumping on moving platforms because the sensor grid was to small. The evolved Marios had issues with enemies where he would not jump over them but run in to them [9].

4.1.3 *Results*

The average maximum level for SRN and MLP were fairly similar. The neural networks with more links did worse then the medium and smaller ones. Overall the smaller network links with the simplest complexity performed better [9]. The HyperGP SRN had similar averages over the sizes, but it did not out perform the other networks.

Neural networks work very well with Infinite Mario Bros. But in the long run, generalization may become a problem [9]. Time and space are also issues. A more advanced architecture might be needed to combat these problems. A way to handle enemies might be to create a separate neural networks that are specifically designed to jump over enemies or jump on them. Items and coins are other networks that would need to be created. This would allow for more diverse behaviors from Mario [9].

Mario is a popular game and has many fans. Updating games like this could appeal to a wide audience. Neuroevolution can add new and interesting scenarios that may bring the player back to an older game.

5. ISSUES

The issue that may arise is that games are not designed with NPCs learning through evolution in mind. Getting the necessary information from such games could be challenging.

A way to get around this is to create an interface that could be used across many games. TIELT (Testbed for Integration and Evaluation of Learning Techniques) is in development [5]. This Java application connect a game engine to a decision system. The system learns about the game and allows the researchers to test many different environments.

Creating neural networks that appeal to an individual player could prove difficult [1]. Evolving NPCs is a way to appeal to the player, but there are limitations. The game has to gather a large audience to sell well, which may cause more generic scenarios to appear.

Another issue, is the unexpected results that neuroevolution can generate. This can lead to erratic game behavior and cause bugs to surface [1]. Testing and debugging this is quite challenging, because it is difficult to reproduce the errors that occur.

6. CONCLUSIONS

Applying neural networks to games can make them more interesting and increase their playability [6, 7]. The neural networks adapt well to new situations and can even react to situations that developers did not predict. Some games are already taking advantages of neural networks and neuroevolution. Fight or Flight featured two different neural networks evolving to defeat a bot. The NERO video game had agents evolve using feed-back from the players. Older games could benefit from neural networks. Infinite Mario Bros had neural networks and neuroevolution woven into it to have Mario complete his own game [9].

7. ACKNOWLEDGMENTS

I would like to thank Elena Machkasova, Kristin Lamberty, and Lucas Ellgren. I would also like to thank my parents, Jim and Karen Reddemann for supporting me through out my life.

8. REFERENCES

[1] D. Charles. Enhancing gameplay: challenges for artificial intelligence in digital games. University of Utrecht, The Netherlands, November 2003. ACM.

[2] C. Delgado-Mata and J. Ibáñez Martínez. AI opponents with personality traits in Überpong. In *Proceedings of the 2Nd International Conference on INtelligent TEchnologies for Interactive enterTAINment*, INTETAIN '08, pages 1:1–1:8, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[3] D. Johnson and J. Wiles. Computer games with intelligence. In *Procs. 10th IEEE Intl Conf. on Fuzzy Systems*, pages 61–68. IEEE, 2001.

[4] V. Ltd. USA yearly chart: The year's top-selling games at retail ranked by unit sales, 2014.

[5] R. Miikkulainen, B. D. Bryant, R. Cornelius, I. V. Karpov, K. O. Stanley, and C. H. Yong. Computational intelligence in games. In G. Y. Yen and D. B. Fogel, editors, *Computational Intelligence: Principles and Practice*. IEEE Computational Intelligence Society, Piscataway, NJ, 2006.

[6] J. Schrum and R. Miikkulainen. Evolving multi-modal behavior in NPCs. In *Proceedings of the 5th International Conference on Computational Intelligence and Games*, CIG'09, pages 325–332, Piscataway, NJ, USA, 2009. IEEE Press.

[7] K. O. Stanley, B. D. Bryant, I. Karpov, and R. Miikkulainen. Real-time evolution of neural networks in the nero video game. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-2006)*, pages 1671–1674, Boston, MA, 2006. Meno Park, CA: AAAI Press.

[8] J. Suarez. Intro to neural networks, 2009.

[9] J. Togelius, S. Karakovskiy, and J. Koutník. Super mario evolution. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 156 – 161, Milano, 2009.

[10] Wikipedia. Pareto efficiency, 2015.

[11] Wikipedia. Video game industry, 2015.