Exploring Custom Scenarios and AI in StarCraft II Editor and Learning Environment

Micro Management in StarCraft II

Cai Yang u6625166

A report submitted for the course COMP3740
Supervised by: Dr. Penny Kyburz, Dr. Sabrina Caldwell
The Australian National University

November 2020

Except where otherwise indicated, this report is my own original work.

Cai Yang u6625166
6 November 2020

Acknowledgments

Many thanks to my supervisors.

Abstract

Research on RTS games has been popular for many years due to their complexity and popularity. As one of the most successful RTS games, StarCraft II®[1] has attracted attention from AI community given its game mechanism and many interesting sub-problems arise from it. A scenario in StarCraft II refers to a specific situation in the actual game and it can provide good practice for players to master some essential skills such as commands of units. In this project, we focus on the scenarios for combats in StarCraft II. Moreover, we want to find out the possible scenarios for modelling combats in the real game and the methods we can use to help ally units to win the combats.

To achieve the goal, we create and explore custom maps using StarCraft II Map Editor. These maps vary with different types of units along with different terrains, with the purpose of covering as many scenarios as possible. Meanwhile, we also explore different algorithms to help ally units to win the combat in the scenarios. To obtain programmatic control over the maps, we provide a learning environments SCMM (StarCraft II Micro Management). We first show some commonly used scripted behaviours in details, which are the simplest but still efficient and effective approaches to manage ally units. Second we present genetic algorithm, which is a bio-inspired algorithm and has been widely used in optimization. They can be adapted to solve problems in games given that no mathematical formulation is required. Then we present a linear-system based potential field, which is inspired by robotics planning. In the end we will talk about gaming tree search, which has been used a lot in games but flawed for this project. The motivation behind the choices of algorithms is that they are usually easy to implement and faster to deploy.

It is worth mentioning at the beginning that the purpose of this project is not to invent state-of-art algorithms but rather explore different scenarios along with different solutions to them. We hope our work, especially the agents, can provide valuable intuitions and can be used as baselines for future research. We also released the source code, along with all the scenario maps and environment[2] [3].

[1] © 2020 Blizzard Entertainment, Inc. All rights reserved.
[2] https://github.com/caiyangcy/SCMM
[3] https://pypi.org/project/SC2MM/

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Real time strategy (RTS) game, whose root can be traced back at 1980s, is a sub-genre of multiplayer game. In such games, players usually have to gather resources to make sure economic growth, construct buildings and build an army to conquer enemies. StarCraft II, initially released at 2010 by Blizzard Entertainment, has become one of the most popular RTS games. Fig 1.1 shows a screenshot of actual game play. In the past years, StarCraft and StarCraft II have become rather popular in AI game community for many reasons [1]: 1. It provides measurements of gaming results. 2. It has high complexity, popularity and well established environment. 3. It can be decomposed into multiple sub-problems, which can be studies independently from each other. 4. The game is supported by multiple platforms, which makes it easier to share researching results with others [1].

1.1 Motivations and Project Scope

Many works have been done on modelling the full game of StarCraft II [2; 3; 4]. As a contrast, this project focuses only on micro management in StarCraft II. The motivation behind this is that excellent low-level control over individual units can maximise the amount of damage dealt to enemies while can help to minimise the damage that ally units have received, which can further help us to win the game as eliminating enemies is an unavoidable step in the game. Micro management can make it possible for weak armies to defeat much stronger armies and it is an important skill can differentiate good players from top professional players.

To obtain better control of the game, we introduce SCMM (StarCraft II Micro Management) environment, which is built on top of SC2LE [1] and SMAC[5]. The environment provides basic information about the game such as positions of units and their abilities. Meanwhile, it also provides a set of agents that can be used as baselines to win the combat in each scenario.

Last, It is worth mentioning that the scope of the project is mainly focused on scenarios creation, exploration and non-learning algorithms, where few learning algorithms are involved and none of the reinforcement learning algorithms will be discussed

Figure 1.1: Overview of SC II

here. The reason is because we are looking for methods with low cost so that they are faster to apply, which is usually not the case for reinforcement learning given that they need large amount of time and data to train.

1.2 Report Outline

The structure of the report is as follows: In chapter 2, we talk about the background and related work on StarCraft II that have been performed in recent years and state our contributions. Chapter 3 shows the design of different scenarios and our environment SCMM along with details of agents. Chapter 4 covers the methodology including experiments and data collection. Chapter 5 presents the results obtained from different algorithms and conclusions and future work are proposed in chapter 6.

Chapter 2

Background and Related Work

Management refers to the interactions and commands between players and gaming objects. There are two types of managements in StarCraft II: macro management and micro management. Macro management refers to the high level control of groups of the gaming objects. Some common tasks are decision making of tactical strategy, units training, building order optimisation, evaluation of global game states, etc. As a contrast, micro management usually refers to low level control of each individual unit in game such as the control of units to and attack enemies or retreat, positioning units.

2.1 Related work

In this section, some related work in terms of algorithms and environment used for mico management in the past research are presented. Some of the related work is highly related to our work and has motivated some ideas of ours.

2.1.1 Algorithms

Significant amount of work has been done over the past years in the field of micro management in order to design effective AI. In [6; 7] the authors talked about scripted behaviours of agents which in general are commonly used in StarCraft II Ladder competitions.

Work of [8; 9; 10] demonstrates some heuristic based searching algorithms such as Alpha-Beta Considering Durations (ABCD), which is a modification of traditional Alpha-Beta algorithm taking into considerations of simultaneous and durative moves in games, UCT Considering Durations (UCT-CD), which is a modification of the UCT Monte Carlo Tree Search and Portfolio Greedy Search (PGS), which is a new hill climbing algorithm.

Potential field, which has been used a lot in path planning and obstacle avoidance, has been applied for micro management. Some recent work has combined potential field and influence map to represent information on units and terrain [11]. In this

case, the influence map is essentially a gird where each cell is assigned a value calculated by influence function and later used by AI to determine actions.

NEAT, stands for Neuro Evolution Augmenting Topology, is original proposed by [12] for evolving neural network topologies along with weights using genetic algorithms. NEAT along with its real time version have been used for micro control of game units in [13].

Bayesian Networks have also been used to imitate players' decision making and mental status in StarCraft II. A Bayesian Network in [14] was fitted on information gathered from players' combat micro-management and the results have shown increased performance compared with built-in AI.

2.1.2 Environment

Besides algorithms, multiple environments have also been released in the past years. *SparCraft* was released by [8], which is a simplified simulation package of StarCraft. The author also released *UAlbertaBot* based on it, which implements the heuristics mentioned in [8]. In [15] the authors presented a Markov environment for cooperative tasks but the environment is never published. The gridworld environment made by [16; 17] mainly focuses on many-agent tasks and supports tasks and applications that require hundreds to millions of agents. This environment is also highly scalable and can hold up to one million agents. A set of grid-like environments was released by [18] which focus on low-level continuous control on competitive and cooperative tasks.

SC2LE, implemented by DeepMind and Blizzard [1], is a learning environment specially designed for StarCraft II. It targets at full games and provides programmatic control of the game along with different feature layers of the game. Built on top of SC2LE, [5] released SMAC (StarCraft II Multi-Agent Challenge) recently, which targets at multi-agent reinforcement learning for micro management in StarCraft II.

2.2 Contributions

As there is a lot of previous work exsiting before our work, it is important to state the contributions and differences from previous work:

1. None of previous work implements scripted behaviours in such a thorough way like our work. Meanwhile, Some of previous work is either not open-sourced or outdated, which makes the future work difficult to carry on since there is no reference.
2. The algorithms used in our work are user-friendly and efficient to deploy. Although some of them requires training, they are fast to be trained due to their size.
3. Although our environment does not support reinforcement learning, it can be easily adapted to provide states and observation information and our Python-based

work also makes it easy to use modern popular deep learning frameworks.
4. We provide a set of maps for different scenarios and an unit tester map, which are helpful for both designing and modelling possible situations in actual game and they can be reused for future research.

Chapter 3

Design and Implementation

In this chapter, We introduce our new environment: **SCMM** (StarCraft II Micro Management) along with its features. Meanwhile, we will describe the considerations during scenario creation and agent design.

3.1 SCMM

As a modification and extension from SMAC [5], we release an variant: **SCMM**, which is an environment only focused on micro management and provides more control over units.

SMAC was originally developed for multi-agent reinforcement learning, which requires observations and states of the each unit. However, all the related functions have been removed in SCMM. Action space has been enlarged to give more control over units. Unlike SMAC, where only each unit can only move, stop, attack, SCMM allows units to use their special ability to win the game, which is closer to the situation in real games.

Meanwhile, SCMM also provides a visualization functionality, which allows the users to visualize the decision making process. What is different from SC2LE is that we suppress redundant features so that users can pay attention to actions and units themselves.

3.1.1 Actions

The action space in SCMM is a discrete set of actions, which consists of *move*, *attack*, *heal*, *no-op* and special abilities such as *force field*, *build auto-turret*. Macro moves such as *attack-move*, *patrol* are disabled since such actions can be achieved by using combinations of basic actions. There are actions that will not be used in the environment but still provided for referencing purpose.

3.1.2 Reward

Given the purpose of each scenario is to win the combat, it is necessary to define measurements of rewards. One way of doing this is to use the default sparse outcome provided by StarCraft II, where +1, 0, -1 are used for indicating victory, draw and defeat. However, sparse rewards are not useful for analysing the behaviours of units, given that narrow victory and superior victory will lead to the same result.

To resolve such a problem, a continuous reward is calculated based on the sum of the ratio of the HP of remaining ally units with the amount of damage dealt to enemy units excluding shields. A reward of 0 means the ally units do not manage to hurt the enemy units at all. We also provide a scaling factor, which is simply the sum of HP of all units at the beginning, that can be used to scale the reward into [0, 1].

3.2 Scenarios

Similar to SMAC, SCMM also provides a set of scenarios that are likely to show up in the actual games, which are also used for testing different algorithms. There are two groups of unis in all the scenarios, where the first group is controlled by algorithms, also known as ally units while the other group is controlled by StarCraft II built-in AI, also known as enemy units.

One important factor that must be taken into account while designing maps for scenarios is the balance of the skirmish. In general imbalanced scenarios are not helpful for modelling the actual game and studying the algorithms. For example, if there are 5 ally Marines and 10 enemy Marines in the map, it is highly unlikely for those Marines to win the combat in major cases, which is due to the imbalanced amount of units. Similarly, if there are 10 ally Marines and 5 enemy Marines, we can easily win the combat given the advantage in amount, which has nothing to do with the algorithms. Some important factors that have to be considered when designing maps for scenarios include but not limit to: health point, damage, shield, enegy and shooting range.

A complete list of scenarios is listed in Appendices. Otherwise mentioned, all the maps are of tiny size (32×32). A map is said to be homogeneous if it contains only one type of units and heterogeneous if it contains more than one type. A map is symmetric if it contains the same ally units and enemy units or asymmetric if there are different units on both sides in terms of either amount or types. The design of scenarios has been focused on heterogeneous and asymmetric cases since there are more common than other types in the actual game. Meanwhile, the built-in *fog-of-war* has been disabled by default to make scenarios only pay attention to combats.

In the end, it is also worth mentioning the ally units and enemy units can be swapped in some scenarios, which provides more possibilities for combats. For example, we

can control Marines to defeat Zealots but the scenario with the control of two units swapped is also of interest.

3.3 Agents

What comes with SCMM is a set of modularized agents. Different from SMAC, where agents are merged into environment, agents in this project are separated from SCMM. These agents are designed in a way such that they can be fit into any environment with similar functionality as SCMM, which makes it possible to use on multiple platforms. In the following subsections, the details of the agents are presented.

3.3.1 Scripted Behaviours

Scripted behaviours are some predefined rules, which have been used for bots in StarCraft II ladder competitions. These behaviours are usually implemented as baselines given that they are usually specially designed for each scenario. Some common used scripted behaviours are listed below and examples are also given in each subsection for helping visualizing and understanding the behaviours.

Attack/Heal Closest

Each unit attacks the enemy that is closest to it.

$$Target(A) = \underset{E}{\operatorname{argmin}}\{Distance(A, E)_E\}$$

It does not make any sense for units with healing ability to attack. For such units like Medivacs, they will be scripted to heal the closest unit.

As shown in 3.1, the blue Marines will attack enemies based on the distance, whose attacking directions have been indicated by arrows.

Attack/Heal Weakest

Each unit attacks the enemy that has lowest health.

$$Target(A) = \underset{E}{\operatorname{argmin}}\{HP_E\}$$

As shown in figure 3.2, the blue Marines will attack enemies based on the health point, whose attacking directions have been indicated by arrows. Marine without upgrading have health points 45 while Maurauders have health point of 125. Hence the red marine will the attacking target in this scenario.

Similar to Attack Closest, units with healing ability will be scripted to heal weakest units in ally, which makes more sense than healing closest ones.

Figure 3.1: Attack Closest

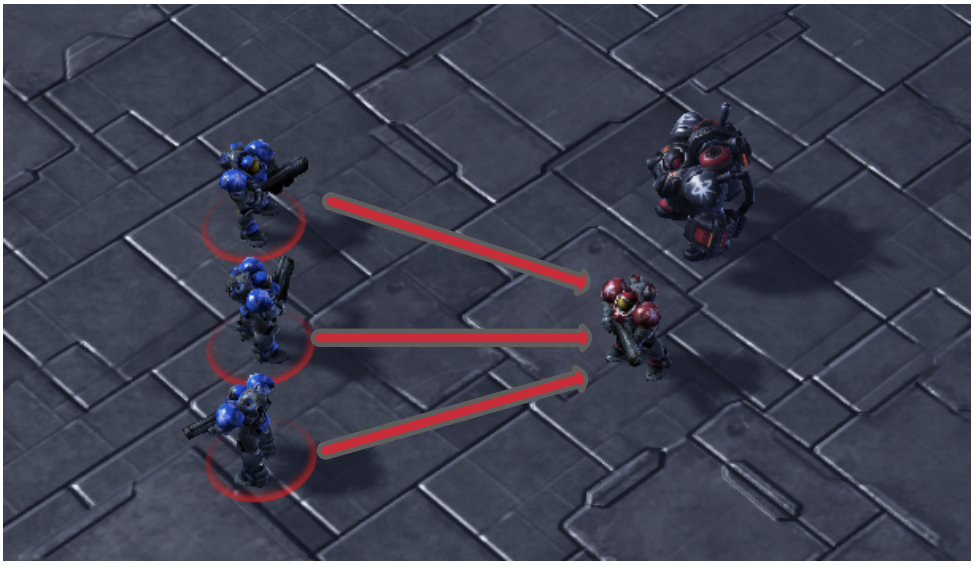


Figure 3.2: Attack Weakest.

Hybrid Attack/Heal

A function is used to calculate scores based on each enemy's health and distance, where some hyperparameters are used as coefficients. More formally, it can be written as

$$Target(A) = \underset{E}{\operatorname{argmin}}\{\alpha \times HP_E + (1 - \alpha) \times Distance(A, E)\}$$

Using scores rather than health and distance directly provides a way for combining information about distance and health, which may be more useful in terms of deciding which enemy to attack (heal). This method is at least as general as previous two methods with the same behaviours as *Attack Closest* when $\alpha = 0$ and *Attack Weakest* when $\alpha = 1$.

Focus Fire

Compared with the case where each unit only focuses on attacking the enemy that is closest or weakest to it, all ally units concentrate on attacking some units can be more effective in terms of elimination since enemy units receive more damage and get killed much faster.

One important issue that has to be considered when focusing fire on enemies is *no overkill*, which means ally units never do more damage to an enemy than what is required to kill it since extra damage does not contribute anything to the reward. To avoid this problem, in each game step, the accumulated damage on each enemy unit in next step will checked to see if it is larger than its current health point. If the enemy has been confirmed to be dead in the next game step, the ally units which have not perform attacking yet will focus their fire to other enemy units.

Figure 3.3 shows a scenario where there are 11 Marines and 3 Drones. Each Marine can do 6-point damage and each Drone has 40 HP in total. If all the Marines focus on attacking one Drone, it will take 3 game steps to eliminate all Drones. However, if 7 of them focus on attacking one of the Drones and the rest 4 focus on attacking the other one, they can eliminate Drones in 2 game steps. This scenario is unlikely to happen in the actual game given its simplicity but the purpose is to show how focus fire with no over kill can improve efficiency.

Dying Retreat

Protecting ally units from being eliminated is as important as killing enemy units. A professional player in general will always try to keep all units alive during a combat. In each game step, each ally unit is checked based its current health and damage accumulated in the past game time. If it is unlikely for the unit to survive at the end of next step with its remaining HP, then the unit is marked as dying and will retreat towards the opposite direction of the enemies. To prevent units from retreating over and over again, each of them is assigned a threshold for maximum number of

Figure 3.3: Focus Fire. 7 Marines should focus on one Drone while the others focus on another.

retreating. Figure 3.4 shows a scenario where the selected blue Marine will move left given it is in dying status.

Block Enemy

Block enemy, also known as choke play, refers to the scenario where ally units make use of force fields to block enemies, which happens a lot where there are ramps and narrow-passing corridors. Figure 3.5 shows the two scenarios. There are two types of choke playing: offensive one and defensive one.

Defensive choke playing refers to the scenario where force field is used to block attacking enemies and can be helpful where there are not enough ally units. The most common strategy is units with force field ability (e.g. Sentry) split enemies into half and attack on enemies trapped.

On the other hand, **Offensive** choke playing refers to the scenario when ally units are attacking and force fields can be used to block reinforcements from enemies.

Attract Fire

Although it is important for players to keep their units alive but sometimes using units to attract fire from enemies can be a better strategy since it can provide enough time for the rest of ally to eliminate enemies. There are two different ways for attracting fire:

HP as attractor: High HP units such as Thors, Siege Tanks are used as attractors for enemies given it usually takes longer to kill them. In the meantime, other units start to attack enemies given they will not receive any damage. This method works well

Figure 3.4: Dying Retreat. The marine with lowest HP will move along the arrow direction.

when there is a small to medium amount of enemies.

Amount as attractor: Instead of using high HP units, units with low HP can also used as attractors. A large amount of such units during combat can make enemy units distracted easily while placing high-attack units in the back can help to eliminate enemies quickly. This method usually works well when dealing with cases where there are high-attack units on enemy's side.

Kiting

Kiting, as the name suggests, is similar to fly a kite. Ally units trigger enemy units to give a chase while maintain a distance between them. This is extremely helpful in a scenario where ally units have longer shooting range and faster moving speed than enemies. One example of such scenario is displayed in figure 3.6. Stalkers move really fast and have a large shooting range while Zealots are melee units which means they only attack units next to them. Stalkers can shoot at Zealots when there is a relatively large distance to Zealots (3.6(b)) and run away when Zealots get close (3.6(a)).

Positioning

Some professional players may refer positioning and choke playing as the same trick but here we would like to separate them. The idea of positioning is similar to choke playing but it usually refers to a more general way of making use of terrain to defeat enemies.
An example is displayed in figure 3.7. Initially, there are 6 Zealots and 24 Zerglings, as shown in the top figure. Once the combat starts, Zerglings will attack from all

(a) Force field on corridor to chock Zerglings

(b) Force field on ramp to choke Zerglings

Figure 3.5: Choke playing in different terrain.

(a) Running Away in Kiting

(b) Attacking in Kiting

Figure 3.6: Kiting

directions, which make it hard to defence, as shown in the middle one. The best way to do in this scenario is move Zealots to the other end of the corridor immediately at the beginning where Zerglings can only attack from one direction, as shown in the bottom one.

Another positioning play is shown in figure 3.8. Colossi can easily move on/off the cliff while Zerglings can only follow the ramps. To eliminate Zerglings, Colossi have to keep moving on and off the cliff to keep Zerglings moving around but unable to attack them.

Alternating Fire

Alternating fire refers to units attack at enemy in an alternative manner. This makes sure that enemies are being attacked all the time and hence receive continuous damage. This method works well when dealing with melee units since melee units always move towards the unit that is currently attacking it and shooting at it in an alternative way could make it walk between units but unable to attack any of them.

Figure 3.9 shows a scenario where a Zealot is trying to attack on two Marines. By shooting at Zealot alternatively, two Marines can stay alive and kill Zealot in the end. It is also worth mentioning kiting on Zealots could also work in this case.

As mentioned earlier in Section 3.2, we can swap the control of Zealot and Marines in this case. To make Zealot kill both Marines, it is important for it to focus on attacking one of them without being distracted by the other's attack. When being kited by Marines, Zealot could use its special ability(Charge) given it has sufficient energy and all the prerequisites are satisfied. This ability can increase Zealot's speed temporarily and allows it to intercept any enemies nearby, which can be helpful for avoiding being kited.

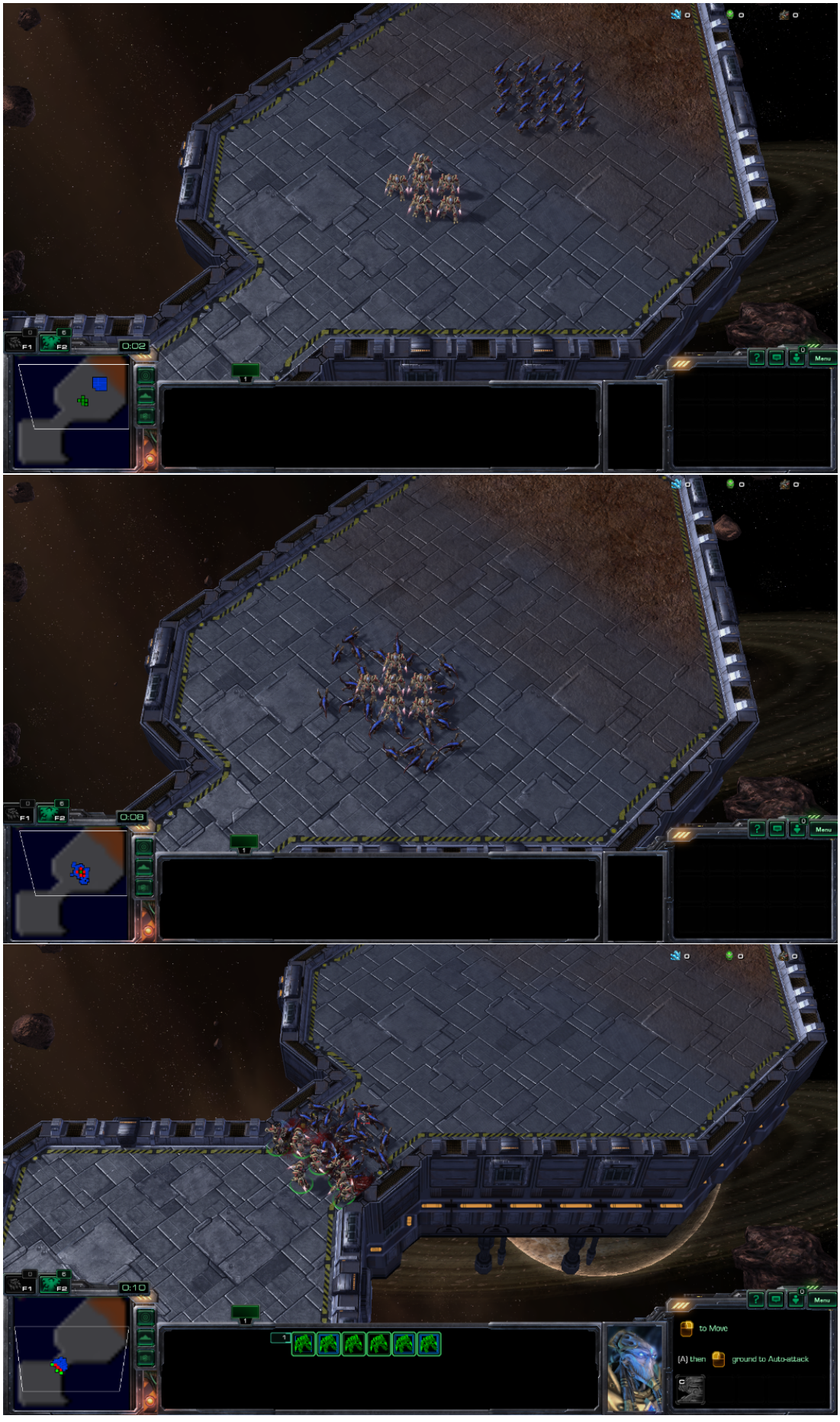


Figure 3.7: Corridor Positioning

(a) Wall-Off: Colossi on the wall

(b) Wall-Off: Colossi off the wall

Figure 3.8: Wall-Off

Figure 3.9: Alternating Fire

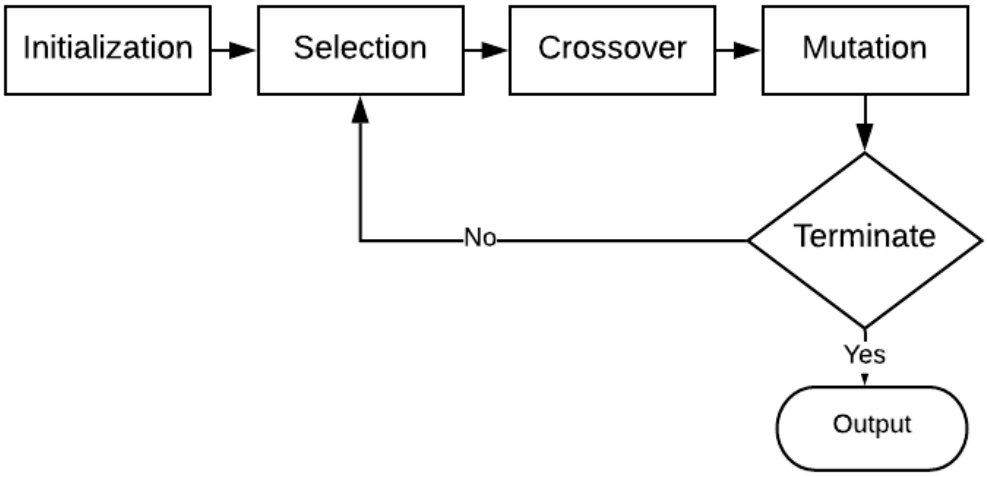


Figure 3.10: Genetic Algorithm Process

3.3.2 Genetic Algorithms

Genetic Algorithm is a meta-heuristic that has been commonly used in areas such as optimization and operations research. It borrows the idea from biological behaviours such as selection, crossover and mutation. It is able to produce some high-quality solutions. In genetic algorithm, bit string representation is adopted, which make it easy and fast to produce results. The general process for genetic algorithm is displayed in figure 3.10, where the evaluation of fitness values has been incorporated into selection part.

The algorithm starts with an randomly initialized population and then evaluate the fitness of each chromosome, where fitness is defined the rewards of each round of game. Probability of a chromosome is selected is proportional to its corresponding fitness value. Crossover operators are performed on the parents selected to generate offspring, during which mutation might happen with small probability to increase the diversity of the population. Termination conditions are usually some predefined threshold on fitness value or number of generations. The algorithm terminates once the condition is satisfied.

In this project, our attention is focused on using genetic algorithms for actions selection. In order to make sure each chromosome carrys enough information about an unit and its surrounding environment, each chromosome is designed as what shows in figure 3.11, inspired by [19] :

In each chromosome, *Segment 1* encodes a threshold for the difference between shooting range of a unit and the distance to it closest enemy. If the actual difference is

Figure 3.11: Chromosome Encoding

smaller than the threshold then the corresponding unit will move towards its closest enemy. *Segment 2* encodes a threshold for sum of all enemies within the unit's shooting range. If the actual number of enemies within the unit's shooting range is higher than the threshold, then the unit will attack its closest enemy. *Segment 3* encodes a threshold for the ratio of weakest enemy's HP divied by the unit's damage. If the actual value is lower than the threshold then the unit will attack the weakest enemy within its shooting range. *Segment 4* encodes a threshold for the difference between number of enemies and number of allies, which is a global information. *Segment 5* encodes the priority of the previous three actions. If multiple actions need to be performed, then the action will be decided based on its priority.

3.3.3 Neural Network with Evolving Weights

Training neural networks using gradient descent requires loss, which is dependent on the actual labels. However, due to sparsity of games, labels will not be available at the end of each episode. One way of solving this problem is to use NEAT, which evolves the weights and topology of neural network. However, in RTS games, one main weakness for NEAT is the evolution process of neural network may takes long time to perform well on different tasks. To alleviate such problem, we designed an *lazy* evolution on weights of neural network, which can be regarded as a compromise between NEAT and genetic algorithms.

Initially, a predefined number of neural networks with random weights are used to predict actions of units. Then in each iteration, the weights of neural network with the best performance from previous iteration will be copies into a predefined number of neural networks along with a multiplication factor that increases or decreases weights. We believe once a neural network manage to find a good combination of actions, the following generations will be able to improve it.

Similar to previous genetic algorithms, the input of neural network has to carry local and global information of a unit. We extend the idea from [11], which computes a 40-element input based on 8 regions surrounding a unit, as shown in figure 3.12.

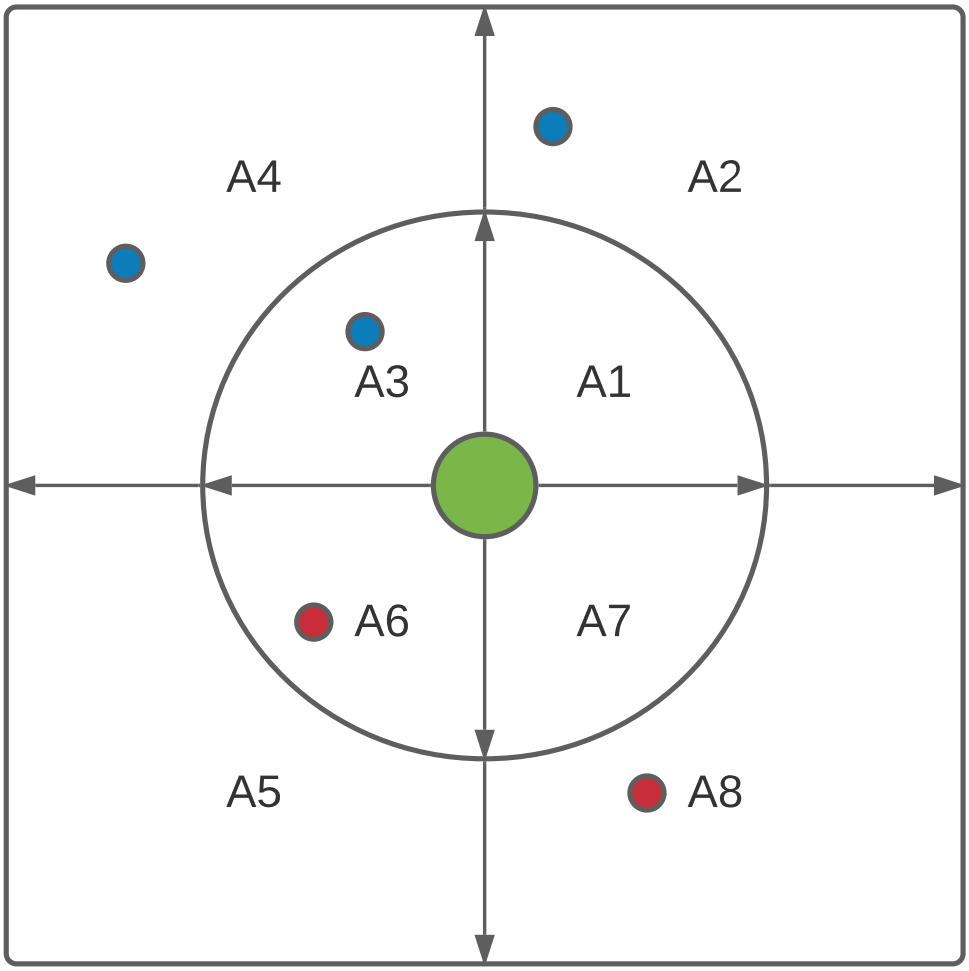


Figure 3.12: Neural Network Input

The details of each element of the input is as follows:

Positions	Values
1 - 8	Mean position of enemies in each region
9 - 16	Mean position of allies in each region
17-24	Number of enemies in each region
25 - 32	Number of allies in each region
33 - 36	Map boundaries
37	Weapon Cooling down
38	Unit damage
39 - 40	Previous two moves

Table 3.1: Input to NN

The dimension of the output is set to be 3. The first value of the output controls the movement on East and West and the second value controls the movement on North and South while the last value determines whether the unit should attack or not. Action selected by neural network is based on its output values and predefined threshold. Sigmoid avtivation function is used in the end to map the output into [0, 1] and the threshold here is defined to be 0.5. If the last value from the output is higher than 0.5, then the unit will attack its closest unit. Otherwise, the action will be moving one of the four directions depending out their value compared to 0.5. It is worth mentioning that the attacking method here can be selected from other possible methods such attck weakest or hybrid attack, but attack closest is chosen for simplicity.

3.3.4 Potential Fields

We adopt a linear system for calculating potential fields. There are three different types of potential: attractive potential from ally units, attractive potential and repulsive potential from enemy units. The formulas for calculating the three potentials are displayed below:

$$P_e^{attractive} = \begin{cases} 0, & \text{if } dist \leq \text{D} \\ \alpha \cdot dist & \text{Otherwise} \end{cases}$$

$$P_e^{repulsive} = \begin{cases} \frac{\alpha}{hp_a} + \beta dps_e + \gamma \#Enemies + dist, & \text{if } dist \geq \text{D} \\ 0, & \text{Otherwise} \end{cases}$$

$$P_a^{attractive} = \begin{cases} \frac{\alpha}{hp_a} + \beta dps_e + \gamma dist, & \text{if } dist \geq \text{D} \\ 0, & \text{Otherwise} \end{cases}$$

where D is a predefined threshold on distance and α, β, γ are hyperparameters (They

do not have to be the same in different potential). The final potential is calculated as the sum of the three potentials above. For each unit, when there is no enemy within its shooting range, then it will move along the direction pointed by the final potential. Otherwise it will attack the closest enemy.

3.3.5 Game Tree Searching

From mathematical perspective, game trees is a directed graph where each node stores information about each state of the game and edges represent moves. Moreover, the decision making process generated by game tree searching algorithms can be viewed as a decision tree, where each layer corresponds to a round of a specific player. One common algorithm for tree search is *MinMax*, where player takes a move that maximizes the minimum value returned by the opponent's possible moves.

However, it is difficult to perform a game tree search on StarCraft II and other RTS games and it will not be tested in this project due to the following reason:
1. It does not make sense to undo an action in a real time space, which is contrary to turn-based strategy games.
2. Each action cannot guarantee the same results due to non-determinism of the game.
3. The low-level protocol does not allow modify gaming information manually, which is different from [8], where a different environment and protocol is used.

Chapter 4

Experimental Methodology

In this chapter, we discuss the setup and conduct of the experiments.

4.1 Algorithm Setup

As mentioned in section 3.1.2, a reward is associated with each round of the game and we make use of it to measure each game. The higher the reward is, the more likely the algorithm is able to work well on that specific scenario. For each scenario, each algorithm is run for 10 episodes and the final value of the reward of each game is collected.

4.1.1 Scripted Behaviours

Attack Closest, Attack Weakest, Focus Fire and Dying Retreat will be run over all the scenarios since they are the most fundamental agents over all scenarios. The remaining agents will be run some their own specific scenarios as a comparison.

4.1.2 Genetic Algorithms

To avoid extreme values for each segment, each segment is restricted into a predefined range. *Segment 1* is limited to [10.25, 11.75]. *Segment 2* is limited to [0, 5]. *Segment 3* is limited to [0, 6]. *Segment 4* is limited to [-1, 20]. These values have been tested to show relative good results and stability among different scenarios but the readers are encouraged to test different range of values.

What is different from scripted behaviours is that, the evaluation of genetic algorithms takes longer to finish. Hence, it is run on some representative homogeneous maps and some of the heterogeneous maps. Moreover, in order to show the algorithm is indeed evolving, we collect the maximum, minimum and mean fitness values in the population of each episode.

4.1.3 Neural Network with Evolving Weights

A simple 4-layer feed forward neural network is used here and we choose the number of neural networks to be 3 in this case. In fact, we believe any number would be reasonable as long as the *lazy* evolution process can finish quickly. The random weight adjustment factor is drawn from a uniform distribution between 0.99 and 1.01 each time. Similar to genetic algorithm, to show the overall performance of neural network is increasing, the maximum, minimum and mean rewards is collected in each episode.

4.1.4 Potential Fields

There is no general guide for selecting the hyperparameters used in potential field. Compared with grid search, a set sample parameters is left for demonstrating purpose. The experiments will not be run using potential field given the demonstrating parameters are not expected to work well. The algorithm is left there as a prototype for reference purpose.

4.2 Hardware and Software Information

All the experiments are run under Python 3.6.10 along with Pytorch 1.4.0 and Numpy 1.19.3. StarCraft II is of version 5.0.3 at the moment the experiments were conducted. The experiments were conducted under a device with Intel(R) Core(TM) i7-7700HQ CPU (2.8 GHz) along with 32 GB RAM.

Chapter 5

Results

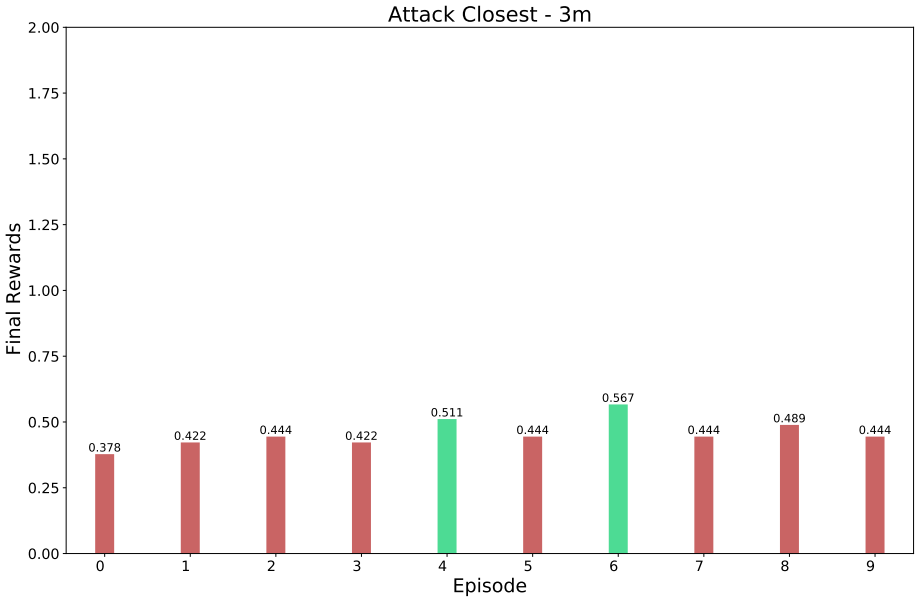
In this chapter, we present and analyse some representative figures of collected rewards/fitness values from previous experiments. Then we will make discussions based the results and show some possible future work in the end.

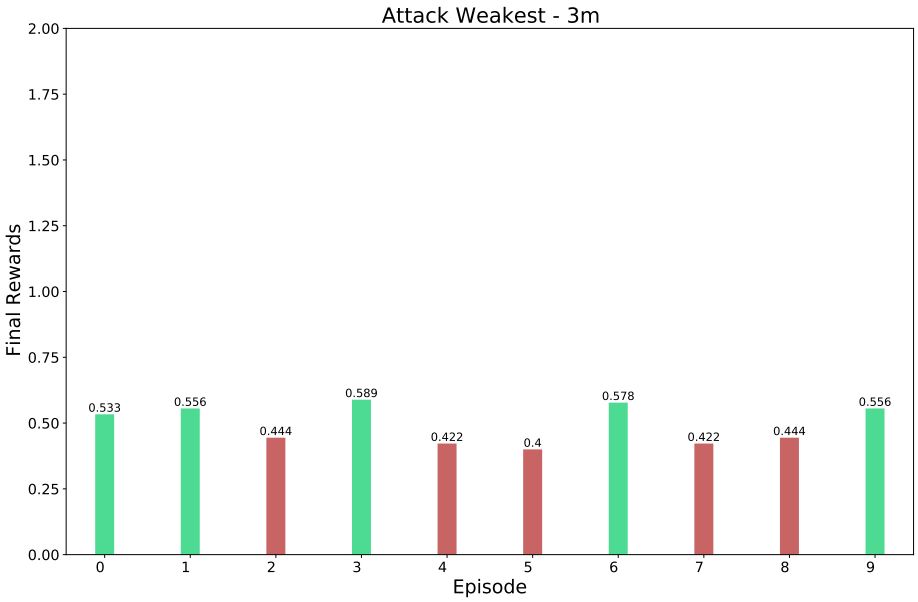
5.1 Results Analysis

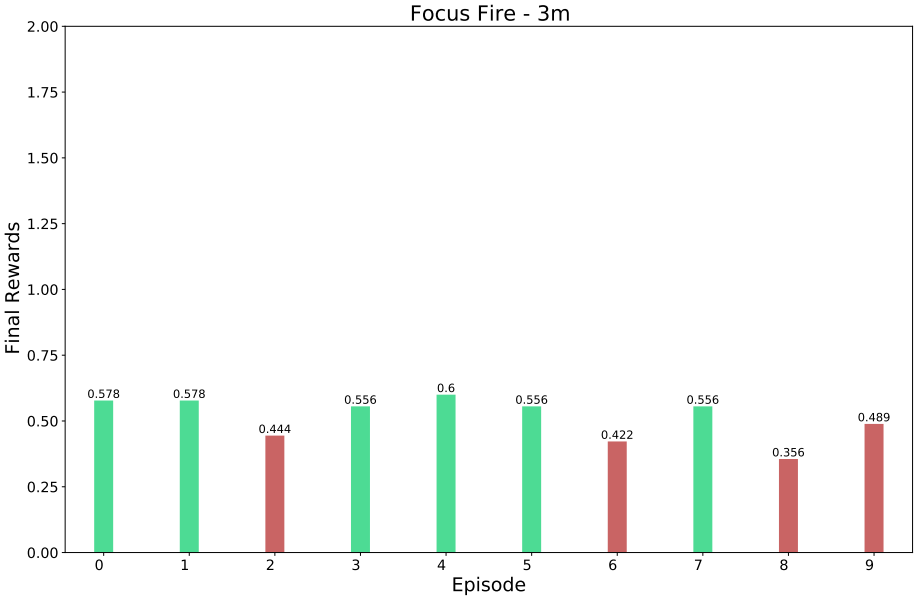
The rewards collected from scripted behaviours are plotted using bar chart given each episode is not related to each other. The fitness values and rewards collected from genetic algorithm and neural network is plotting using lines to show the change of them. Due to space limit, readers can find a detailed set of plots of rewards at the code repo[1] and only some of the representative results are shown below. The horizontal axis is the number of episodes while the vertical axis represents the rewards/fitness values. The title of each figure consists of the abbreviation of the agent and scenario. For all the plots except last one, green color indicates victory of a combat while red color indicats defeat

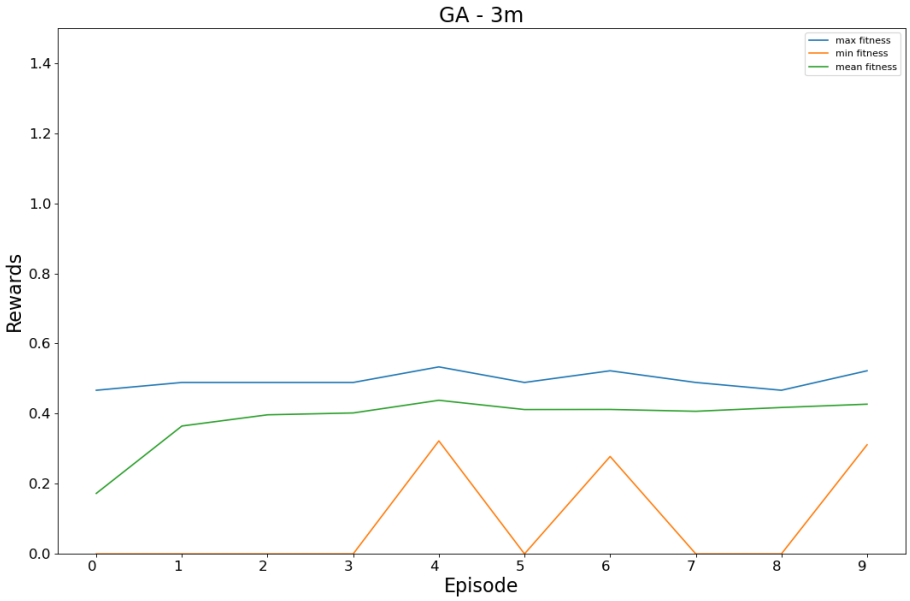
The following are some results on homogeneous scenarios: 3m, 8m, 25m, where there are 3, 8 and 25 Marines respectively.

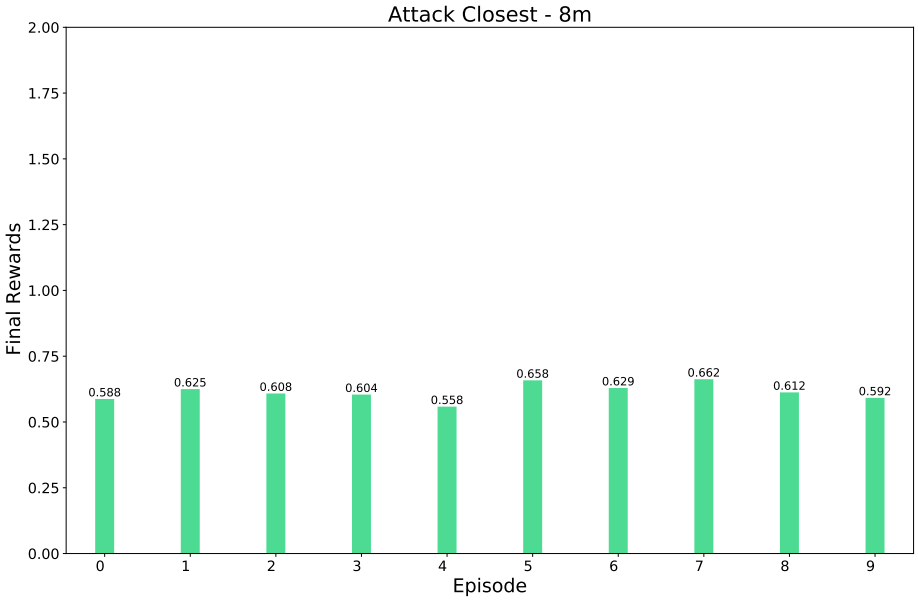
[1] https://github.com/caiyangcy/SCMM/tree/master/plots

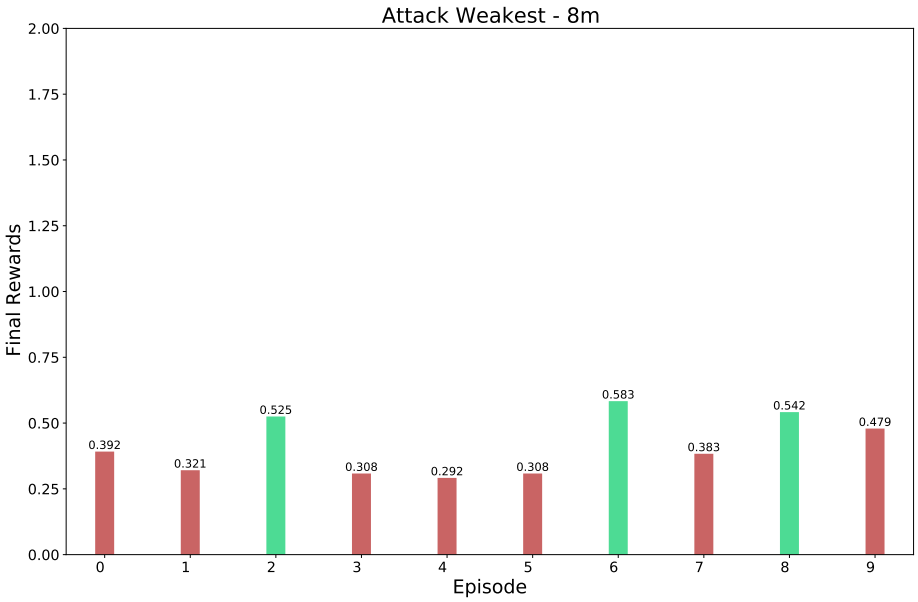

Attack Closest - 3m
Final Rewards
Episode
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0 1 2 3 4 5 6 7 8 9
0.378
0.422
0.444
0.422
0.511
0.444
0.567
0.444
0.489
0.444

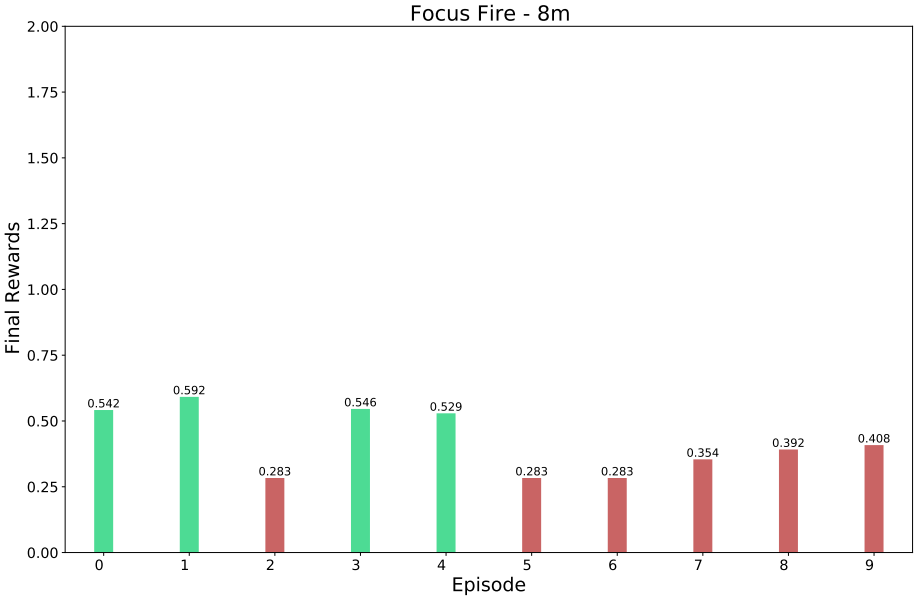

Attack Weakest - 3m
Final Rewards
Episode
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0 1 2 3 4 5 6 7 8 9
0.533
0.556
0.444
0.589
0.422
0.4
0.578
0.422
0.444
0.556

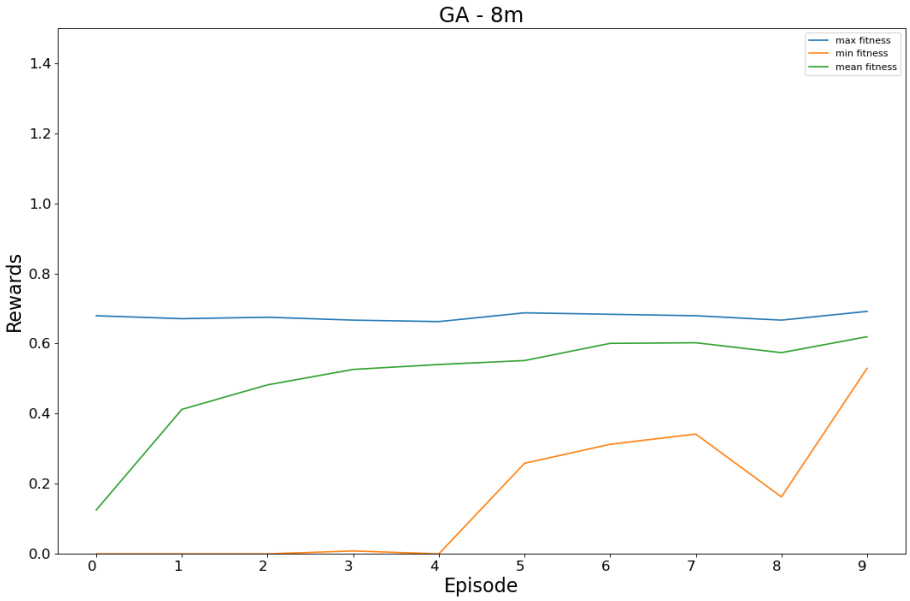

Focus Fire - 3m
Final Rewards
Episode
0.578
0.578
0.444
0.556
0.6
0.556
0.422
0.556
0.356
0.489

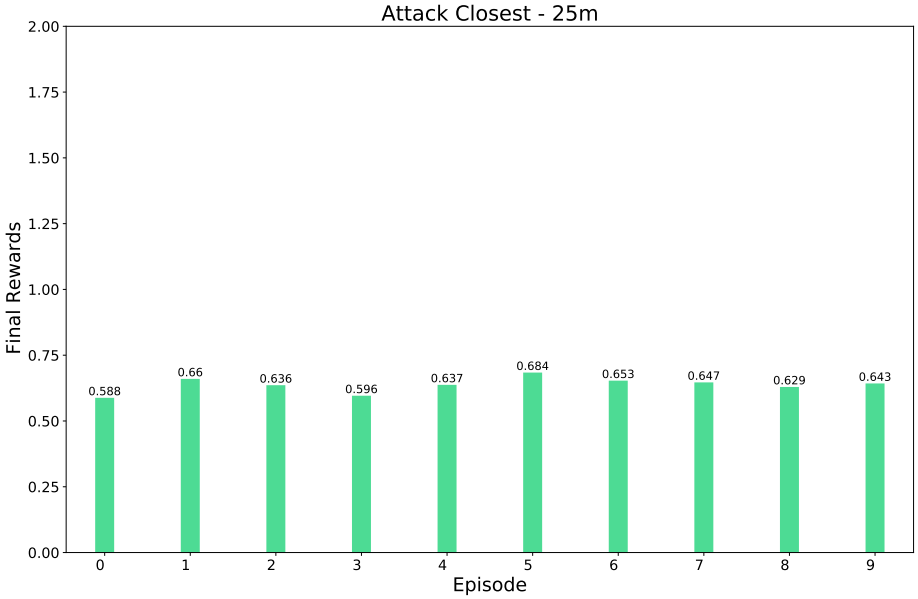

GA - 3m
Rewards
Episode
max fitness
min fitness
mean fitness

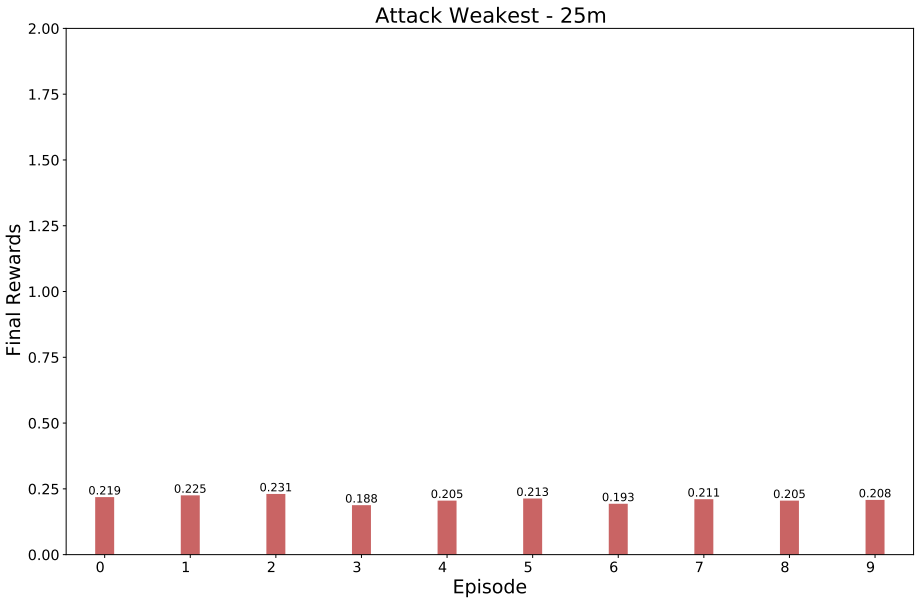

Attack Closest - 8m
Final Rewards
Episode
0.588
0.625
0.608
0.604
0.558
0.658
0.629
0.662
0.612
0.592

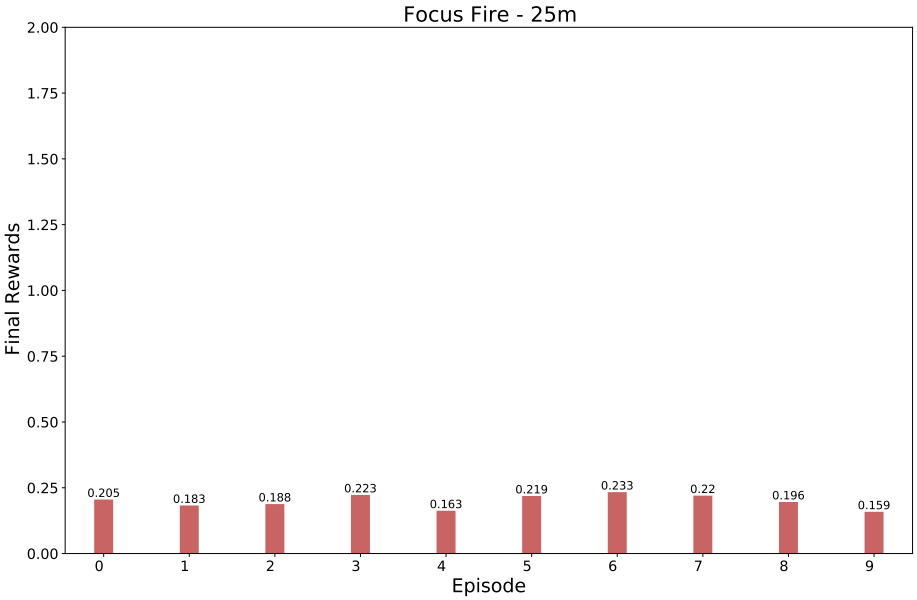

Attack Weakest - 8m
Final Rewards
Episode
0.392
0.321
0.525
0.308
0.292
0.308
0.583
0.383
0.542
0.479

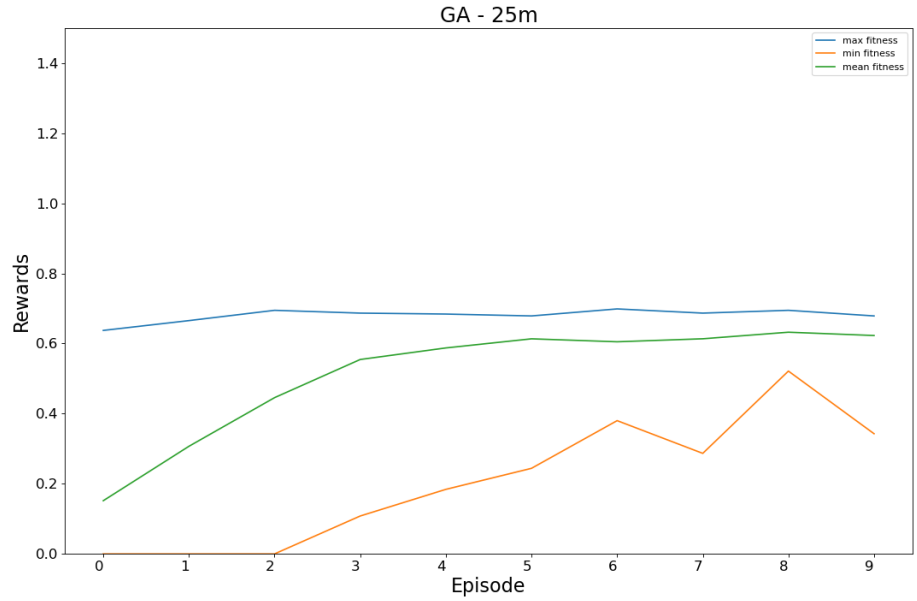

Focus Fire - 8m
Final Rewards
Episode
0.542
0.592
0.283
0.546
0.529
0.283
0.283
0.354
0.392
0.408


GA - 8m
Rewards
Episode
max fitness
min fitness
mean fitness

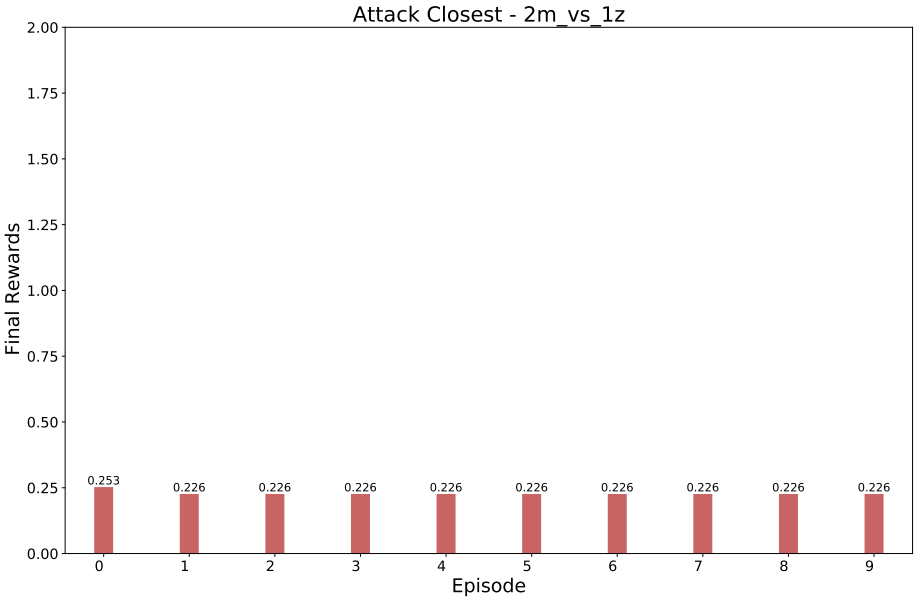

Attack Closest - 25m
Final Rewards
Episode
0.588
0.66
0.636
0.596
0.637
0.684
0.653
0.647
0.629
0.643

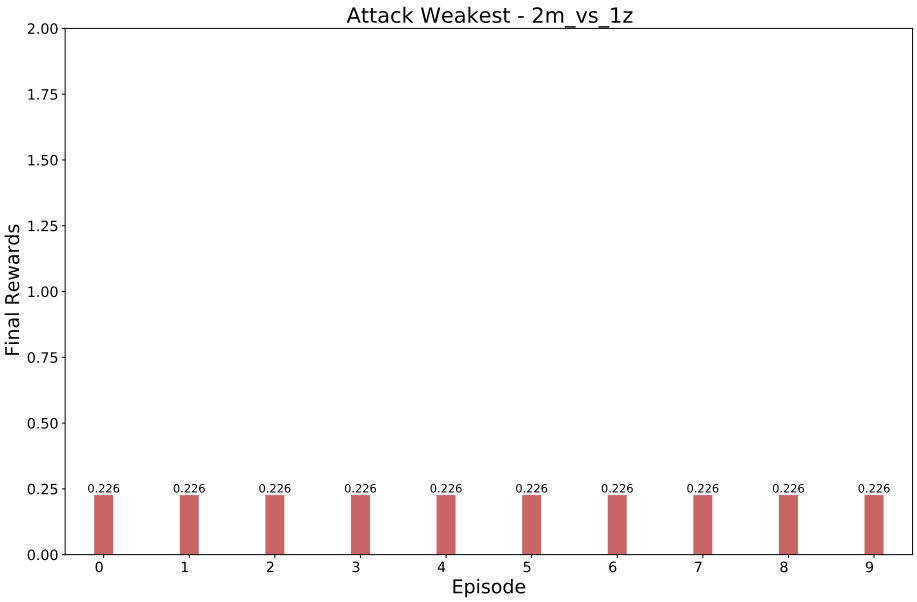

Attack Weakest - 25m
Final Rewards
Episode
0.219
0.225
0.231
0.188
0.205
0.213
0.193
0.211
0.205
0.208

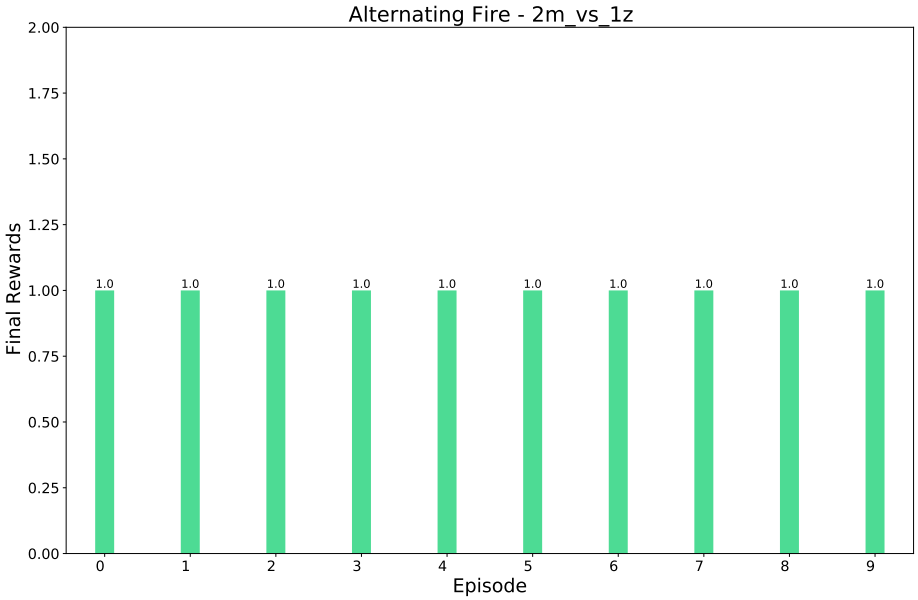




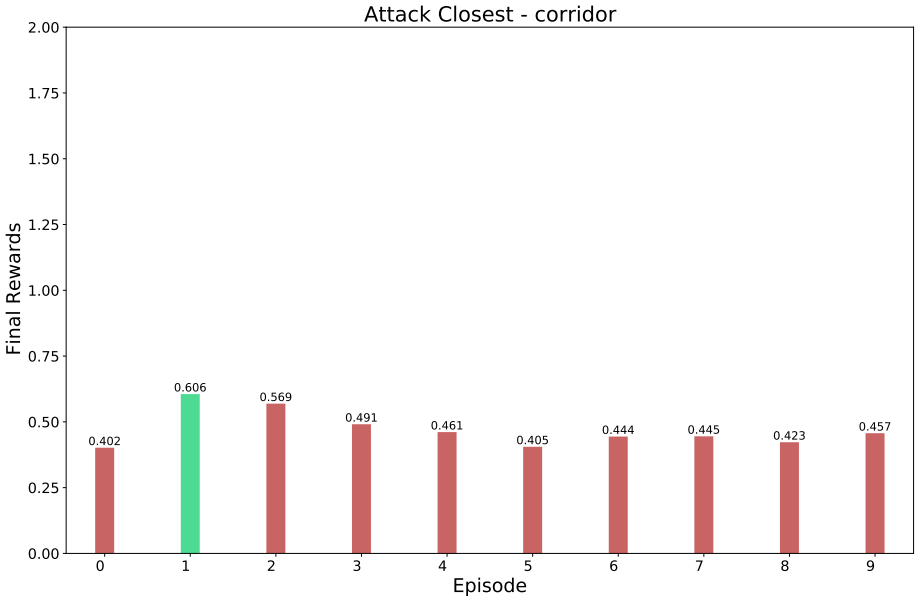
The following are some results on a heterogeneous scenario: 2m_vs_1z, where there are 2 Marines and 1 Zealot.

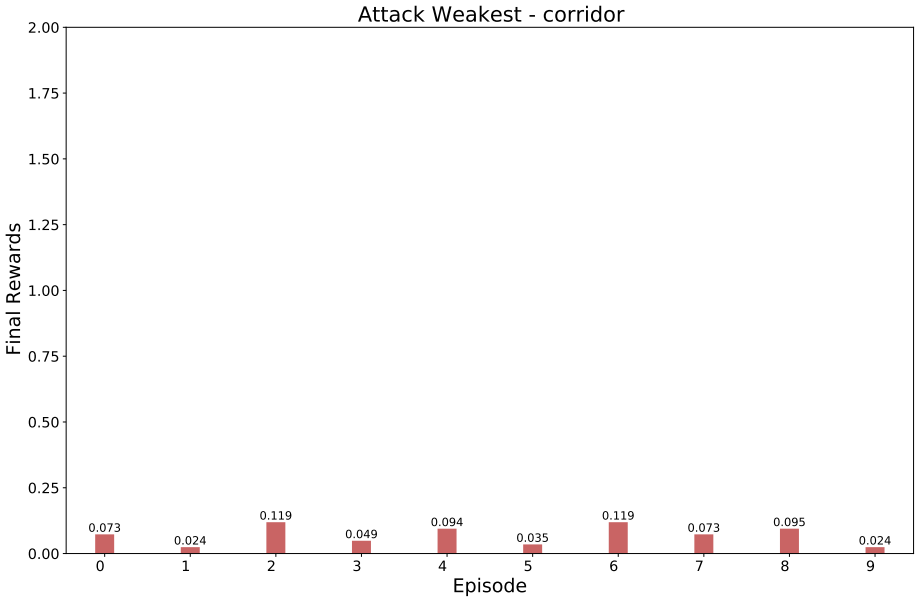

Attack Closest - 2m_vs_1z
Final Rewards
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.253
0.226
0.226
0.226
0.226
0.226
0.226
0.226
0.226
0.226
0
1
2
3
4
5
6
7
8
9
Episode

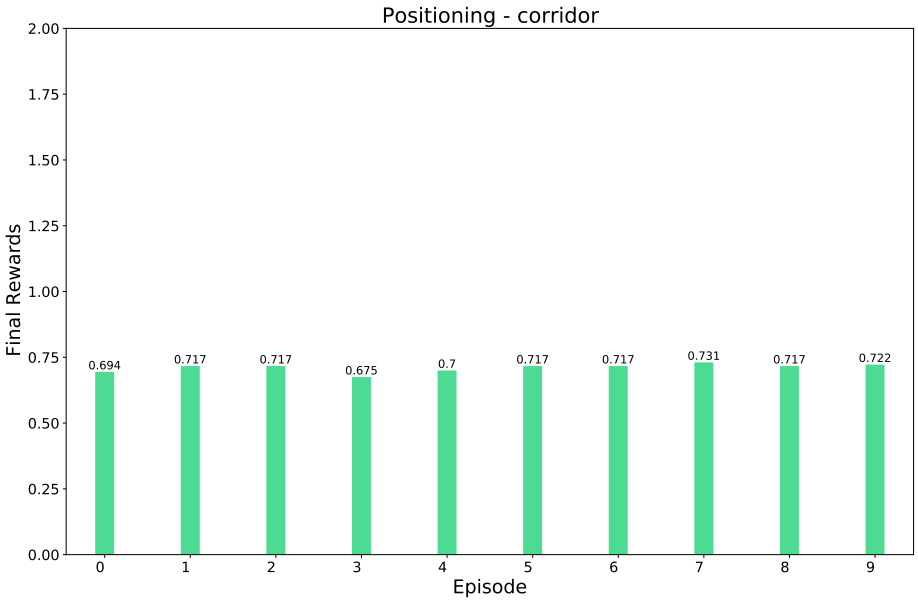

Attack Weakest - 2m_vs_1z
Final Rewards
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.226
0.226
0.226
0.226
0.226
0.226
0.226
0.226
0.226
0.226
0
1
2
3
4
5
6
7
8
9
Episode



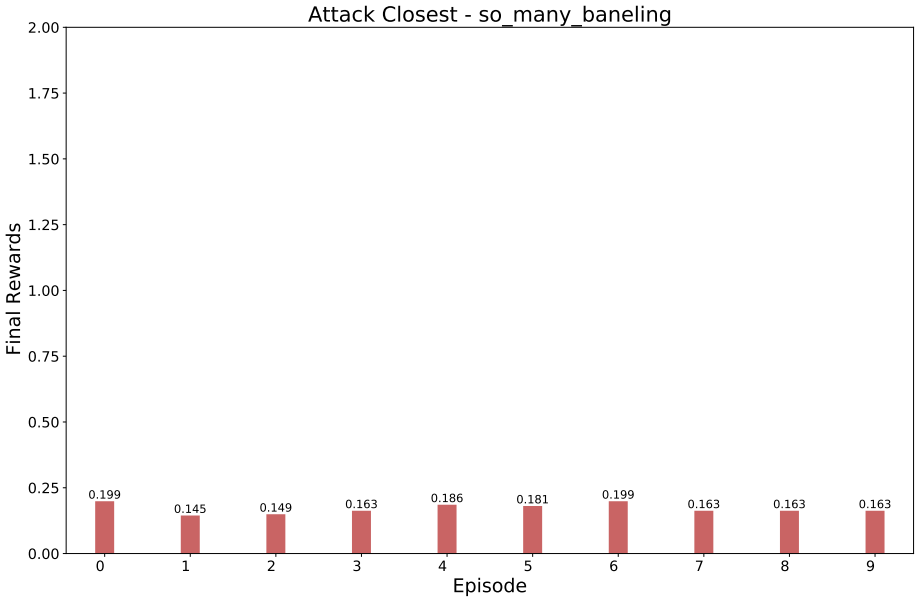
The following are some results on a heterogeneous scenario: corridor, where there are 6 Zealots and 24 Zerglings.

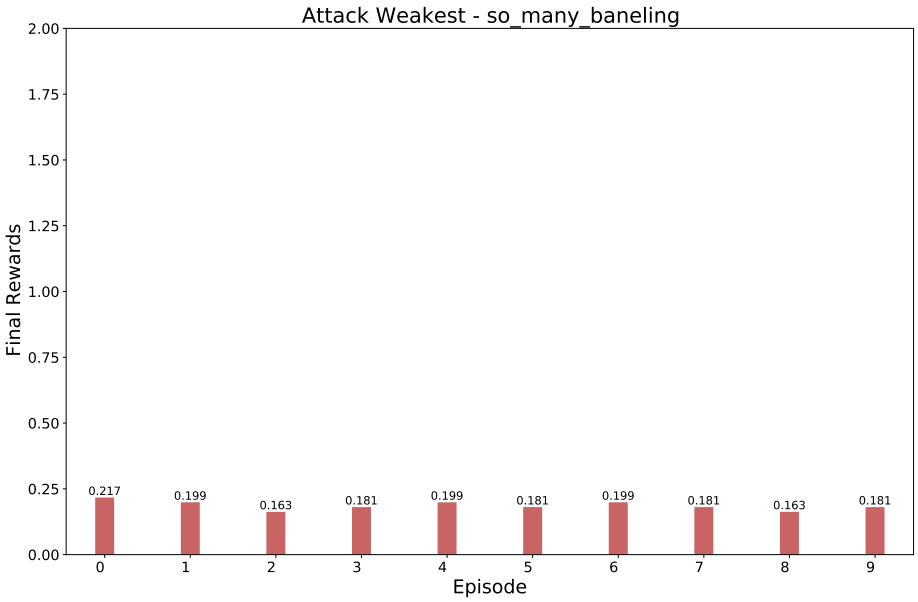


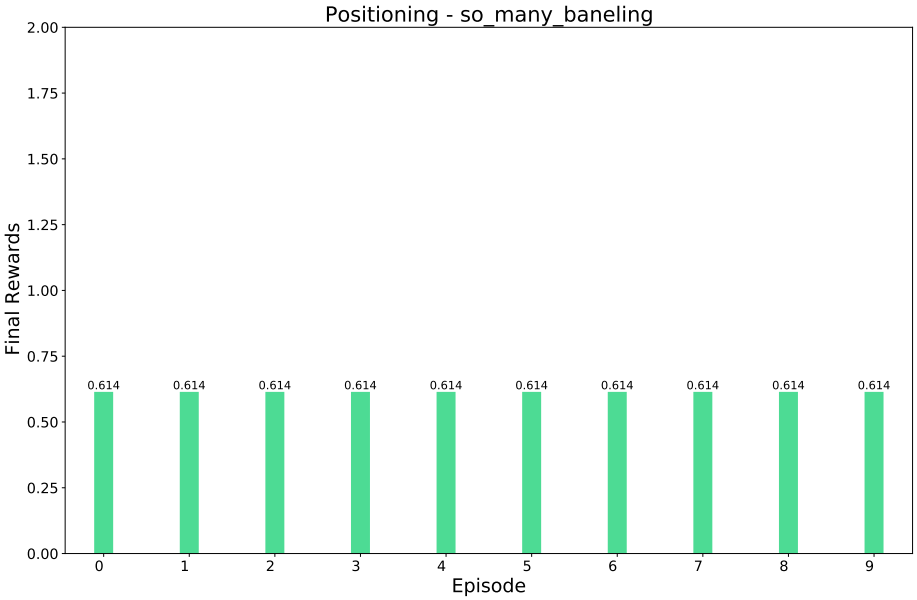




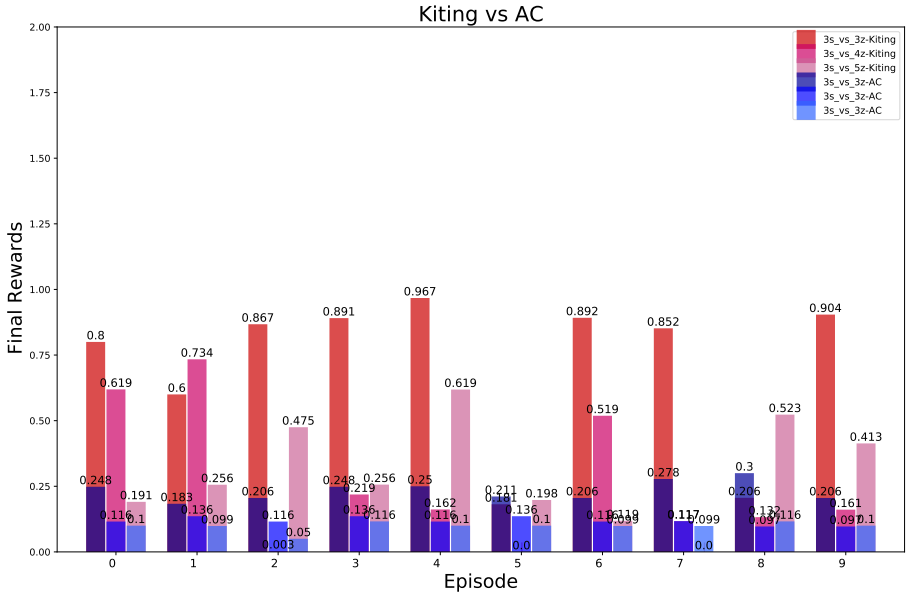
The following are some results on a heterogeneous scenarios so_many_baneling, where there are 7 Zealots and 32 Banelings.

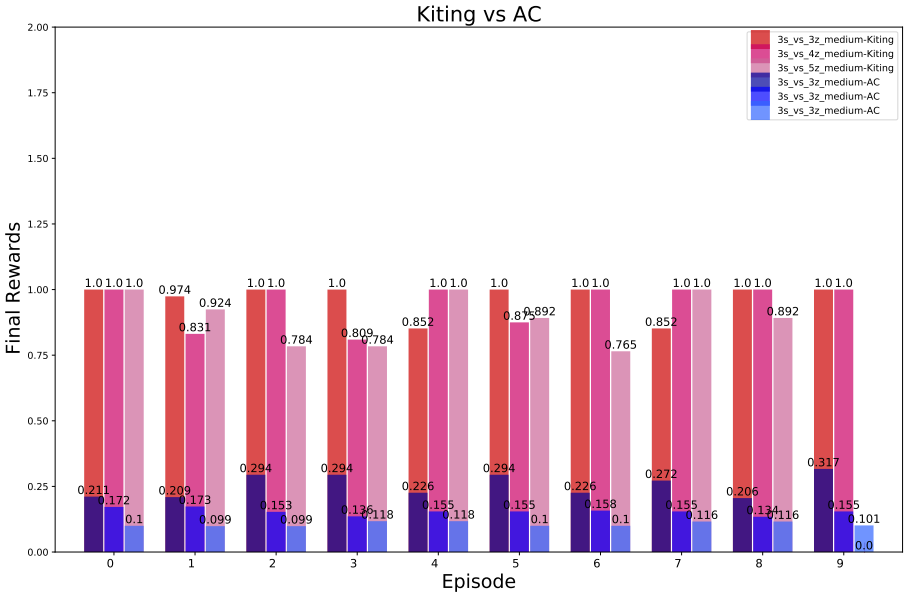

Attack Closest - so_many_baneling
Final Rewards
Episode
0.199
0.145
0.149
0.163
0.186
0.181
0.199
0.163
0.163
0.163


Attack Weakest - so_many_baneling
Final Rewards
Episode
0.217
0.199
0.163
0.181
0.199
0.181
0.199
0.181
0.163
0.181



The following are some results on heterogeneous scenarios: 3s_vs_3z 3s_vs_4z and 3s_vs_5z along with a medium size verison of them.





5.2 Summary

We can see Attack/Heal Closest tend to work well when the size of scenario increases. On the contrary, the performance of Attack/Heal Weakest and Focus Fire tends to decrease when size increases. For heterogeneous scenarios, extra tricks such as Kiting, Alternating Fire, Positioning usually outperform usual methods. Such changes on rewards indicate the scripted behaviours can be used as complements of each other for tackling different tasks. In fact, scripted agents can always be used for baselines when comparing to new algorithms due to their deterministic rules.

The line plots in rewards especially the mean values and minimum values, on genetic algorithms and neural network have shown the model is indeed learning. Algorithms involving evolution of models such as genetic algorithms tend to work well and stay stable on homogeneous scenarios of medium to large size, where the maximum of final rewards/fitness values can stay around 0.7 and mean values can stay at round 0.6. The reason why they did not work well on small size is likely to be caused by some delays in executing actions using these models. Attacking enemies once encountered is the only way to win the combats when the size is small and delays in actions will result in less damage. However, when size is large, the negative effects brought by delays can be alleviated by focusing on attacking single enemies.

We believe the results shown from previous section are representative and they can generalize over other scenarios.

Chapter 6

Conclusion and Future Work

This project focus on StarCraft II micro management, which contains an environment SCMM along with a set of agents and scenarios for exploring micro management during combats in StarCraft II. The results from experiments of in different scenarios have shown different capabilities the agents have to win the combats.

In the near future, the author aims to merge the agents into the maps using StarCraft II editor, which allows more interactions between human players, agents and the actual games. In this way, the agents can be more helpful for human players to perform micro management in the actual games.

Micro management is a sub-task of decentralised control which also includes tasks like cooperation, coordination and communication. A further upgrade on SCMM could include more diverse and challenging scenarios and tasks along with more actions. With more difficult tasks, SCMM along with the agents have the potential to be used as baselines for research with methods involving minimum learning and can become a bridge between non learning methods and deep learning algorithms.

References

[1] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van 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," 2017. (cited on pages 1 and 4)

[2] "Alphastar: Mastering the real-time strategy game starcraft ii." https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii. Accessed: 2020-10-18. (cited on page 1)

[3] P. Sun, X. Sun, L. Han, J. Xiong, Q. Wang, B. Li, Y. Zheng, J. Liu, Y. Liu, H. Liu, and T. Zhang, "Tstarbots: Defeating the cheating level builtin ai in starcraft ii in the full game," 2018. (cited on page 1)

[4] Z.-J. Pang, R.-Z. Liu, Z.-Y. Meng, Y. Zhang, Y. Yu, and T. Lu, "On reinforcement learning for full-length game of starcraft," 2019. (cited on page 1)

[5] M. Samvelyan, T. Rashid, C. S. de Witt, G. Farquhar, N. Nardelli, T. G. J. Rudner, C.-M. Hung, P. H. S. Torr, J. Foerster, and S. Whiteson, "The starcraft multi-agent challenge," 2019. (cited on pages 1, 4, 6, and 44)

[6] S. Rabin, *Game AI Pro 3: Collected Wisdom of Game AI Professionals*. 2017. (cited on page 3)

[7] S. Rabin, *AI Game Programming Wisdom 4*. 2008. (cited on page 3)

[8] D. Churchill, *Heuristic Search Techniques for Real-Time Strategy Games*. PhD thesis, 2016. (cited on pages 3, 4, and 22)

[9] D. Churchill, A. Saffidine, and M. Buro, "Fast heuristic search for rts game combat scenarios," 2012. (cited on page 3)

[10] D. Churchill and M. Buro, "Portfolio greedy search and simulation for large-scale combat in starcraft," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pp. 1–8, 2013. (cited on page 3)

[11] A. Gajurel, S. J. Louis, D. J. Mendez, and S. Liu, "Neuroevolution for rts micro," 2018. (cited on pages 3 and 19)

[12] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002. (cited on page 4)

[13] J. S. Zhen and I. Watson, "Neuroevolution for micromanagement in the real-time strategy game starcraft: Brood war," in *AI 2013: Advances in Artificial Intelligence* (S. Cranefield and A. Nayak, eds.), (Cham), pp. 259–270, Springer International Publishing, 2013. (cited on page 4)

[14] R. Parra and L. Garrido, "Bayesian networks for micromanagement decision imitation in the rts game starcraft," in *Advances in Computational Intelligence* (I. Batyrshin and M. G. Mendoza, eds.), (Berlin, Heidelberg), pp. 433–443, Springer Berlin Heidelberg, 2013. (cited on page 4)

[15] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel, "Multi-agent reinforcement learning in sequential social dilemmas," 2017. (cited on page 4)

[16] L. Zheng, J. Yang, H. Cai, W. Zhang, J. Wang, and Y. Yu, "Magent: A many-agent reinforcement learning platform for artificial collective intelligence," 2017. (cited on page 4)

[17] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, "Mean field multi-agent reinforcement learning," 2018. (cited on page 4)

[18] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," 2020. (cited on page 4)

[19] W. Hsu and Y. Chen, "Learning to select actions in starcraft with genetic algorithms," in *2016 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pp. 270–277, 2016. (cited on page 18)

[20] T. Wang, H. Dong, V. Lesser, and C. Zhang, "Roma: Multi-agent reinforcement learning with emergent roles," 2020. (cited on page 44)

.1 Final Project Descriptions

Project Description:

The purpose of this project is to explore different possibilities for Swarming scenarios and/or Layered Missile Defense scenarios in the StarCraft II editor and learning environment. The steps I will take to complete this project include:

• Read relevant literature surveys and investigate existing methods.
• Create different maps/scenarios for custom AI using StarCraft 2 editor.
• Create scripted and/or machine-learning based bots and analyse their performance in corresponding maps.
• Design new algorithms and compare their performance under custom maps.
• Write a report.

Learning Objectives :

• Familiarity with scenario creation using StarCraft 2 editor.
• Demonstrate understanding of learning algorithms and be able to create suitable maps to test given algorithms for a particular scenario.
• Demonstrate adequate familiarity of learning environment and APIs of StarCraft 2.
• Demonstrate understanding of existing literature.

.2 Study Contract

Australian National University

INDEPENDENT STUDY CONTRACT

Note: Enrolment is subject to approval by the Honours/projects co-ordinator

SECTION A (Students and Supervisors)

UniID: u6625166

FAMILY NAME: Yang PERSONAL NAME(S): Cai

PROJECT SUPERVISOR (*may be external*): Dr. Penny Kyburz, Dr Sabrina Caldwell

COURSE SUPERVISOR (*a RSCS academic*):

COURSE CODE, TITLE AND UNIT: COMP3740 <Project Work in Computing> 6 units

SEMESTER: S2 YEAR: 2020

PROJECT TITLE: Exploring custom scenarios and AI in SC2 editor and learning environment

LEARNING OBJECTIVES:

- Familiarity with scenario creation using StarCraft 2 editor.
- Demonstrate understanding of learning algorithms and be able to create suitable maps to test given algorithms for a particular scenario.
- Demonstrate adequate familiarity of learning environment and APIs of StarCraft 2.
- Demonstrate understanding of existing literature.

PROJECT DESCRIPTION:

The purpose of this project is to explore different possibilities for Swarming scenarios and/or Layered Missile Defense scenarios in the StarCraft II editor and learning environment. The steps I will take to complete this project include:

- Read relevant literature surveys and investigate existing methods.
- Create different maps/scenarios for custom AI using StarCraft 2 editor.
- Create scripted and/or machine-learning based bots and analyse their performance in corresponding maps.
- Design new algorithms and compare their performance under custom maps.
- Write a report.

ASSESSMENT (as per course's project rules web page, with the differences noted below):

☐ Honours (24 credit)	(fixed)	☒ Projects (6-12 credit)	6 / (fixed)
Assessed project components:	**% of mark**	**Assessed project components:**	**% of mark**
Thesis	(85%)	**Thesis** (reviewer mark)	45 45-60%
Presentation	(10%)	**Artefact** (supervisor project mark)	45 30-45%
Critical Feedback	(5%)	**Presentation**	(10%)

MEETING DATES (IF KNOWN):

Weekly

STUDENT DECLARATION: I agree to fulfil the above defined contract:

Cai Yang — 07/31/2020

Signature — Date

SECTION B (Supervisor):

I am willing to supervise and support this project. I have checked the student's academic record and believe this student can complete the project.

Sabina B Caldwell — 31/07/2020

Signature — Date

Reviewer:

Name: Zakir Hossain — Signature:

Reviewer 2: (for Honours only)

Name: — Signature:

REQUIRED DEPARTMENT RESOURCES:

SECTION C (Honours / Projects coordinator approval)

Xing Zhenchang — 31/07/2020

Signature — Date

.3 Description of software / artefacts

The codes used in this project consists the follwing:

scmm/__init__.py

scmm/agents/genetic/__init__.py
scmm/agents/genetic/ga.py
scmm/agents/nn/__init__.py
scmm/agents/nn/nn.py
scmm/agents/potential_field/__init__.py
scmm/agents/potential_field/forces.py
scmm/agents/scripted/__init__.py
scmm/agents/scripted/agent_demo.py
scmm/agents/scripted/alternating_fire.py
scmm/agents/scripted/blocking_enemy.py
scmm/agents/scripted/dying_retreat.py
scmm/agents/scripted/focus_fire.py
scmm/agents/scripted/hybrid_attack.py
scmm/agents/scripted/kiting.py
scmm/agents/scripted/positioning.py
scmm/agents/scripted/wall_off.py

scmm/bin/__init__.py
scmm/bin/map_list.py

scmm/env/__init__.py
scmm/env/multiagentenv.py
scmm/env/micro_env/__init__.py
scmm/env/micro_env/mm_env.py
scmm/env/micro_env/maps/__init__.py
scmm/env/micro_env/maps/mm_maps.py
scmm/env/micro_env/maps/micro_maps/*.SCMap

scmm/utils/__init__.py
scmm/utils/game_show.py
scmm/utils/game_utils.py

Among the above codes, codes under **scmm/env** and **scmm/bin** folder are modified based on [5]. Some of maps in **scmm/env/micro_env/maps/micro_maps/*.SCMap** are from [5; 20]. However, those maps are flawed in terms of functionality and all of them were redesigned. The rest are implemented from scratch.

.4 README

Readme file can also be found in the code repo.

SCMM StarCraft II Micro Management

Installing StarCraft II

You can find lastest verson of StarCraft II here

After installing the game, navigate to `installing_path/StarCraft II/Maps` and move a copy of all the maps in SCMM there.

If `Maps` folder does not exist, which is due to the first time of running the game, then you can manually create one

Installing SCMM

```
$ git clone https://github.com/caiyangcy/SCMM.git
$ pip install SCMM/
```

You also have to install PyTorch, version 1.4.0 is one used in the project

```
$ pip install torch===1.4.0 -f https://download.pytorch.org/whl/torch_stable.html
```

Maps

- You can find a list of maps here

Alternatively, you can run:

```
$ python -m scmm.bin.map_list
```

View a Map

All the maps can be viewed by StarCraft II Editor

Change a Map

The terrain and functionality of a map can be changed by StarCraft II Editor

Create a Map

Create a map using StarCraft II Editor. After creation, make sure add the map to `scmm/env/micro_env/maps/mm_maps.py` and also make sure the map is added to the game folder

Create an unit

The most important thing when creating units on a new map is to disable some reactions of them.

To do this (taken from SMAC):

```
1. Open editor, data editor, unit tab

2. Right click and click add new unit

3. Name the new unit, click suggest right below it

4. Leave the "parent:" row alone. That determines what we're making. We want to make a unit

5. Select the unit you want to copy (bottom of the new opened window, "copy from" row) e.g. zealot if you're copying zealot

6. Set the "Object family:," "Race:," and "Object Type:" as desired. THESE DO NOTHING but make it easier for you to find your ne

7. Press okay, you're almost done

8. Click the plus sign on the data editor tabs, go to edit actor data, actors

9. Click the new actors tab

10. Right click and click add new actor

11. Name it and click suggest like before

12. Change the "Actor Type:" row to unit

13. Select what you want to copy from (bottom of the new opened window again) e.g. zealot if you're coping a zealot

14. Press okay

15. Click on your new actor

16. At the bottom right of the window where it says "Token" and then "Unit Name," change the unit name to the name of your unit

17. Go back to the Unit tab, find the new unit and modify the following fields:

    i.   (Basic) Stats: Supplies - 0
    ii.  Combat: Default Acquire Level - Passive
    iii. Behaviour: Response - No Response
```

Unit Tester Map

A unit tester map can be found at unit tester map folder. Source at unit-tester.

The purpose of this map is to help design some new scenarios.

Run

Refer to the names of agents to find out the details of running agents.

Make sure you are under the correct folder `cd SCMM`

Scripted

Make sure the agent name and map names match when run the scripted agents. Agents like `FocusFire`, `HybridAttackHeal`, `DyingRetreat` work on all maps but other agents like `Kiting`, `Positioning`, `BlockingEnemy` only works for some specific maps.

```
$ python -m scmm.agents.scripted.agent_demo --n_episodes=10 --map_name=3m --difficulty=7 --plot_level=0 --agent=FocusFire
$ python -m scmm.agents.scripted.agent_demo --n_episodes=10 --map_name=8m --difficulty=6 --plot_level=2 --agent=HybridAttackHeal --a
$ python -m scmm.agents.scripted.agent_demo --n_episodes=10 --map_name=3s_vs_3z_medium --difficulty=A --plot_level=0 --agent=Kiting
```

Genetic

```
$ python -m scmm.agents.genetic.ga --n_episodes=10 --map_name=8m --difficulty=7 --plot_level=0
```

NN

```
$ python -m scmm.agents.nn.nn --n_episodes=10 --map_name=25m --difficulty=7 --plot_level=0
```

Potential Field

```
$ python -m scmm.agents.potential_fields.forces --n_episodes=10 --map_name=25m --difficulty=7 --plot_level=0
```

Plots

You can generate plots of rewards using `eval.py` under `scmm.agents`. You can find some pre-generated plots here

Acknowledgement

- The coding on environment and part of the maps were based on SMAC. Refer to the repo for details and license.

.5 A List of Scenarios

Name	Type	Symmetry	Ally Units	Enemy Units
1m[1]	homogeneous	symmetric	1 Marine	1 Marine
3m	homogeneous	symmetric	3 Marines	3 Marines
8m	homogeneous	symmetric	8 Marines	8 Marines
25m	homogeneous	symmetric	25 Marines	25 Marines
5m_vs_6m	homogeneous	asymmetric	5 Marines	6 Marines
8m_vs_9m	homogeneous	asymmetric	8 Marines	9 Marines
10m_vs_11m	homogeneous	asymmetric	10 Marines	11 Marines
27m_vs_30m	homogeneous	asymmetric	27 Marines	30 Marines
half_6m_vs_full_4m	homogeneous	asymmetric	6 half-HP Marines	4 full-HP Marines
half_11m_vs_full_7m	homogeneous	asymmetric	11 half-HP Marines	7 full-HP Marines
half_17m_vs_full_11m	homogeneous	asymmetric	17 half-HP Marines	11 full-HP Marines
bane_vs_bane	homogeneous	asymmetric	20 Zerglings & 4 Banelings	20 Zerglings & 4 Banelings

Table 1: SCMM Homogeneous Scenarios

[1]Mainly used for testing environment setup

Name	Type	Symmetry	Ally Units	Enemy Units
MMM	heterogeneous	symmetric	1 Medivac, 2 Marauders & 7 Marines	1 Medivac, 2 Marauders & 7 Marines
MMM2	heterogeneous	asymmetric	1 Medivac, 2 Marauders & 7 Marines	1 Medivac, 2 Marauders & 8 Marines
2s3z	heterogeneous	symmetric	2 Stalkers & 3 Zealots	2 Stalkers & 3 Zealots
3s5z	heterogeneous	symmetric	3 Stalkers & 5 Zealots	3 Stalkers & 5 Zealots
3s5z_vs_3s6z	heterogeneous	asymmetric	3 Stalkers & 5 Zealots	3 Stalkers & 6 Zealots
3s_vs_3z	heterogeneous	asymmetric	3 Stalkers	3 Zealots
3s_vs_3z_medium[2]	heterogeneous	asymmetric	3 Stalkers	3 Zealots
3s_vs_4z	heterogeneous	asymmetric	3 Stalkers	4 Zealots
3s_vs_4z_medium	heterogeneous	asymmetric	3 Stalkers	4 Zealots
3s_vs_5z	heterogeneous	asymmetric	3 Stalkers	5 Zealots
3s_vs_5z_medium	heterogeneous	asymmetric	3 Stalkers	5 Zealots
1c3s5z	heterogeneous	symmetric	1 Colossus & 3 Stalkers & 5 Zealots	1 Colossus & 3 Stalkers & 5 Zealots
2m_vs_1z	heterogeneous	asymmetric	2 Marines	1 zealot
corridor	heterogeneous	asymmetric	6 Zealots	24 Zerglings
6h_vs_5z	heterogeneous	asymmetric	6 Hydralisks	5 Zealots
2s_vs_1sc	heterogeneous	asymmetric	2 Stalkers	1 Spine Crawler
so_many_baneling	heterogeneous	asymmetric	7 Zealots	32 banelings
2c_vs_64zg	heterogeneous	asymmetric	2 Colossi	64 Zerglings
1s2m_vs_5m	heterogeneous	asymmetric	1 Siege Tank & 2 Marines	5 Marines
1s3m_vs_8m	heterogeneous	asymmetric	1 Siege Tank & 3 Marines	8 Marines
1s3m_vs_4r	heterogeneous	asymmetric	1 Siege Tank & 3 Marines	4 Roaches
6m1r_vs_4g	heterogeneous	asymmetric	5 Marines & 1 Raven	4 Ghosts
12m2r_vs_7g	heterogeneous	asymmetric	12 Marines & 2 Ravens	7 Ghosts
6s4z_vs_8b15z	heterogeneous	asymmetric	6 Stalkers & 4 Zealots	8 Baneling & 15 Zerglings
10z5b_vs_2z3s	heterogeneous	asymmetric	10 Zerglings & 5 Baneling	2 Zealots & 3 Stalkers
6s1s_vs_10r	heterogeneous	asymmetric	6 Stalkers & 1 Sentry	10 Roaches

Table 2: SCMM Heterogeneous Scenarios

[2]map is of medium size