香　港　大　學

THE UNIVERSITY OF HONG KONG

COMP 4801

Evolving Human-like Micromanagement in StarCraft II with Neuro-Evolution and Reinforcement learning

---

Supervisor: Dr. Dirk Schnieders

Written by: Zain Ul Abidin (3035305590)

Group partner: Fawad Masood Desmukh (3035294478)

TESTING and EVALUATION REPORT

April 14, 2019

Abstract

Video games have always been a popular proving ground for artificial intelligence techniques. Traditional AI agents have relied on scripted, rule-based approaches that have several flaws such as the inability to handle massive state spaces and the specificity of logic to a game that can be exploited. Recent breakthroughs in this domain have come through the application of novel approaches in machine learning such as reinforcement learning and neuro-evolution. After preliminary testing in the Arcade and Real-Time Strategy genre, we chose to further explore the applications of machine learning in StarCraft II which is claimed to be the next "grand challenge" for AI research.

In our project, we implement neuro-evolution using NEAT and reinforcement learning using Sarsa(λ) on micromanagement scenarios in StarCraft II involving the small-scale precise control of combat units. Using our developed training framework for applying NEAT to StarCraft II, we evolved neuroevolutionary agents that learned to demonstrate precise hit-and-run strategies to beat the in-game AI in ranged vs melee matchups. Our reinforcement learning agents using Sarsa(λ) learned to be successful in more complex micromanagement scenarios involving enemy engagement selection and timing. Our results serve as a proof-of-concept of the benefits and potential of the applications of these techniques in video games and represent meaningful contributions to the wider video gaming and artificial intelligence communities.

Acknowledgment

We would like to express our sincere gratitude to our supervisor, Dr. Dirk Schneiders, for his support and guidance throughout the project.

We would also like to thank members of the StarCraft AI Discord community who entertained extensive discussions on the use of the various frameworks used as well as provided suggestions and useful resources for the machine learning approaches implemented. In addition, we would also like to thank Aavaas Gajurel at the University of Nevada who entertained our questions on applying neuro-evolution on StarCraft II.

Lastly, we would like to thank our good friend Jamal Ahmed Rahim for his help and support throughout the project.

Table of Contents

List of Tables

List of Figures

Abbreviations

Abbreviation	Full Form
A3C	Asynchronous Actor-Critic Agent
AI	Artificial Intelligence
API	Application Programming Interface
CEM	Cross Entropy Method
DQN	Deep Q-Network
MDP	Markov Decision Process
MLP	Multi-Layer Perceptron
NEAT	Neuro-Evolution of Augmenting Topologies
RL	Reinforcement Learning
RTS	Real Time Strategy
SARSA	State-Action-Reward-State-Action
SC2LE	StarCraft II Learning Environment
TD	Temporal Difference

1 Introduction

The project documentation is divided into two reports discussing the development and testing phases of the project respectively. The development report is written by my group member, Fawad Masood Desmukh and the testing report is written by myself, Zain Ul Abidin.

The development report introduces the project, elaborates on the existing work and then comprehensively details the methodology and the developmental stages of the project.

This report follows from the development report and the reader is recommended to read the development report first. This report first focuses on the testing and evaluation phase of the project and discusses all the results in detail. This is followed by a comparative analysis of the various implemented approaches where all the results are summarized. It finally ends with concluding remarks on the goals, contributions and future directions of the project.

For sake of brevity, we have not included detailed information from the development phase in this report and instead refer to the development report at several points of this report to give context to our tests. Therefore, to get a complete understanding of the project, it is recommended to refer to the development report, wherever it is stated. For convenience we have used a reference notation, *DEV-XX-YY*, where XX is the section number and YY is the page number of the development report.

StarCraft II is claimed as the next *"grand challenge"* for AI research. Keeping in line with the significance of that claim, this project aimed to implement various machine learning approaches on StarCraft II. This report details the successes and failures from our findings.

2 Preliminary Testing and Exploration

The preliminary testing and exploration phase of this project consisted of the application of the neuro-evolution approach to one of the DeepMind mini game [1], ***Move To Beacon.*** The following subsection focuses on the details of the mini game scenario and the preliminary testing results.

2.1 Move To Beacon

2.1.1 Game Description

The mini game consists of one Marine unit and one Beacon. The agent can control the marine unit that gets a reward every time it reaches the beacon. Upon reaching the beacon, the beacon position is randomly reset at least 5 units away from the Marine. The final score in the game is the number of beacons collected in a specified number of episode steps. For our validation test on Move To Beacon we had the number of episode steps set to 1800 steps per episode which takes roughly 60 in game seconds (these can be sped up for training). Figure 1 shows an instance of the 64 x 64 Move To Beacon map.

Figure 1: Move To Beacon map. The blue unit is the Marine. The green circle is the Beacon.

2.2 Validation Test

Before carrying out exploratory testing, we wanted to know whether neuro-evolution will work for the mini game scenario or not. Hence, we carried out an initial validation test. The details of the test are as follows.

2.2.1 Inputs

The neural network was given two basic inputs which were the minimum requirement for the neural network to learn whether moving to beacon is beneficial or not. The two inputs were:

1. Marine selected. (boolean)
2. Marine on beacon. (boolean)

2.2.2 Genome Network Outputs

The validation test had six outputs:

1. No operation.
2. Select marine.
3. Deselect marine.
4. Move to beacon. (move towards the center of the beacon)
5. Move random. (move randomly anywhere on the map)
6. Move middle. (move towards the center of the map)

2.2.3 Results

The results were encouraging. The agent learnt to always choose the output 4 (Move to beacon). This meant that given the exact coordinates of center of the beacon, the agent acknowledged that moving towards and collecting the beacon was rewarding. Figure 2 shows the visualization of the results that we achieved. It is visible from the increasing blue line that on average the genomes learnt to collect the beacon. The agent learnt the desired behavior in just about 40 generations.

Figure 2: Validation test results over Move To Beacon mini game

Hence, our validation test was successful and proved that the neuroevolutionary approach has the potential to produce desired results.

2.3 Exploratory Tests

After establishing that the neuro-evolution approach had learning potential on the mini game scenario, we decided it imperative to do further exploration with various inputs, outputs, fitness functions, and configuration settings. Table 1 details the various tests that were conducted over the Move To Beacon mini game scenario.

2.3.1 Inputs

There were two types of input used in our tests. The feature input extraction is elaborated in detail at DEV-3.4.3.2-38

1. **Pixel**

Trainings were launched with (64, 64), (32, 32) and (16, 16) screen resolution with 4096, 1024 and 256 inputs respectively.

2. **Handcrafted**

Trainings were launched with handcrafted features that took into account the player and beacon's positional coordinates.

2.3.2 Genome Network Outputs

We used the second set of outputs in our tests from the two output sets that we designed at DEV-3.4.3.2-39. This would allow us to specify both the magnitude and direction of movement of the marine as well as reduce the number of outputs.

2.3.3 Configurations

The DEV-3.4.3.2-33 mentions the notable configuration parameters that are used for NEAT evolution. The configuration setting for the exploratory test are as follows:

Activation function	**Aggregation function**	**Population Size**	**Initial Connectivity**	**Network types**	**Number of input nodes**	**Number of output nodes**
Clamped	Sum	100	Partial 0.5	Feedforward/ Recurrent[1]	Corresponds to features	2
Number of hidden nodes	**Node add probability**	**Node delete probability**	**Connection add probability**	**Connection delete probability**	**Weight max value**	**Weight min value**
0	0.5	0.2	0.5	0.5	30	-30
Weight mutate rate						
0.8						

2.3.4 Reward Functions

Two fitness functions were used in our tests to assist the agent in learning the desired behaviour.

$$fitness \mathrel{+}= fitness + beaconreward \qquad (1)$$

$$fitness \mathrel{+}= fitness + (beaconweight * beaconreward) + (distanceweight * \Delta\, distancereward) \qquad (2)$$

[1] Refer to Table 1 in Appendices

2.3.5 Results

Table 1 is a subset of the tests carried out for which the results were recorded. In other cases, incremental changes were made in an attempt to find improvements in results. The test results were not very encouraging. None of the attempts managed to achieve an intelligent strategy to the game.

Initial tests used the pixel-based features. However, no substantial learning was observed. Most of the genomes picked a direction and kept moving in that direction. Eventually all of them got stuck in the corner of the map.

Figure 3 shows the results of the run over 160 generations for test 2 which uses 256-pixel inputs.

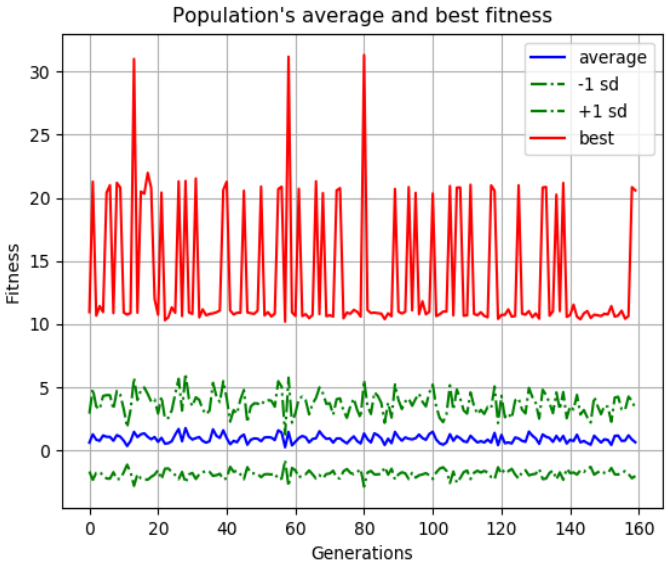

Figure 3: Fitness against Generations for Test 2. 256 Pixel inputs with a Fitness Function (2)

The average population fitness remained stable. Some genomes managed to pick up three beacons which may have been picked up by chance. However, in training it was observed some genomes understood to directly go for the first beacon when it first spawned but do not subsequently go for the other ones. This can be seen with the best genome in figure 2 which consistently picked up one beacon to get a fitness above 10.

Due to the poor results from the pixel based runs we decided to do some further research. [2] indicated that NEAT faced difficulty in learning from raw pixel-based input when applied to Atari games. Therefore, we decided to test out handcrafted features.

Figure 4 shows results from Test 6 according to Table 1 The graph only shows the progression from the last 150 generations.

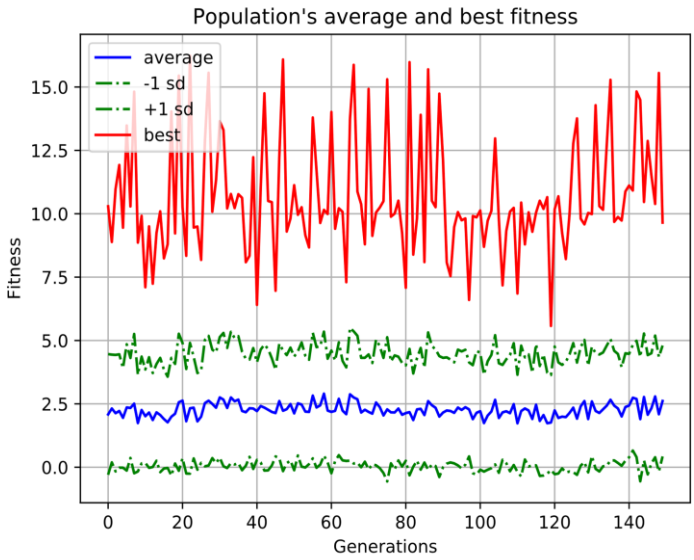

Figure 4: Fitness against Generations for Test 6. Handcrafted feature inputs with a Fitness Function (2)

The average population fitness remained stable. None of the genomes managed to learn a consistent policy for solving the environment.

However, a significant majority of the genome population learnt an interesting strategy which was displayed by the best genome in figure 4 as well. The strategy was to go left and right (oscillate) but over a much wider range to pick up beacon in their traversal rather than actively seeking out the beacon. Some of the marines started oscillating from their start position. However, some displayed understanding of the objective by approaching some position near the beacon and then oscillating. This does represent a valid policy if the genome can learn to align with the beacon first before oscillating. This interesting behavior is encouraged by our fitness function that rewards reducing the distance to the beacon.

As can be seen, the reward function has a great impact on the training efficiency and can lead to suboptimal strategies as was seen in our tests. Handcrafted features have proven to be more

efficient for agents to learn policies and going forward we will focus on using handcrafted features for NEAT. Having gained a better understanding of the neuroevolutionary approach, we moved on to more meaningful combat-based micromanagement scenarios.

3 MicroManagement Tasks

In StarCraft II micromanagement (micro for short) refers to the selection and control of individual units on a precise level to extract the maximum value out of the units. In a combat scenario some units are better matched against the other units. However, the weaker units might have certain advantages over the enemy units. Micromanagement in such situations means to exploit these advantages and inflict the maximum damage possible upon the stronger enemy units. Micromanagement involves different key techniques and behaviors such as *"kiting"*.

Kiting, a behavior which alternates between attacking and running from an opponent, is an example of good micro. It is usually favorable in scenarios where a unit with greater attack range (ranged unit) is against a unit with a relatively smaller attack range (melee unit). Hence, the ranged unit can consistently hit and run from the melee unit, by staying out of the enemy attack range and wins the battle by exploiting its attack range advantage.

Another example of good micro includes enemy selection where a player must be aware of the strengths, weaknesses and abilities of the units in play, in order to engage various enemy units with the most effective units.

Micro also includes the engagement time of the enemy units. This accounts for the position of own units as well as the enemy units and determines the precise time of attacking the enemy to ensure the maximum damage. An example would be attacking the enemy when it is clumped together at a narrow section of the map.

Such management of units on a micro level is the essence of StarCraft II combat scenarios which allows the player to outplay his opponent in competitive battles. Provided that the micro is perfectly executed, it can shift the tide of the battle, and even result in favorable results for a weaker player. However, this requires high levels of skill and dexterity which only the most professional players of StarCraft II can demonstrate.

In the remainder of this report, we demonstrate neuro-evolution and reinforcement learning agents learning intelligent micromanagement strategies.

3.1 Neuro-Evolution

First, we focused on training the agents in adversarial scenarios using the neuro-evolution approach. In this section we will discuss the tests that were conducted and evaluate the results.

3.1.1 Map Specification

We designed a standard map for testing our neuroevolutionary approach using the SC2 map editor DEV-3.4.5.1-49 The same map was used in all our tests, however, some changes were required to change the units and the starting configurations, as required. A detailed set of maps that were used in our tests are mentioned in the development report DEV-3.4.5.2-49.

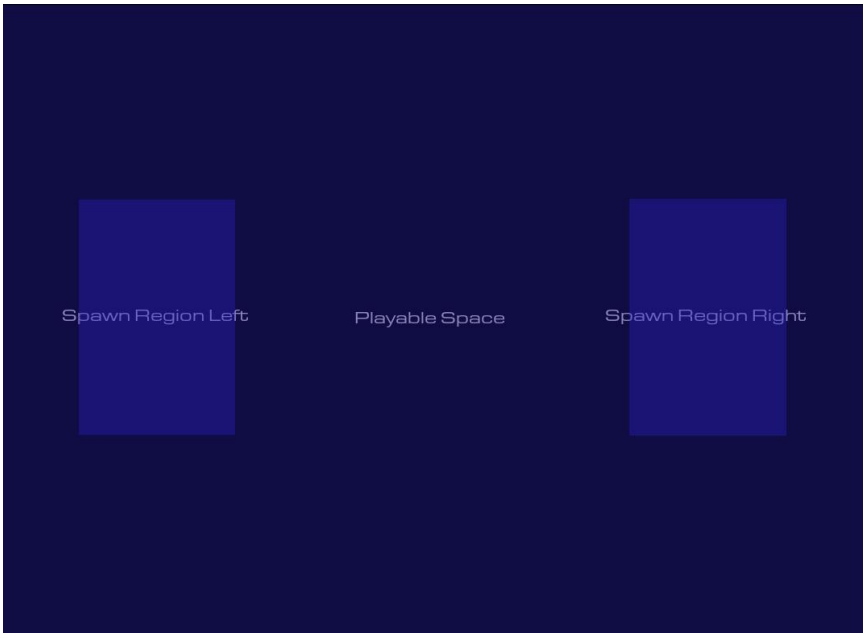

Figure 5: The layout of the standard map used for neuro-evolution tests

The standard map had the dimensions of 64 x 64 with two spawn regions, Spawn Region Left and Spawn Region Right, and a Playable Space. Figure 5 shows the layout of the standard map.

The map specifies two players, player 1 and player 2, where player 1 is controlled by the learning agent and player 2 is controlled by the scripted in-game AI. To remove unfair advantage for the learning agent, we disabled the auto attacking feature of the units controlled by the agent.

At the beginning of the test, the map is initialized, and an episode timer starts for 120 game time seconds. The units are spawned randomly in groups in each of the spawn regions and then engage each other by moving towards the opposite spawn region. Each time a unit from either

side dies, the cumulative score for the episode is updated. In a case where either of the player loses all its units, the units are again randomly spawned in either of the spawn regions. At the end of each episode the episode timer is restarted. This carries on for the desired number of generations.

3.1.2 Initial Test

3.1.2.1 Combat Agent 1

Our initial set of tests were conducted using the Combat Agent 1. Table 2 shows the various tests that were run over the 5v5 map instance of Hellions V Zealots.

- Hellions are ranged units with a fast speed. However, they have low armor and health which requires them to be microed effectively.
- Zealots are melee units with a relatively slower speed but a very strong attack. At a close range they can defeat hellions with ease.

In our tests Hellions were controlled by player 1 (agent) whereas Zealots were controlled by player 2 (scripted AI). The desired result was precise kiting demonstrated by Hellions against Zealots.

i. Inputs

This combat agent used the following handcrafted feature inputs scaled to a range of [0,1]:

1. Current hp - The current hit points of our units
2. Weapon cooldown - Boolean that is true when at least half of our units' weapons are on a cooldown
3. Enemy in range - Boolean stating whether the enemy is in attacking range or not.

ii. Genome Network Outputs

Using the above inputs the agent was able to make a decision of either engaging in a fight or fleeing from the enemy. The outputs are as follows:

1. Fight/Flee - boolean
2. Displacement in x
3. Displacement in y

iii. Configuration

Activation function	**Aggregation function**	**Population Size**	**Initial Connectivity**	**Network types**	**Number of input nodes**	**Number of output nodes**
Sigmoid	Sum	100	Partial 0.5/Full[2]	Feedforward/ Recurrent[3]	3	3
Number of hidden nodes	**Node add probability**	**Node delete probability**	**Connection add probability**	**Connection delete probability**	**Weight max value**	**Weight min value**
0	0.5	0.2	0.5	0.5	30	-30
Weight mutate rate						
0.8						

3.1.2.2 Reward Functions

The following reward function was used to evaluate the reward in these tests without reward scaling.

$$fitness = ((total\ damage\ dealt - hit\ point\ loss)/\ initial\ self\ hit\ points) + 1 \quad (3)$$

[2] Refer to Table 2 in Appendices
[3] Refer to Table 2 in Appendices

3.1.2.3 Results

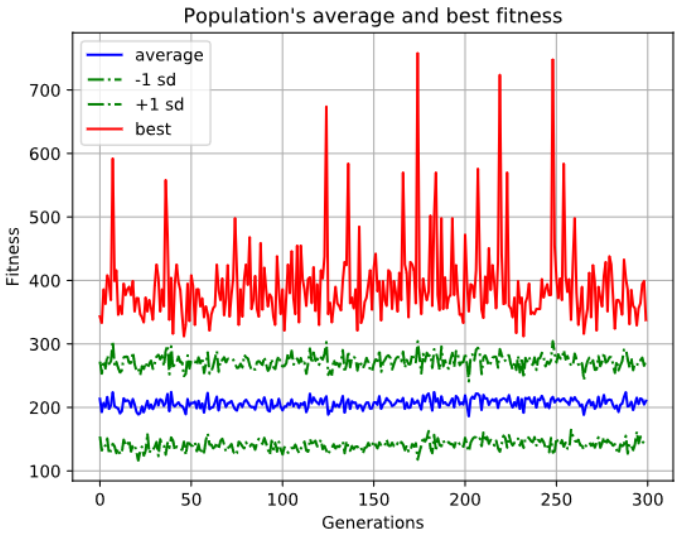

Figure 6: Test 3 results from Table 2 of Combat Agent 1. Handcrafted feature inputs with a Fitness Function (3)

All the tests had very similar results. There was inconsistency observed in the results with no substantial learning observed. Figure 6 shows the results of last 300 generations of test 3. The average population fitness remained stable at 200. None of the genomes managed to learn a consistent policy for solving the environment.

However, some of the genomes learnt to engage the Zealots at the start of the episode. After the initial engagement they demonstrated a fleeing behavior from the enemy and eventually got stuck in the north west corner of the map. Hence, the enemy was lured towards Hellions in a clump which would allow the Hellions to do splash damage over the Zealots and maximize the damage caused. Figure 7 shows an instance of the behavior. This resulted in occasional wins for the agent which are observed by the peaks in Figure 6.

This lack of learning was attributed towards insufficient inputs for the agent to determine the map bounds. Hence, we increased our inputs in the combat agent and carried out a new series of tests.

Figure 7: Hellions lure Zealots into the north west corner of the map

3.1.3 Further Tests

Although the initial set of tests failed, it proved the importance of appropriate feature input extraction for the neural network. An updated version of the previous combat agent, Combat Agent 2 was used to conduct these tests. The units, Hellion and Zealots, were unchanged. The summary of the notable tests can be found in Table 3.

3.1.3.1 Number of genomes per episode

In these set of tests, we made a key decision to make the fitness of a genome evaluation reliant on more than one episode by running the same genome on multiple episodes and then averaging the fitness across the episodes run. A genome was classified as a good genome only if it reached the threshold over the specified number of episodes. We hoped that this would reduce the probability of winning by chance and emphasize good strategies that consistently performed well. This proved to be a significant breakthrough in terms of the quality of results.

3.1.3.2 Combat Agent 2

i. Inputs

In addition to the inputs used by the Combat Agent 1, the updated Combat Agent 2 used additional handcrafted feature inputs. The new inputs accounted for the map bounds in four directions, directions in which the enemy was present, and the previous command chosen by the neural network. All inputs were scaled to a range of [0,1]:

Current hp	Weapon cooldown	Enemy in range	Previous command
North bound	South bound	West bound	East bound
North West enemy presence	North East enemy presence	South West enemy presence	South East enemy presence

ii. Outputs

The outputs remained the same as in Combat Agent 1:

1. Fight/Flee - boolean
2. Displacement in x
3. Displacement in y

iii. Configurations

For this series of tests, we came up with seven different configuration settings. Below we mention the two successful configuration settings.

a) Configuration 1

The first successful configuration settings (Config 1) were as follows:

Activation function	**Aggregation function**	**Population Size**	**Initial Connectivity**	**Network types**	**Number of input nodes**	**Number of output nodes**
Clamped	Sum	150	Full	Feedforward	12	3
Number of hidden nodes	**Node add probability**	**Node delete probability**	**Connection add probability**	**Connection delete probability**	**Weight max value**	**Weight min value**
0	0.15	0.1	0.15	0.1	2	-2
Weight mutate rate						
0.95						

b) Configuration 2

The second successful configuration settings (Config 2) were as follows:

Activation function	**Aggregation function**	**Population Size**	**Initial Connectivity**	**Network types**	**Number of input nodes**	**Number of output nodes**
Clamped	Sum	150	Full	Feedforward	12	3
Number of hidden nodes	**Node add probability**	**Node delete probability**	**Connection add probability**	**Connection delete probability**	**Weight max value**	**Weight min value**
0	0.02	0.01	0.04	0.025	3	-3
Weight mutate rate						
0.95						

3.1.3.3 Reward Functions

The following reward function was used to evaluate the reward in these tests with scaling in the range [0,1]:

$$fitness = initial\ enemy\ hit\ points + current\ self\ hit\ points - current\ enemy\ hit\ points \quad (4)$$

3.1.3.4 Successful Results (Trained Agent)

Multiple tests were run with the seven different configuration files and changing number of episodes per genome. We divided these tests into two different series.

- Series 1 consisted of tests with number of episodes per genome set to 1
- Series 2 consisted of tests with number of episodes per genome set to 3.

The results from the first series of tests were not very successful as they demonstrated a behavior similar to the initial set of tests by moving into a corner of the map and causing splash damage to the Zealots.

However, tests in the second series had successful results. Two of the test results demonstrated the perfect kiting behavior which is impressive considering the space constraint in the map.

These tests used the configuration setting 1 and 2. Figures 8 to 10 show the kiting behavior exhibited by the hellion units.

Figure 11 shows the results of the test with config 1. An increasing fitness can be seen throughout the 200 generations until the fitness is maximized.

Figure 8: Hellions start to kite

Figure 9: Hellions attack while kiting

Figure 10: Hellions continue kiting

Figure 11 shows the results of the test with config 1. An increasing fitness can be seen throughout the 200 generations until the fitness is maximized. Similarly, Figure 12 shows the test results with config 2. The agent learnt the desired behavior relatively faster as the fitness of the best genomes converges to approximately one in just about 100 generations. The average fitness also has an increasing trend throughout the 200 generations indicating that a significant proportion of the genome population is learning the kiting strategy.

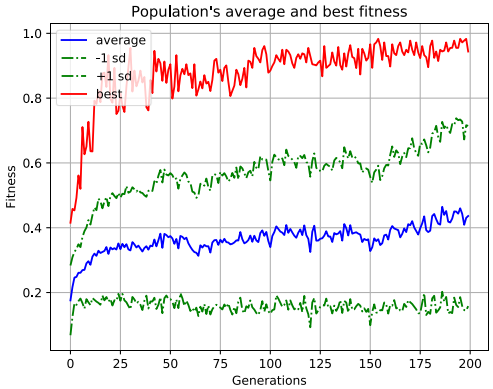

Figure 11: Test results of Combat Agent 2 from further tests series 2 with config 1 on Hellion V Zealot map with a Fitness Function (4)

Figure 12: Test results of Combat Agent 2 from further tests series 2 with config 2 on Hellion V Zealot map with a Fitness Function (4)

The evolved network looks as follows (Figure 13):

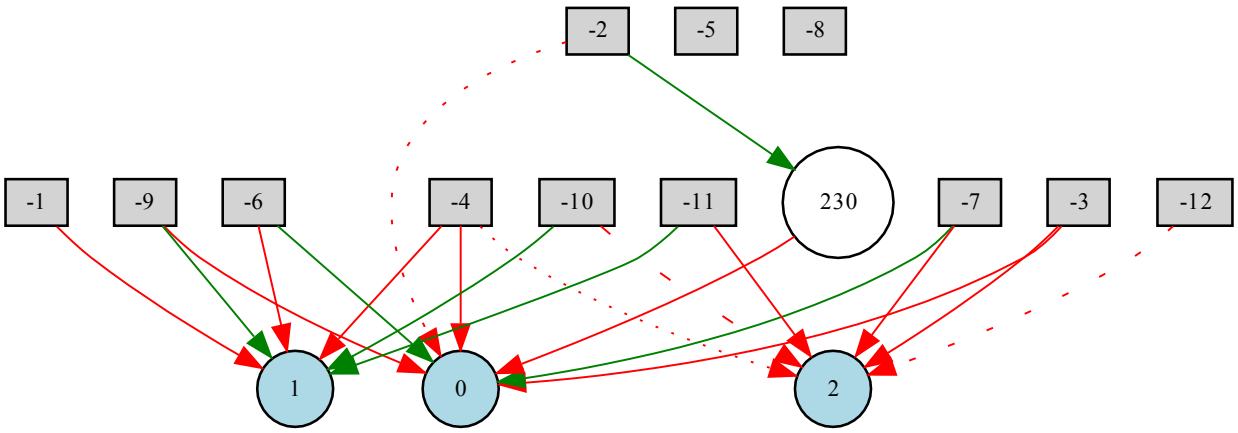

Figure 13: Evolved net from training over Hellion V Zealot NEAT map

NEAT successfully learns the behavior with a small network with only a few hidden nodes. The network optimizes for the fitness function and thus does not evolve more should it sufficiently build up for obtaining a high reward.

Overall, we were able to achieve successful results in our further test series because of a few additional factors:

1. Multiple episodes per genome as mentioned in section 3.1.3.1
2. Increased inputs accounting for the map bounds and the enemy presence.
3. New configuration setting, namely, the weight range and the weight mutate rate.

For here onwards we refer to the agent that was successfully trained in this section as the ***Trained Agent.*** In the following tests over, adversarial scenarios we used Config 2 and Combat Agent 2

3.1.4 Evaluation performance of Trained Agent

This section focuses on evaluating the performance of the Trained Agent that learnt kiting on the Hellion V Zealot map. It was imperative to test the robustness of our agent and observe its behavior in different adversarial scenarios. Hence, we tested the agent by changing the spawn regions, units and engagement strategies respectively.

3.1.4.1 Different number of Zealot units

We evaluated the Trained Agent by changing the number of zealots. Our evaluation was done by incrementing the number of Zealots from five (the number of zealots in the training). The agent performed well even if the zealot units were more than the Hellions. Figure 14 shows the remaining hellion units vs increasing zealot units and Figure 15 shows the remaining zealot units vs increasing zealot units. As can be seen from the figures, the agents do well even with an increasing number of enemy zealot units. Performance goes down due to difficulty of kiting in such a restricted space, so it becomes difficult to avoid enemies when their numbers become too big.

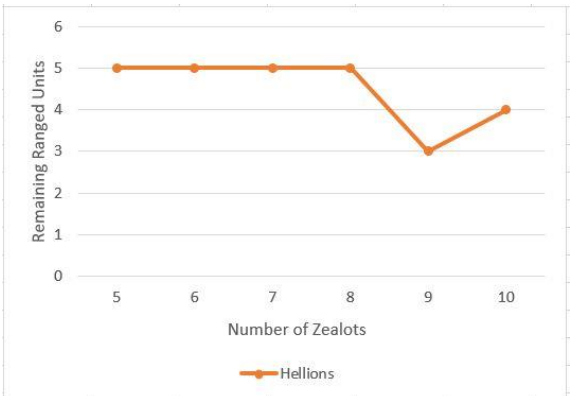

Figure 14: Remaining number of Hellion units against Number of Zealots.

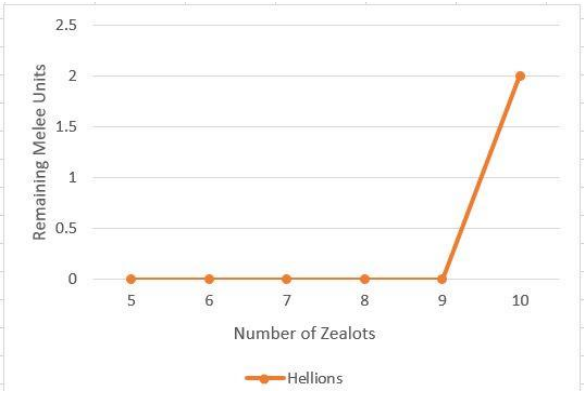

Figure 15: Remaining number of Zealot units against Number of Zealots

3.1.4.2 Different starting configs

We modified the standard map (section 3.1.1) to include new spawn regions in addition to the left and right regions. Figure 16 shows the map layout. With respect to the center of the map, these included:

Up	Down	Left Diagonal 1	Left Diagonal 2	Right Diagonal 1	Right Diagonal 2

Figure 16: The layout of the standard map with additional spawn regions used for neuro-evolution tests

We evaluated the ***Trained Agent*** on the new map by randomly spawning the units in the new regions. The agent only performed kiting if it was spawned in the left or right regions. In any other scenario the agent failed to defeat Zealots.

3.1.4.3 Different ranged units

Next, we evaluated the ***Trained Agent*** by changing the units that it controls. Since it originally learnt over the ranged unit, Hellion, we limited ourselves to the ranged unit types in our evaluation. The new units used in our evaluation were:

1. Stalkers - Relatively slower than hellions with a longer attack range
2. Roaches - Relatively slower than hellions with a small attack range

The learned agent displayed the same kiting behavior. However, performance improved or deteriorated based on the difference in speed and range of those units in comparison to hellions

using which the network was trained. Therefore, there is a difference in how often the unit should attack which affects performance if the same network is reused.

3.1.4.4 Different engagement strategies

The ***Trained Agent*** was further evaluated with different engagement strategies. Three engagement strategies were evaluated:

1. The agent engages the enemy first.
2. The enemy engages the agent first.
3. Both do not engage each other.

The agent did well only on the first engagement strategy since it was trained using that strategy. The other two engagement strategies lead to defeat for the agent in all of the runs.

3.1.5 Fresh Trainings

Since our ***Trained Agent*** was not performing well if the environment and the conditions were changed, we conducted new tests, which are discussed in the following sub section. As mentioned previously we continued to use config 2 and Combat Agent 2 over the following trainings since our previous trainings showed the best results when they were used.

3.1.5.1 Different engagement strategies

Since our evaluation with different engagement strategies failed, we trained the agent using the engagement strategy 2 and 3 (section 3.1.4.4), over the same Hellion V Zealot map used in our initial and further test sets. (section 3.1.2)

Figure 17 shows the results of the training using engagement strategy 2. We can see that the average fitness is increasing but it is rather slow as compared to the training with engagement strategy 1 which converged to the maximum fitness of one within 100 generations.

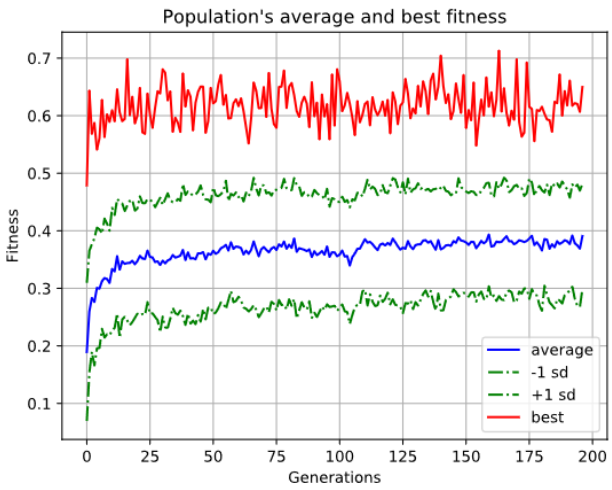

Figure 18: Test results of Combat Agent 2 using config 2 with engagement strategy 3 on Hellion V Zealot map with a Fitness Function (4)

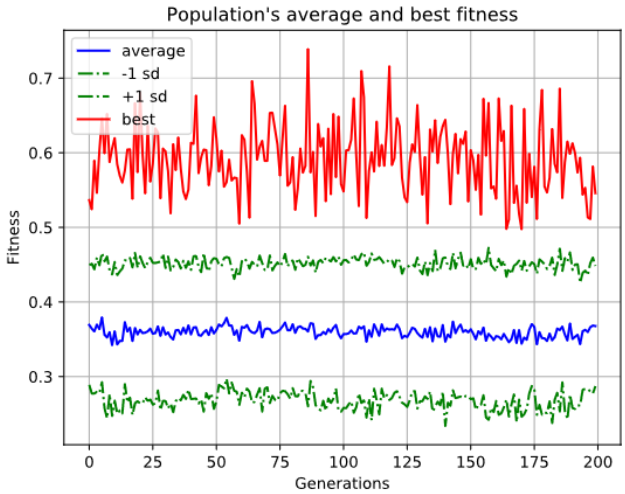

Figure 17: Test results of Combat Agent 2 using config 2 with engagement strategy 2 on Hellion V Zealot map with a Fitness Function (4)

Whereas engagement strategy 2 showed some learning within 200 generations, the agent with engagement strategy 3 did not learn too much since the two enemy groups do not come into contact often enough to accelerate learning. Figure 18 shows that the average fitness in almost constant at 0.35.

After establishing that engagement strategy 1 produced the fastest learning rate for the agent, we decided to use the strategy 1 in all subsequent trainings

3.1.5.2 Different self-units

After training using different engagement strategies, we trained the agent in adversarial scenarios by changing the units that the agent controls.

i. Stalker V Zealot

The training was done using five Stalker units against five Zealot units. The results are shown in Figure 19

Figure 19: Test results of Combat Agent 2 using config 2 with engagement strategy 1 on Stalker V Zealot map with a Fitness Function (4)

As seen from the figure, the average fitness increased, and the agent was able to maximize the fitness in just about 75 generations. Hence, the agent learnt to kite using Stalkers and was able to defeat the Zealots. This kiting was more effective than when the *Trained agent* was used to control Stalkers in section 3.1.4.3 and accounted for the difference in speed and range of the stalkers.

ii. Roach V Zealot

This training was similar to the Stalker V Zealot except for the self-units which were Roaches in this case. Roaches have a lower attack range and a slower speed as compared to Hellions. This combined with the space constraint in the map made it harder for the Roaches to kite properly. However, the agent was able to learn the desired results in about 200 generations. Figure 20 visualizes the results. Once again, this kiting was more effective than when the ***Trained Agent*** was used to control roaches in section 3.1.4.3 and accounted for the difference in speed and range of the roaches.

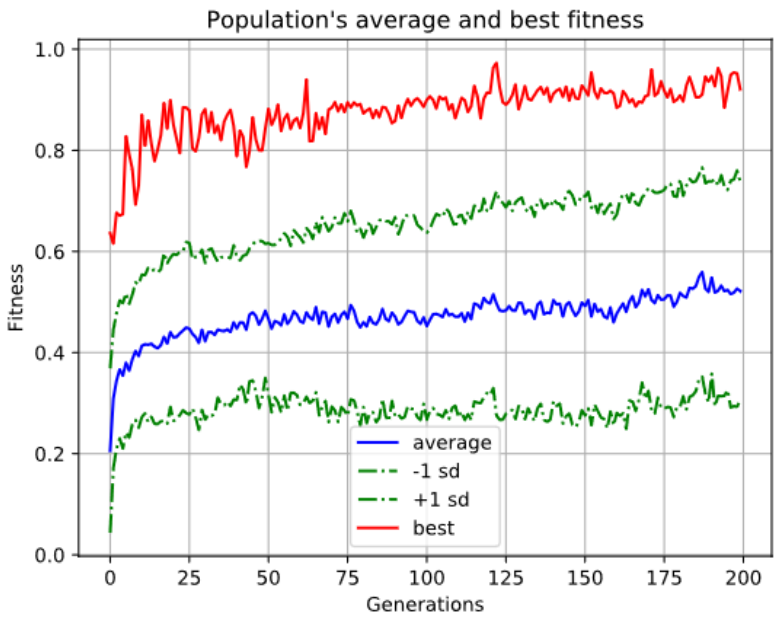

Figure 20: Test results of Combat Agent 2 using config 2 with engagement strategy 1 on Roaches V Zealot map with a Fitness Function (4)

iii. Zealot V Hellion

This training switched the ranged and melee units between the agent and the in-game AI. The agent now controlled Zealots against the scripted Hellions. The ideal strategy would be to head on attack Hellions since they are weak if they are not properly kited. The agent quickly learnt the desired strategy and charges and attacks the ranged enemy units as can be seen from Figure 21. The in-game AI does not know how to kite and therefore loses against the melee units if they engage.

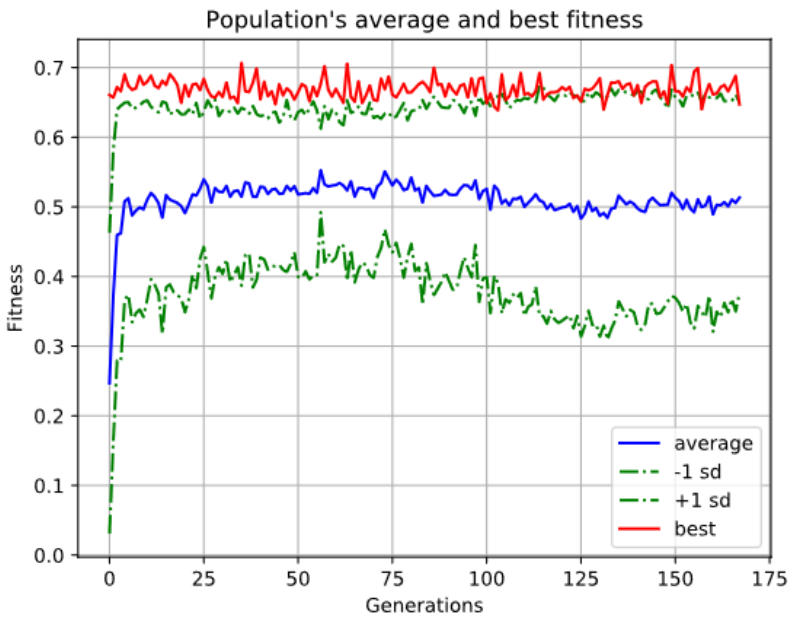

Figure 21: Test results of Combat Agent 2 using config 2 with engagement strategy 1 on Zealot V Hellion map with a Fitness Function (4)

3.1.6 Generalization Trainings

After achieving successful results in the individual trainings, we worked towards generalizing our previously Trained Agent from (section 3.1.3.4) over different environmental conditions. In the subsequent trainings we used config 2, combat agent 2 and engagement strategy 1. We hoped for our ***Trained Agent*** to retain the previously learnt strategies and additionally learn new strategies according to the units in play. Our attempts to generalize the agent included the following:

3.1.6.1 *Trained Agent* over Stalker V Zealot

The ***Trained Agent*** was retrained on the Stalker V Zealot map. Figure 22 shows the result of the training over 400 generations. It can be clearly seen that the average fitness remains constant without decreasing and the best genome continues to perform well over the new setting.

The new agent was evaluated over the Hellion V Zealot map and the Stalker V Zealot map and it performed optimally on both of them. It achieved a high level of micro in both scenarios and therefore we manage to get a single network that performed well on different types of ranged units.

Figure 22: Test results of Trained Agent using config 2 with engagement strategy 1 on Stalker V Zealot map with a Fitness Function (4)

3.1.6.2 *Trained Agent* over Cycling maps

A new set of cycling maps were developed. These maps changed the units in play every time it was reset. Below is a list of the maps with the unit details:

Map	Self-Units	Enemy-Units	Micromanagement Strategy
Cycling enemy melee units	5 Hellions	5 Zealots, 10 Zerglings, 5 Ultralisks	Kiting
Cycling enemy melee and ranged units	5 Hellions	5 Zealots, 5 Roaches	Kiting against Zealots and Stutter Stepping against Roaches
Cycling enemy melee and self ranged units	5 Hellions, 5 Stalkers, 5 Roaches	5 Zealots, 10 Zerglings, 5 Ultralisks	Kiting
Cycling self ranged units	5 Hellions, 5 Stalkers, 5 Roaches	5 Zealots	Kiting
Cycling self melee and ranged units	5 Hellions, 6 Zealots	5 Zealots	Kite with Hellions, Engage with Zealots

The agent failed to learn optimally over any of the cycling maps. This may be because the network does not get enough training time to learn any one particular strategy due to the high frequency of change in training scenarios. This could also be due to insufficient amount of information provided by inputs. In the next section we discuss a new combat agent that we developed to attempt to solve this.

3.1.6.3 Refined Combat Agent

i. Inputs

Refined Combat Agent had inputs in addition to the Combat Agent 2. These additional inputs accounted for the unit speeds, weapon ranges and their types. Below is the list of the inputs. For details of these inputs please refer to DEV-3.4.3.3-41.

Current hp	Weapon cooldown	Enemy in range	Previous command
North bound	South bound	West bound	East bound
North West enemy presence	North East enemy presence	South West enemy presence	South East enemy presence
In enemy range	Self-unit type	Enemy unit type	Self-weapon range
Enemy weapon range	Self-Movement Speed	Enemy Movement Speed	

ii. Outputs

The outputs were the same as used in Combat Agent 1 and 2:

1. Fight/Flee - boolean
2. Displacement in x
3. Displacement in y

iii. Hellion V Zealot

To validate our new inputs, we first trained the new agent on the Hellion V Zealot map from which we previously got our ***Trained Agent*** (section 3.1.3.4) with combat agent 2. The new agent was rather slow to maximize the fitness and in the first 200 generations it was only able to maximize the fitness to 0.65. Figure 23 shows the results of the training for the first 200

generations. Since the fitness was not maximized we continued the training for another 200 generations as shown by Figure 24. After training for a further 200 generations the agent was close to maximizing the fitness. Hence, it was proven that the new inputs help the agent achieve the desired result and the agent learnt to perform kiting to win the battle every time. However, due to an increase in the inputs it took more generations to learn.

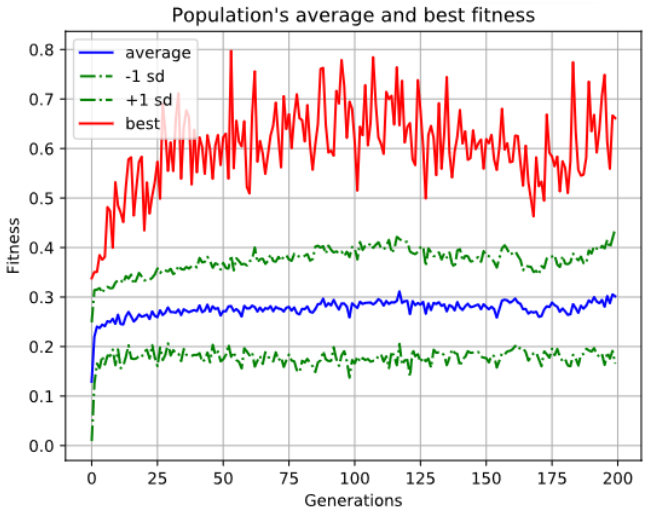

Figure 23: Test results of first 200 generation of Refined Combat Agent using config 2 with engagement strategy 1 on Hellion V Zealot map with a Fitness Function (4)

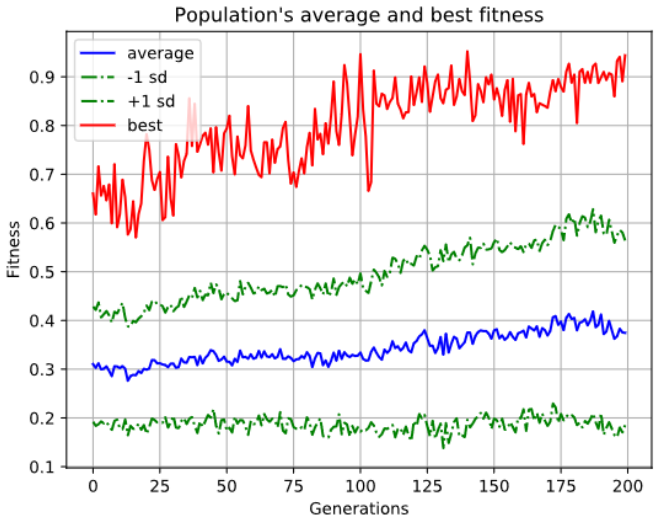

Figure 24: Test results of the continued 200 generation of Refined Combat Agent using config 2 with engagement strategy 1 on Hellion V Zealot map with a Fitness Function (4)

This same network was further trained on the Stalkers vs Zealot map. The fitness was maintained, and the newly evolved network learnt optimal kiting strategies for both hellions and stalkers. The results are left out for sake of brevity.

Similarly, the same network was further trained by switching the units such that the agent now controlled zealots against AI hellions. Figure 25 shows the training. Upon evaluation, it was observed that the network learnt to attack head-on with the zealots to win the battle consistently. However, when re-evaluated using Hellion units, it no longer knew how to kite and kept losing.

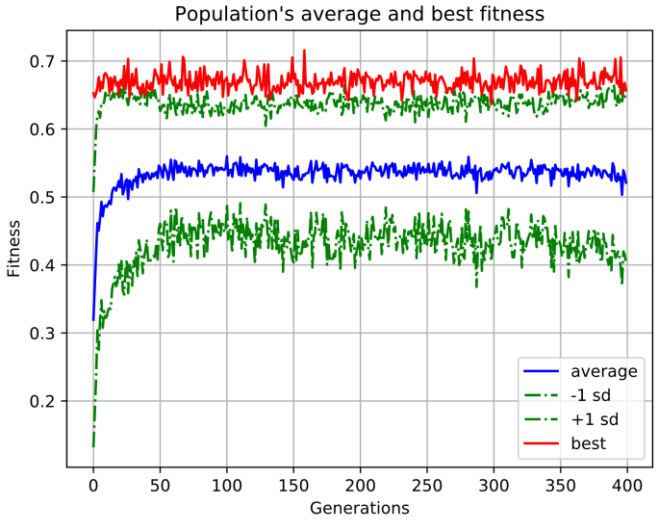

Figure 25: Test results of the continued training of Refined Combat Agent using config 2 with engagement strategy 1 on Zealot V Hellion map with a Fitness Function (4) over Trained Agent

iv. Cycling maps

Since our previous cycling map tests failed to generalize the agent over different unit matchups, we ran training on the Refined Combat Agent with enhanced inputs. However, the results were not very different than our previous attempt. The agent failed to learn multiple strategies for different types of units and failed to generalize. This may once again be attributed to lack of training time against any one-unit matchup due to the high frequency of change. For example, in a map where Hellions played against cycling Zerglings and Zealots the agent learnt to just attack head on and perform no kiting. This would work for Zerglings (which had low health) but not for Zealots.

However, some learning was observed for the case of cycling self-ranged units. Figure 26 shows the training progress. The agent learnt to kite with the different self-ranged units. However, this kiting was sub-optimal and not as perfect as the networks which were trained independently on those ranged separately.

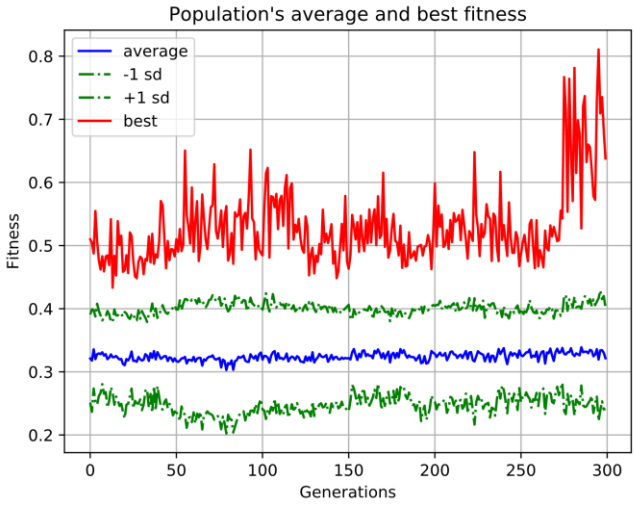

Figure 26: Test results of the continued training of Refined Combat Agent using config 2 with engagement strategy 1 on Cycling Self map with a Fitness Function (4) over Trained Agent

3.1.6.4 Hetero-Combat Agent

So far, we have been training the agent to learn to micromanage homogenous units (all of our units were of the same type). We had successful results whenever the agent was trained individually against different units. In situations where the units over which the agent had learnt the strategy were changed the agents performed sub optimally even though the overall behavior was maintained.

In this section, we focus on our attempts to micromanage heterogenous units (our controlled units are of different types). The control logic for the units after working out available unit types is in a cycle:

1. Select unit group.
2. Retrieve inputs with respect to that unit group and give a fight or flee command
3. Repeat step 1 and 2 for the next unit group

i. Inputs

We mention the inputs below. For a detailed understanding of the inputs you may refer to (DEV-3.4.4.3-41)

Current hp	Weapon cooldown	Enemy in range	Previous command
North bound	South bound	West bound	East bound
North West enemy presence	North East enemy presence	South West enemy presence	South East enemy presence
In enemy range	Self-unit type	Enemy unit type	Self-weapon range
Enemy weapon range	Self-Movement Speed	Enemy Movement Speed	Distance to Enemy

ii. Outputs

The outputs were the same as used in Combat Agent 1, 2 and Refined Combat Agent:

1. Fight/Flee - boolean
2. Displacement in x
3. Displacement in y

iii. Hellion and Stalker V Zealot

We modified the standard map (section 3.1.1) to develop a map for our test. In this new map the agent controlled three hellion units and three stalker units against eight scripted zealot units.

We first attempted to use our already trained networks together to control the heterogenous army i.e. we used the trained agent for StalkersVsZealots for controlling the Stalkers and the trained agent for HellionsVsZealots for controlling the Hellions. However, the two networks working together did not manage to produce successful results. This was mainly due to the fact that the enemy was now responding differently to being faced with two types of independently controlled units. This response was significantly different to the enemy response during the training of the networks and so the networks could not accommodate for it. In addition, the individual networks were trained on the assumption that the network would be able to command the units at each step. However, due to the control logic of the Hetero Combat Agent, the network is only able to

command the same unit group every third step. This affects the preciseness of the micro-control leading to poor results.

Due to the failure of the multiple network control we decided to train the agent (so that one network would be responsible for controlling both types of units). This was run on the new map for 1000 generations. Figure 27 shows the results of the last 300 generations of the training. No concrete learning was seen, and the agent failed to increase the fitness. This was the last test that we conducted for neuro-evolution.

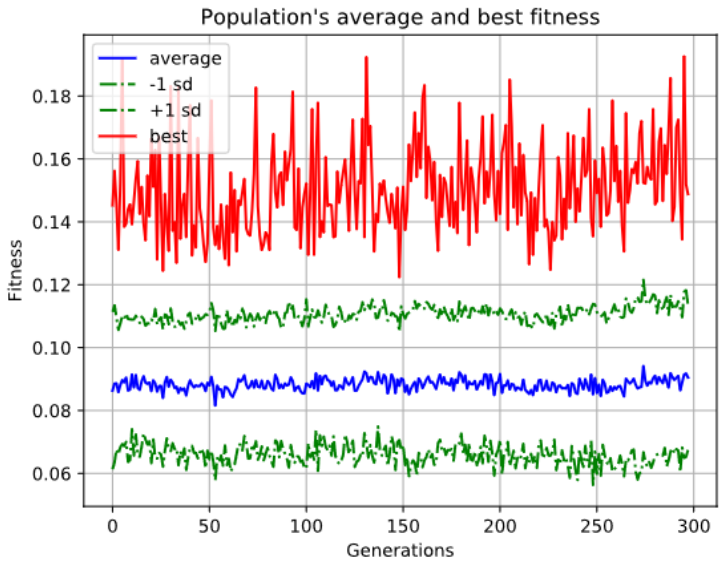

Figure 27: Test results of the last 300 generation of Hetero Agent using config 2 with engagement strategy 1 on Hellion and Stalker V Zealot map with a Fitness Function (4)

3.2 Reinforcement Learning.

This section discusses the testing and evaluations for reinforcement learning. In our project we used the Sarsa (λ) reinforcement learning approach. This approach is described in detail at (DEV-3.4.4.1-44).

3.2.1 Map Specificaiton

The standard map used in Sarsa tests is larger than the one used in neuro-evolution tests. This is due to the fact that we have used Python SC2 (DEV-3.4.4.2-45) for sarsa which allows the agent to obtain observations from the entire map instead of just being bounded by the camera. Hence, the map had dimensions of 128 x 128. The standard map had two players, Player 1 controlled by

the sarsa agent and Player 15 controlled by the in-game AI. An episode is started at the start of the map which is reset each time any of the players loses all of its units. Since the map is huge, some of the episodes would last very long without any progress as the units will go to the opposite corners of the map and stay there. Therefore, we added a reset timer of 60 seconds which resets the whole map if no unit is killed before the timer expires. The choice of the duration of the reset timer was arbitrary as in normal adversarial scenarios at least a unit is killed within 60 seconds. We modified the units for the standard map for various tests as per the requirement. As before, we removed the auto attack feature to remove any unfair advantage for the Sarsa agent. A list of maps developed for sarsa tests can be found at (DEV-3.4.5.3-52)

3.2.2 States

Sarsa agent requires states to abstract the required information. Details about the state definition can be found at (DEV-3.4.4.3-46). Below we mention the state definition briefly for reference:

1. Unit Type of currently selected unit group
2. Unit Type of closest enemy to currently selected unit group
3. Distance to closest enemy
4. Scaled Relative Power
5. Weapon On Cooldown
6. Is Together

3.2.3 Reward Functions

The reward is calculated and used for learning at each step as follows:

$$reward = damage\_dealt - damage\_received * (1 - aggression)$$

The value of aggression can be set before training is launched and is mentioned when reporting results in the following sections.

3.2.4 Parameters

To drive the learning of Sarsa agent we are required to set a few parameters. Details about these parameters can be found at DEV-3.4.4.3-47:

Learning rate α	Reward decay/Discount Factor γ	Trace decay λ	Epsilon ε
0.05	0.9	0.9	0.5 – 0.95

3.2.5 Homogenous Units

3.2.5.1 Hellion V Zealot

i. Unit details

Hellion: Ranged unit with a fast speed. However, it has a low armor and health which requires it to be microed effectively.

Zealots: Melee unit with a relatively slower speed but a very strong attack. At a close-range Zealots can defeat Hellions with ease.

The map consisted of 7 Hellions controlled by the agent and 10 Zealots, controlled by the scripted in-game AI.

ii. Optimal Strategy

To win, the optimal strategy for Hellions is to kite against the relatively slower Zealots as mentioned in section 3.1.3.4. Hellions can use their faster speed to their advantage to stay out of the range of Zealots while attacking them.

iii. Results

The results were successful. The agent was able to learn the micromanagement strategy of kiting optimally. Figure 28 shows the results of the training. It only took the agent 600 episodes of training to learn the strategy, after which the reward remains stable near the maximum value.

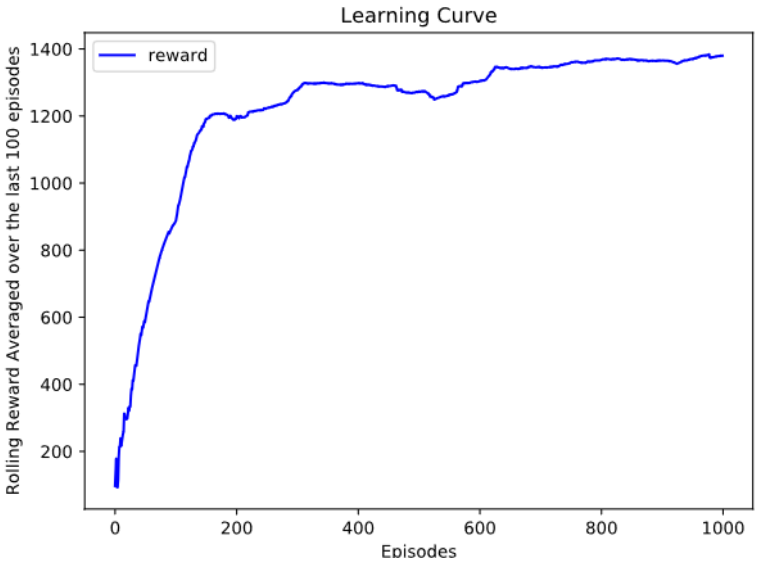

Figure 28: Test results of Sarsa Agent on Hellion V Zealot map

A similar training was launched in which the Hellions were switched to Stalkers. An optimal kiting strategy was learnt once more but the results are not reported here for sake of brevity.

3.2.5.2 Marine V Baneling

i. Unit details

Marine: Ranged unit with a medium speed. It can make use of kiting to defeat weak melee units.

Baneling: It is a fast unit with a very short attack range that involves exploding. However, it causes splash damage to groups of units by suicide bombing.

The map consisted of 25 Marines controlled by agent against 26 scripted Banelings.

ii. Optimal Strategy

In this map instance it is very hard for the Marines to win against the Banelings as Marines are slower than the Banelings. Marines can only do maximum damage when they attack in groups, however, the whole group can be killed by a single blasting Baneling. Hence, in this case it would be ideal for the Marines to scatter and receive the damage individually instead of in groups.

iii. Results

The results were successful for this test as well. Since the optimal strategy was difficult to discover, it took about 25000 episodes for the agent to learn. The results are shown in Figure 29.

As mentioned, the agent demonstrated the scattering behavior to disperse the Banelings and take the damage individually. Figure 30-31 and 32-33 show the initial positioning of the Marines and the scattering behavior respectively.

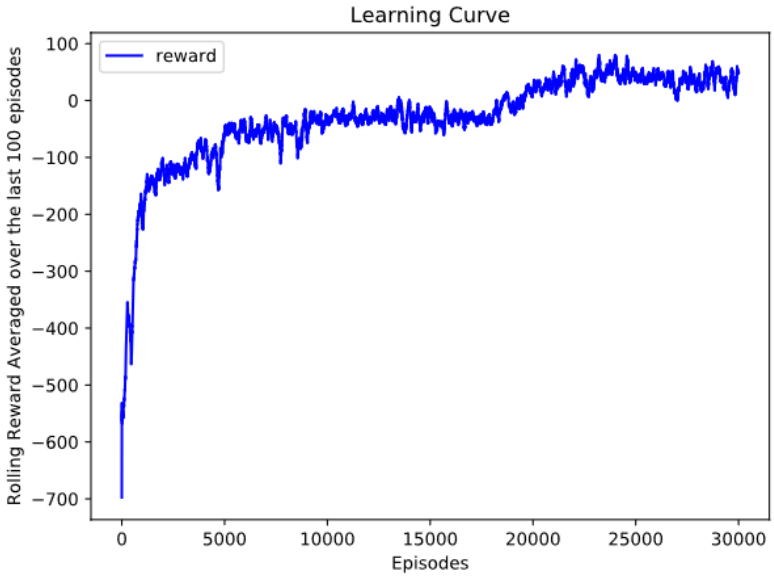

Figure 29: Test results of Sarsa Agent on Marine V Baneling map

Figure 30: Initial positioning of Marines on Marine V Baneling map

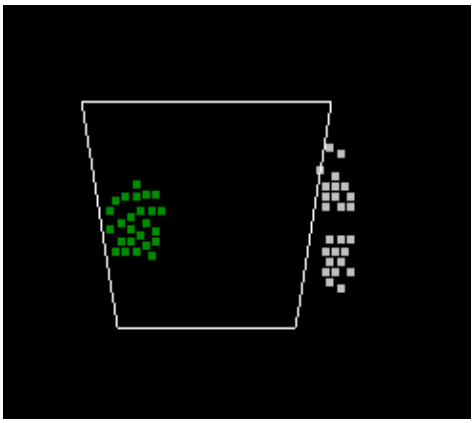

Figure 31: Minimap view of the situation in fig 30. Green dots are Marines, White dots are Banelings

Figure 32: Optimal Strategy of scattering Marines performed by Sarsa Agent on Marine V Baneling map

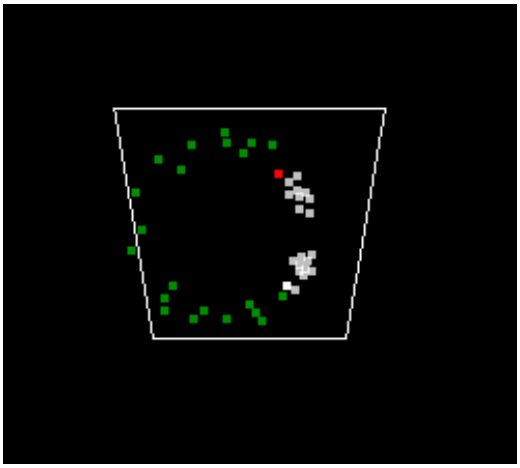

Figure 33: Minimap view of the situation in fig 32. Green dots are Marines, White dots are Banelings

3.2.5.3 Zergling V Marine

i. Unit details

Zergling: A fast melee unit with a very low health.

Marine: Ranged unit with a medium speed. It can make use of kiting to defeat weak melee units.

This map consisted of 30 Zerglings controlled by agent against 15 Marines.

ii. Optimal Strategy

Although Zerglings are low in health, an army of Zerglings can easily defeat a much stronger ranged unit by head on attacking strategy. Hence, it will be ideal to surround the Marines and attack them.

iii. Results

Since the optimal strategy was straightforward the agent was able to learn it in about 250 episodes of training. Figure 34 shows a fast-increasing reward up to 250 episodes until it maximizes and remains constant for the subsequent episodes. This agent was the fastest learning agent so far in our sarsa trainings. Figure 35-36 shows the initial positioning of the Zerglings and Figure 37-38 shows the optimal strategy of attacking and surrounding the Marines.

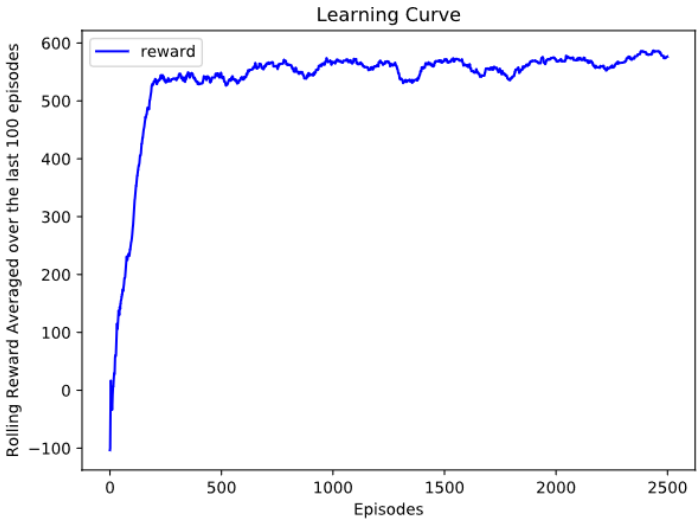

Figure 34: Test results of Sarsa Agent on Zergling V Marine map

Figure 35: Initial positioning of Zerglings on Zergling V Marine map

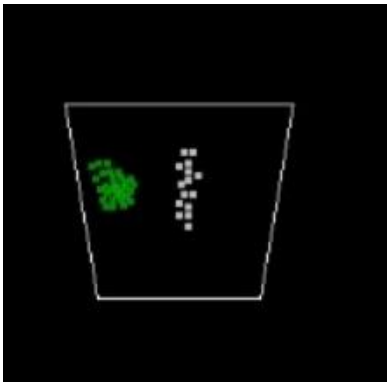

Figure 36: Minimap view of the situation in fig 35. Green dots are Zerglings, White dots are Marines

Figure 37: Optimal Strategy of attacking and surrounding Marines performed by Sarsa Agent on Zergling V Marine map

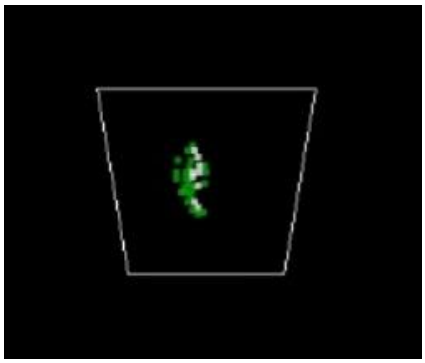

Figure 38: Minimap view of the situation in fig 37. Green dots are Zerglings, White dots are Marines

3.2.6 Heterogenous Units

After successful results with Homogenous units, we decided to move further and test whether the Sarsa agent can learn over Heterogenous scenarios. Our first priority was to tackle the scenario which failed in NEAT as elaborated on in section 3.1.6.4

3.2.6.1 Hellion and Stalker V Zealot

Here we used the same settings as the NEAT test. However, the agent failed to learn any significant strategy. We believe that this was due to the space constraint and since we were using Python-SC2 for Sarsa tests, we had the freedom of using a bigger map. hence, we shifted our training to the standard 128 x 128 Sarsa map (DEV-3.4.5.3-53).

i. Optimal Strategy

The optimal strategy here would be to kite both Stalkers and Hellions. Since Stalkers and Hellions have different speeds, it would be ideal to kite in different directions and break the Zealots into two groups

ii. Results

The results were finally very encouraging as the agent learnt the optimal strategy. Figure 39 shows the increasing trend in the reward that the agent was able to retrieve. Figure 40-41 show the initial positioning of the units and then Figure 42-43 shows the instance where Hellions and Stalkers separate the Zealots into two groups.

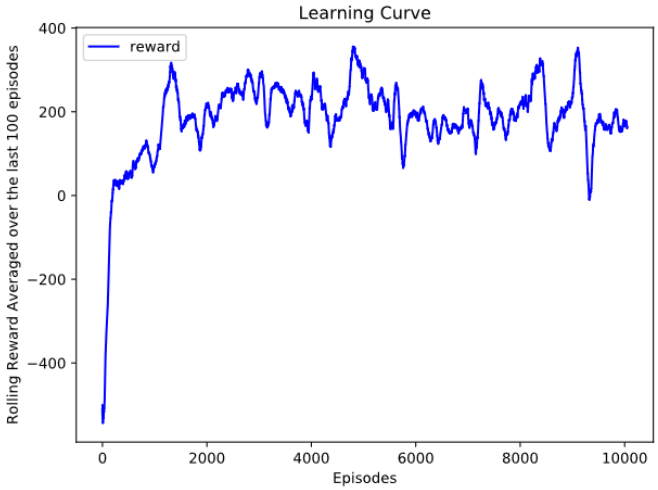

Figure 39: Sarsa Agent on Hellion and Stalker V Zealot

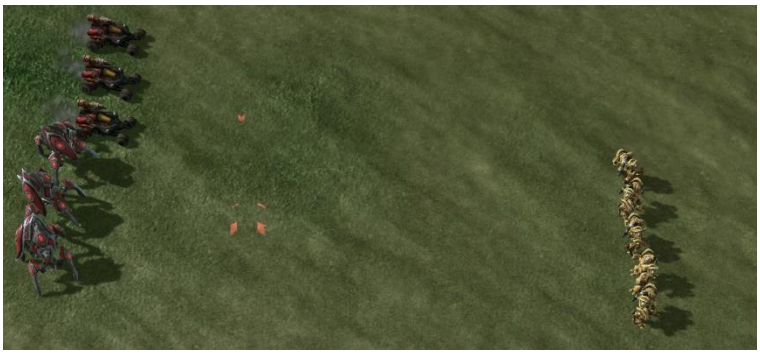

Figure 40: Initial positioning of Hellion and Stalkers on Hellion and Stalker V Zealot

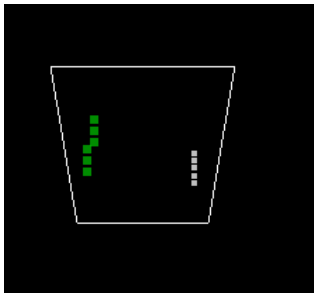

Figure 41: Minimap view of the situation in fig 40. Green dots are Hellions and Stalkers, White dots are Zealots

Figure 42: Optimal Strategy of kiting in different directions performed by Sarsa Agent on Hellion and Stalker V Zealot

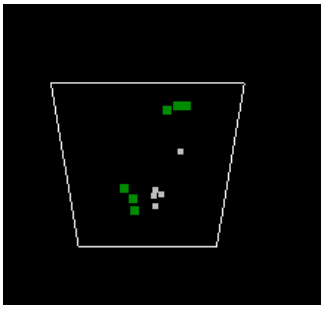

Figure 43: Minimap view of the situation in fig 42. Green dots are Hellions and Stalkers, White dots are Zealots

3.2.6.2 Zergling and Roach V Baneling and Immortal

i. Unit details

Zergling: A small and fast melee unit. An army of Zerglings can easily defeat a much stronger ranged unit by head on attacking strategy.

Roach: Medium ranged unit with a slow speed. However, they have large health values and high damage.

Baneling: It is a fast unit with a very short attack range. However, it causes splash damage to groups of units by suicide bombing.

Immortal: It has a large attack range with a powerful attack. Additionally, it has a strong armor and shield.

The agent controlled 25 Zerglings and 6 Roaches against 7 Banelings and 4 Immortals controlled by the scripted in-game AI

ii. Optimal Strategy

The optimal play would be to pull back the Zerglings at the start of the match and defeat Banelings with Roaches, since Zerglings have a low health which will not be able to sustain the splash damage from the Banelings' bombing. After the Banelings are killed, it is best to attack Immortals with the fast moving Zerglings since they are the most effective in head on attack against ranged units. Additionally, the Immortals are strong against Roaches and so the Zerglings should attract their attention before the Roaches engage.

iii. Results

This training was an attempt to train the sarsa agent to control heterogenous units in complicated scenarios. The optimal strategy was a complicated one, however, Figure 44 shows that the agent was able to improve the reward over episodes until it was maximized. Therefore, the training was a success. Figure 45 shows the instance of pulling the Zerglings back while Roaches face Banelings and Figure 46 shows the instance of Zerglings pushing to attack the Immortals once the Banelings are killed

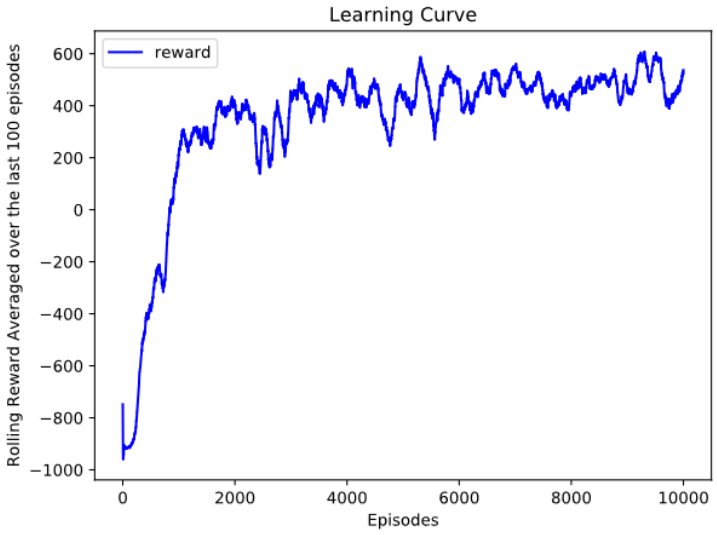

Figure 44: Test results of Sarsa Agent on Zergling and Roach V Baneling and Immortal

Figure 45: Optimal Strategy of pulling back Zerglings performed by Sarsa Agent on Zergling and Roach V Baneling and Immortal

Figure 46: Optimal Strategy of engaging Immortals with Zerglings performed by Sarsa Agent on Zergling and Roach V Baneling and Immortal

3.2.6.3 Banshee and Marine V Corruptor and Ultralisk

i. Unit details

Banshee: It a flying unit which can only attack ground units. In a group it can prove to be very deadly for ground units

Marine: Ranged unit with a medium speed. It can make use of kiting to defeat weak melee units.

Corruptor: Air unit with a strong anti-air capability. However, it has a slow speed which makes it difficult to evade fast moving ground to air units.

Ultralisk: A powerful melee unit with massive health and splash damage capability.

The agent controlled 7 Banshees and 30 Marines against 9 Corruptors and 6 Ultralisks controlled by the scripted in-game AI.

ii. Optimal Strategy

Since Banshee cannot attack air units, the optimal strategy would be to pull back Banshees as the Corruptors approach. The Marines can then engage Corruptors, who cannot engage the ground units in return. Once Corruptors are killed, the Marines should engage the Ultralisks by scattering to avoid the splash damage. Simultaneously, Banshees should move in to support Marines and attack the Ultralisks.

iii. Results

The agent successfully learnt this complex strategy with heterogeneous units as well. Figure 47 supports the claim as the reward increases until it is maximized. Figure 48 shows the instance when Banshees are pulled back to save them from Corruptors and the Marines engaging Corruptors. The optimal strategy of scattering before engaging the Ultralisks by Marines is shown in Figure 49. Later in Figure 50 we can see Banshees engaging Ultralisks to support the Marines since the air threat is now neutralized.

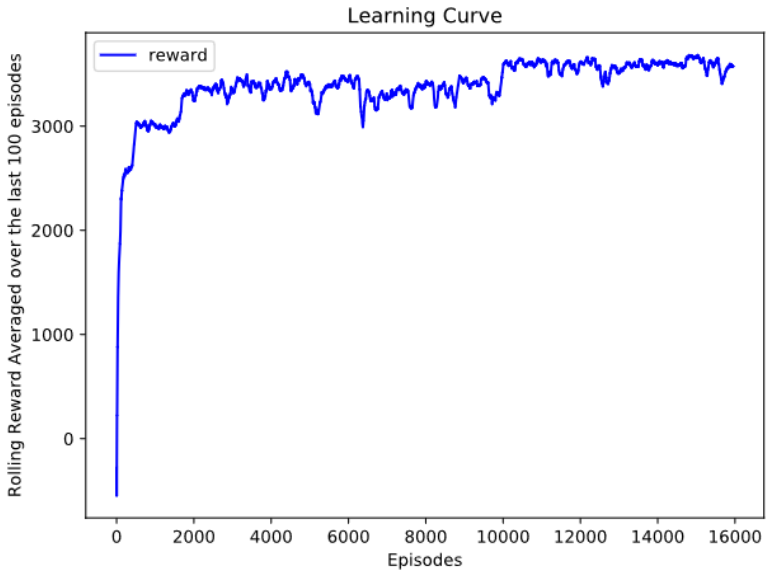

Figure 47: Test results of Sarsa Agent on Banshee and Marine V Corruptor and Ultralisk map

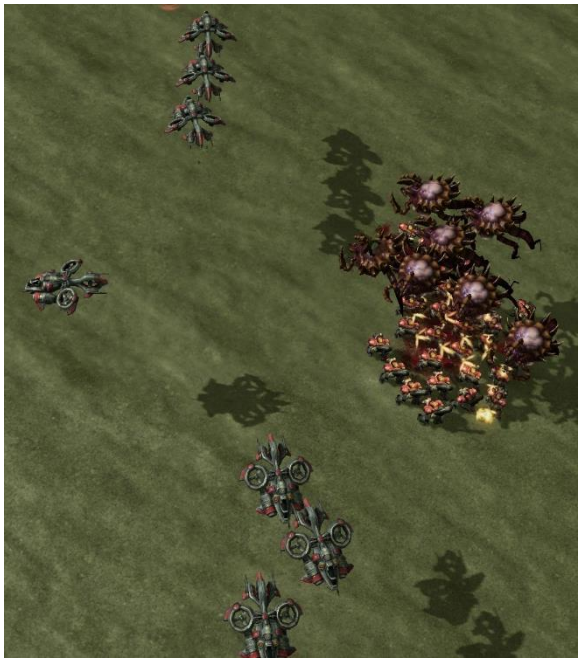

Figure 48: Optimal Strategy of pulling Banshee back performed by Sarsa Agent on Banshee and Marine V Corruptor and Ultralisk

Figure 49: Optimal strategy of scattering Marines before engaging Ultralisks perfomred by SARSA agent on Banshee and Marine V Corruptor and Ultralisk

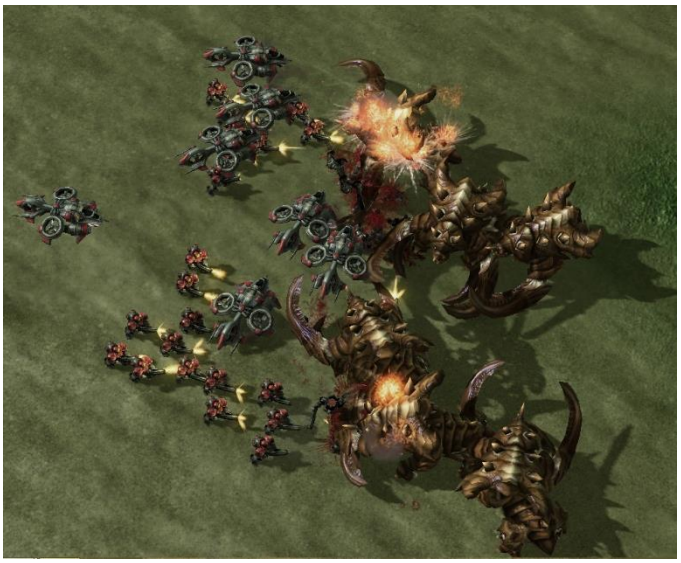

Figure 50: Optimal Strategy of engaging Ultralisks with Banshees to support Marines performed by Sarsa Agent on Banshee and Marine V Corruptor and Ultralisk map

3.2.7 End to End Game

After encouraging results with the Sarsa agent for self-contained micromanagement scenarios we attempted to transfer it to the end-to-end StarCraft II game where two players must build up their armies and then destroy the opponents base. Details of development can be found in the development report at DEV-3.4.4.3-47.

The training was launched against different difficulty levels on the Simple64 StarCraft II map. However, results were not very substantial. Figure 51 shows the progress of training for the bot on the medium difficulty. Unfortunately, training times are significantly longer than in previous cases due to each game lasting at least five minutes even after being sped up. This means that due to time limitations we were not able to run enough games for true learning to emerge. The wins can be attributed to creating a very large enemy that the enemy cannot contest against rather than good micro. However, we are confident that should the training be run for sufficient number of games good micro behavior should emerge.

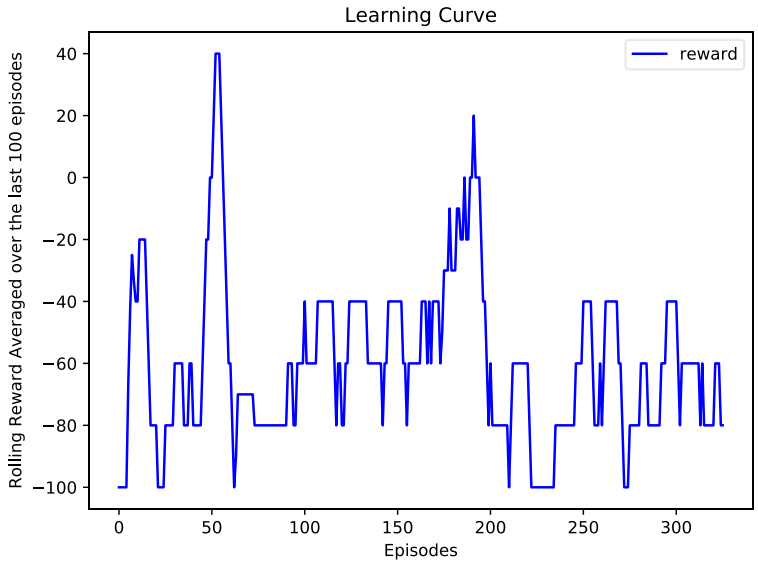

Figure 51: Test results of Sarsa Micro Agent on Simple 64 map in End to End game on Medium difficulty

4 Analysis

In this section, we comment on the success and shortcomings of our implemented approaches.

4.1 Neuro-Evolution

We first applied the neuroevolutionary approach with the NEAT algorithm on micromanagement scenarios. Specifically, we attempted to evolve networks that could learn the appropriate fighting strategy in a ranged vs melee matchup or vice versa.

The neuroevolutionary approach successfully evolved a micro-controller that achieved the desired behavior of kiting/hit-and-run in the case of the ranged vs melee matchup (section 3.1.3.4) and achieved results comparable to [3]. Figure 52 and 53 indicates the strength and skill of the evolved control which maintains good scores even as enemy units are increased. This behavior is impressive considering the fact that the map area is very restricted and thus requires very precise micro-control. In addition, even the in-game scripted *Elite* level AI for ranged units does not know how to hit-and-run and that behavior must be separately hard coded when making AI bots.

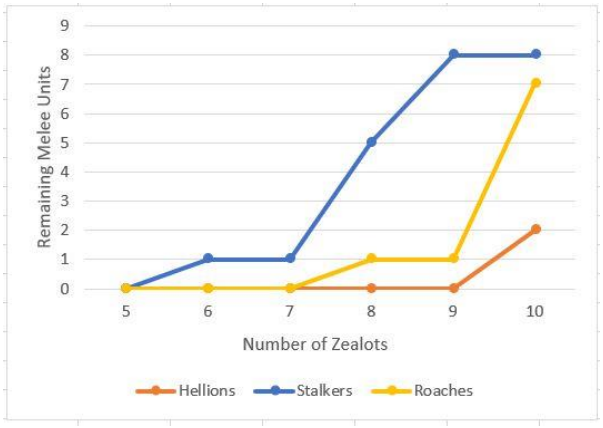

Figure 52: Remaining Zealots against Number of Zealots in battles against Hellions, Stalker and Roaches.

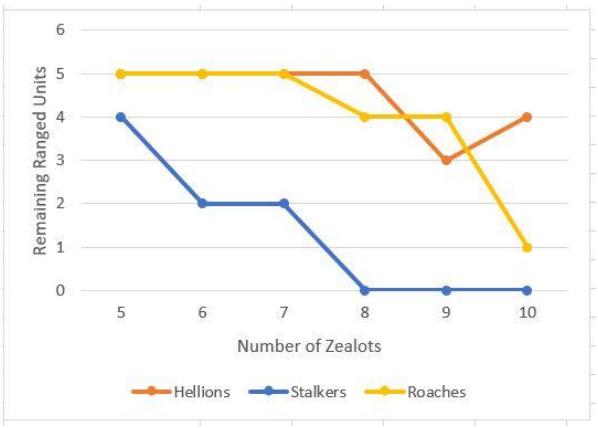

Figure 53: Remaining Ranged Units against Number of Zealots in battles against Zealots.

These networks become more robust to changing conditions if they are incorporated into the training process. For example, incorporating changing starting configurations into the training process means the evolved network learns to perform well regardless of where units are spawned

on the map. Similarly, if an evolved network for one ranged unit (for example Hellion) is further trained for another ranged unit (for example Stalker) then the newly evolved network learns the optimal strategy for both (section 3.1.6.1).

Optimal behavior is evolved fairly quickly, and a successful controller is generally achieved within a hundred generations with the restricted input set as used in Combat Agent 2 (section 3.1.3.2) as shown in figure 54.

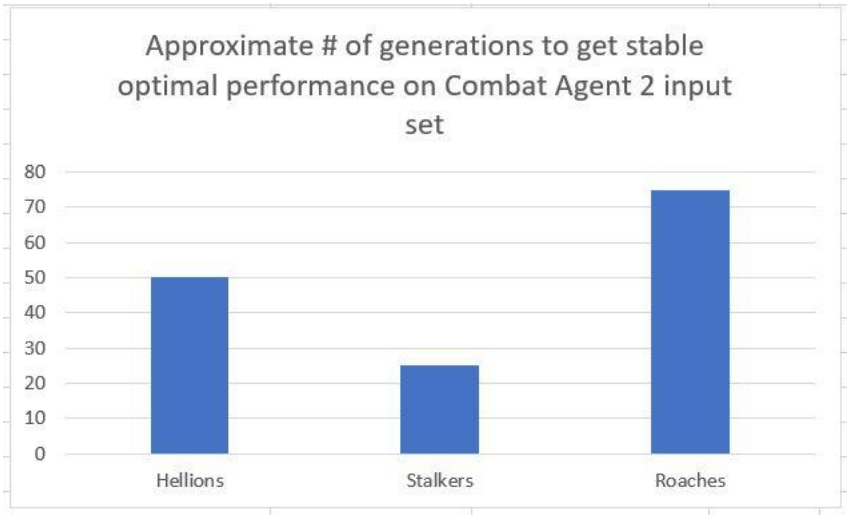

Figure 54: Approximate number of generations to get stable optimal performance of Combat Agent 2 input set

Similarly, the neuroevolutionary approach successfully evolved a micro-controller that achieved the desired behavior of attacking head-on in the case of the melee vs ranged matchup (section 3.1.5.2 (iii)). The evolved melee units consistently win because the in-game AI for ranged units does not know to kite even when set to an *Elite* difficulty level.

Ranged vs Ranged matchup was not emphasized during testing and evaluation since that relies more on unit matchups than good micro-strategy. In the ranged vs ranged matchup, a very advanced technique is *"stutter-stepping"* which means to stay outside the enemy's firing range while using your range advantage to fire. However, this requires very precise control that would be unreasonable on a small map. Likewise, the melee vs melee matchup was not emphasized on during testing and evaluation since the only evolvable strategy is to attack head-on in which case the winner would be whoever managed to get the first hit by chance. Any neural structure

mutations evolved during the evolution process would lead to a drop-in performance if it prioritized running away for melee units.

In an attempt to create more robust agents, the input set was extended (section 3.1.6.3). The agent once again relearnt all the scenarios with the reduced set albeit at cost of an increased training time. However, no significant improvement was seen. The evolved network couldn't learn a strategy that would be optimal for controlling both ranged and melee units. On maps with cycling units, most trainings failed. However, in the case of training during which we cycled between different types of ranged units for itself (Hellions, Stalker, Roaches), the agent learnt kiting behavior though at a suboptimal level (section 3.1.6.3 (iv)).

This extended input set was applied to scenarios in which we controlled different types of units concurrently with the Hetero-Combat Agent (section 3.1.6.4). However, the agent couldn't learn a successful strategy even after a thousand generation training. This could be attributed to the process of selecting and commanding units' groups one after another being too complicated for NEAT to handle especially on such a small map.

The neuroevolutionary approach managed to learn precise micro with rather sparse networks. The evolutionary process just optimizes for the fitness function. Starting from small networks and gradually introducing complications means that simple effective networks are evolved with a reduced search space which decreases the time needed to find an optimal solution. In addition, for the neuro-evolutionary approach one does not need to define the neural network structure before-hand and the best structure is discovered over time.

However, this approach is not without its faults and limitations. The simple networks evolved do not generalize too well and are sensitive to training conditions with performance drops if the conditions (such as map size) change. To transfer these networks to the full end-to-end StarCraft II game would be a significantly hard challenge because of the increase in diversity and complexity. It would take a significantly longer time to evolve large enough networks that could hope to perform competently in such complex scenarios. This increase in training times could however be reduced by parallelizing the training across a large enough distributed computational cluster since the neuroevolutionary approach is trivial to massively parallelize.

The neuro-evolutionary approach through NEAT overcomes some of the issues that are inherent in temporal difference RL algorithms such as Sarsa(λ). The training process isn't as sensitive to the size of the state space in comparison to Sarsa. Furthermore, the evaluation of actions taken in a game are done using the accumulated reward at the end of the game episode and not at each game step as is the case with TD algorithms. This helps avoid the issue of delayed reward for actions which is particularly useful for games like StarCraft II with long reward horizons. In addition, TD algorithms require an initial period of exploration where reward is received infrequently. Neuroevolutionary approaches on the hand do not require the period of exploration and can optimize towards the fitness function from the beginning of the training.

Traditionally, gradient-descent methods with backpropagation have been popular for machine learning applications. Neuro-evolutionary algorithms like NEAT present an alternative approach. In particular, these approaches are well-suited to control problems and can be simulated (making video games a well-suited test environment for them) as one can just define a suitable fitness function that defines good behavior in the simulation. It is harder to apply gradient-descent based supervised learning to the same problems because of the difficulty of generating a dataset that defines good behavior in that environment. In addition, neuroevolutionary approaches have been shown to converge much faster to a solution. [2] demonstrates how deep neuro-evolution can be used to achieve high quality results on the ATARI platform in around 4 hours for a desktop in comparison to the several days of training needed with a deep reinforcement learning algorithm such as Deep Q-Learning. We planned to experiment and compare the performance of NEAT against a Multi-layer perceptron-based reinforcement learning algorithm such as DQN but were unable to do it due to lack of time.

4.2 Reinforcement Leanring with Sarsa (λ)

After exploring NEAT, we applied reinforcement learning with a temporal difference algorithm known as Sarsa (λ). Specifically, we attempted to apply it to a wider variety of micromanagement scenarios than was possible with our implementations for NEAT agents because of shifting from PySC2 to Python-SC2 framework.

The Sarsa agent successfully managed to learn optimal strategies in all of the micromanagement scenarios that were tested. To roughly compare against the tasks that the NEAT agent worked on, the Sarsa agent was first trained on a homogenous ranged vs melee matchup (Hellions vs

Zealots) on a larger map (section 3.2.6.1). The agent effectively achieved the desired behavior of kiting/hit-and-run.

Following this, the Sarsa agent was tested in scenarios in which it had to learn more creative micromanagement strategies. In the Marines vs Banelings scenario, the agent learnt to scatter the marines wide before engaging the enemy in order to avoid the massive area-of-effect damage of the Banelings (section 3.2.5.2). Similarly, in the Zerglings vs Marines scenario, the agent learnt to rush and attack the enemy head-on with the Zerglings (section 3.2.5.3).

NEAT was unable to learn on a heterogenous army composition (our army consists of different types of units) due to a combination of limitations of the PySC2 framework and NEAT itself. The Sarsa agent successfully managed to learn strategies when applied to the hetero scenarios. It not only learnt which of our units are strong against which of the enemy units but also learnt the timing for when to engage which enemy unit. This is impressive considering it takes human players experience to learn these nuanced behaviors that are not obvious at first observation. For example, in the Zerglings and Roaches vs Bane lings and Immortals scenario (section 3.2.6.2) the agent learns to first make the Zerglings retreat to avoid the Bane lings splash damage and engage with Roaches. Then once the Banelings are destroyed, the agent sends in the Zerglings to fight the Immortals before sending in the Roaches. By performing these maneuvers, it wins the battle through good micromanagement. Had it just engaged all units head-on it would have lost. Other hetero scenarios are mentioned in section (3.2.6).

The learnt behavior is obtained in a reasonable amount of time in the space of a few hours quicker than the non-parallelized training for NEAT (this may however be attributed to the fact that a tabular representation is used instead of a neural net). However, the tabular representation used for Sarsa (λ) has its drawbacks. The tabular representation means that the agent fails to perform optimally if it ever encounters a state it has not been trained on. This makes it harder to transfer to a full end-to-end game which consists of many variations in unit compositions for ourselves and the enemy. Nonetheless, it is theoretically still possible to transfer to a full game given enough time to train against all possible unit combinations and an effective state representation.

This was attempted towards the end of the project where the agent-controlled micro in an end-to-end game with scripted macro (section 3.2.7). However, due to lack of time the agent was unable

to play enough number of games to learn effective strategies to beat more than the Easy in-game AI. The natural next step to solve this issue would be to use a neural network for function approximation through deep reinforcement learning as in [4]. In addition, the delayed correlation between actions taken and rewards assigned introduces learning difficulties for a temporal difference method like Sarsa. The weight of this issue is alleviated through temporal assignment with the trace decay parameter in Sarsa(λ).

5 Conclusion

To conclude, we first discussed the importance of artificial intelligence in the domain of video games and vice versa and then went on to discuss the value proposition of using novel machine learning approaches such as neuro-evolution and reinforcement learning after an exploratory phase. As discussed earlier, StarCraft II is referred to as the next *"grand challenge"* for AI research dealing with issues such as navigation, resourcing, micro-control, incomplete information and long-horizon planning to name a few. The complexity of the problems in the StarCraft II environment make it an excellent testbed for machine learning approaches. Thus, solutions and techniques found to be successfully applied in StarCraft II can be transferred to other real-life domains.

In this project we explored two areas of machine learning - *reinforcement learning* and *neuro-evolution* on micromanagement in StarCraft II. We built and open-sourced a novel framework for applying neuro-evolution to StarCraft II using which we built several iterations of our neuro-evolution agents. The trained neuro-evolution agents successfully learnt precise *hit-and-run* strategies to achieve a significantly high win rate against the in-game AI.

Subsequently, we implemented the Sarsa(λ)reinforcement learning algorithm and applied it to a set of diverse micromanagement scenarios requiring more complex strategies. The trained Sarsa agents successfully learnt more complex strategies that involved kiting, effective positioning and enemy engagement selection and timing. Towards the end of the project, we attempted to transfer the Sarsa agent into a full end-to-end game scenario where the agent was in control of the micromanagement of the army created by the scripted macro of the bot. It was able to win sometimes on the Very Easy and Easy difficulty but consistently lost on the Medium and above difficulty. The wins could be attributed to the strength of the size of the army and less so because of good micro. However, it should be noted that these results on the end-to-end game were from

training on a very small set of games and true learning for a TD learning algorithm would only emerge with a large number of games played. Unfortunately, this was not possible due to time constraints.

There are many directions to take in the future to attempt to overcome the drawbacks of the approaches used as mentioned in the analysis section. Newer versions of NEAT such as rtNEAT[5], HyperNEAT [6] and ES-HyperNEAT [7] may be used. The HyperNEAT variants in particular are sensitive to the geometry domain of a problem and thus can also learn from pixel-based feature inputs rather than handcrafted features. All of the networks that were evolved were Feedforward networks modelling all the problems as simple MDPs. However, it may make sense to experiment with recurrent networks and LSTMs to accommodate for more long-term strategic planning and more complex coordination. A distributed version of the developed NEAT-SC2 framework can be written to massively parallelize and speed up training.

The Sarsa agent was able to generate some of these more complex behavior at the loss of being able to generalize outside of training conditions. The Sarsa agent can therefore be extended with the use of neural nets as function approximators for the action-value pairs as in [4] for an application of deep reinforcement learning. This would also make it more equivalent to compare against the neuro-evolutionary methods. In addition, all our agents applied actions to groups instead of individual units and so it would be interesting to apply multi-agent versions of our approaches to see if any interesting behavior emerges. Finally, work can be continued on incorporating the Sarsa micro agent into the full end-to-end StarCraft II game by training on a larger set of games.

Our implementations have shown the power and potential of the neuroevolutionary and reinforcement learning approaches for complex control problems. Through effective experimentation in the StarCraft II environment, one can transfer the findings of the report to real-world problems such as evolving effective controllers for robots. Our open-source Neuro-evolution-StarCraft II training framework along with the extensive testing done with our agents establishes neuro-evolution and reinforcement learning as promising machine learning techniques in this domain and represents meaningful contributions to the StarCraft II and artificial intelligence communities upon which the community may build upon.

6 References

[1] O. Vinyals, T. Ewalds, S. Bartunov, A. S. Georgiev, Petko Vezhnevets, M. Yeo, A. Makhzani, H. Kuttler, J. Agapiou, J. Schrittwieser, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. v. Hasselt, D. Silver, T. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing., Starcraft II: A new challenge for reinforcement learning. *arXiv preprint* arXiv:1708.04782, August 2017.

[2] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, "A neuroevolution approach to general Atari game playing," IEEE Transactions on Computational Intelligence and AI in Games, vol. 6, no. 4, pp. 355–366, 2014.

[3] Aavaas Gajurel, Sushil J Louis, Daniel J Mendez and Siming Liu. Neuroevolution for RTS micro. arXiv: 1803.10288v1

[4] K. Shao, Y. Zhu, D. Zhao. StarCraft Micromanagement with Reinforcement Learning and Curriculum Transfer Learning. arXiv:1804.00810v1.

[5] K. Shao, Y. Zhu, D. Zhao. StarCraft Micromanagement with Reinforcement Learning and Curriculum Transfer Learning. arXiv:1804.00810v1.

[6] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. Artif_cial Life, 15(2):185{212, 2009.

[7] Sebastian Risi and Kenneth O. Stanley (2012) An Enhanced Hypercube-Based Encoding for Evolving the Placement, Density and Connectivity of Neurons In: Artificial Life journal. Cambridge, MA: MIT Press, 2012 (Manuscript 54 pages)

7 Appendices

7.1 Glossary

Baneling – Melee unit. Suicide bombing attack. Medium health

Banshee – Air to ground unit. Medium speed.

Corruptor – Air to air unit. Slow speed.

Genome – The set of genes that together code for a (neural network) phenotype.

Hellion – Fast ranged unit. Splash damage. Low armor and health.

Heterogenous – One player controls different groups of units at a time.

Homogenous – One player controls the same group of units at a time.

Immortal – Ranged unit. Slow speed. Long attack range. Strong armor.

Kiting – A micromanagement strategy which demonstrates the repetitive behavior of attacking the enemy unit and then fleeing.

Marine – Ranged unit. Medium speed. Medium health

Micromanagement – Low-level control of individual units in the player's army

Melee – A type of unit in SC2 which can attack only if it is near the enemy unit.

Neuro-Evolution – Artificial Intelligence approach using evolutionary algorithms to generate artificial neural network, topology, parameters and rules.

Ranged – A type of unit in SC2 which can attack the enemy units at a distance from themselves.

Roach – Ranged unit. Medium speed. Medium attack range. High health

Stalker – Ranged unit. Medium speed. Long attack range.

Ultralisk – Melee unit. Slow speed. Massive health.

Zealot – Melee unit. Slow speed. Strong attack.

Zergling – Melee unit. Fast speed. Low health.

7.2 Config File

7.2.1 Config 1

```
#--- parameters for the neat adversarial experiment ---#

# 2 -2 weight range

[NEAT]

fitness_criterion     = max

fitness_threshold     = 150000

pop_size              = 150

reset_on_extinction   = False


[DefaultGenome]

# node activation options

activation_default      = clamped

activation_mutate_rate  = 0.0

activation_options      = clamped

# activation_options      = sigmoid gauss relu


# node aggregation options

aggregation_default     = sum

aggregation_mutate_rate = 0.0

aggregation_options     = sum


# node bias options

bias_init_mean          = 0.0

bias_init_stdev         = 1.0

bias_max_value          = 2.0

bias_min_value          = -2.0

bias_mutate_power       = 0.8
```

```
bias_mutate_rate       = 0.4
bias_replace_rate      = 0.02

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 1.0

# connection add/remove rates
conn_add_prob          = 0.15
conn_delete_prob       = 0.1

# connection enable options
enabled_default        = True
enabled_mutate_rate    = 0.01

# NEEDS TESTING
feed_forward           = True
initial_connection     = full
# initial_connection     = partial_nodirect 0.5

# node add/remove rates
node_add_prob          = 0.15
node_delete_prob       = 0.1

# network parameters
num_hidden             = 0
num_inputs             = 12
num_outputs            = 3
```

```
# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 2.0
response_min_value      = -2.0
response_mutate_power   = 0.01
response_mutate_rate    = 0.1
response_replace_rate   = 0.0

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 2
weight_min_value        = -2
weight_mutate_power     = 0.5
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 2.5

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 50
species_elitism      = 4

[DefaultReproduction]
elitism           = 3
survival_threshold = 0.3
```

7.2.2 Config 2

```
#--- parameters for the neat adversarial experiment ---#

[NEAT]

fitness_criterion     = max

fitness_threshold     = 150000

pop_size              = 150

reset_on_extinction   = False


[DefaultGenome]

# node activation options

activation_default      = clamped

activation_mutate_rate  = 0.0

activation_options      = clamped

# activation_options      = sigmoid gauss relu


# node aggregation options

aggregation_default     = sum

aggregation_mutate_rate = 0.0

aggregation_options     = sum


# node bias options

bias_init_mean          = 0.0

bias_init_stdev         = 1.0

bias_max_value          = 2.0

bias_min_value          = -2.0

bias_mutate_power       = 0.8

bias_mutate_rate        = 0.4

bias_replace_rate       = 0.02
```

```
# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 1.0

# connection add/remove rates
conn_add_prob          = 0.04
conn_delete_prob       = 0.025

# connection enable options
enabled_default        = True
enabled_mutate_rate    = 0.01
feed_forward           = True
# initial_connection     = full
initial_connection     = partial_nodirect 0.5

# node add/remove rates
node_add_prob          = 0.02
node_delete_prob       = 0.01

# network parameters
num_hidden             = 0
num_inputs             = 12
num_outputs            = 3

# node response options
response_init_mean     = 1.0
response_init_stdev    = 0.0
response_max_value     = 2.0
response_min_value     = -2.0
```

```
response_mutate_power   = 0.01
response_mutate_rate    = 0.1
response_replace_rate   = 0.0

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 3
weight_min_value        = -3
weight_mutate_power     = 0.5
weight_mutate_rate      = 0.95
weight_replace_rate     = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 2.5

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 50
species_elitism      = 4

[DefaultReproduction]
elitism           = 3
survival_threshold = 0.3
```

7.2.3 Refined Combat Agent Config

```
#--- parameters for the neat adversarial experiment ---#

[NEAT]

fitness_criterion     = max

fitness_threshold     = 150000

pop_size              = 150

reset_on_extinction   = False


[DefaultGenome]

# node activation options

activation_default      = clamped

activation_mutate_rate  = 0.0

activation_options      = clamped

# activation_options      = sigmoid gauss relu


# node aggregation options

aggregation_default     = sum

aggregation_mutate_rate = 0.0

aggregation_options     = sum


# node bias options

bias_init_mean          = 0.0

bias_init_stdev         = 1.0

bias_max_value          = 2.0

bias_min_value          = -2.0

bias_mutate_power       = 0.8

bias_mutate_rate        = 0.4

bias_replace_rate       = 0.02
```

```
# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 1.0

# connection add/remove rates
conn_add_prob          = 0.04
conn_delete_prob       = 0.025

# connection enable options
enabled_default        = True
enabled_mutate_rate    = 0.01
feed_forward           = True
# initial_connection     = full
initial_connection     = partial_nodirect 0.5

# node add/remove rates
node_add_prob          = 0.02
node_delete_prob       = 0.01

# network parameters
num_hidden             = 0
num_inputs             = 18
num_outputs            = 3

# node response options
response_init_mean     = 1.0
response_init_stdev    = 0.0
response_max_value     = 2.0
response_min_value     = -2.0
```

```
response_mutate_power   = 0.01
response_mutate_rate    = 0.1
response_replace_rate   = 0.0

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 3
weight_min_value        = -3
weight_mutate_power     = 0.5
weight_mutate_rate      = 0.95
weight_replace_rate     = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 2.5

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 50
species_elitism      = 4

[DefaultReproduction]
elitism           = 3
survival_threshold = 0.3
```

7.3 Tables

Table 1: A Summary of notable tests executed using NEAT on Move To Beacon mini game

Test ID	Inputs Type	Inputs Definition	Outputs	Fitness Function	Random exploration, Network Type, Config details	Generations	Results
1	Pixel	1024 inputs Pixel values	2 outputs 1. Displacement_x 2. Displacement_y	Fitness function (2) Beacon_weight = 5 Distance_weight = 1	Feedforward network No Random Exploration	300	No learning observed for the majority. Marine just picks one direction and gets stuck at the edge of the map. Few genomes learn the oscillation strategy but to much less effectiveness.
2	Pixel	256 inputs Pixel values	2 outputs 1. Displacement_x 2. Displacement_y	Fitness function (2) Beacon_weight = 10 Distance_weight = 0.5	Recurrent Network	2a) 16 2b) 160	2a) First run terminated at 16 gens with a genome somehow reaching 4 beacons. Winner pickle can't replicate results. It seems some of them understand to approach the first beacon directly. 2b) Some genomes managed to pick up 3 beacons but results cant be replicated and the 3 beacons may have been picked up by chance. However, in training it was observed some genomes

							understood to directly go for for beacon when it first spawned but do no subsequently go for the other ones. No oscillation behaviour observed.
3	Pixel	1024 inputs Pixel values	2 outputs 1. Displacement _x 2. Displacement _y	Fitness function (1)	Feedforward network No Random Exploration	300	No learning observed for the majority. Marine just picks one direction and gets stuck at the edge of the map. Few genomes learn the oscillation strategy but to much less effectiveness.
4	Pixel	4096 inputs Pixel values	2 outputs 1. Displacement _x 2. Displacement _y	Fitness function (1)	Feedforward network No Random Exploration	300	No learning observed. Marine just picks one direction and gets stuck at the edge of the map.
5	Handcrafted	4 Inputs - Scaled [0,1] 1. Player_x 2. Player_y 3. Beacon_x 4. Beacon_y	2 outputs 1. Displacement _x 2. Displacement _y	Fitness function (2) Beacon_weight = 5 Distance_weight = 1	Feedforward network No Random Exploration	500	The majority of "good genomes" that achieved scores of 15+ (meaning 2 beacons and some distance reward) had the same policy - just move left and right on a wide range from the starting position. A valid policy would be to approach the same y as beacon center and then oscillate but none seemed to have learnt that.

							Some odd ones stand around and do nothing while the rare one stops right next to the beacon and does not take it.
6	Handcrafted	4 Inputs - Scaled [0,1] 1. Player_x 2. Player_y 3. Beacon_x 4. Beacon_y	2 outputs 1. Displacement_x 2. Displacement_y	Fitness function (2) Beacon_weight = 5 Distance_weight = 1	Feedforward network No Random Exploration	1500	Population extinct before 1500 generations. 1400 generations done. Most genomes oscillate from starting position. Some stand in place. No genome approaches beacon. No learning observed.
7	Handcrafted	4 Inputs - Scaled [0,1] 1. Player_x 2. Player_y 3. Beacon_x 4. Beacon_y 1 Input - distance to beacon	2 outputs 1. Displacement_x 2. Displacement_y	Fitness function (2) Beacon_weight = 5 Distance_weight = 1	Feedforward network No Random Exploration	1500	Population extinct before 1500 generations. 1400 generations done. Most genomes oscillate from starting position. Some stand in place. No genome approaches beacon. No learning observed.
8	Handcrafted	4 Inputs - Scaled [0,1] 1. Player_x 2. Player_y 3. Beacon_x 4. Beacon_y	2 outputs 1. Displacement_x 2. Displacement_y	Fitness function (2) Beacon_weight = 5 Distance_weight = 1	Recurrent network	1500	The agent learnt to move around in a circle over the map with the circle edge near the boundary of the map. It was observed that the agent decreases the radius of the circle to whenever the beacon is spawned to collect the beam and hence the reward.

Table 2: A Summary of notable initial tests executed using Combat Agent 1 on HellionVZealot map

ID	Inputs Type	Inputs Definition	Outputs	Fitness Function	Random exploration, Network Type, Config details, Map	Generations	Results
1	Handcrafted	Total Self Current hp, Weapon cooldown (majority), enemy in range (avg position) [0,1]	Displacement (x,y), Fight/flee (boolean)	Enemy Initial hp + Current self hp - Current Enemy hp	Feedforward, Partial 0.5 initial connection, 5v5 HellionVZealots	500	No consistent results. Most tend to approach and then get stuck in north west corner where they get some score hitting the zealots as they approach in a group
2	Handcrafted	Total Self Current hp, Weapon cooldown	Displacement (x,y), Fight/flee (boolean)	Enemy Initial hp + Current self hp - Current	Feedforward, Partial 0.5 initial connection, 5 v 5 HellionVZealots	1000	No consistent results. Most tend to approach and then get stuck in north west corner where they get some score hitting the zealots as they approach in a group

		(majority), enemy in range (avg position) [0,1]		Enemy hp			
3	Handcrafted	Total Self Current hp, Weapon cooldown (majority), enemy in range (avg position) [0,1]	Displacement (x,y), Fight/flee (boolean)	Enemy Initial hp + Current self hp - Current Enemy hp	Feedforward, full initial connection, 5v5 HellionVZealots	1000	No consistent results. Most tend to approach and then get stuck in north west corner where they get some score hitting the zealots as they approach in a group

Table 3: A Summary of notable further tests executed using Combat Agent 2 on HellionVZealot map

Series	Inputs Type	Inputs Definition	Outputs	Fitness Function	Random exploration, Network Type, Config details, Map	Generations	Results
1	Handcrafted	Total Self Current hp, Weapon cooldown (majority), enemy in range (avg position), 4 boundary sensors, 4 enemy regional sensors [0,1]	Displacement (x,y), Fight/flee (boolean)	Enemy Initial hp + Current self hp - Current Enemy hp [0,1]	Feedforward, 5v5 HellionVZealots, 7 different configs, notable change is weight min max range, 1 eps per genome	200	None of the config files showed any learning. Wins by chance by going in to the corner of the map and when a group of zealots if formed it splash damages them all
2	Handcraf	Total Self Current hp,	Displacement (x,y),	Enemy Initial hp +	Feedforward, 5v5 HellionVZealots, 7	200	Config file 1 and 3 showed much promise. Both of them showed an

	ted	Weapon cooldown (majority), enemy in range (avg position), 4 boundary sensors, 4 enemy regional sensors [0,1]	Fight/flee (boolean)	Current self hp - Current Enemy hp [0,1]	different configs, notable change is weight min max range, 3 eps per genome		upward learning trend in the average and the best genomes. The agent learnt to show some form of kitting. It hits and then runs away and carries this behaviour out until it wins. Proven and tested against 6 enemies as well. However a change of map changed the behaviour of the agent and it was unable to win which is weird.