Making a Better Opponent:
Machine Learning for Video Games

Kraig McFadden

May 7, 2018

Advisor: Professor Andrew Houck

Submitted in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Engineering

Department of Electrical Engineering

Princeton University

Making a Better Opponent: Machine Learning for Video Games

Kraig McFadden

ABSTRACT

The purpose of this work is to develop a novel game artificial intelligence (AI) system. The current, standard AI system for action games – behavior trees – has been in use for over a decade, and there are calls for innovation coming from players and developers alike. The new system must meet four goals – adaptability to allow for natural game progression, simplicity regarding both content and code, computational feasibility so it can be run in a commercial game on consumer computers, and above all, it must provide believability, engagement, and fun for the player, which is summed up as “fun factor”. To meet these four goals of adaptability, simplicity, computational feasibility and fun factor, a machine learning model has been selected to control AI agents in the game. Specifically, a neural network was selected to model AI agent behavior. While neural networks have been shown to be able to approximate any function arbitrarily well, there are practical considerations and limitations, including how best to train the network. The bulk of the engineering work in this thesis was directed towards answering that question. After settling on a feasible network architecture, the network was trained using genetic algorithms, reinforcement learning, and a third learning algorithm that was a hybrid of those two. Data was collected during the training process to see how agents improved according to the fitness function applied to them, and data was also taken at regular intervals during the training to see how agents performed against control opponents (that is, opponents that behaved deterministically). Also, player testing was conducted to determine player perceptions of the AI agents, their believability, difficulty, and fun factor. Overall, it was determined that each training method produced agents that were capable of learning and improving in fitness, but the genetic algorithm and hybrid approaches to training produced AI agents that were the most enjoyable to play against. Additionally, the genetic algorithm was quite simple to implement compared to the other two because it does not require backpropagation, and that reduced the computational power required to run it. All things considered, a neural network trained with a genetic algorithm seems the most promising for controlling NPC AI enemies in an action game.

Acknowledgements

I would like to thank Professor Andrew Houck and Professor Yoram Singer for agreeing to advise and read this thesis. I appreciate you both taking a gamble on an unconventional, self-directed project. Also, I'd like to thank Levent for his advising, and for keeping me on track throughout this year. Also, to the department at large, I am very grateful for the education I received during my time here at Princeton.

I also would like to recognize and thank my family back home in Vermont – to my parents and my sisters, I really appreciate your love and support. I know I didn't call home nearly often enough, but I know you all have been thinking of me and I have been thinking of you too. I'll be home soon!

Thank you also to everyone involved in making this thesis happen. Specifically, thank you to BC and Grant for being there to bounce ideas off of, and to struggle together through these last four years, and thank you to all of the other senior guys who changed my life in so many ways. Also, to my play testers and everyone else who aided or prayed for my thesis, I really thank God for you and your willingness to serve. Even something so small doesn't go unnoticed.

Of course, my gratitude doesn't stop there. I'd be forgetting a lot of important people and important moments in my life if I didn't remember the Dod and PCF communities! Dod has had countless visitors this year, and as averse to distraction as I used to be, I see now that it was an incredible blessing to have so many great people come through the door every day. I think I finally get why God made us for community, and I feel very grateful for the community that he has surrounded me with here at Princeton. You all know who you are – know also that you're the reason I thank God every day, the reason I long for heaven, and the reason I can really call this place home. Forget GPA, forget my classes, forget this thesis even – you're the reason I am grateful for this year. You're the people I'd like to acknowledge. I echo Paul:

> [3] I thank my God in all my remembrance of you, [4] always in every prayer of mine for you all making my prayer with joy, [5] because of your partnership in the gospel from the first day until now. [6] And I am sure of this, that he who began a good work in you will bring it to completion at the day of Jesus Christ. [7] It is right for me to feel this way about you all, because I hold you in my heart, for you are all partakers with me of grace, both in my imprisonment and in the defense and confirmation of the gospel. [8] For God is my witness, how I yearn for you all with the affection of Christ Jesus. [9] And it is my prayer that your love may abound more and more, with knowledge and all discernment, [10] so that you may approve what is excellent, and so be pure and blameless for the day of Christ, [11] filled with the fruit of righteousness that comes through Jesus Christ, to the glory and praise of God.

- Philippians 1, verses 3-11

Finally, I thank God for his steadfast love and faithfulness. I dedicate this work to him, and not men, knowing I am serving the Lord Christ.

Contents

Chapter 1: Introduction 7

Why do we need a better opponent? 7

What does a better opponent look like? 9

Why is machine learning interesting for game AI? 10

Chapter 2: Game Setup 13

Genre & Gameplay 13

Game Modes 14

Chapter 3: AI Implementation 19

Neural Networks 19

Reinforcement Learning 22

Genetic Algorithms 24

Hybrid 27

Chapter 4: Results 28

Training 28

Testing Against Control Opponent 29

Player Review 36

Chapter 5: Discussion & Future Work 44

Works Cited: 49

Chapter 1: Introduction

Why do we need a better opponent?

At the 2006 Game Developers Conference (GDC), lead artificial intelligence (AI) programmer at Monolith Productions, Jeff Orkin, delivered a talk about a new AI system his team had developed for the game *F.E.A.R.* In place of what had come before – "A* and Finite State Machines (FSMs)"[1] according to him – they created a planning system that AI agents could use to determine the best actions to take to meet in-game goals.

The system, called Goal-Oriented Action Planning (GOAP), was a huge step forward in making AI that behaved realistically. It allowed for dynamically choosing actions based on the situation, and even reduced programming complexity by doing away with the large-scale FSMs developers had been using before. He noted,

> With each layer of realism, the behavior gets more and more complex. The complexity required for today's AAA titles is getting unmanageable. Damian Isla's GDC 2005 paper about managing complexity in the Halo 2 systems gives further evidence that is a problem all developers are facing [Isla 2005].[2]

There he cites another developer, Damian Isla, who also contributed to the sweeping changes in game AI happening around that time. Isla used a different system called Behavior Trees for the landmark game *Halo 2* by Bungie Studios (published by Microsoft Games in 2004). This system also reduced complexity for programmers, and allowed AI agents in game to behave more realistically than ever before. However, at the recent 2017 GDC AI Summit, AI developer Ben Sunshine-Hill commented on the state of the industry, claiming "And here we are, more than a decade down the road [from *Halo 2*], and guess what? We're still using behavior trees"[3]. He continues, "I believe we are killing game AI as a creative discipline"[4]. Though behavior trees were sufficient for action games for many years, they may be reaching the end of their life in

[1] Jeff Orkin, "Three States and a Plan: The A.I. of F.E.A.R.," *Proceedings of the Game Developers Conference,* (2005), 1. http://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf
[2] Ibid., 3.
[3] "Turing Tantrums 2017: AI Devs Rant," Proceedings of the Game Developers Conference, January 6, 2018, https://www.youtube.com/watch?v=i00Yu2zHbbU
[4] Ibid.

terms of realism and novelty they can offer to gamers. Sunshine-Hill ended his talk with a call for continued growth and innovation in the field of AI, and fellow developers echoed his sentiment.

Turning back to Orkin, it's clear there is a need for continued innovation in game AI. He says in his 2006 paper,

> In the early generations of shooters, such as Shogo (1998) players were happy if the A.I. noticed them at all and started attacking. By 2000, players wanted more, so we started seeing A.I. that can take cover, and even flip over furniture to create their own cover. In No One Lives Forever (NOLF) 1 and 2, A.I. find appropriate cover positions from enemy fire, and then pop randomly in and out like a shooting gallery. Today, players expect more realism, to complement the realism of the physics and lighting in the environments.[5]

That was in 2006. Today, games are being produced with photorealistic graphics and advanced physics engines, but the same AI used in 2004. Clearly it is time for some innovation if the industry wants to sustain its growth. Players expect realism, and if behavior trees cannot keep pace with the graphics and other game features, they have to be improved on.

Further, players have been looking for novelty in their games, and this is quite clear from the types of games that have been doing well in the past few years. Independently produced games are gaining more traction than ever with platforms like Steam allowing small developers to get their games in front of large audiences.[6] Other games from small developers have seen huge success, like *Minecraft*. Not only are these games typically cheaper than AAA titles, they also take risks that those large games do not. Financially speaking, it is risky for a large company like EA to take a gamble on a game with a weird story and strange mechanics, when tried and true titles and genres have been making them money for so long. For players however, the tried and true is becoming tiresome, as evidenced by flagging sales of long-running series, such as *Call of Duty*.[7] In their place, there is becoming much more demand for novelty in games, making now a very strategic time to be thinking about new ways of doing game AI.

[5] Orkin, *Three States and a Plan*, 3.
[6] Richard Cobbett, "From shareware superstars to the Steam goldrush", PC Gamer, https://www.pcgamer.com/from-shareware-superstars-to-the-steam-gold-rush-how-indie-conquered-the-pc/
[7] "All time unit sales of selected games in Call of Duty franchise", Statista, https://www.statista.com/statistics/321374/global-all-time-unit-sales-call-of-duty-games/

What does a better opponent look like?

Game AI has transformed immensely from its humble beginnings in FSMs through GOAP and on to behavior trees today. Increasingly complex combinations of actions and states are achievable without needing millions of lines of code. Players can immerse themselves in game worlds and feel like NPCs behave realistically, however, there are still drawbacks to these fully-deterministic systems (complex though they may be). Dr. Harbing Lou commented in a blog post for the Harvard Graduate School of Arts and Sciences that "All NPCs' behaviors are pre-programmed, so after playing an FSM-based game a few times, a player may lose interest."[8] He makes an important point that applies to all deterministic AI systems, and it also ties in with what Jeff Orkin stated regarding AI keeping pace with the rest of game development. If every action game has relied on behavior trees for its AI since *Halo 2*, there will come a point when players will lose interest in the next game coming out, knowing that the AI will behave according to the same patterns that it did in the last game. When the enemies are so crucial to the gameplay, they must be interesting and add to the experience, rather than make the player feel like they are having the same experience they have had with action games for 10 years.

Dr. Lou continues, and offers a potential improvement for game AI: "Although AI designers in the 1990s worked very hard to make NPCs look intelligent, these characters lacked one very important trait: the ability to learn."[9] He suggests that this feature is what distinguishes a human player from an AI, and can be the difference between a novel experience and a boring one. He sums up by saying, "In the future, AI development in video games will most likely not focus on making more powerful NPCs to more efficiently defeat human players. Instead, development will focus on how to generate a better and more unique user experience."[10] This is the key to good game AI – it is not about defeating the player as quickly as possible, but instead it is about making the experience as rich and enjoyable as possible.

So what makes a gaming experience enjoyable? There are many factors, but a very important one in games is progression. Players like to feel as though they're improving or growing in the game in some way, whether that be completing levels, leveling up a character, or

[8] Harbing Lou, *AI in Video Games: Toward a More Intelligent Game* (blog). August 28, 2017. http://sitn.hms.harvard.edu/flash/2017/ai-video-games-toward-intelligent-game/
[9] Ibid.
[10] Ibid.

moving forward in a story. Challenge needs to stay about constant too, so as the player gets better at the game, or levels up their character, it doesn't become too easy and boring.

When using traditional AI techniques that are fixed and deterministic, not only can they lose player interest, they can also take up more and more developer time. The only way to add challenge and variety with AI agents in these systems is to add many different types of agents to the game, or to change some surface-level aspect of that agent. For example, in a first-person shooter game like *Call of Duty*, enemy soldiers often behave in the same way. This is because their backing AI is the same and does not change during the game. The way that enemies are kept interesting is by adding different types, such as a close-combat infantry, long-distance sniper, or heavily-armored soldiers for instance. Alternately, surface-level attributes are changed, such as what the infantry has for a weapon, body-armor, or camouflage. This means creating a lot of content and a lot of code to provide this feeling of progression, which goes back to the problem that Orkin described regarding complexity. On top of that, it costs developer hours, which can translate to many real dollars for a company.

Ideally, a new AI system for opponents will be able to provide progression for the game without adding complexity or costing more money. Additionally, it needs to meet the requirements normally posed on game AI systems: they must be fun to play against and believable for the player so the experience of immersion isn't disrupted. In terms of technical requirements, it must also run smoothly and reliably on consumer computers (or game consoles).

Altogether, this new opponent looks like this: it can change over the course of the game, it isn't hard or messy to implement, it is computationally feasible to run, and above all, it is fun and believable to play against. The next section will discuss how this kind of opponent might be implemented using a machine learning model. Specifically, it is implemented in this thesis as a neural network. Chapter 3 builds upon the next section and discusses ways that the neural network could be trained for a game. The final two chapters detail the results of training and testing, as well as future work in the area.

Why is machine learning interesting for game AI?

Recently, there has been a surge of interest in machine learning. John Giannandrea, former Vice President of Engineering at Google, commented "We've seen extraordinary results in fields that hadn't really moved the needle for many years. I think we're in an AI spring right now"[11]. Andrea Schiel, from the 2017 GDC AI Summit, remarked that Giannandrea may be seeing the spring, but for game AI, it's still just "slush season"[12]. The purpose of this thesis is to start thinking about how to push games into an AI spring of their own.

Pushing game AI forward is no simple task however. It is easy to ask *why* we need better AI, and slightly harder to ask *what* that AI should look like, but it is very hard to ask *how* the AI should be built. It needs to meet the requirements placed on it in the previous section, including adaptability or built-in progression, simplicity, reasonable computational requirements, and a "fun-factor". If behavior trees are reaching the end of their believability and novelty, it is important to think about how a new system would address those concerns as well.

To address the first point of adaptability and progression, machine learning (ML) models seem like a very promising solution. There's a hope that a well-made model could learn to play over the course of the game, improving with the player and keeping the challenge constant, without the need for extra content and extra code. By foregoing extra code, a ML model could also meet the simplicity requirement and save developer hours. The actual functionality of the code and of the ML model might not be simple, but a dedicated AI person on the team could make it manageable.

In addition to keeping challenge constant, another exciting possibility is the ML model could learn from each individual player as they play, and adapt to their strategies and techniques. This is like what human players do when they play games together. Multiplayer game modes have been well-received by audiences, especially in action or shooter games, and this is evidenced by the release of game series that only have a multiplayer game mode such as *Titanfall*[13] or *Destiny*[14]. If the AI system could produce agents that behave like human players, there is a good chance it would include the "fun-factor" that the system must have as well.

The only remaining quality a ML model must have is reasonable computational requirements such that it can run smoothly on an average consumer computer. This may or may not be attainable with a model, depending on what its architecture is like and how it is trained. The question of training does allow some insight though; because one goal of the AI system is to

[11] Google AI, Google Inc., https://ai.google
[12] *Turing Tantrums 2017.*
[13] Titanfall 2, EA Games Inc., https://www.ea.com/games/titanfall/titanfall-2
[14] Destiny 2, Activision, https://www.destinythegame.com/

have it adapt during gameplay, it needs to work with some form of online training, as opposed to offline or batch training. That means there is no existing dataset, and the model must be able to update during live gameplay. Also, because it's not clear what the model should have as an output at any given time for an AI agent action, supervised learning is out of the question too. The most likely training scheme would have to be some form of online, unsupervised learning, capable of running without lagging on the average consumer computer.

Regarding the actual model being trained, it would have to meet a few requirements. Because gameplay is in some sense a function of whatever is currently happening in the game environment (and maybe what has happened previously), the model must be able to approximate arbitrary functions well. It also must be simple enough for developers to adopt, and able to run on consumer computers. With those requirements in mind, a neural network seems like the ML model to use for modeling the AI agents' behavior. Neural networks are not understood perfectly, but they have been applied in a variety of ways, and unsupervised learning schemes exist, including reinforcement learning and genetic algorithms. Their architecture can be made as complicated or as simple as the developer wants, and this allows the developer to tradeoff between modeling power and runtime. With some tweaking, a neural network architecture could be built that allows the agents to play convincingly, while still running at a normal game speed (typically 30 frames per second). The primary development challenge comes from deciding what that architecture should look like: what are the inputs, what are the outputs, how many layers of neurons are stacked between the input and output, what loss function and transfer functions should be used, etc. Also, a suitable training scheme must be selected, and the bulk of the work in this thesis revolves around answering that question. Three unsupervised training algorithms are tested: reinforcement learning, genetic algorithms, and a hybrid of the two. Chapter 3 will describe the neural network architecture and training methods in more detail.

Chapter 2: Game Setup

Genre & Gameplay

Machine learning-based AI can be developed for a variety of games. Good candidates include strategy games, action and shooter games, or any other type of game that has a discrete set of actions that can be taken at any point in time, where the AI must determine which action is best suited to the current situation. In these types of games, there is a clear end goal for the learning process: figure out *when* (or to some degree, *how*) to perform each action for the most payoff. The loss or error of the learned function must also be trackable, and in these candidate games it is trackable by looking at the performance of the AI agents according to the objectives of the game.

Of course, there are many poor candidates for this type of AI model as well. For instance, the strategy game *Civilization*[15] may be far too complicated for a machine learning model to effectively learn in a reasonable amount of time. Not only are there actions that the model must learn (make units, make buildings, launch attacks, set up defenses, etc.), it must learn how to balance those actions across many different cities and against many opponents at once. On top of that, there are some games where there simply isn't enough complexity for an ML model to be useful. Examples might include platformers where the enemies are intended to be simple and display only one mechanic at a time, or a narrative-driven game where gameplay mechanics are not the focus.

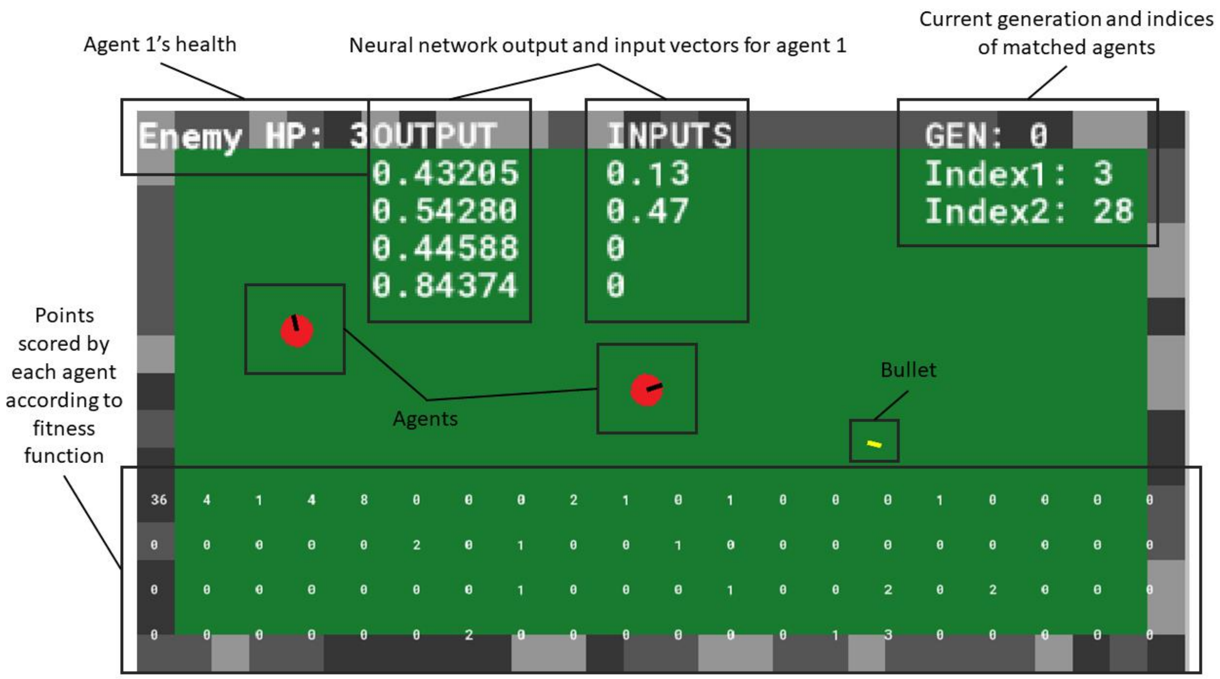
For this project, a 2D, top-down shooter game was selected because it provides a reasonable complexity tradeoff; it is complex enough to benefit from the realism provided by a learning model, but simple enough to be learnable and therefore played competently for the sake of believability. The set of output actions for the model is simple – because it is 2D, aiming amounts to lining up the direction the agent is facing with the direction toward the target and then firing. Moving is as simple as going forward in the current facing direction. Altogether, that is four actions – adjust the facing angle left, adjust it to the right, shoot, and go forward. The interesting aspect of the model is that it should theoretically be able to learn how to combine those output actions to best evade and attack the target.

[15] Civilization VI, 2K Games, https://civilization.com/

Gameplay, with a player involved, is quite simple. The player controls their avatar with the keyboard, including moving and reloading their gun, and they aim and shoot with the mouse. The player tries to shoot the enemies, and gets a kill if the enemy is hit three times. At the same time, the player tries to avoid being shot by the enemies. Wall objects keep the player and enemies contained, and provide some cover from incoming bullets. Ammo refills lie in the corners of the screen as well for the player to pick up. These extra details (walls and ammo) were necessary to simulate a real game setting, and having to collect ammo kept the player on the move, allowing the AI agents to shoot and pursue as opposed to having the player hide behind cover excessively.

Game Modes

The game is separated into two main modes: training and testing (testing can also can be thought of as single player). In training mode, the game is not actually playable; instead, it runs the unsupervised learning methods used to train the neural network underlying the AI agents. In the single player mode, the player controls their avatar and competes against the AI agents that have been trained. The agents continue to learn as the player plays using their respective training methods. This mode is also used to gauge player perception of the AI. A description and screen shot of both modes follow.


Agent 1's health
Neural network output and input vectors for agent 1
Current generation and indices of matched agents
Points scored by each agent according to fitness function
Agents
Bullet

```
Enemy HP: 3  OUTPUT     INPUTS         GEN: 0
             0.43205    0.13           Index1: 3
             0.54280    0.47           Index2: 28
             0.44588    0
             0.84374    0

36 4 1 4 8 0 0 0 2 1 0 1 0 0 0 1 0 0 0 0
0  0 0 0 0 2 0 1 0 0 1 0 0 0 0 0 0 0 0 0
0  0 0 0 0 0 0 1 0 0 0 1 0 0 2 0 2 0 0 0
0  0 0 0 0 0 2 0 0 0 0 0 0 1 3 0 0 0 0 0
```

Figure 1: The training mode screen. From left to right, top to bottom, this figure depicts the health of agent 1, the neural network outputs and inputs for agent 1, the current generation, the indices of the two agents currently competing, the two agents themselves, a stray bullet, and a grid of points scored by each of the agents in the population.

The figure above depicts the training mode. Training was performed in a similar manner for the ML models, whether genetic algorithm, reinforcement learning, or the hybrid algorithm. Essentially, the training period begins with a selection of randomly generated agents that compete against one another round-robin style. Through competition, the two most fit agents are determined and move on to become the parents of the next generation. For a genetic algorithm, the parents produce the next population in three ways: the best parent is mutated to produce some children, the two parents are crossed over to produce more children, and finally the two parents are crossed over and the resulting child is mutated to produce the rest. For reinforcement learning, the two best neural networks are simply copied an equal number of times and allowed to continue training through the competitions. The benefit of this for reinforcement learning, which doesn't need to have children produced or operate in generations, is that it allows for a variety of training stimuli to shake the network out of locally optimal solutions that it sometimes falls in to. Also, having multiple copies of a good network makes it more likely that at least one will train through some local optima to reach a better solution. Lastly, for the hybrid training algorithm, agents compete and are still mutated and crossed over like in the genetic algorithm setup, however the agents are also able to continue reinforcing actions while they compete. Chapter 3 will provide a more thorough description of the training process for each algorithm.

To make training feasible from a computational standpoint, there was a tradeoff to make between number of agents in the population, amount of time allowed to a matchup, and conditions for ending a matchup early. Population size was set at 80 after some testing (and this is described in a bit more detail in Chapter 3). Matchups were set to be 40 seconds in duration at the normal 30 game frames per second. This was to allow less competent (but still somewhat competent) agents the time that they needed to score points, while discouraging overly slow agents from scoring the points necessary to move on. In some sense, the time limit favors agents that can act quickly and decisively. Also, to speed up training, especially at the beginning when most agents were not able to hit one another, a timer was set up that would end the match after 20 seconds of inactivity. This operated on the assumption that if an agent could not land a hit in 20 seconds, it would not land a hit in 40 seconds either, so it would be more efficient to simply move on to the next matchup. The game speed was then set to the fastest possible setting (my desktop with an Intel i7-4770k CPU and Nvidia GTX 970 GPU managed around 2000 game frames per second, meaning each generation took just over half an hour to train).

Additionally, data was collected during the training process. The list of points scored by each agent (that is, the fitness of each agent), was recorded at the end of each generation. In Chapter 4, the results section, this data is displayed visually to get a sense for how each generation progressed in fitness. On top of that, the best agent from each generation was matched up against two control opponents (that is, opponents that acted the same way every matchup) to gauge performance against an objective measure. The first control opponent acted like a turret, only turning and shooting at the AI agent periodically. The second control opponent also turned and shot, but moved toward the AI agent at the same time. By recording the number of hits landed and received by each AI agent during these matchups, it is possible to see progress from generation to generation toward some base level of competency at the game.

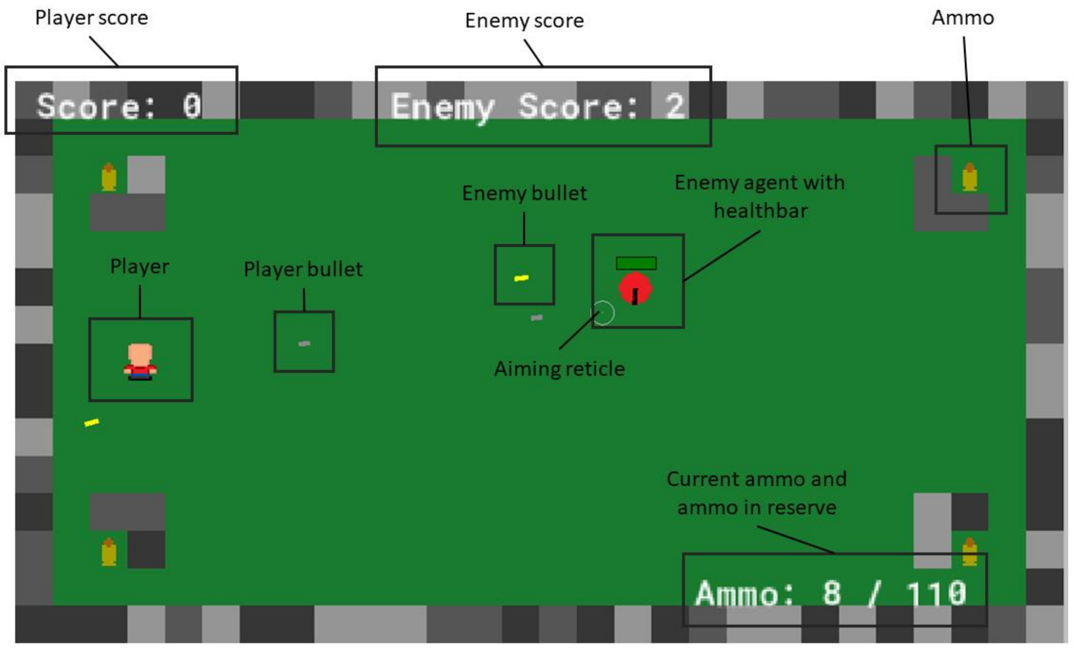


Figure 2: The single player/testing screen. From left to right, top to bottom, this figure depicts the player and enemy score, an ammo refill, the player avatar, player and enemy bullets, the AI agent, and the amount of ammo the player currently has.

Moving on from training, the figure above depicts the testing, or single player, portion of the game. Given what was stated about better game AI in chapter 1, testing needs to reveal three things: does the AI reach a base level of competency at the game, can it improve over time, and is it fun to play against? The data collected during training speaks to two of those points: change in fitness and change in performance against control opponents tells us whether the neural network was improving, and whether it was competent at the game or not. However, getting a full sense of competency, believability, and especially “fun” required having players sit down with the game and play it. Each player testing session lasted 15 minutes, with 5 minutes devoted to competition against agents trained with one of the three training methods. The players were asked questions during the play session to get a sense for player perception of the AI as well.

As they played, the player and AI scores were recorded to get an objective measure of performance. This data is important to compare with player responses to questioning, especially regarding perceived difficulty. If the AI was not able to land any hits on the player, and the player had an easy time defeating the enemies, that suggests that their competency was not sufficient to keep a player engaged. The challenge would be too low and progression through the game would not be satisfying. On the other hand, if the player struggled to land any hits and the AI dominated the match, the game would not be fun to play because the challenge would be too great.

In addition to difficulty, player testing also gave players a chance to decide which enemies were the most enjoyable to play against. Difficulty is a big factor in enjoyment, but as mentioned in Chapter 1, novelty is also very important. Enemies that behave very deterministically become very rote and boring to fight, so it is important to get player feedback on the style of play as well. Difficulty and fitness are not the only factors to take into consideration when deciding which AI setup is best.

Chapter 3: AI Implementation

Neural Networks

As mentioned previously, a machine learning model is the backbone of the game AI system being developed in this thesis. Specifically, the model is an artificial neural network, which can be thought of conceptually as a series of matrices that transform an input vector to an output vector. At each layer in the network, a weight matrix is applied to the vector from the previous layer to produce a new vector. That vector has a non-linear activation function applied to it before being passed to the next matrix. Between the transforms and non-linearities, the neural network acts as a function from some input space to some output space. Training a neural network amounts to figuring out what the weight matrices should be that transform the vectors at each layer in order to get the desired outputs.

A neural network is a good choice for the game AI system because it has properties that are well-suited to the uncertainty of learning to play a game against a human player. Namely, "the family of hypothesis classes of neural networks of polynomial size can suffice for all practical learning tasks", according to the book *Understanding Machine Learning*, by Shai Shalev-Schwartz and Shai Ben-David[16]. That is to say, a neural network can learn any function arbitrarily well provided its structure is sound. Choosing actions during a game is just some function of what is happening in the game at that time, and a neural network has the potential to discover a close approximation to that function – possibly even better than a human would, as evidenced by the Google project AlphaGo.[17]

Shalev-Schwartz and Ben-David continue, "[a neural network] has the minimal approximation error among all hypothesis classes consisting of efficiently implementable predictors"[18]. This statement has two important takeaways – not only is a neural network an 'efficiently implementable predictor', it is arguably the best choice for minimizing 'approximation error' from the class of efficiently implementable predictors. So, the hope with a neural network, given that claim, is that it can learn to play the game and learn it well enough to

[16] Shai Shalev-Shwartz and Shai Ben-David, *Understanding Machine Learning: From Theory to Algorithms* (Cambridge University Press, 2014).
[17] David Silver et al., "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature* 529, no. 7587 (January 27, 2016): 484, https://doi.org/10.1038/nature16961.
[18] Shalev-Shwartz and Ben-David, *Understanding Machine Learning*, 268.

play believably in a finite amount of time. Training time does vary with the size of the neural network and the complexity of the function being learned, so for the sake of training time and in-game run time the network will have to be kept small.

Additionally, a neural network is a good choice for the AI system because there has been wide application of neural networks and there are many training algorithms for them. Three algorithms were mentioned at the end of Chapter 1: reinforcement learning, genetic algorithms, and a hybrid of the two. All three are unsupervised and can be run online (as in, all three can be run while the game plays rather than needing to run separately on collected data). They have all seen success in applications as well. Reinforcement learning was used by AlphaGo to defeat the best Go players in the world, indicating that reinforcement learning is very effective for learning some game functions.[19] Genetic algorithms have also been used by hobbyists to make AI agents that learn to fight.[20]

The architecture of these example networks vary tremendously however. The network for AlphaGo Zero is not laid out explicitly, but it takes in the game board as input, which means the input layer is a vectorized 19x19 matrix, or 361 elements long. It does mention having multiple convolutional layers, as well as layers with rectifier activation functions, so there are clearly multiple layers. Also, training the network requires multiple TPUs, which are specialized pieces of hardware. This suggests the network is quite large.[21] The network used to train the fighting bots is very small by comparison - it only takes 4 inputs, has 5 outputs, and has two 5-node hidden layers in between.

For this project, one requirement is that the AI system can run on average consumer computers. Further, the neural network must be able to learn quickly if it is to be able to learn while in-game. To reach these goals, it is important to have a small network that is sufficient for the type of gameplay that players will find interesting. To that end, the neural network architecture that this project uses is much more like the second, simpler network. After a few iterations of training with different architectures, it became clear that the number of inputs was the most important factor in how fast the network trained and could run. Having many inputs, especially inputs that are commonly 0 or very low values, makes it hard for the network to figure

[19] Silver et al., "Mastering the Game of Go with Deep Neural Networks", 484.
[20] "Combat Learning", Double Zoom, http://doublezoom.free.fr/programmation/AG_Exemple_Fighting.php
[21] David Silver et al., "Mastering the Game of Go without Human Knowledge," *Nature* 550, no. 7676 (October 18, 2017): 354, https://doi.org/10.1038/nature24270.

out what is most important to consider when making gameplay decisions. For instance, the network could track every bullet that might possibly be on screen, but if there is only one or maybe two bullets in flight at any given time, that architecture would have many useless inputs. Not only would the network take time to figure out the importance of those inputs in training, it would take time in-game to do the matrix transforms on those inputs. Clearly, having as few inputs as possible is key to performance needs.

On top of that, keeping the number of hidden neurons and layers low is important for the same reasons. The fewer neurons, the smaller the matrices can be for transforming the vectors, and the fewer weights that would need to be learned to approximate gameplay well. The outputs were also well determined from the setup of the game, and were described in Chapter 2. Essentially, all possible actions the AI agents could take could be summed up in four actions – rotate right, rotate left, move forward, and shoot. That gives four outputs for the neural network. Altogether, the final architecture was determined to be four inputs – angle to target, distance to target, angle to nearest bullet, and distance to nearest input. This is followed by a 10-neuron hidden layer, and then four outputs as described above. With those four inputs, the network should have as much information as it needs to track and shoot its target, as well as enough information to dodge incoming bullets (assuming there are not too many coming at once). This architecture is depicted in figure 3.

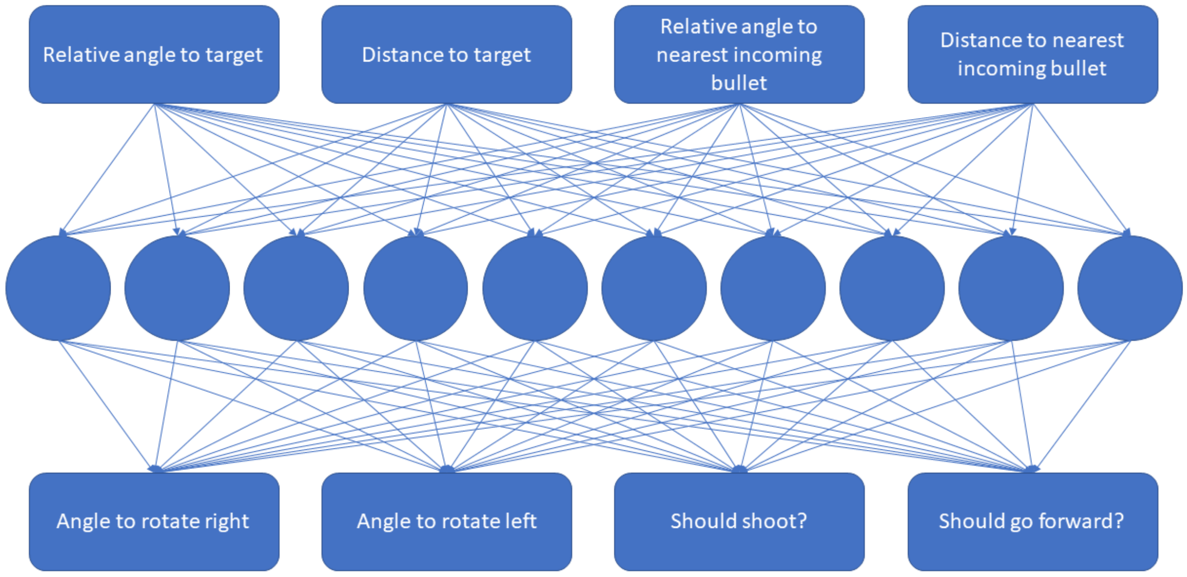


Figure 3: A representation of the neural network architecture used by the AI agents. There is an input layer that takes four inputs: angle and distance to target, and angle and distance to the nearest incoming bullet. Between the input and output, there is a hidden layer with ten neurons. Lastly, there are four outputs that control the rotation of the agent, as well as whether it should move forward or shoot.

Reinforcement Learning

One form of unsupervised learning that has been gaining traction recently is reinforcement learning. Google DeepMind's AlphaGo has demonstrated success against human players in the complex game Go, and it was trained largely through reinforcement learning[22]. A more recent model, AlphaGo Zero, was trained entirely against itself using reinforcement learning, and it has outperformed all previous versions of AlphaGo[23]. What exactly is reinforcement learning though? A rudimentary description might be that reinforcement learning involves making a decision based on the current state of the environment, then determining whether that decision was good or not, and changing the future likelihood of making that same decision to reflect that outcome. To make that a bit more concrete, imagine two AI agents are matched up to fight. One sees that its angle to its target is conducive to shooting (as in, the target would be hit if the agent shot). It chooses to fire, and it lands a hit. This is a good action to take

[22] Silver et al. "Mastering the Game of Go with Deep Neural Networks", 484.
[23] Silver et al., "Mastering the Game of Go without Human Knowledge", 354.

given the situation, so that action should be reinforced to happen more in the future given that same situation. Alternately, if the agent does not take the shot but does something else instead, that other action should probably be negatively reinforced. This same thought process can be applied to any other situations. If the agent gets hit, then it can negatively reinforce whatever it was doing when it got hit, so it chooses to do something else until it eventually learns to dodge. Every action receives feedback, and over time, this (hopefully) results in taking good actions given each possible scenario.

An action in this scheme is selected by looking at which network output is greatest. If the first output is greatest, the agent should rotate right. If the second is greatest, it should rotate left, and so on. This makes reinforcement easier, because only one action needs to be positively or negatively reinforced at a time. Figuring out how to reinforce outputs when multiple actions could be selected gets very messy very quickly. For instance, if an agent decides to rotate right and shoot simultaneously, but it misses, how should each of those decisions be weighed? Was rotating right just as bad a choice in that situation as deciding to shoot? Theoretically, reinforcement learning is a stochastic process, and might be able to figure out a good solution given enough examples, but extended trial and error showed that this method required a long time to train and it often resulted in agents exhibiting divergent behavior, or getting stuck in local minima. To work around that, it was decided that the network should select and reinforce on one action at a time.

Given the method of selecting actions, reinforcement ended up looking like supervised learning, albeit with the labels determined by the effectiveness of a given action. So, like in supervised learning, the network ended up using the backpropagation algorithm to update the weight matrices, but it did this by comparing the outputs to “correct” outputs that were determined by the reinforcement learning framework, as opposed to actual data that was collected beforehand. These ‘correct’ outputs were determined by looking at what resulted from the last series of actions. Each agent could store up to 45 actions that would all be updated on when a reinforcement-worthy event occurred. Because in-game time runs at 30 frames per second though, this is only 1.5 seconds worth of actions. If these actions resulted in a noteworthy outcome, they were updated to either positively or negatively reinforce the last series of actions taken. Each action in that 1.5 second window was reinforced on, even if it did not directly contribute to the outcome. This is because preliminary actions affect the outcome as well. An example of this might be an agent decided to move forward then shoot. Because it moved forward, the shot landed because the target was too close to dodge in time. In this situation, obviously shooting was a good action to take, but so was moving because it led to a successful shot. By reinforcing the whole series of actions, the agents theoretically could learn how to string actions together, rather than just reacting to each individual scenario. To keep the reinforcement from being too extreme though, the amount of reinforcement decreased the further into the stack of actions. The most direct action leading to the reinforcement event was reinforced most, and earlier actions were reinforced less.

Altogether, there were four reinforcement-worthy events – hitting the target, missing, idling, and being hit. Determining how to produce a 'correct' label from those events was done through trial and error (much of the first semester was devoted to this). If an agent landed a hit, this was reinforced strongly positively. Because the point of the game is to shoot and defeat the enemy, the most important thing the AI agents could do is learn to shoot effectively, hence the strong reinforcement. The other three events result in negative reinforcement in varying degrees. Being hit is considered worse than missing or idling, so it results in the strongest negative reinforcement. Idling and missing are treated the same, and are only slightly discouraged.

Genetic Algorithms

As an alternative to reinforcement learning, genetic algorithms offer an attractive approach to training the neural network. They are stochastic, unsupervised methods of training that are loosely based on evolution. Like organisms, the agents being trained with a genetic algorithm have "chromosomes"[24], with the chromosomes being the weights and biases in the neural network. Evolution takes place by mutating and crossing over those chromosomes with other agents that perform well from the population of agents[25]. Performance is measured by a fitness function that the developer must define. Compared to reinforcement learning, this is a much simpler and more open-ended approach to measuring performance because it evaluates on a longer time scale. It determines how an agent performs over its entire lifespan, whereas in reinforcement learning, each individual action has some value and is trained on.

[24] Colin Reeves and Jonathan Rowe, *Genetic Algorithms – Principles and Perspectives*. (Cambridge: Kluwer Academic Publishers, 2002), 20.
[25] Ibid. 25, 31.

Given that the genetic algorithm relies heavily on randomness and does not take the effect of individual actions into account, a developer might be suspicious of its efficacy in training a neural network. For some applications, it is true that using backpropagation to train the neural network is faster and more effective, especially if there is data available to train on and the underlying function being approximated does not have many local optima. However, for applications where there is not much data, or there are many local optima, a genetic algorithm can be an effective means of training.

Researchers David Montana and Lawrence Davis discuss successfully training a neural network using a genetic algorithm in their 1989 paper at the International Joint Conferences on Artificial Intelligence. They say,

> Genetic algorithms should not have the same problem with scaling as backpropagation. One reason for this is that they generally improve the current best candidate monotonically. They do this by keeping the current best individual as part of their population while they search for better candidates. Secondly, genetic algorithms are generally not bothered by local minima. The mutation and crossover operators can step from a valley across a hill to an even lower valley with no more difficulty than descending directly into a valley[26]

This highlights two strengths of the algorithm – though it is stochastic, it will continue moving forward because the current best candidate can only be improved on (or else stay the same if crossover and mutation do not produce anything better). This allows the algorithm to run indefinitely, and given a good fitness function, it will continue to produce better and better agents. Also, the authors say that genetic algorithms are "not bothered by local minima", so the training is not prone to end prematurely like training with backpropagation is. A reinforcement learning scheme can cause an agent to fall into behavior patterns that are good, but not necessarily optimal or interesting, because of the large number of local optima in the gameplay function. Often, tricks and heuristics are needed to get the backpropagation-based algorithm out of the local optima, but Montana and Davis suggest that this can be avoided entirely with a genetic algorithm.

[26] David Montana and Lawrence Davis. "Training Feedforward Neural Networks Using Genetic Algorithms," *International Joint Conferences on Artificial Intelligence*, (1989), 763. https://www.ijcai.org/Proceedings/89-1/Papers/122.pdf

For this project, the implementation is meant to be simple, yet robust enough to handle basic gameplay elements of the test game in a convincing manner. An initial population of 80 agents was selected to start training. Trial and error showed that too few agents in a population (around 50) led the training to start converging towards a poor solution initially, whereas too many agents (greater than 100) made each generation last too long for training to be feasible. The resulting tradeoff was between genetic diversity, and speed of training. From there, agents would fight one another round-robin style, completing 79 matches. The fitness function is simple: 1 point for each bullet landed, and 3 bonus points for winning the match before the timer ran out. In total, an agent could get 6 points in a single match. This function was used to encourage good aim by awarding points for hits landed (plus agents in the early generations were not very likely to win before the timer ran out), but also to award bigger-picture strategy by giving many points for winning. The idea is that the winner is likely a good shooter, but also good at dodging or hunting the opponent down to land 3 hits in the given time. The two best agents from each generation were selected to be parents of the next generation, which is known as tournament-style selection (as opposed to roulette-wheel) [27].

To produce the next generation, the parents were mutated and crossed over in different ways. The first two agents produced were clones of the two parents. This ensured that the best neural networks trained so far would remain in case all children were less fit than the parents. From there, 26 agents were produced by mutating the best parent. Mutations are simple in this context – a few randomly chosen weights or biases in the net get replaced by a random number between 0 and 1. From there, 26 crossovers of the two parents were produced, where crossing over consists of randomly picking weight and biases from one of the two parents and filling a new neural net with those values. Lastly, 26 agents were produced that were crossed over and mutated. The three different methods were selected primarily because it is not clear which would be most effective, so this allows for a variety of approaches.[28]

Determining when to end training is the most difficult question to answer, considering the algorithm can run indefinitely. [29] For this project, the idea is to have agents that exhibit a base-level of competency that can then learn as they play, and the act of training itself is to show that

[27] Reeves and Rowe, *Genetic Algorithms,* 32.
[28] Ibid. 31.
[29] Ibid. 30.

the neural network model is able to learn a gameplay function. As such, training can be terminated as soon as agents appear able to shoot accurately most of the time, and dodge some of the time. This is mostly a qualitative observation, and not rooted in mathematical or learning theory.

Hybrid

A third possible training method for the neural network is a combination of the two approaches. Reinforcement learning provides feedback to the network on individual actions, thereby optimizing in the short term for each scenario the agent encounters. The genetic algorithm, on the other hand, allows the agents to act on a longer time scale and optimizes for that time scale. Together, it seems the two methods would encourage a well-rounded AI agent, capable of acting tactically in combat scenarios, while still executing a longer-term strategy.

To make the hybrid method work, agents would be set up to train in the same format as the genetic algorithm, still crossing over and mutating, but would also be equipped to do backpropagation on each action they take. Actions would be stored on a slightly shorter time scale (10 frames as opposed to 45), to play to the strength of each method. Because the genetic algorithm is already optimizing for long-term play, the reinforcement learning would be geared to train on only the most immediate actions taken before a consequence.

Chapter 4: Results

This section collects and displays the data gathered from training and testing the AI. It is broken into three sections: a section to show the change in fitness of the neural network over time during training, a section for testing against control opponents to get an objective view on the baseline competency of the neural networks at the game, and a section for player testing and review, to see what player perceptions of the AI were. The third section is arguably the most important as any training methods and neural network architectures are useless if the players do not enjoy the game.

Training

The following three graphs summarize the fitness of the population at each generation for the genetic, reinforcement, and hybrid methods of training. They are box and whisker plots of the points scored by all agents in the population at that generation. The average number of points scored according to the fitness function is indicated by the orange marker in each box. The tighter the box and whiskers are, the less variance there was in the population in terms of fitness. Also, by following along the mean, we can get a sense of how the neural network was becoming fitter over time.

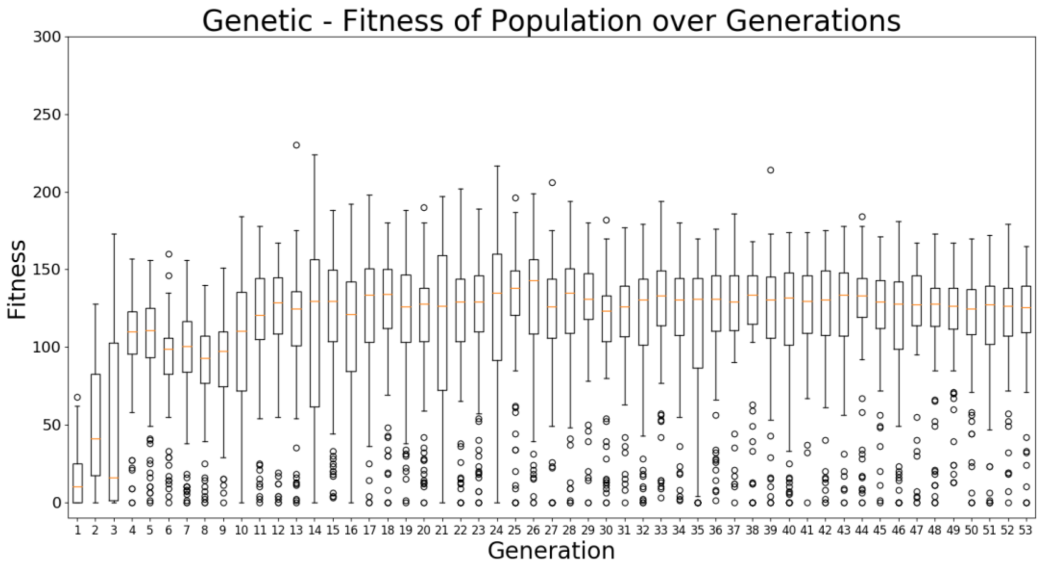


Figure 4

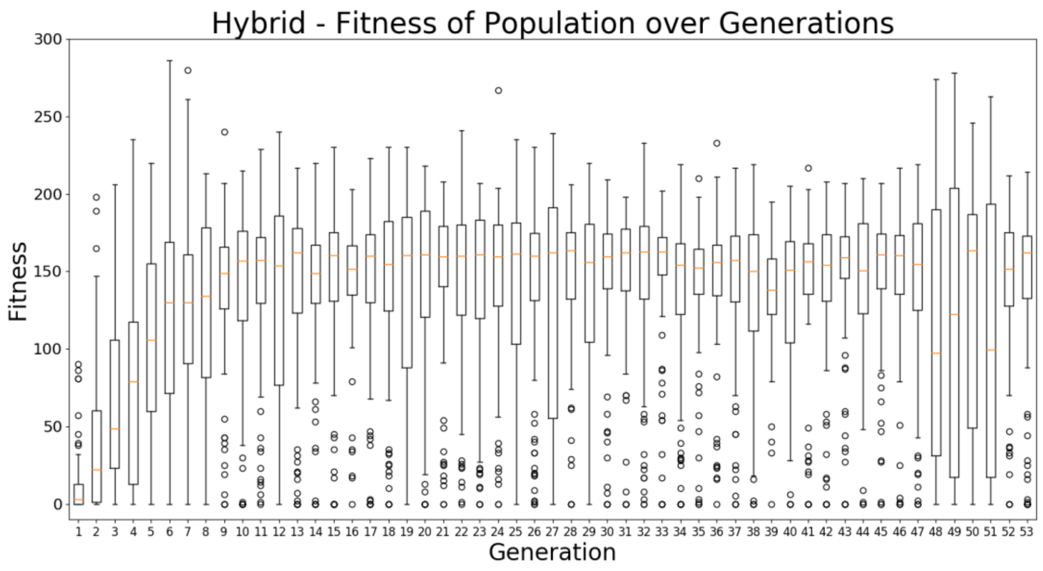


Figure 5

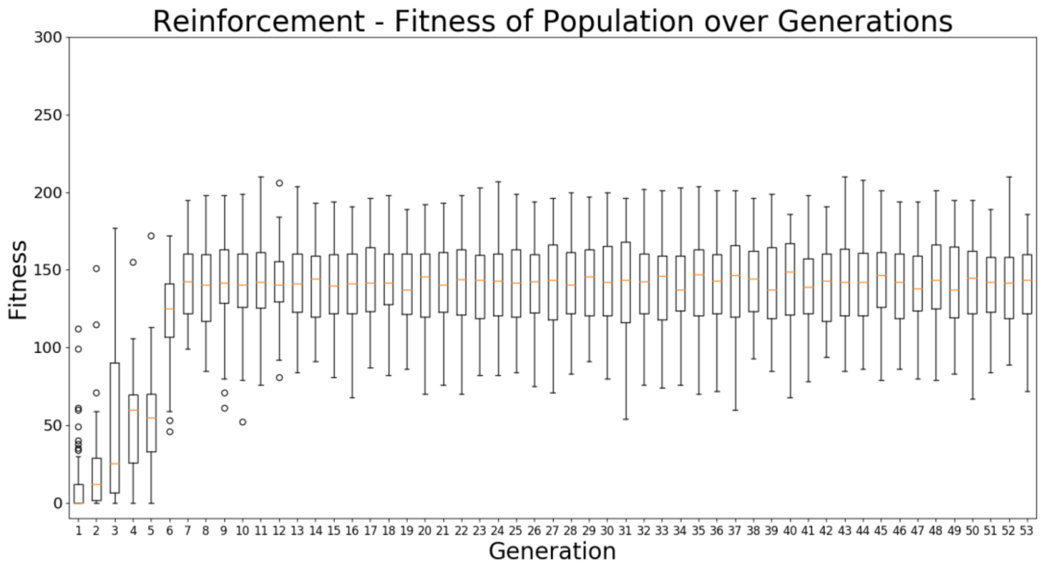


Figure 6

Testing Against Control Opponent

This section summarizes the performance of the neural networks against two types of control opponent. The first opponent acted like a turret – it was stationary and simply turned and shot at the AI agent. The second opponent still turned and shot at the AI agent, but it also moved slowly in the direction of the agent. Both control opponents exhibit very simple competency at the game, and as training progressed, the goal was to see the AI agents perform better and better against them. Seeing that trend would confirm that the neural network was learning and might be capable of baseline competency at the game. Specifically, progress was measured by observing the score of both the control opponent and the AI agent during their 30 second matchup. This was characterized as "hits" landed by the AI agent, and "injuries" sustained by the agent. The graphs below show the number of hits and injuries recorded at each time step in the game, and each line represents a different generation (striding by 5 generations each time so there are not 50 plots on one figure). The alpha of the line indicates what generation it represents. Lighter lines took place earlier in training, and darker lines occurred later. The expected trend for hits is that later generations would score more, and the graph of the score over time would be linear, as the training favors consistent and quick-scoring agents. For injuries, the expected trend is about the same, except the slope should be decreasing over time because the agents should learn to avoid bullets. The graphs are all scaled to the same y-values so it is easy to compare the number of hits and injuries received by all three methods across both types of control opponent.

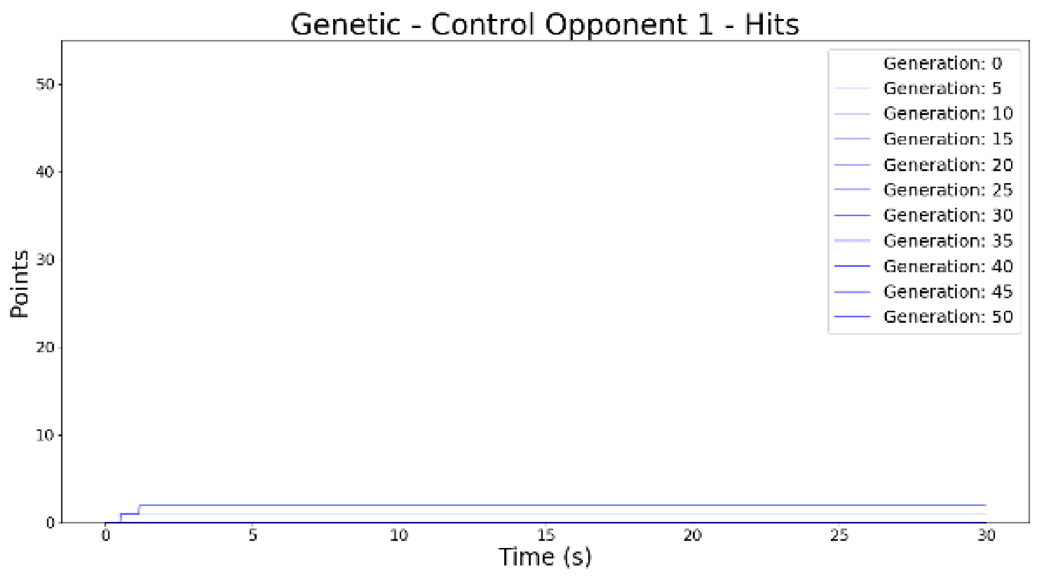


Figure 7

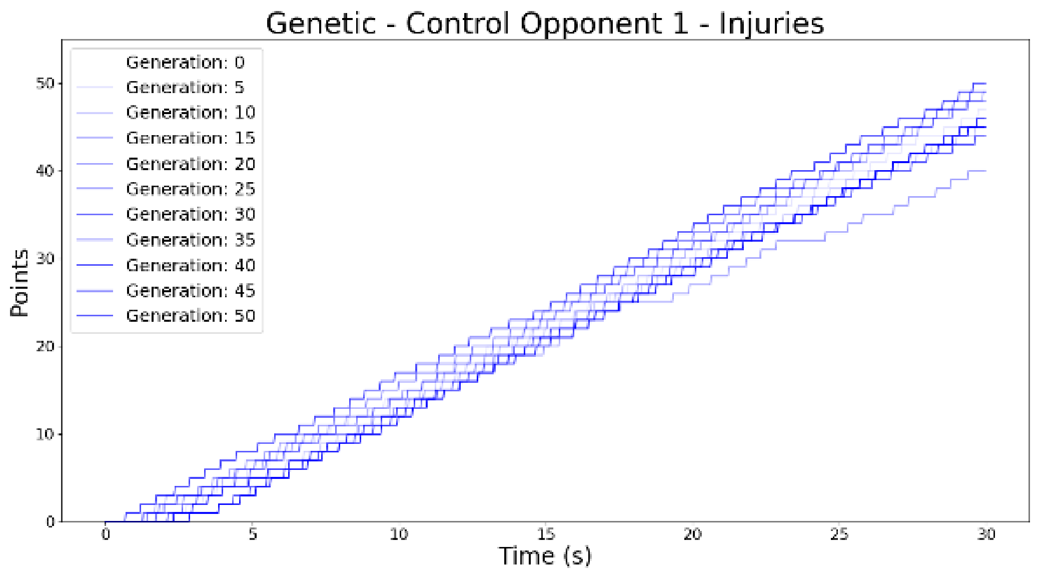


Figure 8

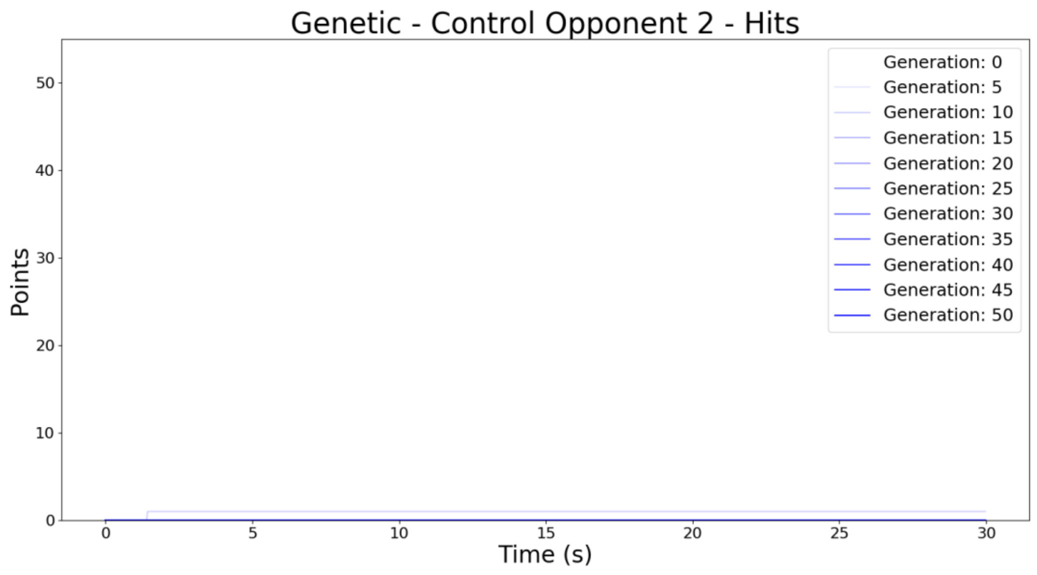


Figure 9

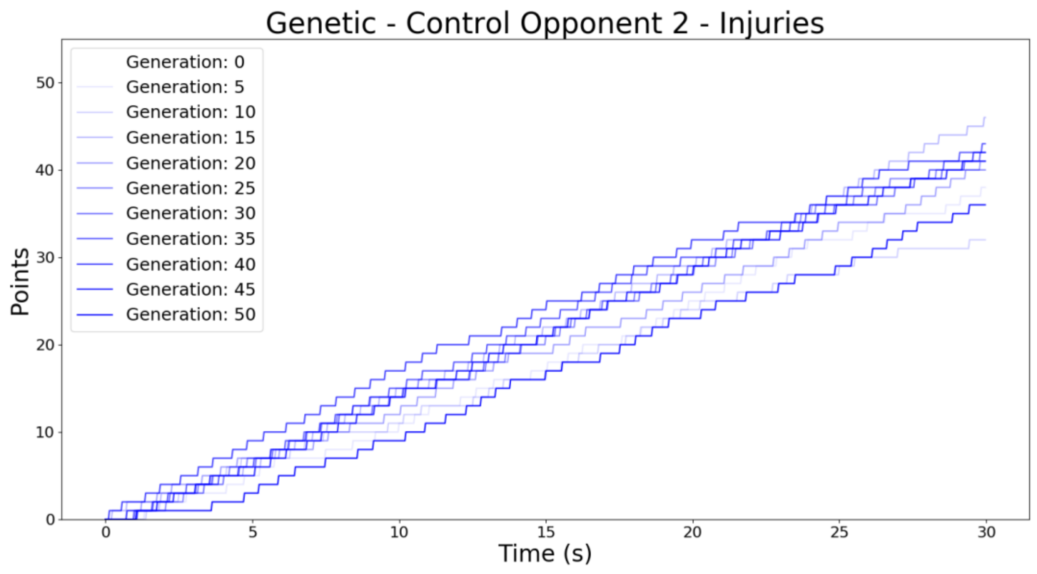


Figure 10

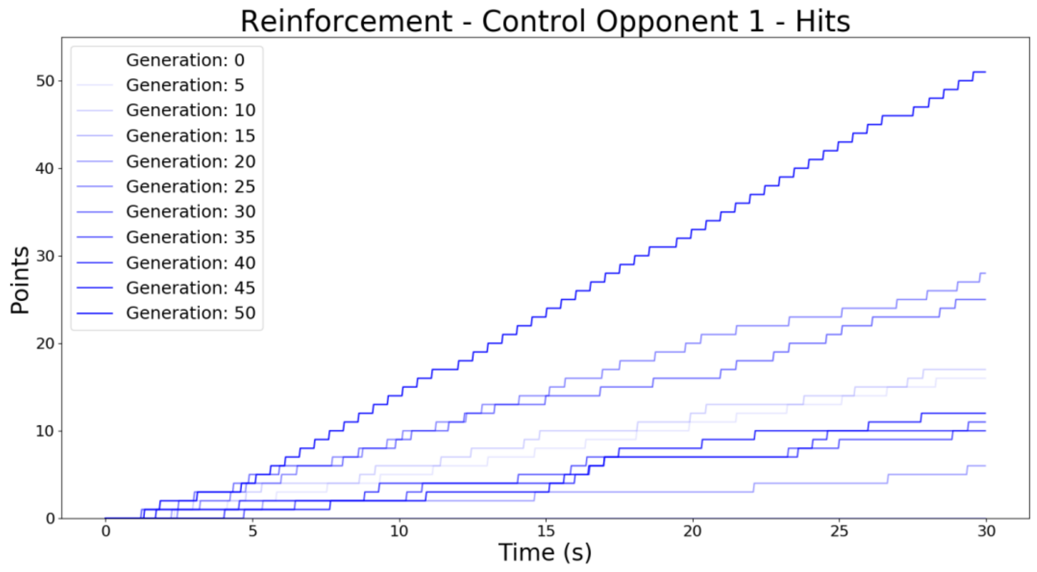


Figure 11

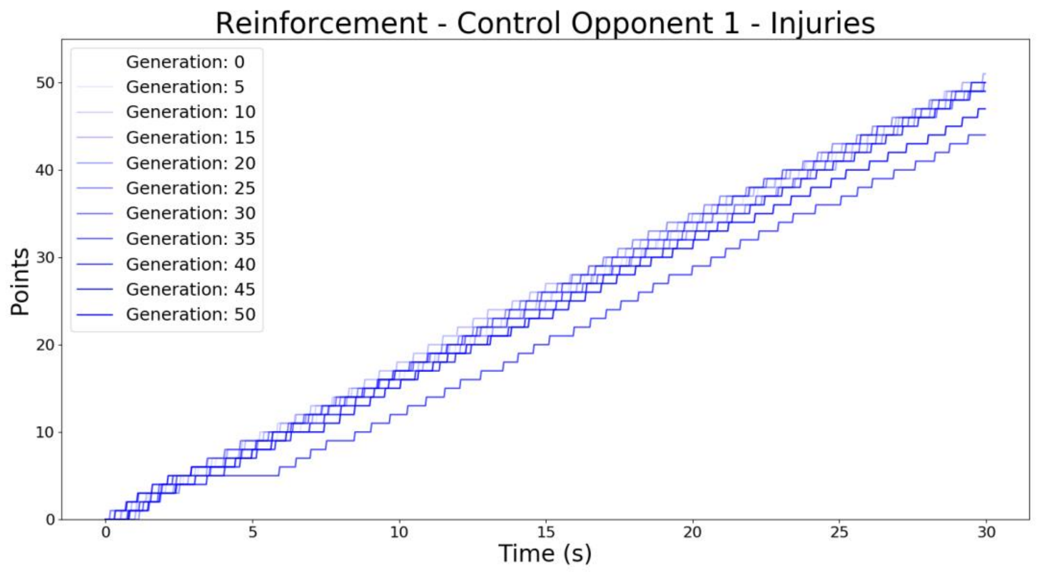


Figure 12

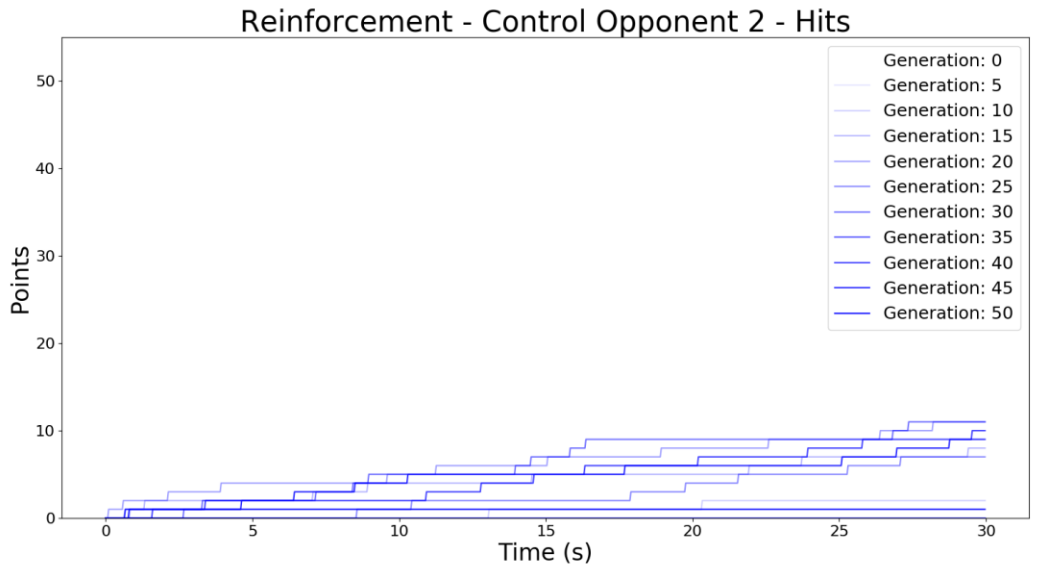


Figure 13

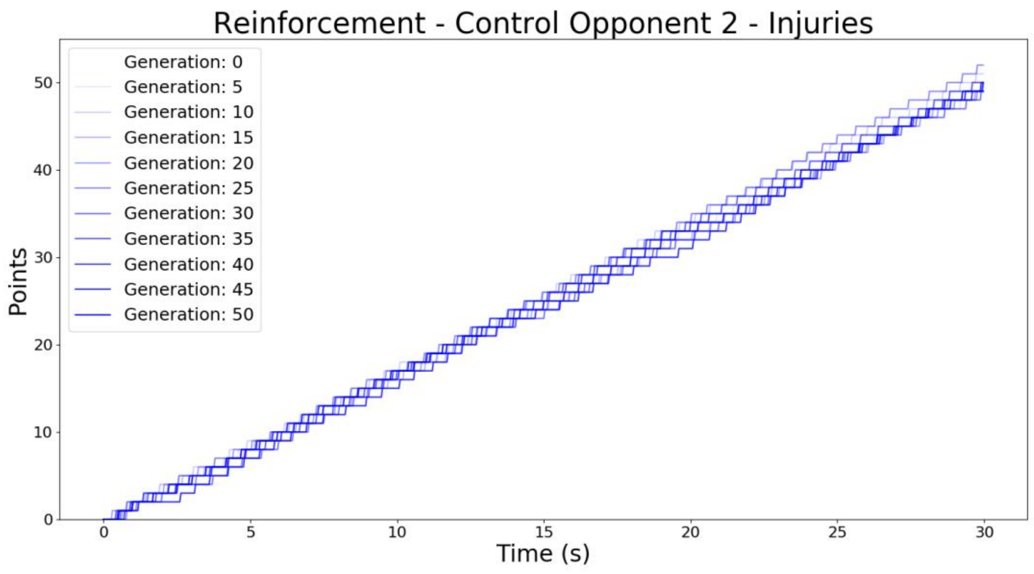


Figure 14

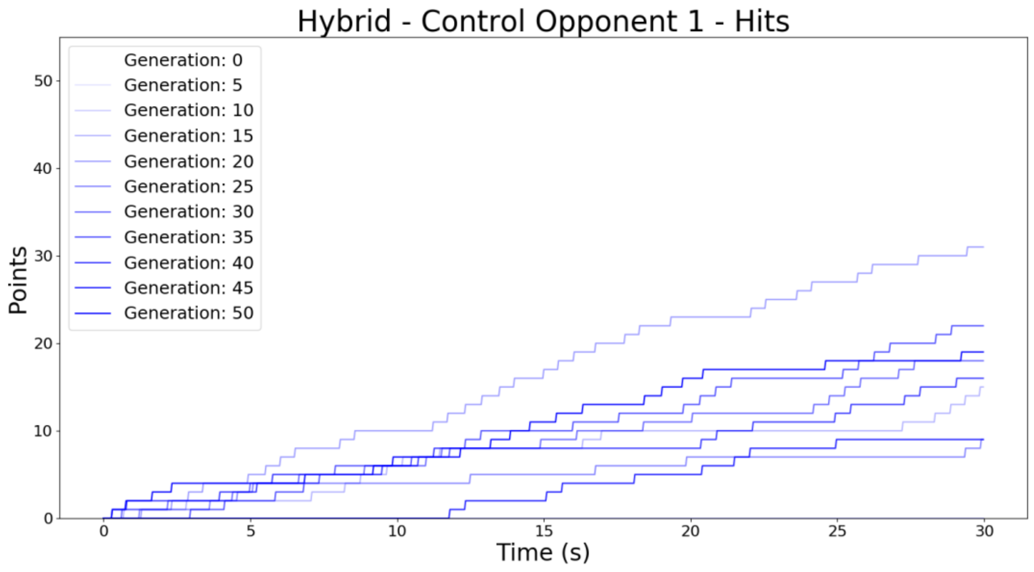


Figure 15

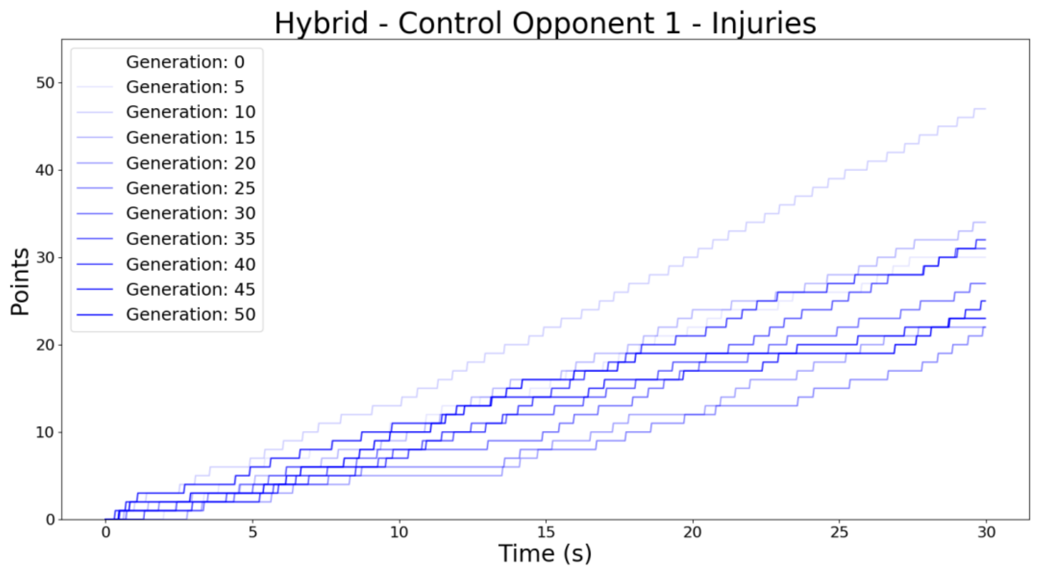


Figure 16

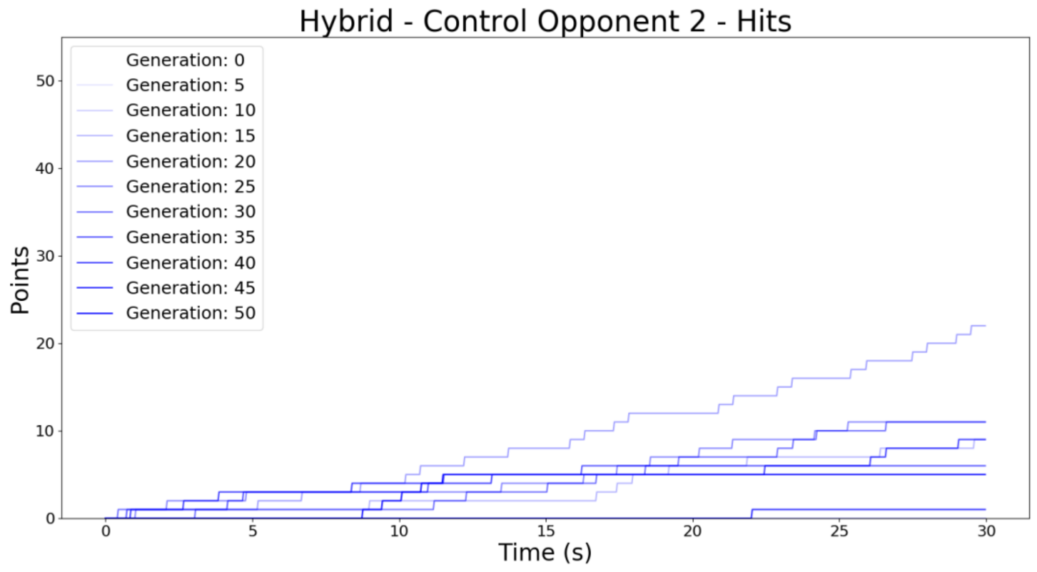


Figure 17

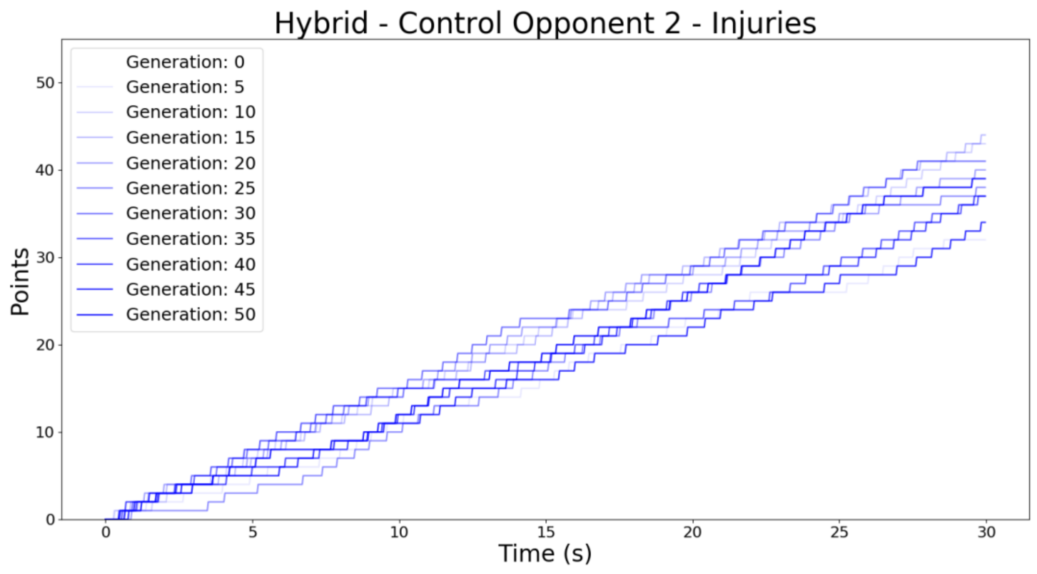


Figure 18

Player Review

The player review section details the player and agent performance during a 15-minute play session. The sessions were divided into three 5-minute chunks, with each chunk devoted to playing against agents trained using one of the three methods. Each method is given its own colored line on the plots of player and enemy score, so that differences in scoring can be seen between the three training methods. These plots serve to corroborate player reported perceptions of difficulty and enemy behavior. The graphs are not scaled to the same y-values because comparisons between players is not as important as comparisons between the methods for each individual player. Each play tester has two graphs – a graph of player score (how many enemies the player defeated) and enemy score (how many times the enemy agent hit the player). Additionally, there is a table summarizing player responses to questions after the graphs.

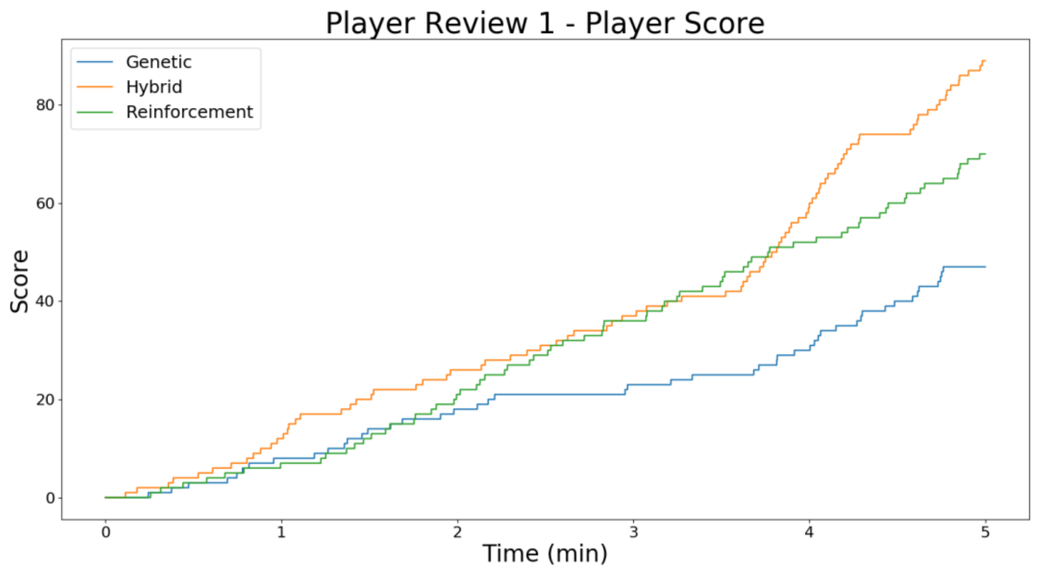


Figure 19

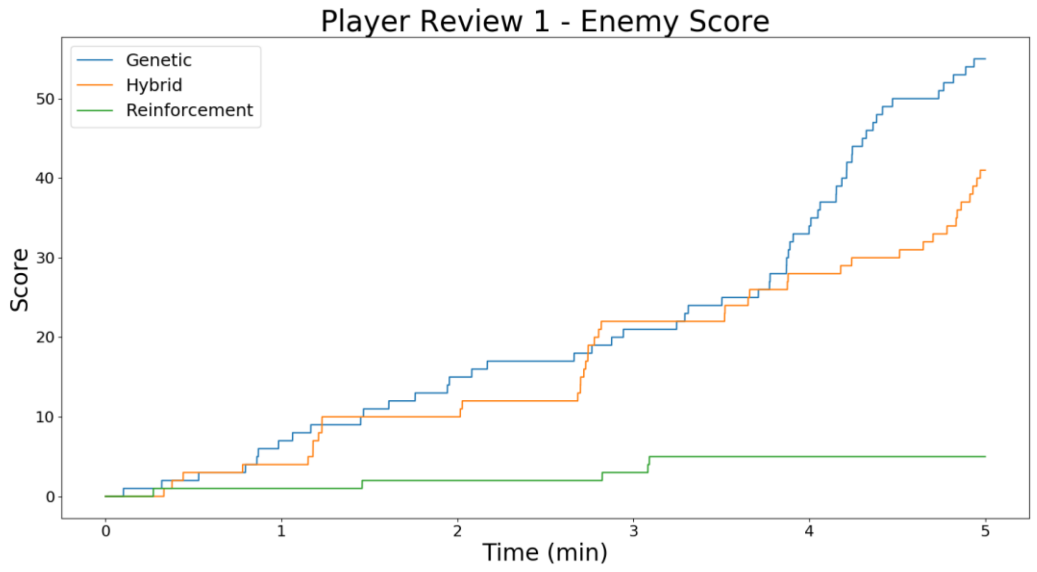


Figure 20

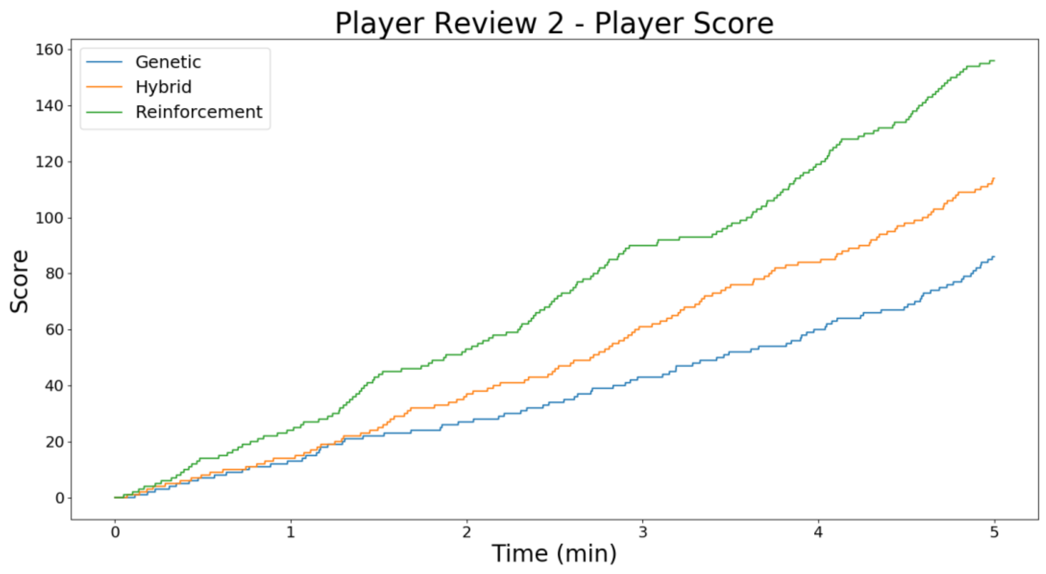


Figure 21

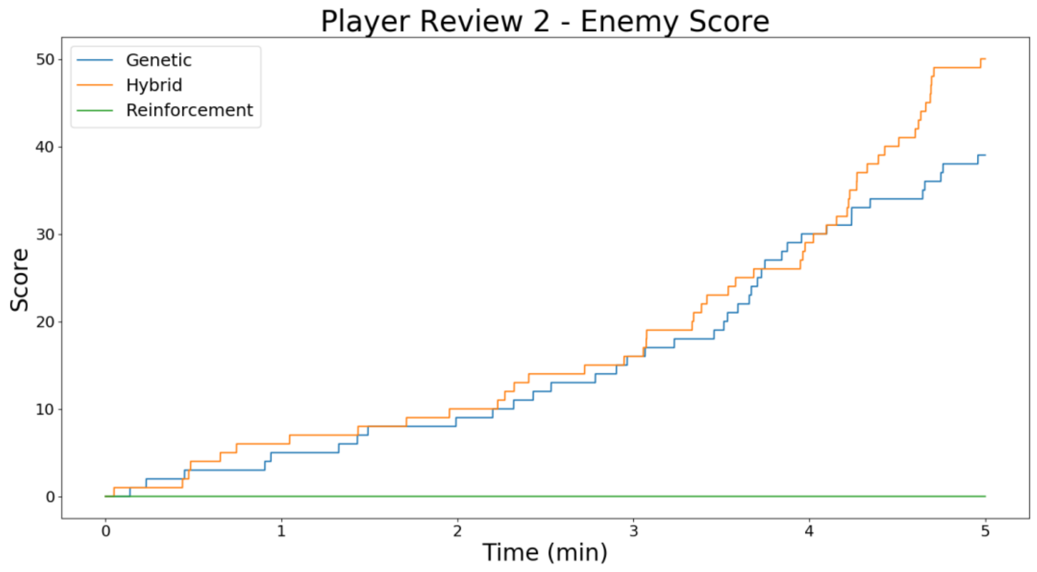


Figure 22

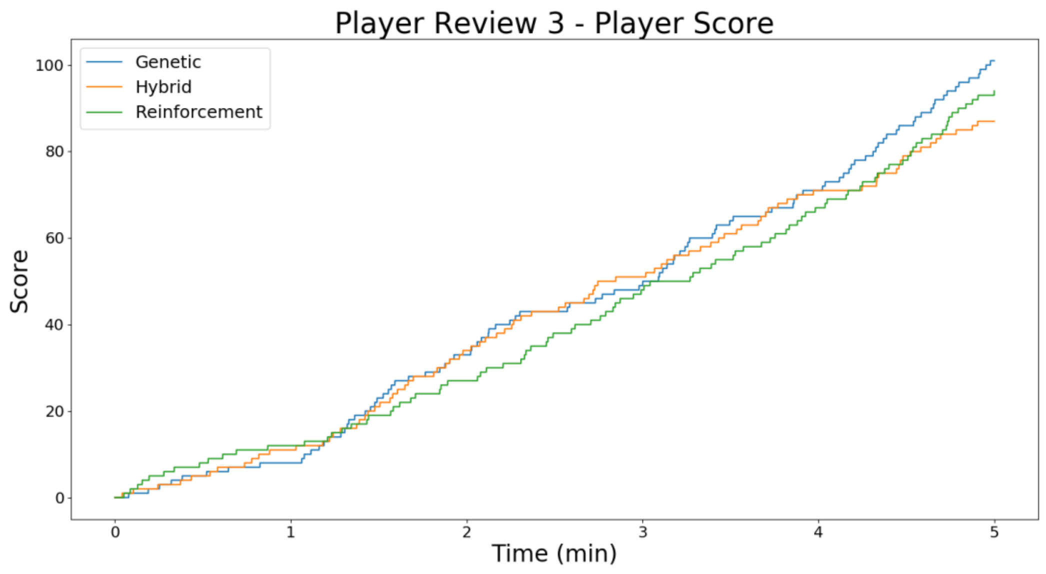


Figure 23

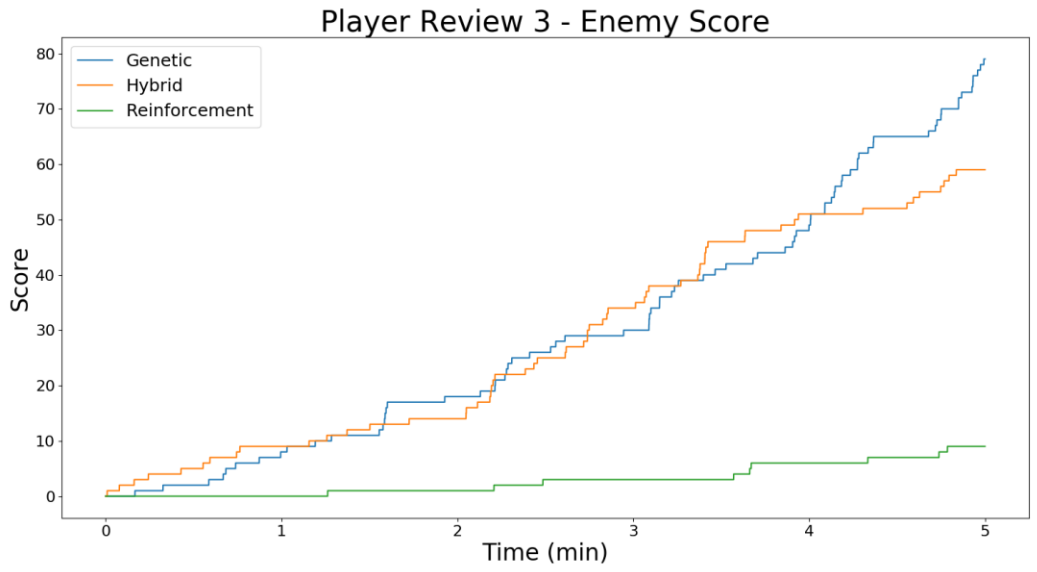


Figure 24

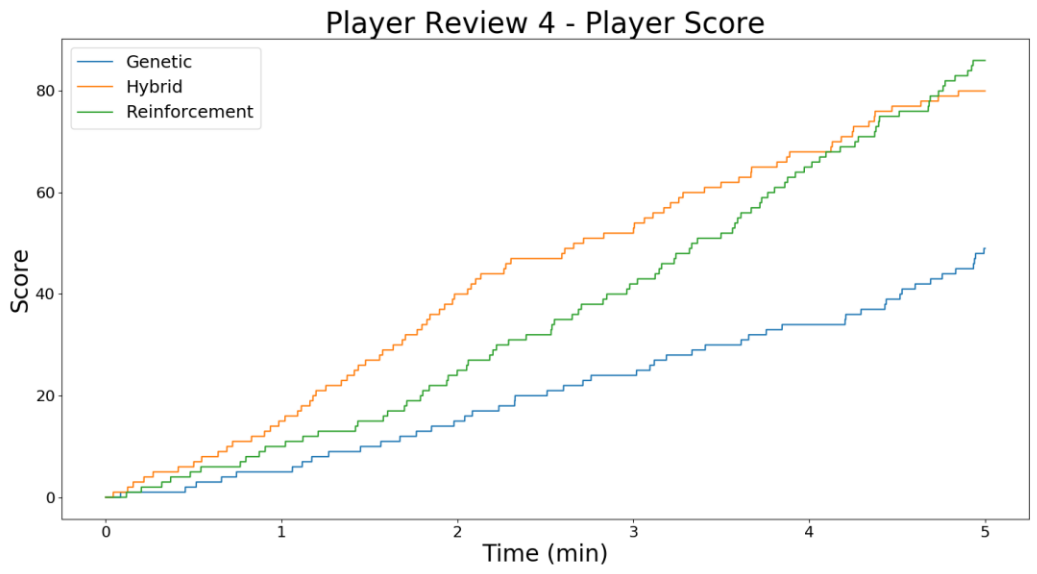


Figure 25

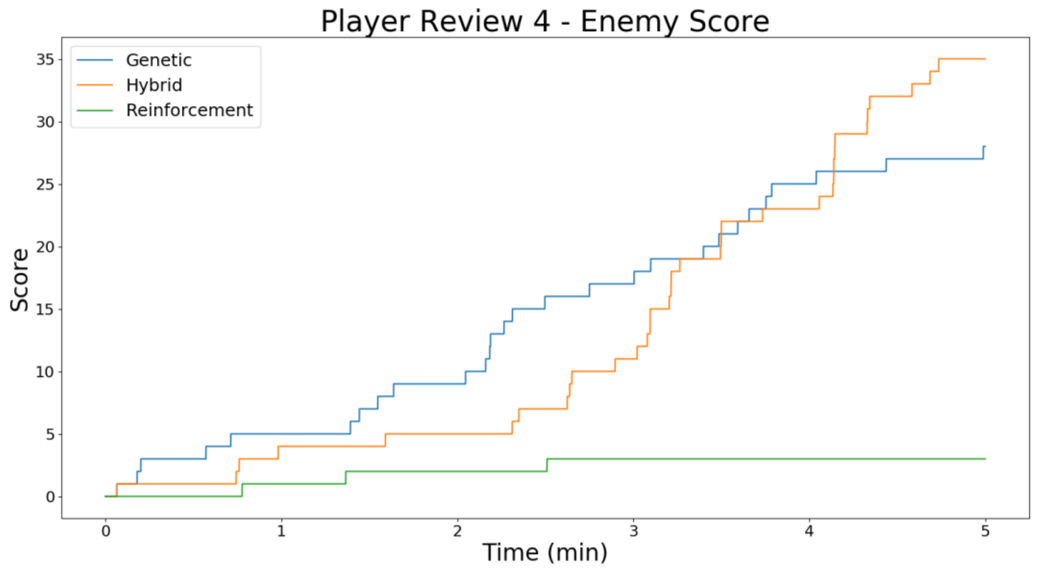


Figure 26

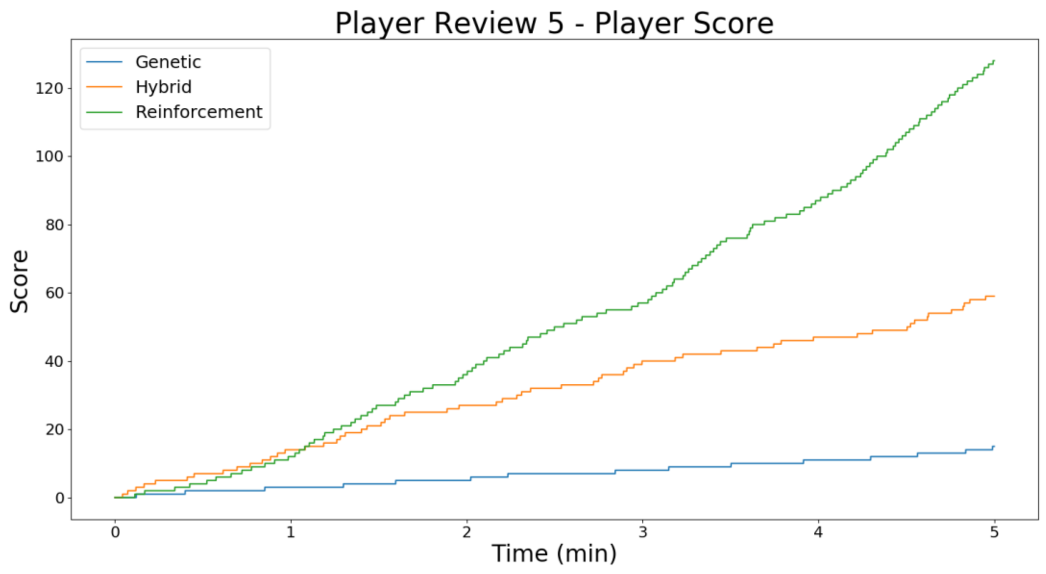


Figure 27

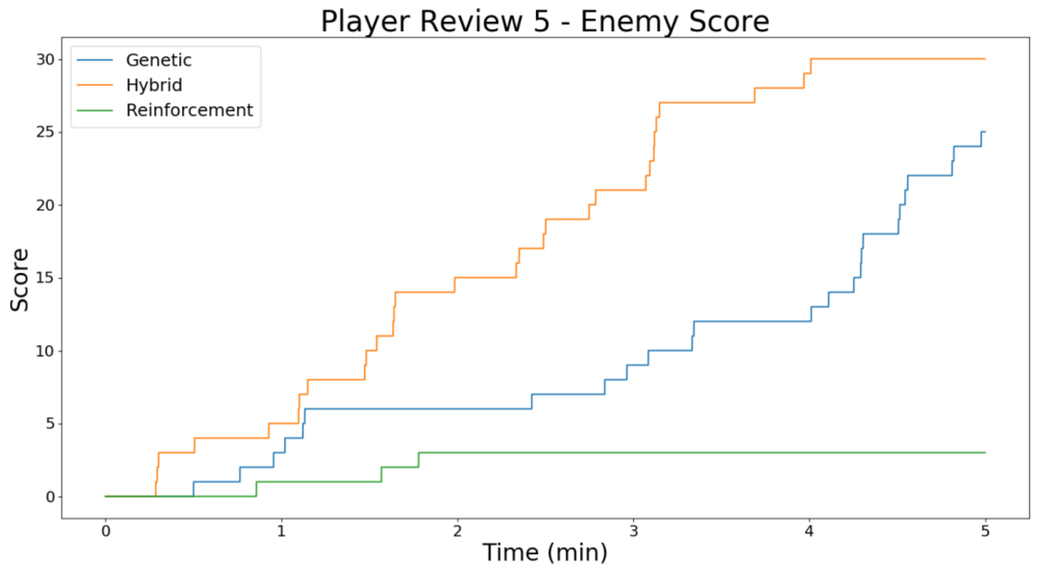


Figure 28

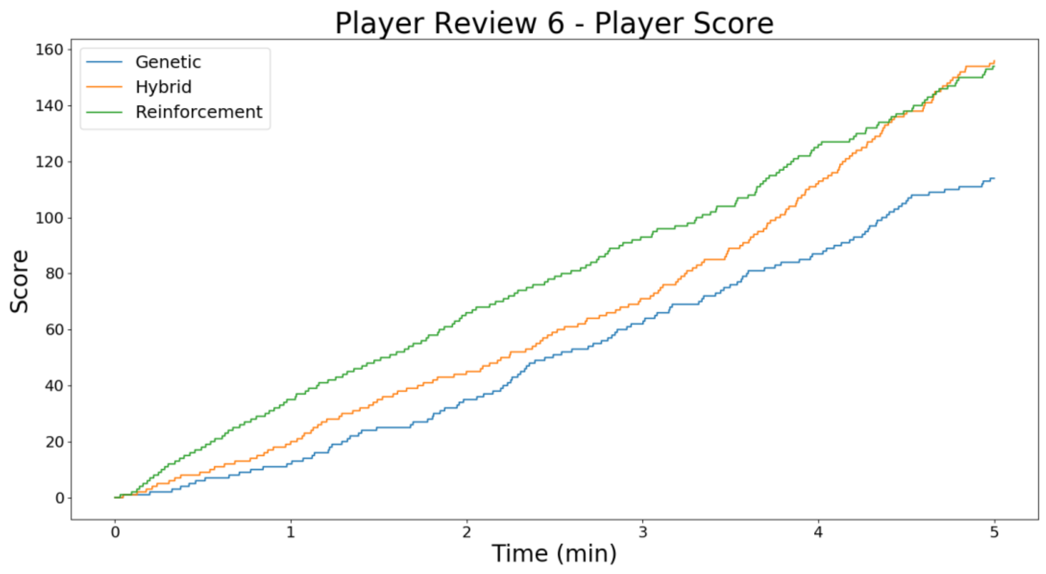


Figure 29

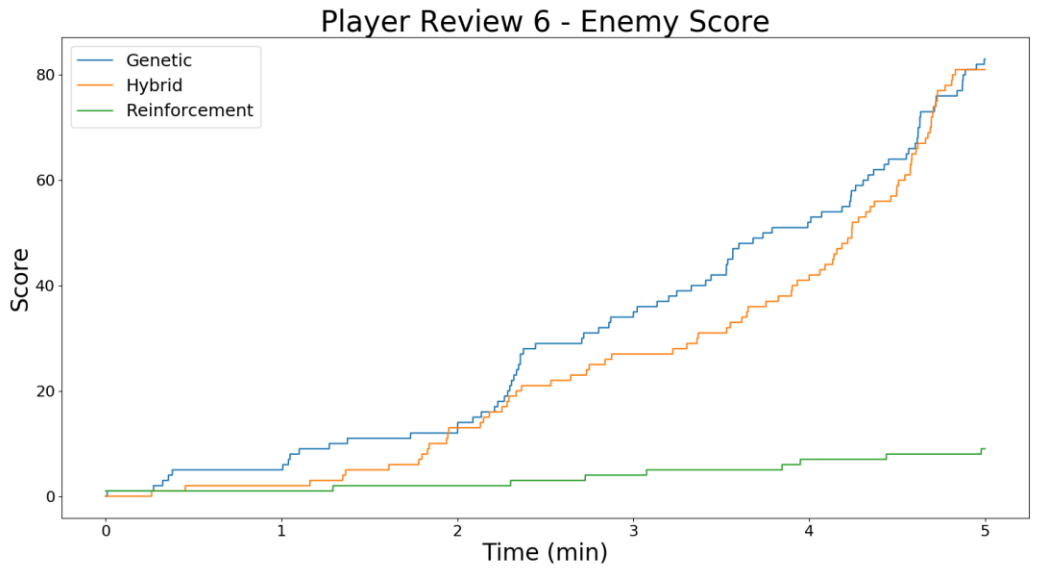


Figure 30

Summary of Player Responses

Player	Perceived agent accuracy (0 – 3, none to always)			Perceived ability for agent to avoid bullets (0 – 3, none to always)			Average difficulty (0 – 2, easy to hard)		
1	2	1	2	1	0	0	1.6	0	0.4
2	2	1	2	1	0	1	1.4	0	1.4
3	2	2	2	1	0	1	1	0.2	1.2
4	3	1	2	1	0	0	0.6	0	1.4
5	2	1	2	0	0	1	0.4	0	0.4
6	3	1	2	0	0	0	0.8	0	0.6
Avg.	2.33	1.17	2	0.67	0	0.5	0.97	0.03	0.9

Table 1: Summarizes the player responses to questions asked during gameplay. Red columns indicate results for the genetic algorithm. Blue columns indicate results for reinforcement learning. Green columns indicate results for the hybrid algorithm. The final row displays a mean of all 6 player responses.

Chapter 5: Discussion & Future Work

This thesis aimed to develop a new system for game AI, specifically for action games, capable of learning from players as they play. This system was designed and built with four goals in mind: allow for natural progression through learning, draw players in with novel or believable gameplay, meet computational requirements of a commercial game while keeping code complexity low, and generally be fun to play against. To meet those goals, a neural network was chosen to be the foundation of the game AI system, and the rest of the work on this project was devoted to determining the architecture of the network, as well as the best way to train it. Training was done through three algorithms: genetic algorithms, reinforcement learning, and a hybrid of the two. Data was collected during testing and training, and was summarized in Chapter 4. Now, based on that data, as well as observation from live gameplay, one method must be selected to use with the system. Looking at player reception, code complexity, run-time constraints, and general flexibility, a genetic algorithm with the given neural network architecture seems to be the best choice for use with this new game AI system. This chapter will explain the pros and cons of each method, and will conclude with directions the work could take going forward.

To start off, the genetic algorithm approach to training was successful, and ranked favorably with players in terms of fun and believability. Though their behavior was erratic – spinning in circles, shooting from odd angles – they did consistently shoot at players and were sometimes able to dodge bullets. Perhaps because they were erratic, they were also novel to players and the gameplay did not become predictable or boring after the 5-minute play test.

The box plot of training data shows that the neural networks could learn basic gameplay quickly, as the mean fitness rose to a steady state in about 15 generations. From there, the mean did not change much, but the variance decreased as the populations became better in general. Though the mean fitness seems to be lower than that of the reinforcement learners or the hybrid learners, this does not necessarily mean the genetic algorithm agents were worse. In fact, because the fitness function rewards agents for landing hits, the fitness of a population that can dodge well will look worse than the fitness of a population that does not dodge at all (like the reinforcement learners). Player testing suggested that the genetic algorithm did lead to agents with the best ability to dodge, and that the genetic algorithm led to agents with the best accuracy and highest average difficulty. This suggests that the fitness achieved in 55 generations of training was sufficient to reach a base level of competency to engage a player.

The most condemning data for the genetic algorithm agents is the performance against the control opponents. The number of injuries sustained varied slightly from generation to generation (more so than the reinforcement learners, but less so than the hybrid learners), but the number of hits landed is abysmally low for both control opponents. For some reason, the agents trained with genetic algorithms could not land hits against the control opponents. To contradict those results though, the data collected from player testing indicates that the agents trained with genetic algorithms *could* land hits, and in fact were able to land far more hits than the reinforcement learners, and more in some cases than the hybrid learners. The box plot of training data also clearly shows that the agents could land hits against each other, so what was going on with the control opponents?

By observing gameplay during player testing, it seems as though the genetic algorithm agents were learning to *put a lead* on their shots, or in other words, shoot slightly ahead of where the target was moving to. This way, the target would end up moving into the location that the bullet was traveling by the time the bullet arrived there. For moving opponents, this was particularly effective, and it explains why the agents were able to score well against each other and against players. It also explains why the agents failed to hit the first control opponent; because it was stationary, putting a lead on the bullets meant they would always miss the target. For the second control opponent, it is harder to say why the agents could not land a hit, but most likely it was because of the direction of travel. The control opponent was moving toward the agent, so a lead still would not be effective, as a lead in that case would be the same as aiming directly at the target.

To contrast the genetic algorithm, agents trained with reinforcement learning were generally considered the least fun to play against by players. The agents trained with reinforcement learning had very predictable and boring behavior: turn toward the player, approach, then shoot. Players universally agreed that these agents made no attempt to avoid bullets, however, when they shot they were typically accurate.

This behavior makes sense on reflection. Given one on one matchups that end when one agent wins, an optimal solution is to be the first to shoot. There is no incentive to dodge if it is possible to land three hits by simply facing and shooting. Then, according to the fitness function, the agent that stood and shot first is the one that moves on, thereby solidifying that behavior. With that in mind, it is possible to say reinforcement learning worked quite well. The neural network learned an optimal (or at least very good) solution to the game. The populations of reinforcement learners exhibited some of the highest mean fitness scores in the box plot of training data, and also the least variance. The reinforcement learners also generally landed the most hits against the first control opponent, and the second most hits against the second control opponent, after the hybrid method. However, the number of injuries received from both control opponents did not change much from generation to generation. This is consistent with the playing style that the reinforcement learners developed. However, despite the otherwise successful training, the agents were not fun to play against. Clearly, optimal does not equal enjoyable.

This result does not completely disqualify reinforcement learning from being used in the game AI system. The network learned with it after all, but what it learned did not translate to enjoyable gameplay. This suggests that the training process was flawed, not that the reinforcement learning algorithm was itself broken. The training process emphasized the wrong traits for gameplay by encouraging standoff behavior in one-on-one duels. Instead, the training process should have given the agents a variety of stimuli that were more like actual human gameplay. The difficulty with this though is that emulating human gameplay for training is not an easy problem to solve. If it were, this entire thesis would be pointless. Playing against humans for the entire training period is also not feasible because it takes a while for the neural network to reach a baseline level of competency at the game. The time it takes to reach that point is more than a player is likely to put in before becoming bored with the game (especially if the AI is very bad that entire time). Also, the code complexity of reinforcement learning is much greater than the code complexity of the genetic algorithm, because there is a lot more code and math involved with the backpropagation algorithm, as well as the system for storing actions. Altogether, the reinforcement learning algorithm is not inherently bad, but because it is hard to train properly and requires more code complexity, the genetic algorithm still seems to be a better choice.

While reinforcement learning alone did not produce ideal agents, the hybrid learning algorithm did lead to positive player reception. In terms of gameplay, the hybrid agents acted similarly to the agents trained with the genetic algorithm. The plots of enemy score show that those two methods produced agents that were equally capable of landing hits against human players. Also, the boxplot of training data shows that the hybrid learners had some of the highest mean and outlier fitness scores. This was coupled with the greatest variance in the fitness of the population however. Looking at the graphs of performance against the control opponents, the hybrid algorithm produced agents that performed well there too. They scored well against the control opponents (rivaled by the reinforcement learners), and managed to reduce the number of injuries they sustained over a few generations. However, in terms of player review, the hybrid algorithm was not significantly better than the genetic algorithm. In fact, it had lower (albeit similar) scores for accuracy and dodging, and a similar difficulty rating. Many of the player plots also show the hybrid and genetic algorithm performances overlapping.

Is this enough to justify the extra code complexity though? The author (and programmer) would argue no, the performance difference is not significant enough to warrant the extra time and code needed to perform hybrid learning. The reinforcement learning code and framework took around 4 months to set up and train on. Building the neural network was done separately, and took around 2 months. This was necessary work regardless of the training method however. In comparison, the genetic algorithm framework took less than a month to set up, debug and run, and could produce competent, fun-to-play against agents after a few days of training. Also, the framework did not require any derivatives, gradients, loss functions, or other difficult theory. Conceptually, the genetic algorithm is simpler than reinforcement learning, and in terms of code, it was much easier to develop. For a game studio on a budget, it is important that the development be simple so developer hours are not wasted, and so, the genetic algorithm is preferable over the hybrid method. While the hybrid learning algorithm proved to be effective, the extra time (and potentially money) required to make the reinforcement learning portion work was not useful enough to warrant adding it.

With almost a year of design and development on this project, the last thing to do before concluding is consider where to go next. At this point, the best candidate for an action game AI system is a simple neural network trained with a genetic algorithm, but game AI, particularly for action games, requires a few more pieces that this system does not yet have. For instance, agents need to be able to navigate, which is a task that has traditionally been handled by the A* algorithm. Currently, the neural network is not able to navigate levels with many walls or obstacles in them. Before the AI could be used in an actual game, this problem would need to be addressed. By focusing on a purely genetic algorithm approach, it seems likely that inputs could be added to the neural network that would allow it to learn how to navigate walls, at least in a rudimentary way. It could be given a longer stretch of time to train as well, and the fitness function could be altered to encourage navigation. Alternately, work could go into combining the network with more traditional AI techniques, such as A*. By altering the inputs and outputs slightly, the network could perhaps learn to choose when to aim and shoot, and when to use A* to navigate closer to the target. This same approach could be used for adding other AI behavior, including line-of-sight and vision cones, or more action game specific behavior, such as flanking or squad behavior.

On top of adding functionality to the AI, there is still a problem that needs to be addressed regarding learning from players during gameplay. This thesis demonstrated that the AI system could learn, could become base level competent at the game, and could give players a challenge. However, it did not demonstrate that the agents could learn specific behaviors for countering specific player strategies. Agents could become better generally by increasing their fitness, but it is not clear that they could become better in player-specific ways. To address this, more work would have to go into the design of the fitness function. By making the fitness function dependent on agent performance against the player, it is theoretically possible that the agent would adapt to the player specifically. Work would also have to go into level design and enemy/player interactions to make sure that each encounter resulted in an accurate assessment of agent performance against the player. Each enemy agent would have to have a significant faceoff against the player to learn anything meaningful, and the layout of the game would have to be conducive to producing that kind of faceoff.

Altogether, this thesis serves as a small proof-of-concept for the use of machine learning in game AI. Specifically, it advocates for the use of simple neural networks trained with a genetic algorithm to drive enemy agents. While the game AI system is not yet complete, the foundation is in place to add functionality, such as navigation or squad behavior, and eventually take the system to use in commercial games. It also keeps code complexity low, which has been a motivating factor for AI programmers since Jeff Orkin gave his talk in 2006. The system certainly will not be suitable for every game or every developer, but the hope is that games will soon be released that can give players an exciting and novel experience because of the AI alone.

Works Cited:

[1] Orkin, Jeff. "Three States and a Plan: The A.I. of F.E.A.R.," *Proceedings of the Game Developers Conference,* (2005), http://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf

[2] Isla, Damian. "Handling Complexity in the Halo 2 AI," *Proceedings of the Game Developers Conference,* (2005), International Game Developers Association.

[3] "Turing Tantrums 2017: AI Devs Rant." Proceedings of the Game Developers Conference. January 6, 2018. https://www.youtube.com/watch?v=i00Yu2zHbbU

[4] Cobbett, Richard. "From shareware superstars to the Steam goldrush." PC Gamer. https://www.pcgamer.com/from-shareware-superstars-to-the-steam-gold-rush-how-indie-conquered-the-pc/

[5] "All time unit sales of selected games in Call of Duty franchise." Statista. https://www.statista.com/statistics/321374/global-all-time-unit-sales-call-of-duty-games/

[6] Lou, Harbing. *AI in Video Games: Toward a More Intelligent Game* (blog). August 28, 2017. http://sitn.hms.harvard.edu/flash/2017/ai-video-games-toward-intelligent-game/

[7] Google AI. Google Inc. https://ai.google

[8] Titanfall 2. EA Games Inc. https://www.ea.com/games/titanfall/titanfall-2

[9] Destiny 2. Activision. https://www.destinythegame.com/

[10] Civilization VI. 2K Games. https://civilization.com/

[11] Shalev-Shwartz, Shai, and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge: Cambridge University Press, 2014.

[12] Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search." Nature 529, no. 7587 (January 27, 2016): 484. https://doi.org/10.1038/nature16961.

[13] "Combat Learning." Double Zoom. http://doublezoom.free.fr/programmation/AG_Exemple_Fighting.php

[14] Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, et al. "Mastering the Game of Go without Human Knowledge." Nature 550, no. 7676 (October 18, 2017): 354. https://doi.org/10.1038/nature24270.

[15] Reeves, Colin R. and Rowe, Jonathan E. *Genetic Algorithms – Principles and Perspectives*. Cambridge: Kluwer Academic Publishers, 2002.

[16] Montana, David J. and Davis, Lawrence. "Training Feedforward Neural Networks Using Genetic Algorithms," *International Joint Conferences on Artificial Intelligence*, (1989), https://www.ijcai.org/Proceedings/89-1/Papers/122.pdf