IMPERIAL

MSc Individual Project

Imperial College London

Department of Computing

Quality and Diversity for Real Time Strategy game AI

Author: Edward Wu

Supervisor: Dr Antione Cully

Submitted in partial fulfillment of the requirements for the MSc Computing (Visual Computing and Robotics) degree of Imperial College London

September

Abstract

The video games industry faces ongoing challenges in developing Artificial Intelligence (AI) agents that are both engaging and fair. Traditional game AIs often fall into two extremes: overly simplistic agents lacking depth, or unrealistically powerful agents with inhuman abilities, such as perfect information or reaction times, which diminish player experience. This report investigates the application of reinforcement learning (RL) and Quality-Diversity (QD) algorithms to train AI agents capable of exhibiting both high performance and diverse, humanlike behaviours in Real Time Strategy (RTS) games—an environment characterized by partial observability and large state spaces. The proposed algorithm is largely inspired by the Policy Gradient Assisted MAP Elites (PGA-ME) algorithm. A novel policy gradient emitter was designed for PGA-ME that was based on the QMIX Multi Agent Reinforcement Learning (MARL) algorithm that made it suited for controlling multiple units in a RTS game collaboratively. For the evaluation of the final algorithm, the SMAX environment, a Jax implementation of the StarCraft Multi-Agent Challenge (SMAC) benchmark, was used to generate episodes and visualised to analyse evidence of unique playstyles.

Acknowledgements

Throughout the project I have received help from others and would like to take this opportunity to acknowledge their contributions towards the project.

I would like to offer my thanks to my supervisor, Dr Antoine Cully, as without him and is expertise this project would not have been possible, and PhD student Richard Bornemann who has offered me assistance during the project.

I would like to also thank the contributors and maintainers for the JaxMARL and QDax repositories as their work has aided greatly in my project and prevented me from having to implement everything from myself from scratch.

Finally, I want to thank my parents who have supported me through the project and tough times. Without them I would have never decided to pursue a Master's in Computing with specialism in Visual Computing and Robotics and gotten the chance to do this project.

Contents

1. Introduction

1.1. Context

The video games industry is a large industry that makes use of many technologies such as Artificial Intelligence (AI) agents to enhance the gaming experience. Currently, a challenge encountered by mainstream video game AIs is that they are problematic to balance in terms of difficulty or complexity. On one end of the spectrum, they are too simple only able to perform simple actions and on the other completely overpowered as they have been given abilities for inhuman performance such as frame perfect game input or access to information not available to the human players such a real time location of everything on a map and these are essentially cheats. These problems were encountered by the AlphaStar Team (1) when they developed their StarCraft AI, shown in Figure 1.1, and had to impose limitations such as an Action Per Minute (APM) cap, limit on number of simultaneous actions, and providing the AI map information only a human would have access to achieve a more humanlike behaviour. These problems have resulted in AI earning the stigma of being inferior to humans in the context of video games whether that be skills due to being too simple or unfairness due to having access to cheats.

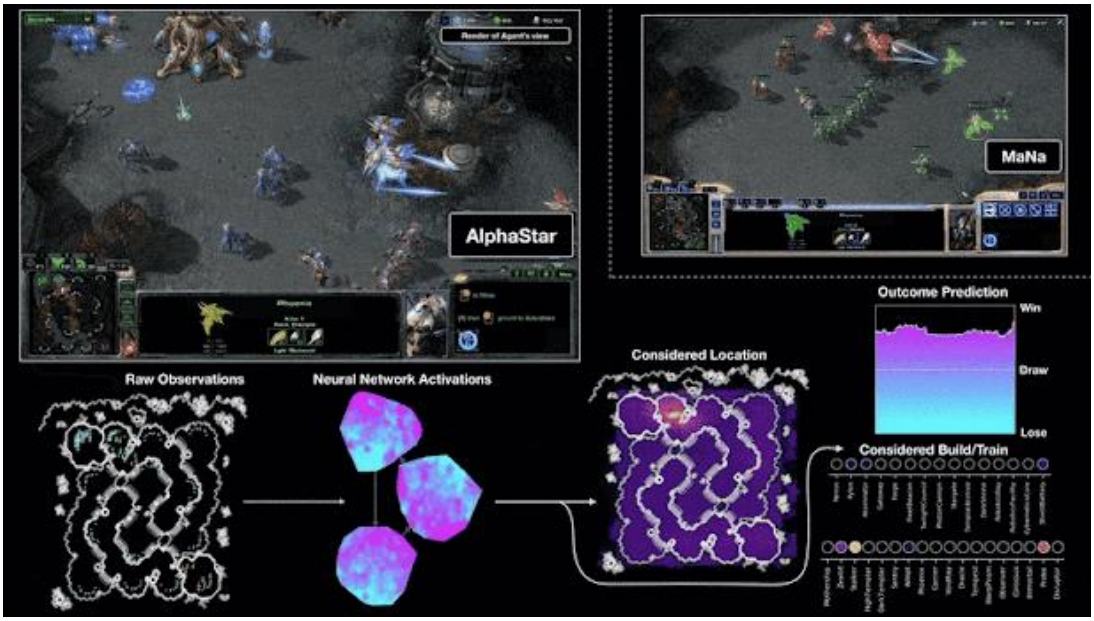


Figure 1.1: Visualisation of the AlphaStar agent during a game from the agents point of view: the raw observation input to the neural network, the neural network's internal activations, some of the considered actions the agent can take such as where to click and what to build, and the predicted outcome. The view in the top right is what is seen by the player MaNa that the agent is playing against, and the agent does not have access to this view (1).

The problem of AIs being too simplistic can partly be attributed to the fact that most game AIs are hand crafted according to (2) meaning that their quality are subject to the amount of resources that have been allocated towards their development and in the wider scheme of video game development they are seen a lower priority according to (3). This means that there is often very little resource allocated to give them complex behaviours as mentioned by (3) and some resort to shortcuts such as cheats which are a quick and easy way to add difficulty.

Machine learning has been increasingly incorporated into the game development pipeline with examples such as (4) due to the recent explosion of generative AI, and platforms such as DALLE-3 (5) and Stable Diffusion (6) for the generation of assets. However, there have been some controversy and push back for this according to (4) and (7) with a big concern regarding the methods of data acquisition for the training of these models as some claim that art being used to train has been stolen. It can be seen as a threat to people's livelihoods as being an artist is a well-established field, whereas design of video game AI is a less established role in video game development and this avoids or lessen the threat posed to people's livelihood. Using reinforcement learning for video game development also avoids the ethical problem with data collection to train the machine learning model as data is generated or collected by the agent itself with the only concern being whether rights have been obtained to collect the data from a specific environment such as the simulator.

Procedural content generation is another area of video game design where machine learning has been used where quality and diversity algorithms are sometimes combined with reinforcement learning according to (8). Examples include generation of bullet hell game levels (9), Role-Playing Games (RPG) (10) and platformers (11). There is also OMNI-EPIC which generated sandbox game environments each with different challenges (12) with examples shown in Figure 1.2.

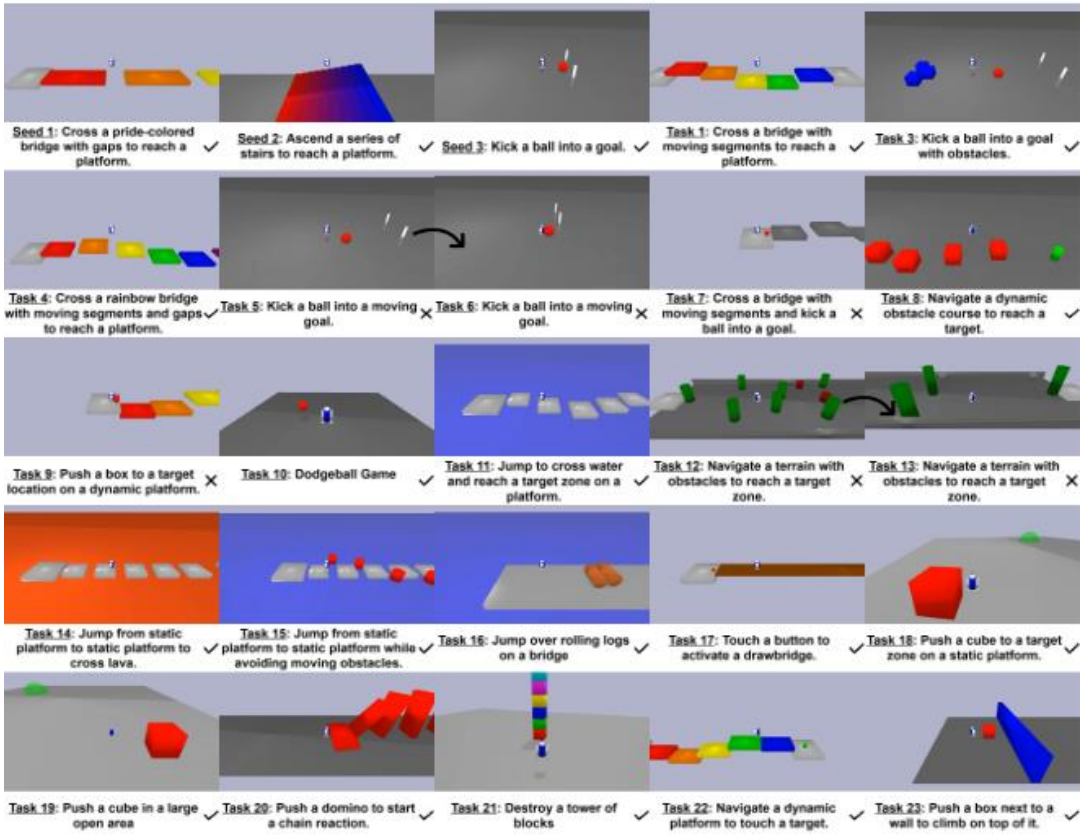


Figure 1.2: A collection of procedurally generated game levels from OMNI-EPIC (12).

Success with reinforcement learning research and video games date back to the breakthrough by Minh (13) who used deep reinforcement learning to train an agent to play Atari games. There was also success on more traditional games such as GO from AlphaZero (14), chess and shogi by DeepMind who then went on to develop AlphaStar (1) that could play the Real Time Strategy (RTS) game StarCraft II. OpenAI achieved a similar success with OpenAI Five (15) which was a team of AI agents that could collaboratively play the Multiplayer Online Battle Arena (MOBA) game Dota 2. Both StarCraft and Dota 2 share many similarities with an important one being that players only get a partial observation of the game state, and this is caused by a "fog of war" mechanic, demonstrated in Figure 1.3, which has been mentioned in research by (1) and (15), and surveys (16) and (17). This means that each unit can only see a fixed distance around itself, therefore the player can only observe the sum of all what is seen by each unit it controls or units on the same team. This breaks the assumptions from the standard Markov Decision Process (MDP), and it is this challenge that has researchers interested in the genre of RTS games.

Figure 1.3: Screenshot of the mini map used in StarCraft: Brood War demonstrating the effects of the fog of war occluding large parts of the map taken from (17).

Another challenge of RTS games is their inherently large state spaces as mentioned in (16) and (17), therefore they can have many optimal or near optimal solutions that are potentially never discovered by reinforcement learning algorithms since they use a single optimisation function or fitness function according to (18). Research (1) and (15) focussed on performance, trying to achieve superhuman performance and not necessarily a diverse set of solution of which there are likely many due to their large and complex state spaces.

Therefore, research into combining quality and diversity algorithms with reinforcement learning (QD RL) is an interesting area of research allowing for greater exploration of the problem and has shown promising results from results by (19), (20), (21) and (22) in the area of robotics control. This has also been attempted in some games as seen by the work from (23) in the video game Gran Turismo and (24) in chess where diversity was added to AlphaZero allowing different playstyles were achieved by both works. However, the majority of QD RL research has been in the context of robotic control which commonly have continuous actions and fully observable environments in contrast to video games such as RTS games which have discrete actions and partially observable environments.

1.2. Objectives

The primary objective of this project is to investigate how reinforcement learning (RL) can be used to train video game AI that exhibit both high performance and behavioural diversity. Specifically, the work aims to overcome the common trade-off in video game AI design where agents are either overly simplistic and predictable or artificially challenging through non-human capabilities such as perfect information or reaction times.

To address this, the project explores the integration of quality and diversity algorithms with reinforcement learning to produce agents that demonstrate a spectrum of effective strategies. These diverse behaviours can then enhance player engagement by offering varied gameplay experiences and perhaps adapting to the different playstyles.

This project will also allow for the implementation of QD RL in a partially observable environment such as an RTS game which is an area that has not been the focus of QD RL which has mainly been used in robotic control. This will involve the added implementation of memory to the architecture to ensure that agents are able to learn from the environment based on its history instead of just current observation.

The aim is to achieve these under a resource constrained scenario to demonstrate such an algorithm is feasible for the game industry which is not likely to access to a large amount of compute just for the design of video game AI.

1.3. Scope of Report

Chapter 2 will introduce what RTS games are and reinforcement learning before going into depth on specific methods of reinforcement learning. Followed by what it means to have high quality and diversity and the associated algorithms that used to introduce such quality and diversity to reinforcement learning. Chapter 3 will then explore the experimental setup used including the Jax python library, the SMAX environment used for the evaluation and the QDax library. The chapter will then propose an initial design, identifying the key parts, and the testing and process taken to get a final design. Chapter 4 will then detail the final design and the evaluation performed to determine whether unique playstyles was achieved with the proposed design. Finally, Chapter 5 will conclude the report noting potential areas for future work.

2. Background

This chapter reviews the literature on research in Real Time Strategy (RTS) games and methods used to try and solve them and explain why RTS game are inherently hard solve. Then it will cover the background on the techniques from reinforcement learning and quality and diversity algorithms. The chapter will then finish detailing the hybrid approach of combining reinforcement learning and quality and diversity algorithms which will form the basis from which the final design is inspired from.

2.1. Real Time Strategy games

RTS is a genre of video games that is much more complicated than the traditional games reinforcement agents are used trained on. The added complexity comes from the fact that unlike chess it is not turned based, and players can perform actions at any time resulting in simultaneous actions as mentioned in (16) and (17). Also, the action space can quickly scale up with the number of available units to control and this is assuming that each unit has the same set of actions but there may be different unit types each with their own set of unique possible actions further complicating the action space according to (16) and (17).

The strategy used in RTS games can be broken down into “macro” and “micro” as mentioned in (16), where “macro” usually refers to the higher-level strategy such as what buildings to construct to spawn a certain unit and getting the resources needed, whereas “micro” refers to the strategy of controlling the units in situations such as combat to ensure victory in an engagement. The focus of the project will be on the micro aspect of RTS strategy and will involve training an AI that can expertly manage any set of given units to ensure victory. The strategies used to micro can be differentiated based on how aggressive or defensive it is so an aggressive strategy could involve directly engaging the enemy to deal and much damage as possible before they have the chance to deal significant damage whereas a defensive strategy would involve much more movement in the attempt to out manoeuvre the enemy unit and minimise the damage taken. A good player would be able to switch between the two approaches and know when to do so in the heat of battle where decisions must be made in under a second.

The unit types available in RTS games tend to have different attributes such as health point HP, attack power AP, attack range, which can be melee or ranged, and many others, each adding a layer of complexity. Some games such as StarCraft also give unique special abilities to a certain unit type adding another layer of strategic complexity. This means that a rock paper scissors relationship can develop between the unit types. A ranged unit would naturally have an advantage when going against a melee unit with the same HP and AP as it has the chance to deal damage before the melee unit closes the distance. However, this could be seen as a soft counter as there is still the chance for the melee unit to overcome the disadvantage with numbers. So, with two melee units there is still the chance for victory if the melee units if it attempts to out manoeuvre the ranged unit by splitting the attention of the ranged unit between them by retreating when HP

is low and reengaging when attentions is switched to another melee unit. There are hard counters, for example a melee unit would never be able to attack an airborne unit regardless of its numbers meaning that it is completely countered. This sort of hard counter usually arises from the macro strategy where one player knowing that the opponent is only spawning melee units spawns airborne units to counter them meaning the outcome has already been decided before the battle has commenced.

Currently, RTS games usually come with hard coded scripted AI agents which players can play with or against. But the problem with these agents is that there is not much variety unless the designers dedicate time to adding more unique playstyles. Earlier research on developing AI agent for StarCraft with reinforcement learning made use of Monte Carlo Search (MCTS) with scripts which were the focus (17). The reason for this was that to solve the complex problem it was broken down into smaller simpler tasks that could be handled by the MCTS and then the complex actions were bundled into hand crafted scripts which played for the agent when selected. This is an area that would be improved with deep reinforcement learning algorithms such as AlphaStar (1).

It is possible for RTS games to be formulated as a Multi Agent Reinforcement Learning problem where each individual unit is an agent, and a policy is learned that allow these agents to work together. A famous MARL benchmark is the StarCraft Multi-Agent Challenge, SMAC (25), which uses the StarCraft RTS game to test different MARL algorithms. Examples of different combat scenarios can be seen in Figure 2.1.

Figure 2.1: Screenshots of several SMAC scenario that could be used for testing (26).

2.2. Reinforcement Learning

The goal of reinforcement learning is to learn a policy for which an agent will maximise the cumulative reward in an environment modelled as a Markov Decision Process (MDP). The tuple (S, A, R, T, γ) can be used to represent the MDP where S is the state space, A is the action space, R is reward function, T is the transition dynamics function and γ is the discount factor. So, for an episode of length T and $t = [0, T]$ where current state is s_t and the taken action is a_t with immediate reward r_{t+1} results in the following cumulative reward R_t in Equation 2.1 for the current step t in the episode (27).

$$R_t = \sum_{i=0}^{\infty} \gamma^i \, r_{t+1} \qquad 2.1$$

However, in a traditional RTS game such as StarCraft, rewards are sparse where the only real reward given is to win at the end of an episode and sequence of actions taken by the agent does not really matter and for this reason it can be hard for reinforcement learning to be stable. The

game implicitly allows the player creativity in how to reach the final state of winning, and this is why RTS games are good for training diverse solutions. To overcome the problem of sparse rewards, reward shaping can be used to give an agent immediate rewards for doing certain actions in the environment that it known to help it reach the winning state. For the example of StarCraft, it could be choosing to do damage to enemy units or structures which while will not directly cause the agent to win brings it closer to winning as to win means to destroy all enemy units and structures and this was used in work by (28).

Deep Reinforcement Learning was used to overcome the problem of large search spaces where explicitly mapping out all the possible combinations of states and actions along with their rewards was infeasible. Instead, a policy or Q value function could be learned through a deep neural network, as seen in Figure 2.2, and was used to great success by (29) in a Deep Q Network (DQN). The DQN introduced several key advancements which included a slower updating target network and replay buffer. As an off-policy method, it had a behavioural policy that was used to collect episodes from the environment which were placed into a replay buffer that was sampled from to update the target network which learned the Q values of the states in the environment.

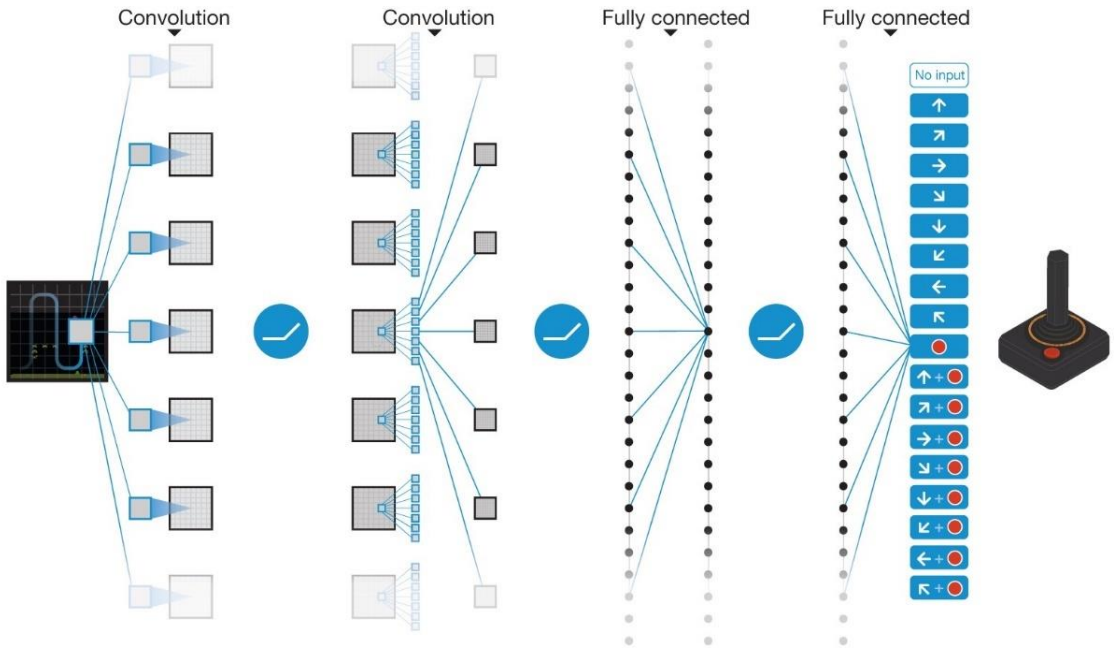


Figure 2.2: Diagram showing the architecture of the neural network that was trained in (29) for playing Atari games.

The actor critic architecture by (30) was then combined with deep learning to get the benefits of both policy-based and value-based approaches to reduce the high variance of policy based deep learning. This was possible through the concept of advantage $A(s_t, a_t)$ which measured how much better an action a_t is compared to the expected value $Q(s_t, a_t)$ of being in a particular state s_t. The critic network learns $V(s_t)$ through the minimisation of the temporal difference of the value function during training as shown in Equation 2.2.

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{2.2}$$

The extension of actor critic by (31), led to the Deep Deterministic Policy Gradient (DDPG) method which extended DQNs to with continuous actions by using a deterministic policy instead of a stochastic policy.

Policy based methods also used the actor critic model with deep learning to directly learn the policy without needing to first learn a model of the environment which could be intractable. However, they suffer from instability as during training it was possible that large updates were made to the policy taking them outside the region of stability. To improve stability of policy gradient methods, Trust Region Policy Optimization (TRPO) (32) was introduced, which constrains policy updates by enforcing a trust region, preventing the new policy from deviating too far from the old one within one update. TRPO can be complex to implement due to its reliance on second-order optimization, so Proximal Policy Optimization (PPO) (33) was later proposed to simplify this idea by replacing the trust-region constraint with a clipped surrogate objective, that limits the policy update by clipping the probability ratio between the new and old policy when multiplied by the estimated advantage making it easier to implement. This idea is demonstrated in Figure 2.3.

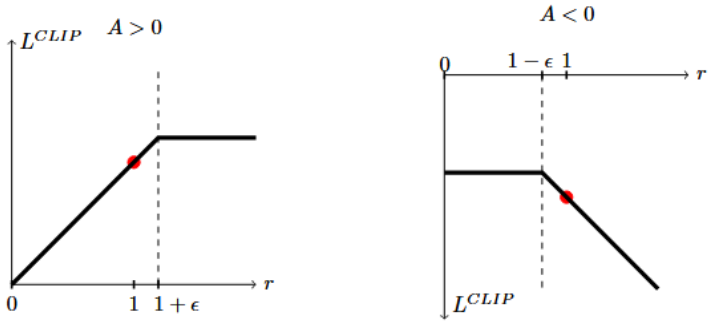


Figure 2.3: Graphs demonstrating the clipping used for the surrogate objective L^{CLIP} of PPO for positive advantage on the left and negative advantage on the right (33).

2.3. Quality and Diversity

The aim of Quality and Diversity algorithms stated by (18) is to find a set of solutions that are not only performant but diverse in contrast to traditional reinforcement learning algorithms which optimise to one solution that maximises cumulative sum of rewards. Instead, inspiration was taken from evolution in nature by research from (34) where a wide variety of animals have evolved adapting to different environments with diverse features all seemingly with the same goal of surviving and reproducing. Key ideas from QD algorithms are the training of a large population of solutions, local competition between solutions and rewarding diversity which were all present in AlphaStar according to (35). Although (1) never explicitly stated that AlphaStar was or made use of QD algorithm it demonstrates just how powerful QD algorithms are, and how it formalised existing concepts in AI research. To determine whether two solutions are different, Behaviour Descriptors (BDs) are used to measure certain behavioural characteristics in the solutions and form a behaviour space. For example, in the work by (34) a 6-dimensional BD was used to quantify the percentage of time each limb was in contact with the ground so the way diversity was expressed was how each solution made use of or not use the 6 limbs to move.

MAP-Elites (ME) is a QD algorithm that maintains a large population of solutions in an archive or repertoire that represents the behaviour space and solutions are stored based on their BDs and this illuminates or highlights region in the behaviour space where solutions have been found and filled according to (26). Since the behaviour space is continuous, it is discretised into local regions which allow for local competition as solutions with the same or similar BDs would end up in the same cell and since each cell had only one cell it stores the higher quality solution (26).

To generate the solutions to populate the repertoire, genetic algorithms (GA) according to (36) are used to mutate existing solutions in the map and these are evaluated to determine whether they should replace solutions already in the map. So, after initialisation the algorithms runs the loop of selection, mutation, evolution and replacement as shown in the pseudocode in Figure 2.4.

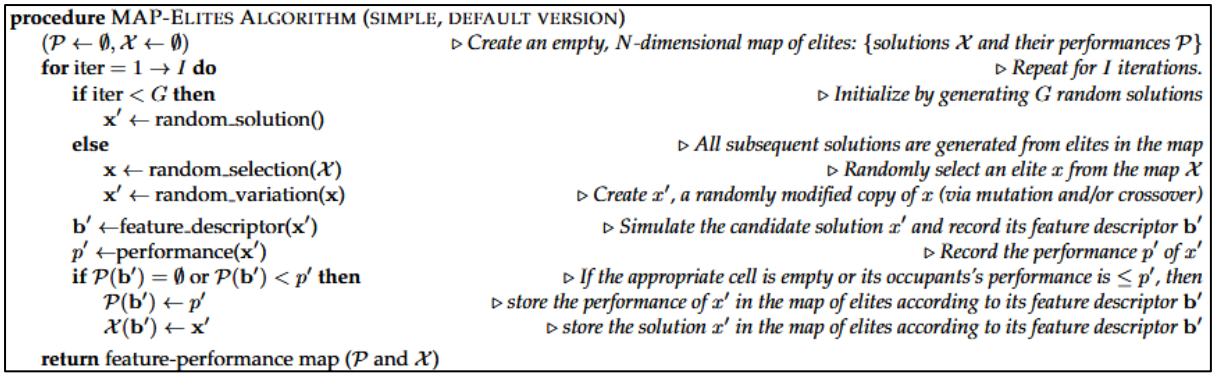


Figure 2.4: Pseudocode of the simple default version of MAP Elites (26).

Evolutionary strategies are another method such as Covariance Matrix Adaption Evolutionary Strategy (CMA-ES) (37), (38), and Exponential Natural Evolution Strategies (xNES) (39) which are less random than GAs used in MAP-Elites where in each generation of offsprings the algorithm tends toward the optimum as it is updated. Although diverse solutions are found, these are not stored, and the methods tend towards an optimum. Covariance Matrix Adaption with MAP-Elites via a Gradient Arborescence (CMA-MEGA) (40) exploits the optimisation algorithm CMA-ES with the use of MAP-Elites to store solutions and balance the exploitation and exploration to get a population of high quality and diverse solutions.

2.4. Hybrid approach

The problem with the MAP-Elites algorithm according to (19) is that it struggles with high dimensionality problems due to the use of GAs as offsprings that are generated although diverse are completely random meaning that convergence to a high fitness solution is slow. One approach to solving this is by taking a hybrid approach with reinforcement learning and quality and diversity algorithms where the aim is to get the diverse solutions from QD but speed up the convergence by using gradient updates from reinforcement learning. This was the approach taken by Policy Gradient Assisted MAP-Elites (PGA-ME) (19) where TD3 (41) was the reinforcement learning chosen. It used an actor and critic architecture and had 2 critics to prevent overly optimistic Q value approximations by taking the minimum of the 2 best value predictions and were trained using episodes from a replay buffer. The choice of TD3, an off-policy method, also gave the PGA-ME method high sample efficiency according to (19). The trained critic then used by the actor to determine the value of the action it has chosen, and this can then be used to get the gradient to update the actor. This happens for half of the policies sampled from the archive with the other half still uses a GA and this strikes a balance between quality and diverse solutions in the offspring population, and each are evaluated to see if it should be added to the archive. A summary of the PGA-ME algorithm can be seen in Figure 2.5.

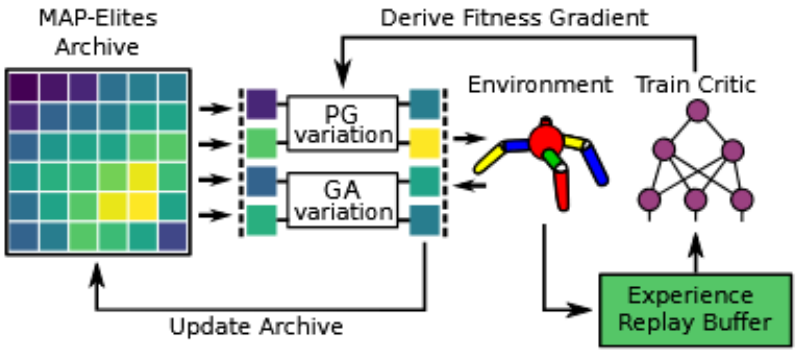


Figure 2.5: Diagram showing how the PGA-ME algorithm works (19).

Descriptor Conditioned Policy MAP-Elites (DCP-ME) from (20) improves upon PGA-ME by conditioning the actor and critic of PGA-ME to the behavioural descriptor and this was done to prevent all the gradient updates going towards the global maximum but a local maximum depending on the descriptor allowing for diversity in offsprings produced from the policy gradient updates. The work by (20) also showed that a byproduct of conditioning the actor to the descriptor is that it ends up distilling the solutions in the archive allowing for the simple change in policy by inputting the desired descriptor to the actor after it has been trained without needing to have to store all the solutions in the archive. The Descriptor Conditioned Reinforcement Learning (DCRL) method from (21) further improves the upon DCP-ME by also utilising the descriptor conditioned actor as a generator for new offspring population by giving as an input the descriptors desired for the policies which there may not already have a solution in the repertoire. This further increases diversity and speedup up the method as it no longer relies on the offspring solution propagating away from parent policies to unexplored regions according to (21).

A different approach is Proximal Policy Gradient Arborescence (PPGA) from (22), which in contrast to PGA-ME and its improved methods, used the on-policy reinforcement learning method PPO, but in doing so loses the benefit of the high sample efficiency. What it gains is a better search for solutions as it was able to find more performant and diverse policies for the MuJoCo Humanoid robot benchmark according to (22) which PGA-ME failed to perform well on. It was suggested that on-policy methods such as PPO offered more stable and effective gradient information for navigating the QD search space than off-policy methods such as TD3 as when PPO was replaced for TD3 in the method a significant loses of performance was observed according to (22). The method was inspired by CMA-MAEGA from (42) which was an improved version of CMA-MEGA by (40).

The PPGA method from (22) introduces Markovian Measure Proxies (MMPs) which in contrast to traditional BDs, which are only calculated and known at the end of an episode, are descriptors that are known for each state during roll out of the episode. For example, a BD could be percentage of time robotic limb is in contact with ground which can only be calculated at the end as the ratio of steps in the episode limb was contact with ground and length of episode. The MMP of that would be is the limb in contact with the ground and be binary, so 1 if true else 0. This makes the descriptors differentiable during rollout according to (22) which is key with an on-policy method such as PPO as now gradient can not only be calculated with respect to the fitness function ∇_f but also the descriptors or MMPs ∇_δ. Equation 2.3 shows the formulation of the final combined gradient, ∇, used for the gradient update step.

$$\nabla = c_0 \nabla_f + \sum_{j=1}^{k} c_j \nabla_{\delta_j} \quad 2.3$$

To take advantage of multiple gradients, xNES (39) is used to sample a set of gradient coefficients c which determine the contribution of each gradient in the final combined gradient value ∇ used for the update. To get diversity and k offsprings, k sets of gradient coefficients are generated that update the parent policy in a different direction for each offspring. The offsprings are then evaluated and added to a MAP-Elites repertoire depending on their fitness and then the offsprings are ranked based on their improvement to solutions in the repertoire. This ranking is used to update the mean and covariance matrix of xNES to produce a mean gradient coefficient for the parent policy to step it in the direction of most potential based on offsprings for further exploration. This is then repeated until the offsprings no longer result in an improvement over the solutions already in the archive, after which xNES is reset and a new parent policy is sampled from the archive to repeat the process. The PPGA algorithm is summarised in Figure 2.6.

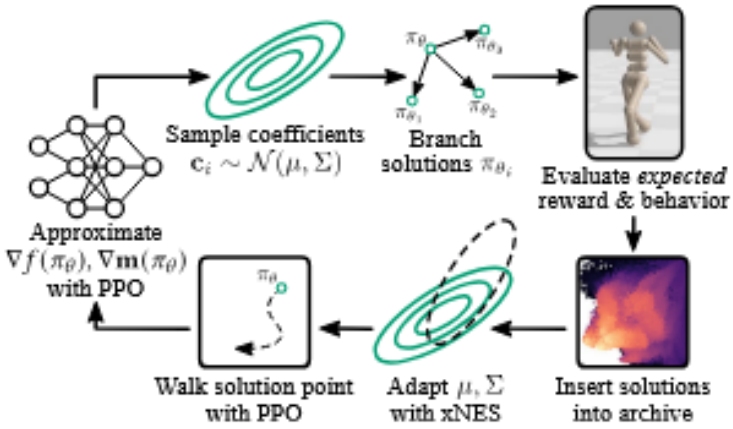


Figure 2.6: Diagram showing how the PPGA algorithm works (22).

3. Methodology

This chapter details the methodology for the design of the algorithm based on the literature explored in the previous chapter. It will start with detailing the experimental setup and environment used before going into the design. The design was divided into two parts, the reinforcement algorithm and the quality and diversity algorithms which were worked on separately. The chapter will detail the testing performed for the design of the two parts to refine them for the final design which will be covered in the next chapter.

3.1. Experimental Setup

The Jax Python Library (43) enables fast training through its JIT complier that generates highly optimised code for the GPU accelerator. It also has high vectorisation of instructions and an efficient auto differentiation. This has led to the development of highly efficient Jax based environments for reinforcement learning research such as JaxMARL by (44) which is a collection of environments specific for Multi-Agent Reinforcement Learning (MARL).

The QDax Library (45) contained many quality and diversity algorithms, leveraging the performance from using Jax, ready to use such as MAP-Elites. It contained a range of "containers" which were the repertoire types that were used to store solutions and "emitters" which were the algorithms used to generate new offspring solution each iteration. These were designed to be modular so that they could easily swap them out for others.

3.2. SMAX environment

SMAX is one of the environments available in JaxMARL (44) and is a Jax implementation of an existing MARL benchmark called SMAC (StarCraft Multi-Agent Challenge) by (25) that made use of the StarCraft Python API. One of the downsides of SMAC was its reliance on the StarCraft game engine through the API which makes training and agent extremely time consuming requiring a large amount of compute. Other environments such as MircoRTS-Py (28) exist that simulated a more complete RTS game but are not implemented in Jax. Hence, SMAX was used due for its ease of efficient parallel rollouts that made train a large population of solutions feasible.

Although SMAX and SMAC are MARL environments, the focus of the project was not designing a new MARL algorithm, instead existing MARL algorithms will be used in the design of the proposed design. One approach for MARL, is that each agent acts independently using the same shared policy and this was shown to be effective by (46) for SMAC with their Independent PPO (IPPO) algorithm as there is implicit information sharing between the agents through the sharing of the same policy weights.

In SMAX an immediate reward is given based on the percentage damage dealt to the enemy health, calculated from difference in health before and after an attack normalised by max health of the unit type, in each time step as shown in Equation 3.1. This was used for reward shaping as otherwise there is only the victory reward at the end of the episode which is only given if all enemies are dead and at least one ally is alive. The reward for each agent on the team was the same as the overall team reward for the team and this was to encourage teamwork between the units so that agents choose actions that benefit the team rather than themselves.

$$R = \sum_{t=1}^{T} r_t = \sum_{t=1}^{T} \left(\sum_{j \in enemies} \left(\frac{H_j^{t-1} - H_j^t}{H_j^0} \right) + 1\{win\} \cdot B \right) \quad 3.1$$

Where:

- R = Return
- T = max episode length
- t = timestep
- r_t = immediate reward
- j = enemy index in set of all enemies
- H_j^t = Heath of enemy j at timestep t
- B = Battle won bonus, 0 if battle not won or 1 is battle has been won

The reward for winning an engagement was only given under the condition that there was at least one ally unit alive and all enemy units are dead, and this was because due to the use of vmap in JAX for parallelisation it was possible for actions to happen simultaneously and therefore a draw could occur (44). The theoretical max return was 2.0 with 1.0 from the win bonus and 1.0 for the damage dealt to enemy total health.

The discrete actions in SMAX consisted of move up, down, left, right, stop, and attack, where the number of instances for attack action was less than or equal to max number of enemies depending on how many were in attack range and still alive. The locations of agent units in 2D space were also discrete represented as a grid as seen in Figure 3.1. Table 3.1 and Table 3.2 summarises other information regarding the SMAX environment including unit types and possible scenarios for the simulation.

Table 3.1: Table showing all the possible unit type along with the shorthand used for the scenario names and some of their stats which include Sight range (radius of the circular observation range), Attack range (radius of the circular attack range), Attack power (the damage inflicted per attack), Attack cooldown (the minimum interval between each attack), Heath Point (max amount of damage that can be taken before death).

Unit Name	Shorthand	Sight Range	Attack Range	Attack Power	Attack Cooldown	Health Points
Marine	m	9.0	5.0	9.0	0.61	45.0
Marauder	M	10.0	6.0	10.0	1.07	125.0
Stalker	s	10.0	6.0	13.0	1.87	160.0
Zealot	Z	9.0	2.0	8.0	0.86	150.0
Zergling	z	8.0	2.0	5.0	0.50	35.0
Hydralisk	h	9.0	5.0	12.0	0.59	80.0

Table 3.2: Table showing all the possible scenarios in the SMAX environment along with the unit compositions for the ally and enemy.

Scenario Name	Ally Units	Enemy Units
2s3z	2 stalkers & 3 zealots	2 stalkers & 3 zealots
3s5z	3 stalkers & 5 zealots	3 stalkers & 5 zealots
5m_vs_6m	5 marines	6 marines
10m_vs_11m	10 marines	11 marines
27m_vs_30m	27 marines	30 marines
3s5z_vs_3s6z	3 stalkers & 5 zealots	3 stalkers & 6 zealots
3s_vs_5z	3 stalkers	5 zealots
6h_vs_8z	6 hydralisks	8 zealots
smacv2_5_units	5 randomly chosen	5 randomly chosen
smacv2_10_units	10 randomly chosen	10 randomly chosen
smacv2_20_units	20 randomly chosen	20 randomly chosen

The scenarios in Table 3.2 could be classified into five distinct types of environments which were: heterogeneous unit types with mirror match up (2s3z and 3s5z), homogenous unit types with number disadvantage for allies (5m_vs_6m, 10m_vs_11m and 27m_vs_30m), heterogenous unit types with numbers disadvantage for allies (3s5z_vs_3s6z) and heterogenous unit types across ally and enemy units with numbers disadvantage for allies (3s_vs_5z and 6h_vs_8z). There was also the option for completely random unit types with equal numbers across ally and enemy units. Most testing performed was using the 2s3z scenario which had 2 stalkers units that were ranged units and 3 zealots which were melee units.

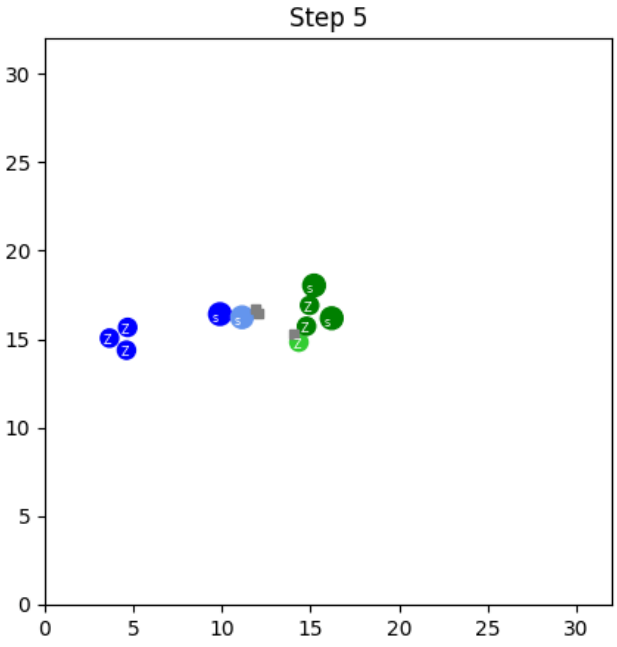


Figure 3.1: Screenshot that shows one frame from the output GIF showing the visualisation of the learned IPPO policy (blue) against a Heuristic enemy (green). Units are represented as circular disks marked by the letter shorthand corresponding to the unit type. Units with a faded colour symbolises them taking damage and the grey squares represent attacks which move from the unit that is attacking to the unit that is being attacked.

SMAX also had a visualiser function that allowed for the visualisation of an episode from a policy and Figure 3.1 shows a frame from the resulting GIF generated by the visualisation function in SMAX for a learned policy against an enemy heuristic policy.

The enemy heuristic policy was handcrafted by (44) such that units it controlled always headed towards the centre of the environment and attacked the closest target it encountered and stuck to this target even if another target reached a closer range to prevent flip flopping between targets that could be exploited. Doing this theoretically allowed the heuristic policy to achieve a win rate close to 50% when playing against itself according to (44) and this was used as a benchmark when evaluating new policies to see it learned something useful during training.

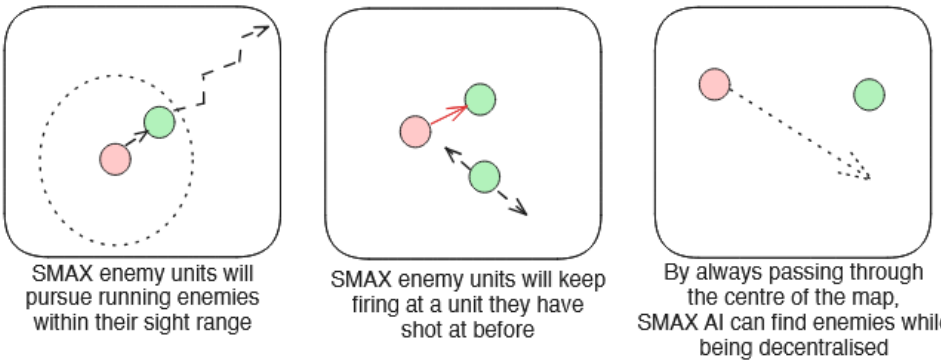


Figure 3.2: Diagram that demonstrates the behaviour of the heuristic policy in SMAX (44).

Due to the possibility of simultaneous actions and collisions, when two units tried to occupy the same position, tie-breaking was used. This made the environment highly stochastic as the same sequence of actions across episodes can lead to very different outcomes depending on the Jax PRNG key used for the SMAX environment. To make sure evaluations are robust guidance was taken from (47) to average the performance across multiple evaluations to get a more representative results for the actual performance of a particular policy.

3.3. Initial Design

The intended use of the algorithm in the video game industry was that the algorithm would train a wide range of high performing and diverse solution stored in a repertoire that could then be used to select the best solution as well as a solution with a specific BD that corresponded to a unique playstyle. This removed the need for the designer to hard code different play styles which may be difficult or cumbersome. A constraint on the amount of GPU resources was also added to try a represent the amount of GPU hardware a game designer may realistically have access to. Therefore, experiments were done using a NVIDIA Tesla T4 or NVIDIA A16 as they both had access to 16GB VRAM which is accessible on consumer GPUs.

From the literature review undertaken in chapter 2, promising methods that combine both reinforcement learning, and quality and diversity algorithms were PGA-ME and PPGA. Focus was placed on PGA-ME but, since it was designed for continuous actions, adaptations were needed so that it worked for the discrete actions of SMAX. The choice of PGA-ME meant that there were two parts to the design: the reinforcement algorithm to provide the gradient information for the policy gradient emitter to refine existing elite solutions to get new offsprings and the quality and diversity algorithm which was responsible for the scoring and storing of the offsprings. MAP Elites

also needed a variation operator was needed as the genetic variation emitter to mutate new random offsprings from existing elite solutions to promote exploration of new potential solutions and increase diversity of episodes in the replay buffer.

The proposed design would train a population of good solutions, for a specific scenario in the SMAX env starting with a batch of randomly initialised policy weights. At the end, it was expected that the final population contained in the repertoire lead to unique behaviours that could be considered playstyles. Solutions in this context referred to the combination of weights in the neural network for the policy network and offsprings are newly generated candidate solutions that have yet to be evaluated to determine whether they should be added to the repertoire of elite solutions. The policy gradient emitter and the genetic variation emitter both contribute to the new population of offsprings each iteration of the MAP-Elites algorithm. A diagram of the proposed design can be seen in Figure 3.3.

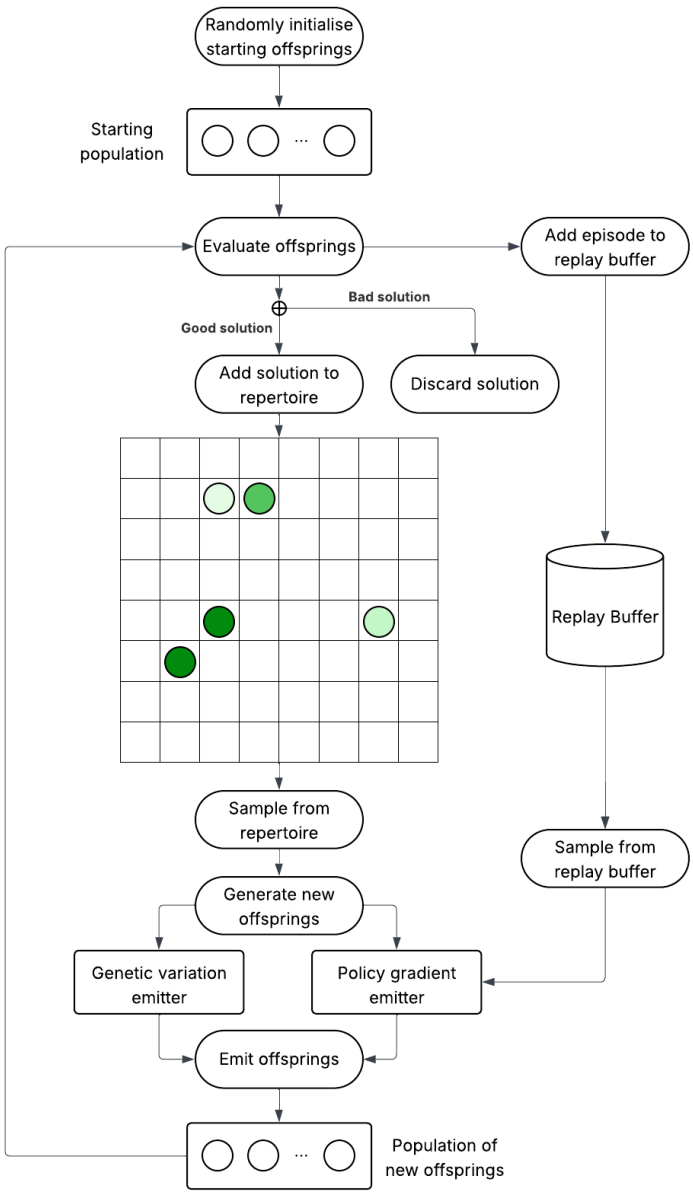


Figure 3.3: Diagram showing the envisioned design for the final design of the algorithm and highlights the parts that needed to be designed such as the policy gradient emitter, genetic variation emitter and evaluation function.

3.4. Reinforcement Learning (RL) algorithm

The decision to use PGA-ME constrained the possible RL algorithm to those that were off policy. It was also preferred that the method produced a deterministic policy which allowed for more reliable evaluation of the offsprings. Stochastic policies could still have been used but would have needed to be made greedy during evaluation. The choice of using the SMAX environment also introduced the constraint that the algorithm had to work with discrete actions instead of the continuous actions expected from robotics control where PGA-ME was originally applied. From Table 3.3, potential candidates were considered and IQL along with QMIX were chosen for initial testing as they were both off policy and produced deterministic policies. Both methods learnt a Deep Q Network and the policy was derived from the Q values (48) (49).

Table 3.3: Table of potential reinforcement learning algorithms for the policy gradient emitter.

Algorithm	On or Off policy?	Deterministic or Stochastic policy?	Centralised training?
IQL (48)	OFF	Deterministic	NO
QMIX (49)	OFF	Deterministic	YES
SD SAC (50)	OFF	Stochastic	NO
IPPO (46)	ON	Stochastic	NO

The choice of IQL and QMIX allowed for the exploration into the use of centralised training and decentralised execution (CTDE) for MARL tasks as QMIX built upon IQL by introducing a mixer network as seen in Figure 3.4. This enabled information sharing between agents during training through the projection of the per agent Q values to a single joint action-value estimate via a hidden representations whose size depended on the mixer embedding layer size (49). This is on top on the implicit information sharing obtained through parameter sharing achieved through all agents using copies of the same agent policy network and weights that both methods had (48). The weights of the mixer network were generated by a hypernetwork which was a two-layer network, whose size depended on the hypernetwork hidden dimension, that took the global state of the environment as input, allowing the mixing function to be conditioned on the global information (49). However, outside of training, the mixer network was not used and during inference only the agent's policy network was used, hence the decentralised execution (49).

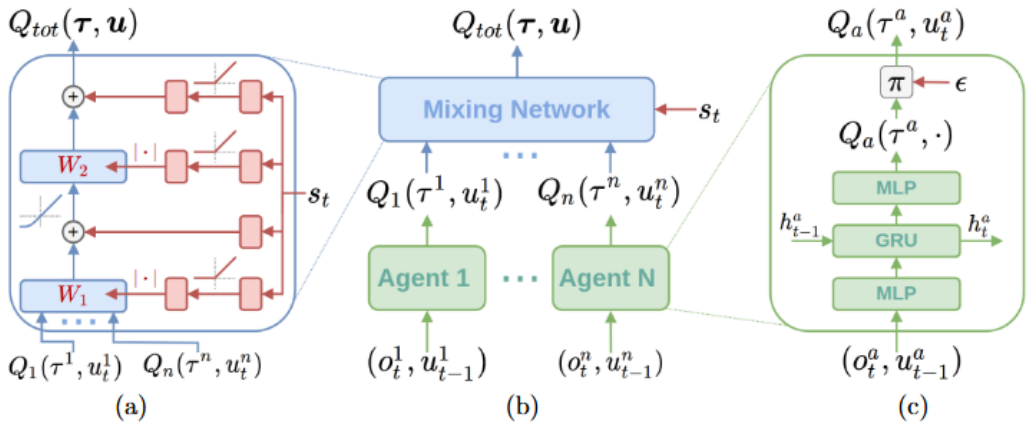


Figure 3.4: Diagram show the structure of the mixing network and agent networks. The hypernetwork for the mixing network is shown in red in the diagram (49).

Since PGA-ME was originally implemented for robotic control, it has only been tested in fully observable environments, whereas SMAX was only partially observable. This meant that a memory mechanism, such as a recurrent layer, was needed for the agent network and experiments performed by (51) and (52) using a long short-term memory (LSTM) and (53) using a gated recurrent unit (GRU) showed increased performance in Partially Observable MDPs (POMDPs). The GRU cell was used for the recurrent layer, as (53) suggested, the GRU-RNN had better convergence than a LSTM-RNN with better performance, however the difference was small. Preliminary testing in SMAX using IPPO with and without a recurrent layer had shown that with a recurrent layer the method was able to learn a useful policy that was able to defeat the heuristic agent in the 2z3s scenario, whereas the method without any memory only learnt to move in one direction. The architecture of the RNN network used by both IQL and QMIX for the agent policy network can be seen in Figure 3.5.

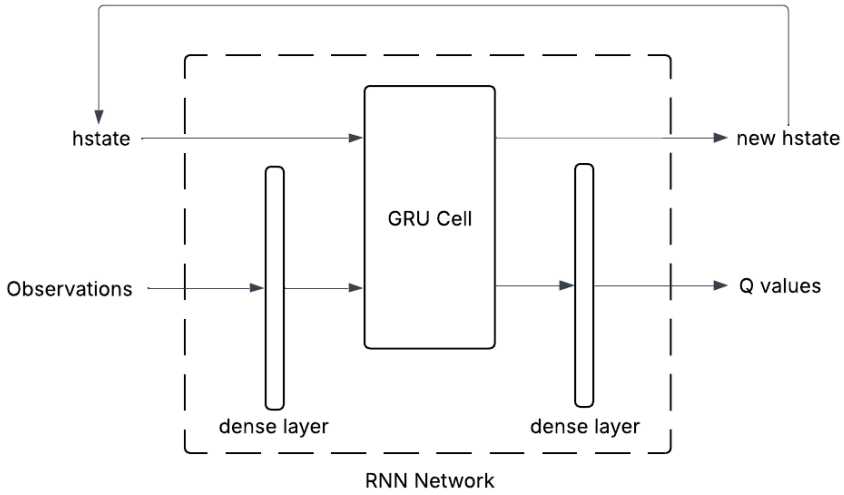


Figure 3.5: Diagram of the RNN network with a GRU cell as the recurrent layer that was used for the agent network for IQL and QMIX, where hstate is the hidden state.

The size of the GRU cell used for the recurrent layer determined how much information, called the hidden state (hstate), was carried between timesteps steps. A hidden state is updated each timestep based on the current observation, and the previous hidden state which is initialised to zero at the start of an episode. Tests were performed using the IQL and QMIX baseline algorithms from JaxMARL with a recurrent layer to determine the most suitable size for the recurrent layer for the SMAX environment. Each algorithm was used to train a policy that were used by the agents to go against a heuristic policy in the 2s3z scenario. During the training process, after every percentage of the max environment steps, which was $1x10^5$ steps for max steps of $1x10^7$, evaluation of the trained policy was conducted by doing rollout using that policy on the same environment. Using Jax it was possible to do parallel rollouts across 512 environments for each evaluation step and the final mean return and win rate across all episodes were recorded. The final mean returns and win rates were plotted against environment steps in Figure 3.6 for IQL and Figure 3.7 for QMIX to observe how performance changed during the training and the final values reached.

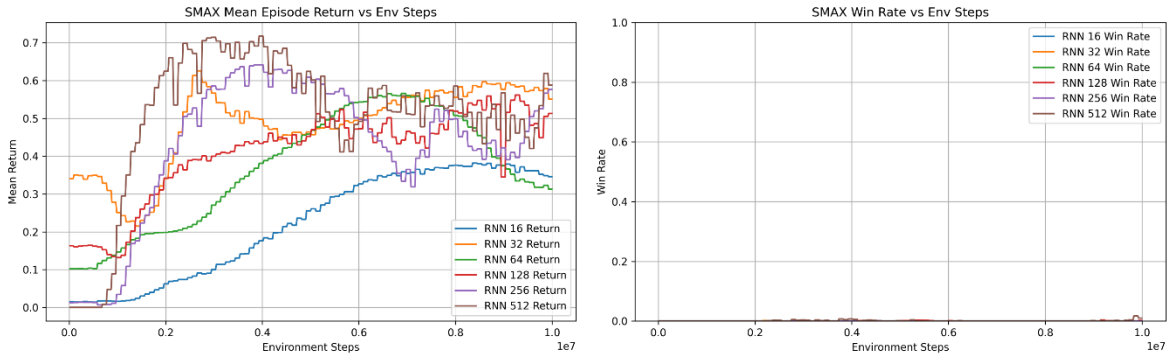


Figure 3.6: Graphs of the final mean returns (left) and the win rates (right) for the IQL algorithm in the 2s3z scenario for $1x10^7$ environment steps.

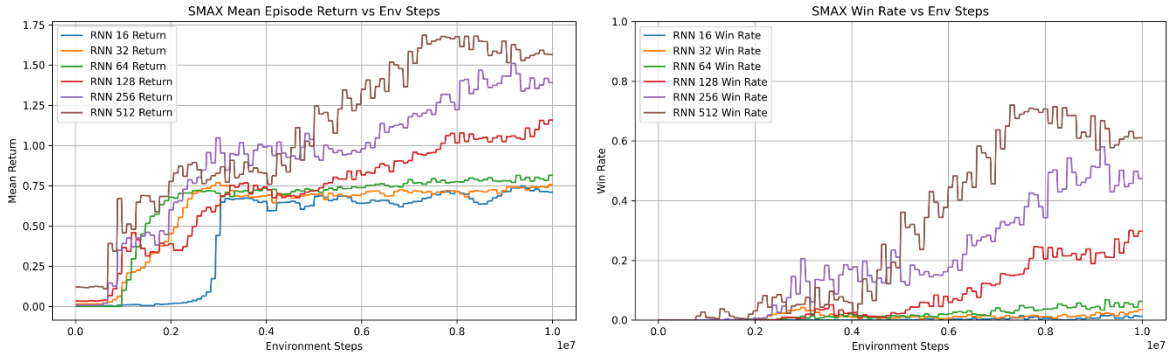


Figure 3.7: Graphs of the final mean returns (left) and the win rates (right) for the QMIX algorithm in the 2s3z scenario for $1x10^7$ environment steps.

Comparing the results from IQL and QMIX in Figure 3.6 and Figure 3.7, QMIX was able to achieve much higher mean returns during training and win rate increased before levelling off. IQL on the other hand struggled and was unable to get a policy that could win. This was not surprising as it showed that the advantage of information sharing during training was able to lead to a better policy as expected. It could also be seen that larger hidden layer sizes for the recurrent layer also resulted in better performance for both IQL and QMIX. The key point from this is that QMIX with a larger recurrent layer size of 512 and 256 was able to reach and surpass the baseline win rate of 50% against the heuristic policy demonstrating that the method learnt something useful. More testing was also performed on the other scenarios for QMIX with different recurrent layer sizes, but this proved to be inconclusive, as it did not score a win rate higher than 50% exposing the limitations of QMIX.

Since QMIX had a better performance than IQL in the 2s3z scenarios, further tests were performed to determine how the size of the mixer network used in QMIX affected the performance of the policies learnt. Different combinations of embedding layer size and hypernetwork layer size were tested, where the embedding layer controlled the capacity of the mixing network itself, and the hypernetwork hidden layer determined how expressive the weights of the mixing function were depending on the global state of the environment (49). The final mean return can be seen in Figure 3.8 and the win rate across the 516 parallel evaluation environments can be seen in Figure 3.9.

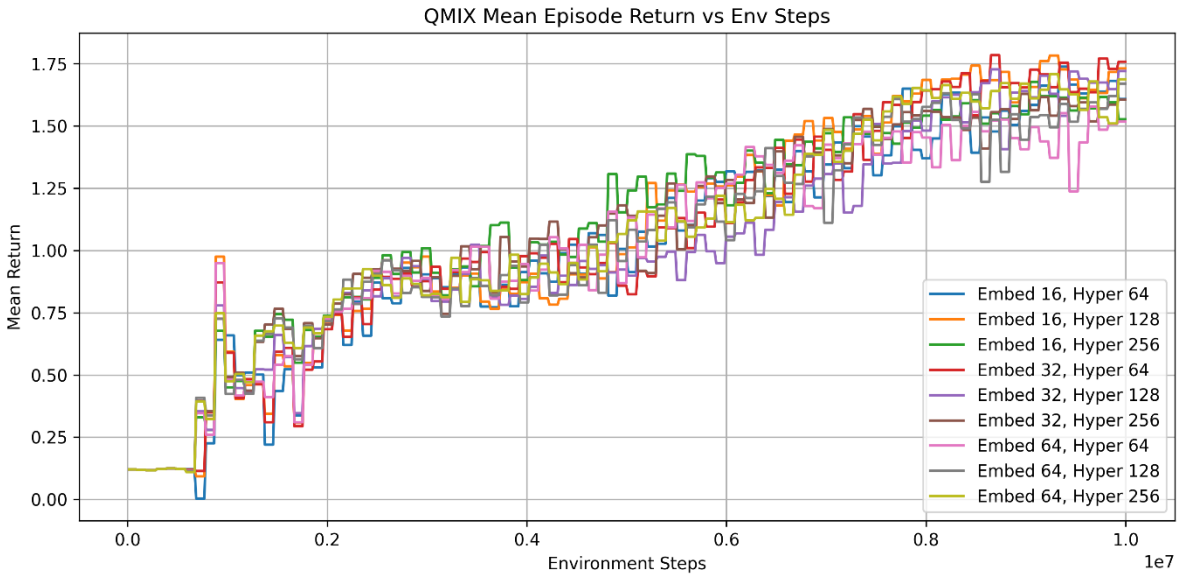


Figure 3.8: Graph of the final mean returns for different embedding layer size and hypernetwork hidden layer size for the QMIX algorithm in the 2s3z scenario for $1x10^7$ environment steps.

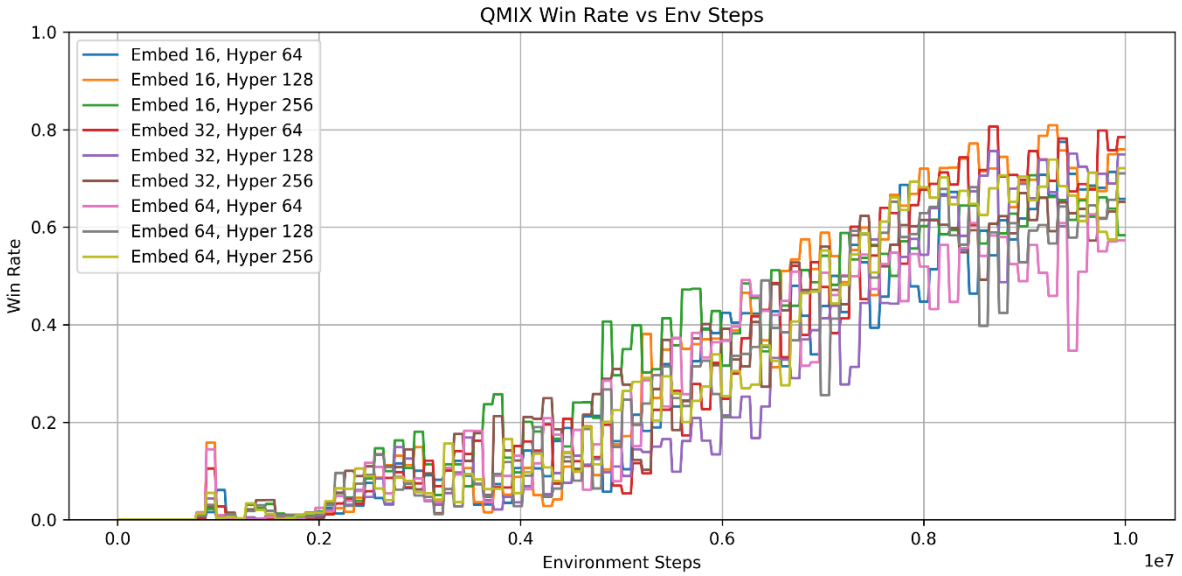


Figure 3.9: Graph of the win rates for different embedding layer size and hypernetwork hidden layer size for the QMIX algorithm in the 2s3z scenario for $1x10^7$ environment steps.

The results from the grid search suggested that, for the 2s3z scenario, varying the embedding layer size of the mixing network and the hidden layer size of the hypernetworks had little impact on overall performance. This implied that the performances gains of QMIX relative to IQL primarily came from the inclusion of a mixer network and its ability to combine the per agent Q values to a single joint action value during training. The lack of increased performance with larger sizes for the mixer network also suggested, that for the 2s3z scenario, a smaller mixer network was sufficient for this function. Tests on another scenario called 5m_vs_6m also yielded the same results supporting this hypothesis.

To summarise, the reinforcement algorithms IQL and QMIX were chosen initially for their properties of being off policy for use with PGA-ME and being able to handle discrete actions for the SMAX environment. Memory architecture, in the form of a GRU cell as the recurrent layer, was added to handle that partial observability of SMAX. QMIX had better performance than IQL and this was due to its CTDE nature, so agents could share information during training to aid the learning of the agent policy. It was also discovered, that for the 2s3z scenario, a smaller mixer network for QMIX was able to sufficiently get the joint action representation and increasing the size of it did not increase the performance. Testing with different sized recurrent layer also revealed that larger sized recurrent layers resulted in better performance.

3.5. Quality and Diversity (QD) algorithm

PGA-ME required the use of the MAP Elites quality and diversity algorithm. This required the design of a scoring function used during evaluation and the Behavioural Descriptors (BDs) which described the features of the solutions. The BDs determined where the diversity was wanted and the behaviour space where the solutions would occupy. The archive or repertoire that was used to store the solutions could have been grid based, where the behaviour space was broken down into fixed intervals to get square cells, or Centroidal Voronoi Tessellation (CVT) based where the cells were determined by equally spaced centroids in the behaviour space. A grid repertoire was chosen that could be individually indexed from 0 to 99, as shown in Figure 3.10. Functions were created that could return indexes of solutions based on its fitness and getting the index of fittest solution within a range of a specific descriptor. Each cell in the repertoire stored the weights for the networks along with the fitness scored and descriptors calculated during the evaluation step.

90	91	92	93	94	95	96	97	98	99
80	81	82	83	84	85	86	87	88	89
70	71	72	73	74	75	76	77	78	79
60	61	62	63	64	65	66	67	68	69
50	51	52	53	54	55	56	57	58	59
40	41	42	43	44	45	46	47	48	49
30	31	32	33	34	35	36	37	38	39
20	21	22	23	24	25	26	27	28	29
10	11	12	13	14	15	16	17	18	19
0	1	2	3	4	5	6	7	8	9

Figure 3.10: Visualisation of how each cell in the 100 cells grid-based repertoire was indexed

Problems were encountered during the design of the scoring function when the final return was used due to the stochastic nature of the SMAX environment. It resulted in a final repertoire that contained unreliable solutions as some had got lucky which a specific Jax PRNG key used during the evaluation for an instance of the SMAX environment. During visualisation of the policy when

another key was used, the policy performed much worse than what the fitness stored in the repertoire indicated. To mitigate this, each offspring was evaluated across several parallel evaluation environments each with a different key, and the fitness was the mean final return across all the parallel environments. A variance term was also added to penalise high variance in final returns across the parallel environment so that better solutions were ones that were more consistent.

Another term was added that rewarded solutions with more ally units alive at the end of the episode to further differentiate between better performing solutions. Therefore, solutions that were unable to beat the opponent would not get this reward, but those that did get an added score based on the ratio of number of alive ally units to the total of ally units across all parallel environments. The designed scoring function can be seen in Equation 3.2.

$$f = r_{mean} - r_{variance} + \frac{n_{alive}}{n_{total}} \tag{3.2}$$

where:

- r_{mean} = mean final return across parallel environments
- $r_{variance}$ = variance of final returns across parallel environments
- n_{alive} = number of alive ally units across parallel environments
- n_{total} = total ally units across parallel environments

The theoretical max score achievable was 3.0, obtained by achieving the max mean final return of 2.0 with no variance in fitness across the parallel environments and all ally agents kept alive by the end of an episode to get a bonus score of 1.0.

The Behavioural Descriptor was designed such that it could quantify the qualitative features that describe a playstyle such as unit formations and techniques such as flanking. The devised BDs, as seen in Equation 3.3, were the percentage of actions being a movement action and the distance between units. A high movement action percentage could indicate a policy that prioritised moving units more often to get a positional advantage whereas a low movement percentage could indicate a policy being highly aggressive focused on attacking. Differing distances between units could also be a sign of different formation or grouping of units to get a tactical advantage.

$$BD = [a_{movement}\%,\ \ d_{allies}] \tag{3.3}$$

Equation 3.4 and Equation 3.5 shows the calculation to get the BDs, movement action percentage and normalised mean distance between allies.

$$a_{movement}\% = movement\ action\ percentage = \frac{total\ movement\ actions}{total\ actions} \tag{3.4}$$

$$d_{allies} = normalised\ mean\ distance\ between\ allies = \frac{mean\ distance\ between\ ally\ units}{0.25 * environment\ length} \tag{3.5}$$

The distance BD required some engineering to get a value between 0 and 1 that could suitably summarise the distance between each unit for a policy rollout. It was calculated by averaging the distance between ally unit across all time steps and normalising it against a quarter of the max length of the environment which was 32. The reason for normalising against a quarter of the max

length instead of the max length was that normalising against the max length compressed the solutions into a very small range as it was unlikely that agents would be spaced on average an environment length apart. This would have resulted in solutions competing over a small region in behaviour space which reduced the diversity.

The genetic variation emitter used to generate the random offspring each iteration for the MAP Elites algorithm was the Iso+LineDD variation operator from (54) which consisted of two types of mutations the isotropic mutation and line mutation. The operator can be expressed as the equation seen in Equation 3.6.

$$x_{new} = x_i + \sigma_1 \cdot \mathcal{N}(0, I) + \sigma_2 \cdot (x_j - x_i) \quad 3.6$$

where:

- x_{new} = new mutated solution
- x_i = the initially selected solution from repertoire to be mutated
- x_j = the second selected solution from repertoire used for the line mutation
- σ_1 = Iso sigma
- σ_2 = line sigma
- $\mathcal{N}(0, I)$ = Gaussian noise

The isotropic mutation applied gaussian noise to a sampled solutions from the repertoire in all directions to create a distribution and its aim was to control the diversity of the new solution. A larger value of iso sigma would increase the noise applied to create the distribution and increase diversity. The line mutation created a distribution by adding perturbations along a line connecting the current solution being mutated and another solution sampled from the repertoire. The aim of line sigma was to exploit the correlation between high performing solutions in the repertoire and prevent the new solution from straying too far from the region of high performance. A larger line sigma would stretch the line distribution closer to the second sampled solution. The two distributions were then combined to create an Iso+LineDD distribution which can be seen in Figure 3.11 that was then sampled from to get the new mutated offspring.

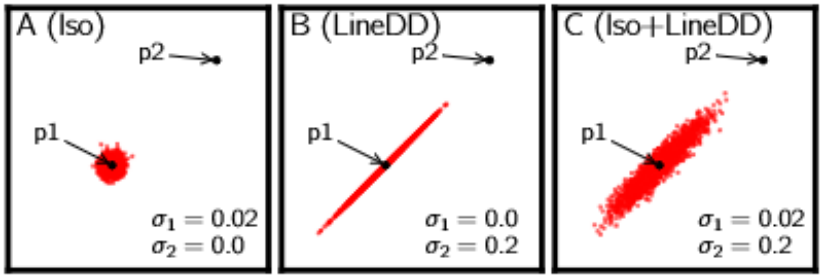
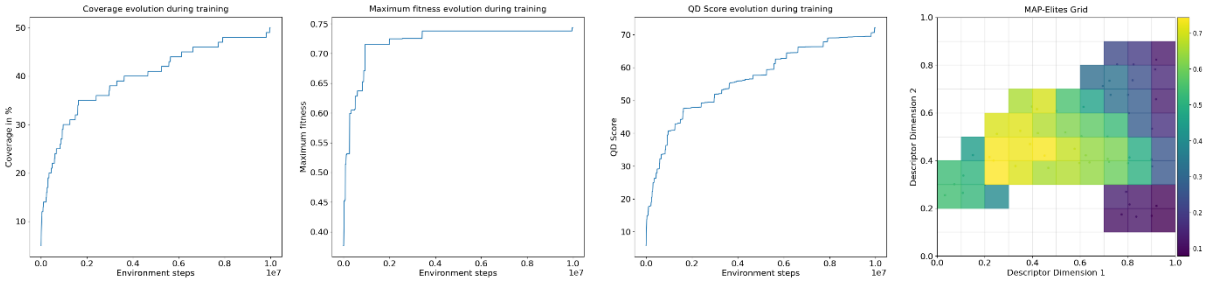


Figure 3.11: Shows the isotropic gaussian distribution (A) controlled by the iso sigma σ_1 and the line distribution (B) controlled by line sigma σ_2. The resulting distribution form combining the two can be seen in (C) which allows samples to explore the region around the first sampled solution from the repertoire p1 and correlation between this and another sampled solution from the repertoire p2 (54).

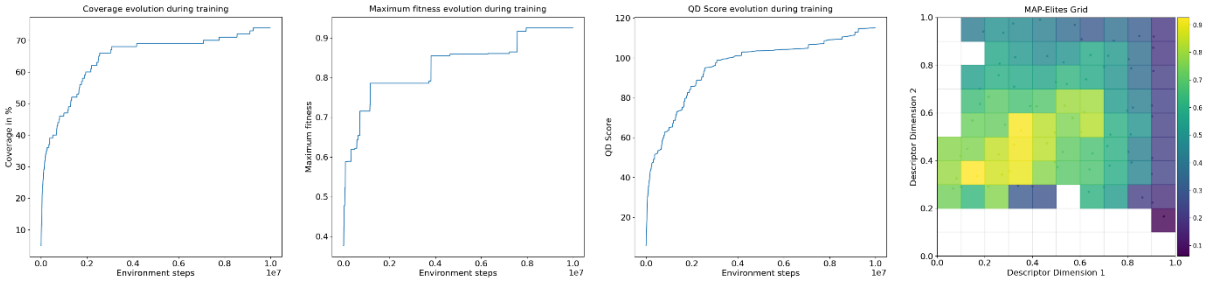
A range of iso and line values were tested to see what values enabled the best exploration without sacrificing performance. The plot for the coverage, max fitness and QD score of the final repertoire for some of the value combinations can be seen in Figure 3.12 evaluated under the 2s3z scenario in SMAX for $1x10^7$ environment steps. Metrics used for evaluation include QD-score (sum of total fitness in the archive), coverage (percentage of non-empty cells in the

repertoire), and max fitness of single best performing solution. These enabled the measure of performance and diversity of the solutions in the final repertoire. Heat maps of the MAP-Elites repertoires were also plotted in Figure 3.12 showing the fitness of the solution according to the BDs of the solution in behaviour space.

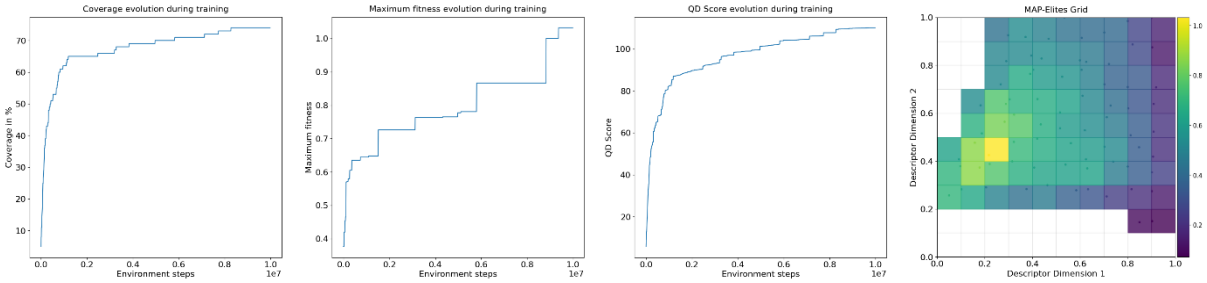
(a) Iso sigma = 0.002, Line sigma = 0.02



(b) Iso sigma = 0.005, Line sigma = 0.05



(c) Iso sigma = 0.01, Line sigma = 0.1



(d) Iso sigma = 0.02, Line sigma = 0.2

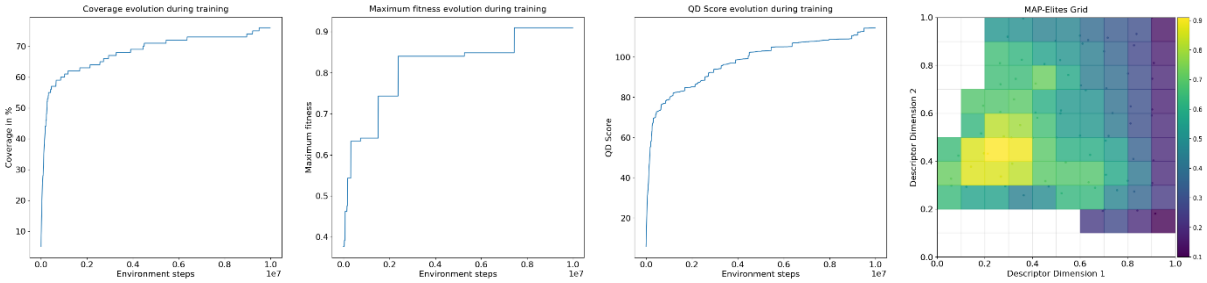


Figure 3.12: Plots for the coverage, max fitness and QD score of the MAP Elites algorithm ran for $1x10^7$ environment steps with different iso sigma and line sigma values for the Iso+LineDD variation operator in the 2s3z scenario. The heat map shows the solutions in behaviour space with the colours indicating fitness of the solution.

Table 3.4 summarised the final values achieved for all iso sigma and line sigma values tested. It was determined that the best values were 0.005 for iso and 0.05 for line. The value pair (0.01, 0.1) came close with the same coverage and higher max fitness, but it had a lower QD score than that of (0.005, 0.05) for iso and line, which was more important than max fitness as the aim was more solutions with higher performance.

Table 3.4: Table of repertoire coverage, max fitness and QD score for different iso sigma and line sigma combinations for the iso+lineDD varitation opertator.

Iso sigma	Line sigma	Coverage %	Max fitness	QD score
0.001	0.01	40	0.719208	52.71672
0.002	0.02	50	0.743392	72.14767
0.005	0.05	74	0.92585	115.1396
0.01	0.1	74	1.03185	110.1063
0.02	0.2	76	0.909567	114.548
0.05	0.5	68	0.870233	97.11987

To summarise, MAP Elites was used as the quality and diversity algorithm for the final design. A scoring function was designed to account for the highly stochastic SMAX environment and BDs were designed that could be used to quantify the qualitative aspects of different playstyles and made and where diversity was wanted. The Iso+LineDD variation operator was used as the genetic variation emitter in MAP Elites. After testing with different iso sigma and line sigma values, it was determined that an iso sigma of 0.005 and line sigma of 0.05 produced the best QD score with a high coverage in the 2s3z scenario in SMAX.

4. Evaluation

This chapter details the modifications needed to combine two parts of QD and RL to get the final design and the tests were conducted to justify certain design aspects of the final design. This is then followed by a discussion of the results obtained for the final design with a quantitative evaluation followed by a qualitative evaluation of the solutions in the final repertoire generated by the algorithm to determine whether there was evidence for unique playstyles.

4.1. Final Design: Combining QD and RL

To combine QMIX with MAP Elites to get the proposed PGA ME QMIX algorithm, a QMIX based policy gradient emitter was designed. This involved forgoing the epsilon greedy exploration used in the QMIX algorithm and instead relying on the randomly generated offsprings from the Iso+LineDD variation operator to generate diverse episodes for the replay buffer and exploration. Each solution consisted of weights for the shared agent policy network and the mixer network. The reason for each solution had its own mixer, instead of one global mixer network shared by all solutions, was that it allowed for more diversity since each mixer network for each solution could mix the agent Q values differently instead of being forced to mix the per agent Q values in a particular way.

Another change was that a soft update to the network weights was used instead of a hard update that was used during previous testing for QMIX. In the original QMIX implementation a target network was hard updated every set number of training step, the target network update interval, where each training step had an epoch which was number of gradient steps applied to the online network before updating the target network. This meant that the number of training steps for every solution that was generated by PGA ME QMIX had to be kept track off which was difficult to do and took up resources that were limited and better spent elsewhere such as the replay buffer. The soft update approach also reduced the number of gradient steps needed by a factor of the target update interval for one target network update reducing the time taken for the algorithm. There were also the benefit of smoother updates helping the solutions converge more reliably. Equation 4.1 shows the equation for a soft update.

$$\theta_{target} = \tau\theta + (1 - \tau)\theta_{target} \tag{4.1}$$

where:

- θ_{target} = target network weights
- θ = online target weights
- τ = tau, fraction of online weight used to update the target weights

Different value combinations of tau and epoch for the soft update with QMIX were tested, where epoch referred to the number of full policy updates to the online network before a soft update is made to the target network. During the tests, the target update interval was fixed at a value of 1

meaning that every time the online network was updated, the target network was immediately updated. From Figure 4.1 for the mean return and Figure 4.2 for the win rate, the value pairs (0.1, 10) and (0.05, 20) for tau and epoch performed best within $1x10^7$ environment steps where evaluation was done across 512 parallel environments for the 2s3z scenario. Therefore, it was decided that tau was 0.1 and epoch was 10 for the final algorithm as that required fewer gradient steps allowing the algorithm to run faster and use less energy.

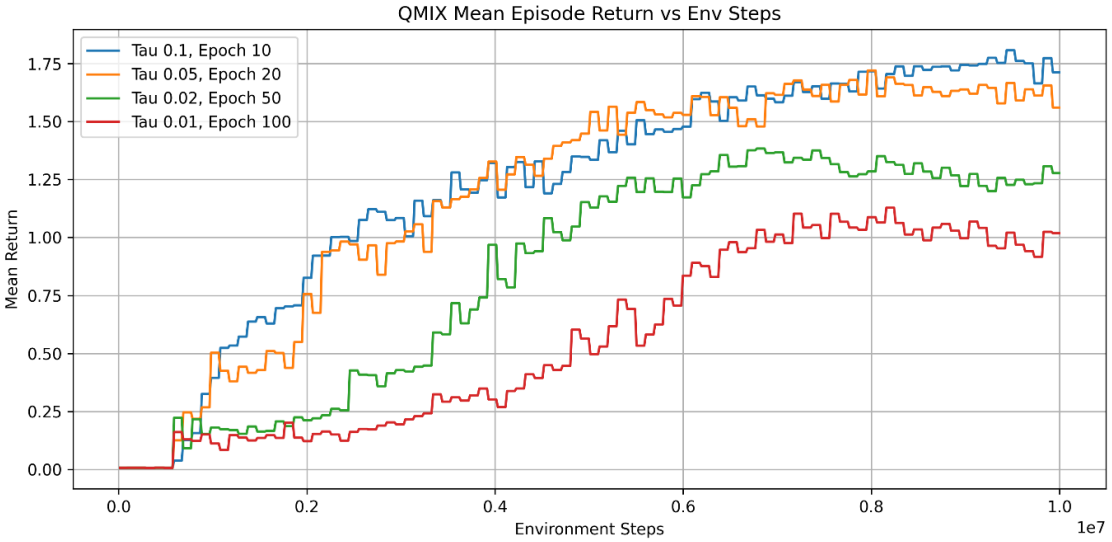


Figure 4.1: Graph the mean returns of different combinations of tau and epoch value for the soft update used for the QMIX algorithm in the 2s3z scenario for $1x10^7$ environment steps.

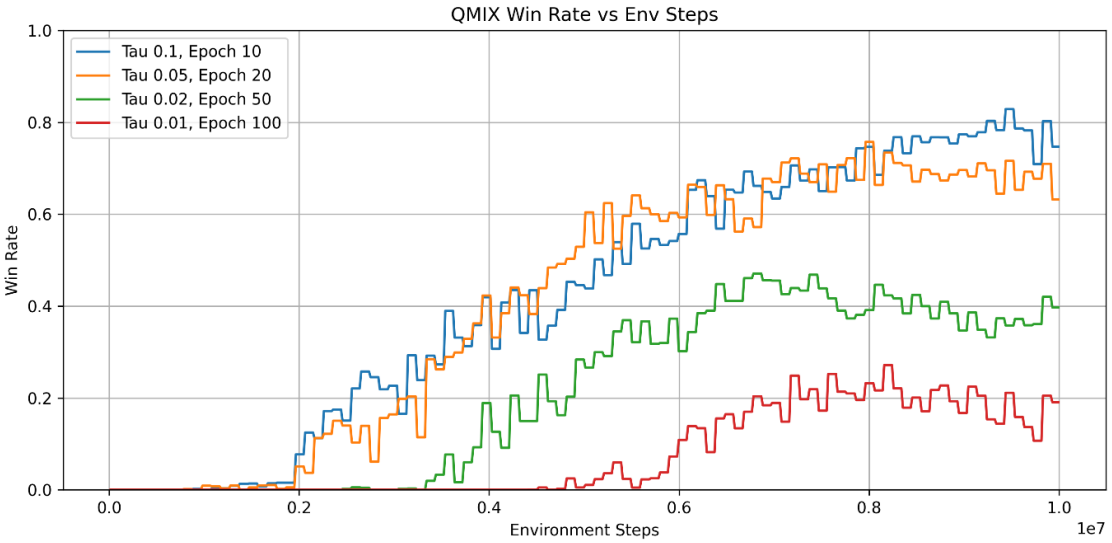


Figure 4.2: Graph of the win rate of different combinations of tau and epoch value for the soft update used for the QMIX algorithm in the 2s3z scenario for $1x10^7$ environment steps.

Using the new soft update, updated tests were performed for different sizes of recurrent layer for QMIX and from Figure 4.3 and Figure 4.4 it showed that the soft update had given better performance for all recurrent layer sizes when compared to hard updates in Figure 3.7. It was also interesting to note that the gap in performance between the recurrent layer sizes 256 and 512

decreased using soft update with both achieving similar performance. This was perhaps due to the smoother updates allowing for faster convergence of the recurrent layer with size 256, so that at least up to $1x10^7$ environment steps it achieved comparable performance to that of size 512. However, it could be that beyond $1x10^7$ steps performance with size 512 increases more than that of 256 as it has more capacity to learn but this was not tested. The potential with a lower recurrent layer size proved to be helpful as it freed up resources for other parts of the algorithm under the constraints of 16GB of VRAM.

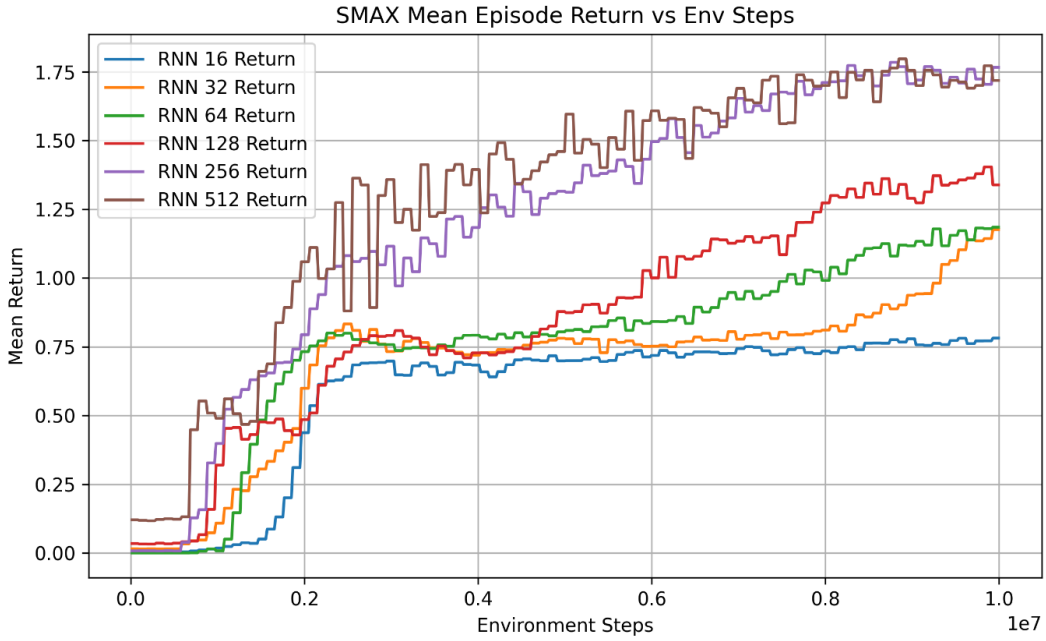


Figure 4.3: Graph of final mean return for different recurrent layer sizes for the QMIX algorithm with soft updates in the 2s3z scenario for $1x10^7$ environment steps.

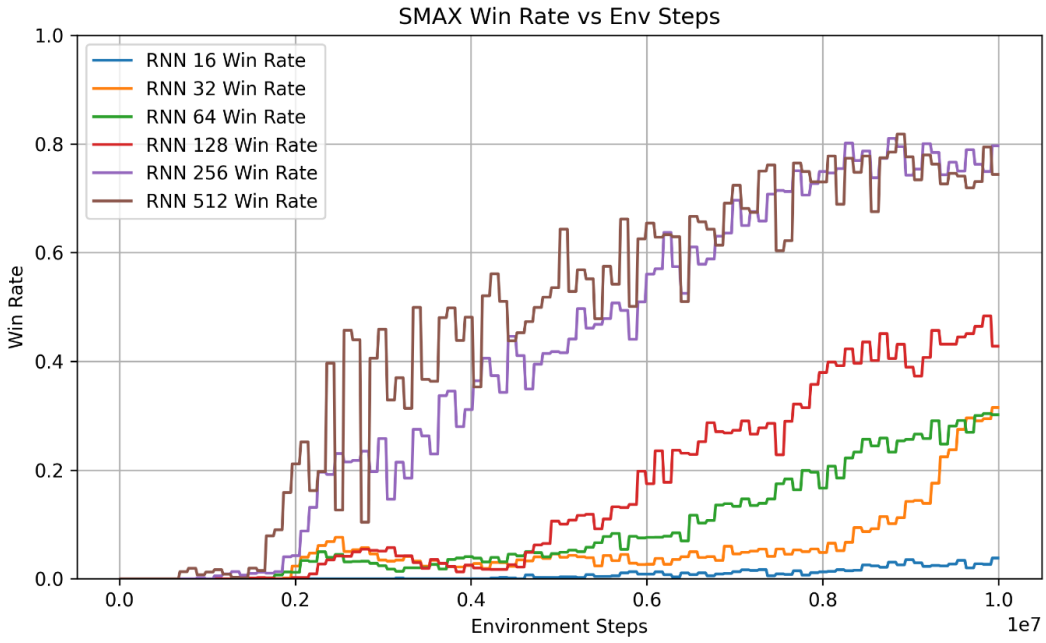


Figure 4.4: Graph of win rate for different recurrent layer sizes for the QMIX algorithm with soft updates in the 2s3z scenario for $1x10^7$ environment steps.

Different mixer embedding sizes and hypernetwork hidden sizes were also tested but performed similarly as in the case with a hard update shown previously in Figure 3.8 and Figure 3.9 with smaller mixer sizes having a negligible better performance that larger mixer sizes.

To summarise, the QMIX algorithm was adapted to work as the policy gradient emitter in the MAP Elites algorithm for the final algorithm and this required several changes to it, which involved removing the epsilon greedy exploration and switching to a soft update. Different tau and epoch values were tested for the soft update, and it was determined that values of 0.1 for tau and 10 for epoch resulted in the best performance. These values were then used to determine the best values for the recurrent layer size which was 256 and mixer network size which was 16 for embedding layer size and 64 for the hidden layer of the hypernetwork.

4.2. Results

The final algorithm was tested on the 2s3z scenario against a heuristic policy with a recurrent layer size of 256 for the agent network alongside a mixer network with embedding size of 16 and hypernet hidden size of 64. The QMIX policy gradient emitter used soft updates with a tau value of 0.1 and epoch of 10 and each iteration to refine the sampled solution.

To evaluate the final algorithm, a run was performed for $1x10^8$ environment steps with 16 parallel environments for scoring with max episode length of 100. This resulted in a total of 12500 iterations of the MAP Elites algorithm. Each iteration 5 offspring were produced, also called the batch size, with three from the QMIX emitter and two from the Iso+LineDD variation operator. The results for the run can be seen in Figure 4.6 and Table 4.1.

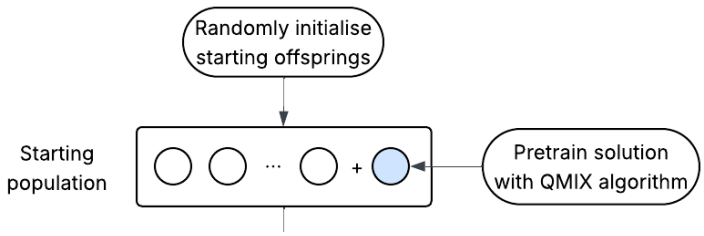


Figure 4.5: Diagram showing the seeding of a pretrained solution into the initial starting population.

Another run was conducted where seeding was used, as shown in Figure 4.5. Seeding referred to the process of adding an initial good solution, trained for $1x10^7$ environment steps, to the randomly initialised solutions at the start of the MAP Elites algorithm was used to try and improve the overall fitness of the solutions in the repertoire. The plots for the repertoire coverage, max fitness and QD score can be seen in Figure 4.7 and Table 4.2 for the run with seeding.

Table 4.1: Table summarising the final coverage, max fitness and QD score achieved from the designed algorithm without seeding.

Coverage %	Max fitness	QD score
70	1.63	160.88

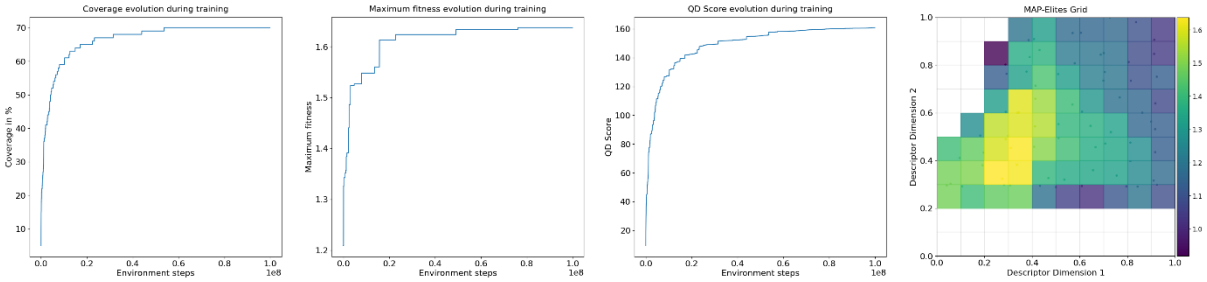


Figure 4.6: Graphs of coverage, max fitness and QD score for the final repertoire generated from the designed algorithm without seeding ran for $1x10^7$ environment steps in the 2s3z scenario. The heat map shows the solutions in behaviour space with the colours indicating fitness of the solution.

Table 4.2: Table summarising the final coverage, max fitness and QD score achieved from the designed algorithm with seeding.

Coverage %	Max fitness	QD score
62	2.88	157.28

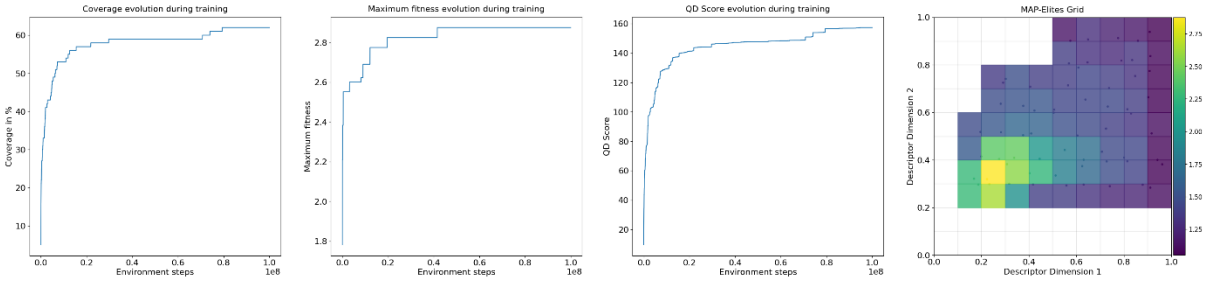


Figure 4.7: Graphs of coverage, max fitness and QD score for the final repertoire generated from the designed algorithm with seeding ran for $1x10^7$ environment steps in the 2s3z scenario. The heat map shows the solutions in behaviour space with the colours indicating fitness of the solution.

Due to the VRAM constraint, only 16 parallel environments could be used during the scoring of the offsprings during the MAP Elites algorithm. To get a more reliable measure of the performance for the solutions in the repertoire from the run with seeding, solutions that achieved a fitness higher than 2.0 were taken and reevaluated at the end of the algorithm to get their win rate for one episode across 512 parallel environments.

Table 4.3: Table containing all the solutions with a fitness higher that 2.0 from the repertoire generated by the algorithm with seeding along with their reperotiore index, BDs and win rate achieved in the reevaluation.

Repertoire Index	BD	Fitness	Win rate %
21	[0.188, 0.297]	2.28	49.2
22	[0.234, 0.299]	2.63	72.9
31	[0.169, 0.321]	2.27	40.6
32	[0.224, 0.319]	2.88	74.4
33	[0.308, 0.382]	2.56	44.3
34	[0.403, 0.344]	2.25	28.9
42	[0.275, 0.404]	2.41	50.8
43	[0.338, 0.409]	2.44	40.2
44	[0.446, 0.410]	2.02	30.9

The solutions in Table 4.3 were then all visualised for one episode in the SMAX environment with the same Jax PRNG key to qualitatively evaluate them and determine whether any unique playstyles were present among the best performing solutions. The reason not all solutions in the repertoire were visualised, was that due to their low fitness, there was low confidence when determining their playstyle, whether an action taken by the policy was deliberate or random.

4.3. Discussion

Comparing the results in Figure 4.7 and Table 4.2 those in Figure 4.6 and Table 4.1, it was noted that the addition of seeding with pretrained solution resulted in significantly better performance achieving a max fitness of 2.88 compared to 1.63 with little sacrifice to coverage in the repertoire getting 62% instead of 70% when compared to the run without seeding. Although the coverage was lower for the case with seeding being used, a larger percentage of the solutions are of a higher fitness with nine achieving a fitness above 2.0 and none in the case with no seeding. This was also be evidenced through the QD score of 157.28 for the run with seeding being quite close to that of 160.88 for the run without seeding despite the lower coverage demonstrating that there was a smaller population of fitter solutions.

It was speculated that the reason the solutions were not as good as expected without seeding, and hence why seeding was used, was that the QMIX algorithm required many training steps to get a good solution and this was likely due to the highly stochastic environment. Relying on the MAP Elites algorithm to train a solution entirely was difficult as it was unlikely to get a good solution in one iteration and each iteration random solutions were chosen to be refined and with a large repertoire it was unlikely a particular solution would be consistently chosen. Therefore, inspiration was taken from AlphaStar (1) where imitation learning was used to get initially good solutions before using RL algorithms. Hence, a similar approach was taken where a good solution was trained and then seeded among the other randomly initialised solutions at the start of the MAP Elites algorithm to bypass the difficulties encountered in the early training of the solutions.

However, the revaluation of the top performing solution in Table 4.3 showed that even though the fitness score may seem high their reevaluated performance was not as strong. The worst

reevaluated solution only achieving a win rate of 28.9% against a heuristic policy which was lower than the 50% baseline. This result demonstrated the difficulty of training a good solution for the SMAX environment using the QMIX emitter and suggests that a better reinforcement algorithm was needed. Running the algorithm on the more difficult 5m_vs_6m scenario also showed bad performance even with seeding as the pretrained solution using the same $1x10^7$ environment steps was unable to achieve a win rate greater than the baseline of 50%. An alternative explanation could be that the solutions are still undertrained, and more iterations of the MAP Elites algorithm were needed.

Regardless, to determine whether distinct playstyles were learnt using the algorithm all solution with a fitness greater than 2.0 for the run with seeding, shown in Table 4.3 were visualised using the same Jax PRNG key for the environment to see if they performed differently. Doing this revealed that the solutions could be broadly classified into several different playstyles, however there were still characteristics shared among the different playstyles that ensured they performed well. The solutions were visualised multiple times with different keys and all solutions exhibited consistent behaviour across the keys, however the less fit solutions with lower win rates in Table 4.3 were unable to consistently win with that behaviour.

From the visualisations, most solutions learnt to focus damage dealt on one or two enemy units, instead of spreading the damage out across more enemy units, to quickly take down an enemy unit and reduce the damage potential of the enemy. This ensured that more ally units were kept alive. Some of the solutions also learnt to group units together based on their unit type and this allowed them to get into formation to execute certain playstyle. Most playstyles learned to retreat certain units when their health started getting low to reset the targeting of the enemy to another ally unit keeping the retreating ally units alive for longer so that they could do more damage. These behaviours seem to have been inherited from the seeded solution suggesting that these were good tactics that the offsprings decided to keep them.

The unique playstyles identified were named: Flanking melee unit, Flanking ranged unit, Single unit tanking, Mobile ranged unit and Full aggression. Table 4.4 summaries the playstyles with a short description and logs the repertoire indexes of the solutions that exhibited these playstyles.

Table 4.4: Table summarising the playstyles observed along with a description and the repertoire indexes of the fittest solutions from the reperoire generated by the algorithm with seeding that achieved them.

Playstyle	**Description**	**Repertoire Indexes**
Flanking melee unit	Takes a melee unit and manoeuvres it towards the back of the enemy formation	21
Flanking ranged unit	Takes a melee unit and manoeuvres it towards the back of the enemy formation	33, 34, 42, 44
Single unit tanking	Separates out a single unit to engage the enemy and draw attention away from other ally units	43
Mobile ranged units	Consistently moves the ranged unit around weaving in attacks between movement actions	22, 32
Full aggression	Charges all units into the enemy and continuously attacks enemy with very few movement actions	31

Looking at the solution with the highest win rates in Table 4.3, both of the highest performing solutions, indexed 22 and 32, have the same playstyle of Mobile ranged unit suggesting that, at least when playing against a heuristic policy in the 2s3z scenario, it is the best policy. This was likely due to keeping the ranged units mobile allowed it to take advantage of the heuristic policy changing targets when their target exited their attack range. The policy would draw attention of the enemy units to the ranged units and then when their health gets low it would retreat while dealing damage to maximise the damage dealt. Then, when they were out of enemy range and the enemy switched target, they reengaged into combat. This tactic worked more effectively with a ranged unit due to their larger attack range hence why a similar playstyle that involved melee units was not learnt. To do this strategy with melee units required a much finer control of units due to the lower attack range that was much harder to learn.

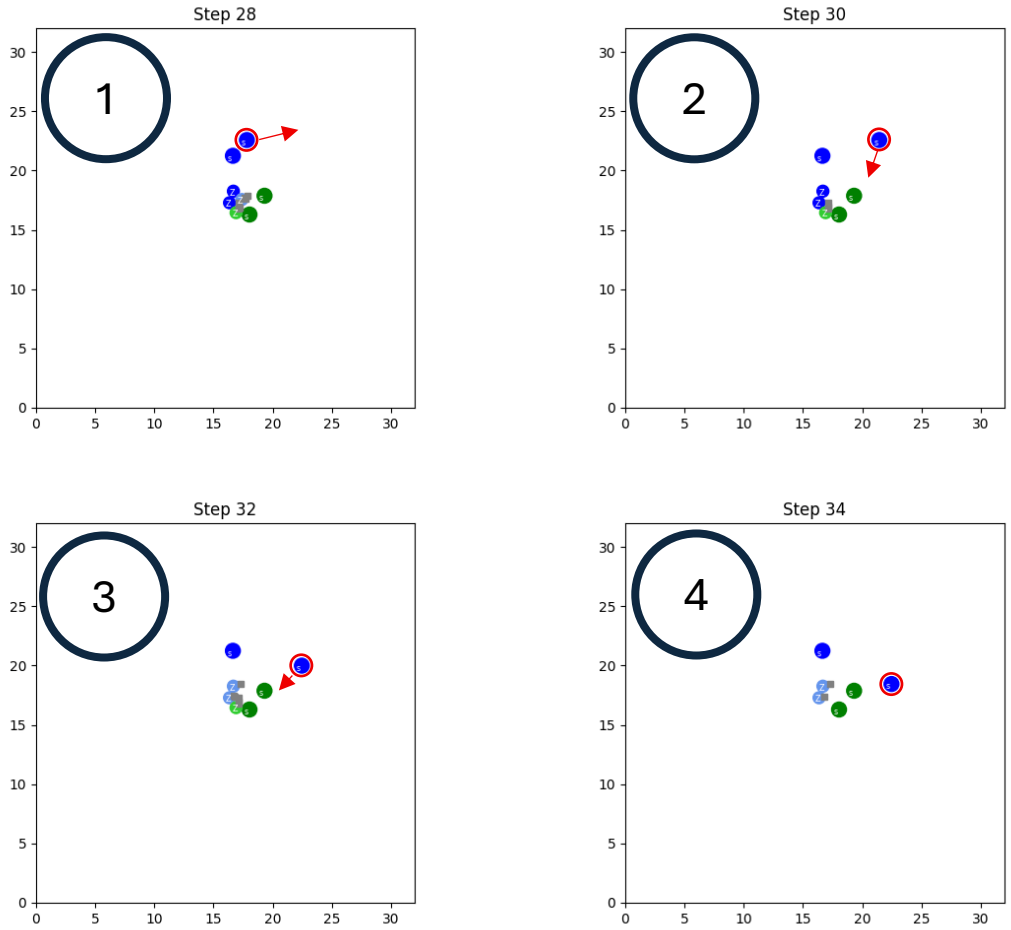


Figure 4.8: Image sequence (1 to 4) showing the flanking manoeuvre in red arrows with a ranged unit, highlighted in red, from a visualisation of the solution indexed 34.

The Flanking ranged unit playstyle also had multiple solutions with varying BDs for each solution, implying that the playstyle was easy to learn. A demonstration of the playstyle can be seen in the image sequence in Figure 4.8. A melee unit version of the playstyle was also learnt by only one solution which reinforced the idea that better control of melee units was more difficult to learn. A demonstration of the playstyle can be seen in Figure 4.9. The advantage of the flanking playstyle was that it allowed the ally unit to reach behind the enemy formation and attack the enemy ranged

units which are much more powerful having a higher attack power as shown in Table 3.1. Removing them early reduced the damage potential and ensured the teams survival.

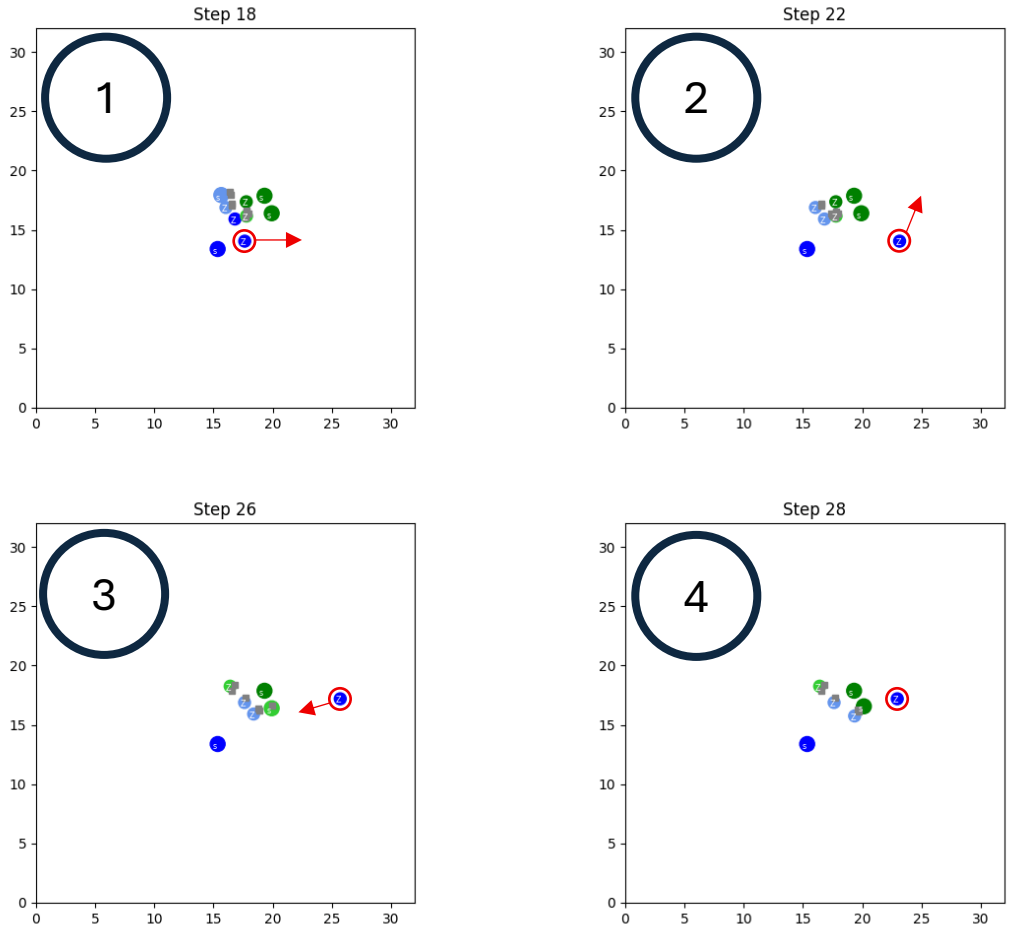


Figure 4.9: Image sequence (1 to 4) showing the flanking manoeuvre in red arrows with a ranged unit, highlighted in red, from a visualisation of the solution indexed 21.

The Single unit tanking playstyle was quite interesting as it diverged in behaviour very early into the episode. Unlike the other playstyles that kept all the ally units bunched up in the early steps of the episode. This playstyle separated out one ally unit and kept the rest together and engages with the single unit first. This behaviour allowed it to take advantage of the heuristic policy always attacking the first target it sees so the single unit could draw all attention towards itself and minimise the damage the other ally unit took. The remaining ally units could then follow up and attack the enemy while they were distracted. Figure 4.10 demonstrates the separation of the single ally unit from the rest of the ally units created at the start of an episode. The success of this tactic depended on whether it was then able to retreat this ranged unit without it being killed so that it could continue to participate in doing damage to the enemy.

The final playstyle was the Full aggression playstyle which was very simple having the ally units focussed on only attacking to maximise the damage dealt to the enemy. This behaviour was likely driven by the reward function of the SMAX environment where it got immediate rewards for doing damage to enemy health points.

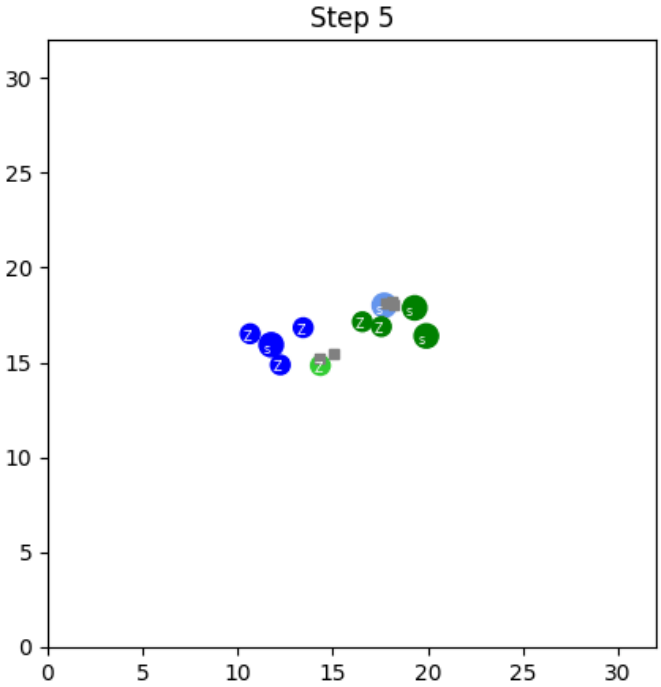


Figure 4.10: Screenshot showing the gap between the single unit used to tank the enemy attack and the rest of the allies at the start of an episode for solution indexed 43.

Even though distinct playstyles were achieved, the amount of diverse playstyle was not as high as the coverage of the repertoire suggested, as many solutions were excluded due to low fitness. From the nine fittest solutions shown in Table 4.3 there were only five distinct playstyles. A potential explanation was that the descriptors were not expressive enough to encourage different playstyles, and the BDs may not necessarily line up with the playstyles. Some solutions received a high movement percentage by have units constantly move forward and backwards on the spot which was not a desired behaviour. This observation demonstrated the need to design better BDs to represent solutions. This misalignment in playstyle can be also seen in the many solutions that have different BDs that sharing the same Flanking ranged unit playstyle in Table 4.4. There is also the problem of subjectiveness to what is considered a playstyle as someone may consider a certain action being indicative of a new playstyle while another may not. For example, someone may class both flanking playstyles as the same, but another might not. Hence, there might be more playstyles the five named one from Table 4.4 present than what is initially suggested based on the criteria used.

Another problem was that the fittest solutions had low diversity in terms of BD as they were clustered around the seeded pretrained solution which seemed to suggest that they were slight permutations of that seeded solution that have been refined. This was a problem that can be encounter by PGA-ME and addressed by DCRL which tries to force diversity my making the solutions conditioned on the descriptor preventing the solutions from collapsing into a single solution (20). However, this would require a RL algorithm that implements an actor and critic and not a Q-network based like QMIX.

During the design of the final PGA-ME QMIX algorithm there were significant challenges with regards to VRAM resource available from the GPU. The used GPU was either an NVIDIA A16 or

NVIDIA Tesla T4 which both had 16GB of VRAM that was very quickly used up. This meant that significant sacrifice was made with regards to the size of the replay buffer as it was only able to store the rollouts of the offspring from the last one or two iterations. There was still diversity in the replay buffer as each iteration there were randomly mutated offsprings, but the quality of these episodes was likely very low. All the higher quality episodes likely were from the seeded solution and its refined version meaning that only these solutions were able to get good learning data. The other solutions instead learnt at a much slower rate as they did not have a head start. This theoretically could be overcome by seeding more pretrained solutions with different BDs, but the problem arises of how to obtain the initial set of good performing diversity solutions with different BDs. Larger unit compositions also required more resources to do rollouts which meant that less had to be allocated to the already under resourced replay buffer, this also restricted the scenarios that the algorithm could be tested on.

The constraint of resources and other factors with the design such as the QMIX algorithm had limited many aspects of the project. The QMIX algorithm was only able to get good results for the 2s3z scenario which was the easiest available scenario as it was a mirror matchup. This meant that the current algorithm design was only able to produce different playstyle for one specific scenario. The original intention was that a repertoire could have been trained for all the different types of scenarios available in the SMAX environment to get diverse solutions for diverse environment scenarios and then a method could have been designed to combine these repertoires to get one repertoire that held truly diverse solutions for different combat scenarios. There was also the limitation of only training against a heuristic policy meaning that all the solutions are potentially overfitted to the heuristic policy.

To overcome this limitation, the algorithm could, after a set number of MAP Elite iterations, swap out the current enemy policy played against for the current best performing solution in the archive and then the repertoire is wiped keeping only the N best solutions. However, problems were encountered with this since the policies were trained as allies, which always started on the left side of the environment, and when they were made an enemy policy, they started on the right. This meant that the trained policy never learnt to move towards the left to find the enemy and always moved right which caused the units to move to the boundary of the environment and died when set as the enemy policy. This highlighted problems with QMIX being unable to learn a policy that was position invariant and the SMAX environment being designed to work with a heuristic policy and implementing another learned policy or self-play involved time that was not available.

5. Conclusion

The aim of this project was to combine reinforcement learning algorithms with quality and diversity algorithms to achieve an algorithm that could train different playstyles for RTS games. A PGA-ME inspired algorithm was designed using a novel QMIX emitter to replace the Actor Critic based TD3 emitter used with PGA-ME. The architectural addition of the recurrent layer through a GRU cell was used to overcome the problems encountered in a POMDP present in RTS games. The proposed PGA-ME QMIX algorithm was designed and tested using the SMAX environment from the JaxMARL library which made heavy use of JAX for compiling the python script to efficient machine code and parallelised computing on a GPU to significantly speed up the execution and training the large number of solutions needed in quality and diversity algorithms. Starting with a seeded pretrained solution among other randomly initialised solutions a repertoire with diverse solutions was generated. From the fittest solutions selected from the repertoire, there was evidence to suggest that different playstyles were learned. These included a highly aggressive playstyle that mostly focussed on attacking, one that aims for positional advantage by flanking the enemy units which is the act of manoeuvring units behind the enemy units while thy are distracted by other ally units at the front, and others.

The final PGA-ME QMIX algorithm demonstrates that there is potential in the application of quality and diversity algorithms such as MAP Elites during game design such as the design of game AI. However, there were several problems encountered that limited the potential of the final algorithm designed and potential improvements that could be explored.

5.1. Future Work

Although there was potential with seeding a single pretrained, this combined with the PGA-ME QMIX algorithm collapsing to a global maximum, meant the most good solutions were clustered around a single point in the behaviour space around the BD of the seeded solution. The seeding of multiple pretrained solutions could increase the diversity of the good solution but it can only work if their BDs are different. One idea that could be tested could be imitation learning which was utilized by AlphaStar (1) to similarly get initial good solutions before training started. So perhaps solutions could be trained offline with a buffer containing only episodes with a certain playstyle to get solutions with different BDs. The proposed algorithm could be used to see if it can discover new novel playstyles that the game designers did not come up with, instead of trying to learn playstyles that are known as done in this paper. But it is still likely that the good solutions are still likely to be clustered around the seeded solutions.

The proposed algorithm could also take ideas of conditioning the solutions to the descriptors from DCRL (21) and this could allow for great diversity as good solutions are not likely to be clustered around the seeded solutions. However, this would require replacing the QMIX emitter with another reinforcement learning emitter that utilised the actor and critic architecture as DCRL needs to train a descriptor conditioned actor. The SD-SAC (50) algorithm could be a suitable replacement as it is off policy and worked with discrete actions, however it loses the information sharing from the adoption of CTDE by QMIX that was made possible by the mixer

network. There has been research into adapting soft actor critic for MARL through the adoption of CTDE by (55) and (56). Perhaps this new method could perform better in the scenarios outside of the 2s3z scenario such as the 5m_vs_6m scenario where the enemy units outnumbered the ally units. This has potential for unique strategies as the disadvantage could force the solution to learn playstyles that somehow gains an advantage such as through positioning or fine control to offset the deficit and win.

Work could also be done to improve the BDs for the solutions, so that they better lined up with the playstyles, and increase the dimensions of the BD as the BD used for the final design was only 2 dimensional. An example BD could be using distance to specific unit types instead of all ally units to encourage combined arms approach where units of the same type work together among themselves and cover for weaknesses of other unit types. Adding the unit type encoding to the observation the policy receives for the scenarios with heterogenous unit types could potentially help the policy learn more complex control and take advantage of properties unique to them such as attack range.

Another area of improvement could be training a positional invariant policy which would allow for it to be used as both an ally and enemy policy as transformation to the starting position should not affect performance. This could open the possibility to more diverse enemy behaviour instead of being constrained to the heuristic policy and open the possibility for diverse solutions for diverse enemy behaviour.

The ideas in this paper could be extended further by exploring the feasibility of designing a game AI capable of dynamic adaptation through the adjustment of their strategies over time, either between or during matches, to maintain challenge and engagement from the players. This could be achieved by training another reinforcement learning agent that is able to select different playstyles from a repertoire of diverse solutions using a BD to get the best solution to tackle the current situation in real time. Work could also be extended to the macro component of RTS strategy by allowing agent to select their own unit composition knowing unit composition.

References

(1) Team A. AlphaStar: mastering the real-time strategy game StarCraft II. *DeepMind blog*. 2019; 24 .

(2) Li Z, Ji Q, Ling X, Liu Q. A Comprehensive Review of Multi-Agent Reinforcement Learning in Video Games. *Authorea Preprints*. 2025; .

(3) Jacob M, Devlin S, Hofmann K. "it's unwieldy and it takes a lot of time"—challenges and opportunities for creating agents in commercial games. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment,* ; 2020. pp. 88–94. .

(4) Perkel S. *Popular 'Call of Duty' franchise confirms what players suspected: AI was used to make some of the games*. https://www.businessinsider.com/call-of-duty-activision-ai-in-some-games-2025-2 [Accessed Jun 2, 2025].

(5) *DALL·E 3*. https://openai.com/index/dall-e-3/ [Accessed Jun 2, 2025].

(6) *Stable Diffusion Online*. https://stablediffusionweb.com/ [Accessed Jun 2, 2025].

(7) McGowan C. 'One day I overheard my boss saying: just put it in ChatGPT': the workers who lost their jobs to AI. *The Guardian*. -05-31 2025. https://www.theguardian.com/technology/2025/may/31/the-workers-who-lost-their-jobs-to-ai-chatgpt. [Accessed Jun 2, 2025].

(8) Gravina D, Khalifa A, Liapis A, Togelius J, Yannakakis GN. Procedural content generation through quality diversity. *2019 IEEE Conference on Games (CoG),* : IEEE; 2019. pp. 1–8. .

(9) Khalifa A, Lee S, Nealen A, Togelius J. Talakat: Bullet hell generation through constrained map-elites. *Proceedings of The Genetic and Evolutionary Computation Conference,* ; 2018. pp. 1047–1054. .

(10) S. -G. Nam, C. -H. Hsueh, K. Ikeda. Generation of Game Stages With Quality and Diversity by Reinforcement Learning in Turn-Based RPG. *IEEE Transactions on Games*. 2022; 14 (3): 488–501. 10.1109/TG.2021.3113313.

(11) S. Nam, C. -H. Hsueh, P. Rerkjirattikal, K. Ikeda. Using Reinforcement Learning to Generate Levels of Super Mario Bros. With Quality and Diversity. *IEEE Transactions on Games*. 2024; 16 (4): 807–820. 10.1109/TG.2024.3416472.

(12) Faldor M, Zhang J, Cully A, Clune J. OMNI-EPIC: Open-endedness via Models of human Notions of Interestingness with Environments Programmed in Code. *arXiv preprint arXiv:2405.15568*. 2024; .

(13) Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*. 2013; .

(14) Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*. 2018; 362 (6419): 1140–1144. .

(15) Berner C, Brockman G, Chan B, Cheung V, Dębiak P, Dennison C, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*. 2019; .

(16) S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, M. Preuss. A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in Games*. 2013; 5 (4): 293–311. 10.1109/TCIAIG.2013.2286295.

(17) Z. Yang, S. Ontañón. An Experimental Survey on Methods for Integrating Scripts Into Adversarial Search for RTS Games. *IEEE Transactions on Games*. 2022; 14 (2): 117–125. 10.1109/TG.2021.3065313.

(18) A. Cully, Y. Demiris. Quality and Diversity Optimization: A Unifying Modular Framework. *IEEE Transactions on Evolutionary Computation*. 2018; 22 (2): 245–259. 10.1109/TEVC.2017.2704781.

(19) Nilsson O, Cully A. Policy gradient assisted map-elites. *Proceedings of the Genetic and Evolutionary Computation Conference,* ; 2021. pp. 866–875. .

(20) Faldor M, Chalumeau F, Flageat M, Cully A. Map-elites with descriptor-conditioned gradients and archive distillation into a single policy. *Proceedings of the Genetic and Evolutionary Computation Conference,* ; 2023. pp. 138–146. .

(21) Faldor M, Chalumeau F, Flageat M, Cully A. Synergizing quality-diversity with descriptor-conditioned reinforcement learning. *ACM Transactions on Evolutionary Learning*. 2025; 5 (1): 1–35. .

(22) Batra S, Tjanaka B, Fontaine MC, Petrenko A, Nikolaidis S, Sukhatme G. Proximal policy gradient arborescence for quality diversity reinforcement learning. *arXiv preprint arXiv:2305.13795*. 2023; .

(23) G Leon B, Riccio F, Subramanian K, Wurman P, Stone P. Discovering Creative Behaviors through DUPLEX: Diverse Universal Features for Policy Exploration. *Advances in Neural Information Processing Systems*. 2024; 37 49625–49648. .

(24) J. Scheiermann, W. Konen. AlphaZero-Inspired Game Learning: Faster Training by Using MCTS Only at Test Time. *IEEE Transactions on Games*. 2023; 15 (4): 637–647. 10.1109/TG.2022.3206733.

(25) Samvelyan M, Rashid T, De Witt CS, Farquhar G, Nardelli N, Rudner TG, et al. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*. 2019; .

(26) Mouret J, Clune J. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*. 2015; .

(27) Sutton RS, Barto AG. *Reinforcement learning: An introduction*. : MIT press Cambridge; 1998. .

(28) S. Huang, S. Ontañón, C. Bamford, L. Grela. Gym-μRTS: Toward Affordable Full Game Real-time Strategy Games Research with Deep Reinforcement Learning. *2021 IEEE Conference on Games (CoG),* ; 2021. pp. 1–8. 10.1109/CoG52621.2021.9619076.

(29) Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. *Nature*. 2015; 518 (7540): 529–533. .

(30) Konda V, Tsitsiklis J. Actor-critic algorithms. *Advances in neural information processing systems*. 1999; 12 .

(31) Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, et al. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*. 2015; .

(32) Schulman J, Levine S, Abbeel P, Jordan M, Moritz P. Trust region policy optimization. *International conference on machine learning,* : PMLR; 2015. pp. 1889–1897. .

(33) Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*. 2017; .

(34) Cully A, Clune J, Tarapore D, Mouret J. Robots that can adapt like animals. *Nature*. 2015; 521 (7553): 503–507. .

(35) Arulkumaran K, Cully A, Togelius J. Alphastar: An evolutionary computation perspective. *Proceedings of the genetic and evolutionary computation conference companion,* ; 2019. pp. 314–315. .

(36) B. L. Miller, M. J. Shaw. Genetic algorithms with dynamic niche sharing for multimodal function optimization. *Proceedings of IEEE International Conference on Evolutionary Computation,* ; 1996. pp. 786–791. 10.1109/ICEC.1996.542701.

(37) Hansen N. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*. 2016; .

(38) N. Hansen, A. Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary computation*. 2001; 9 (2): 159–195. 10.1162/106365601750190398.

(39) Glasmachers T, Schaul T, Yi S, Wierstra D, Schmidhuber J. Exponential natural evolution strategies. *Proceedings of the 12th annual conference on Genetic and evolutionary computation,* ; 2010. pp. 393–400. .

(40) Fontaine M, Nikolaidis S. Differentiable quality diversity. *Advances in Neural Information Processing Systems*. 2021; 34 10040–10052. .

(41) Fujimoto S, Hoof H, Meger D. Addressing function approximation error in actor-critic methods. *International conference on machine learning,* : PMLR; 2018. pp. 1587–1596. .

(42) Fontaine M, Nikolaidis S. Covariance matrix adaptation map-annealing. *Proceedings of the genetic and evolutionary computation conference,* ; 2023. pp. 456–465. .

(43) Bradbury J, Frostig R, Hawkins P, Johnson MJ, Leary C, Maclaurin D, et al. JAX: composable transformations of Python NumPy programs. 2018; .

(44) Rutherford A, Ellis B, Gallici M, Cook J, Lupu A, Ingvarsson Juto G, et al. Jaxmarl: Multi-agent rl environments and algorithms in jax. *Advances in Neural Information Processing Systems*. 2024; 37 50925–50951. .

(45) Chalumeau F, Lim B, Boige R, Allard M, Grillotti L, Flageat M, et al. QDax: A Library for Quality-Diversity and Population-based Algorithms with Hardware Acceleration. *Journal of Machine Learning Research*. 2024; 25 (108): 1–16. http://jmlr.org/papers/v25/23-1027.html. .

(46) De Witt CS, Gupta T, Makoviichuk D, Makoviychuk V, Torr PH, Sun M, et al. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*. 2020; .

(47) Flageat M, Chalumeau F, Cully A. Empirical analysis of pga-map-elites for neuroevolution in uncertain domains. *ACM Transactions on Evolutionary Learning*. 2023; 3 (1): 1–32. .

(48) Tan M. Multi-agent reinforcement learning: Independent vs. cooperative agents. *Proceedings of the tenth international conference on machine learning,* ; 1993. pp. 330–337. .

(49) Rashid T, Samvelyan M, De Witt CS, Farquhar G, Foerster J, Whiteson S. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*. 2020; 21 (178): 1–51. .

(50) Zhou H, Lin Z, Li J, Fu Q, Yang W, Ye D. Revisiting discrete soft actor-critic. *arXiv preprint arXiv:2209.10081*. 2022; .

(51) Hausknecht MJ, Stone P. Deep Recurrent Q-Learning for Partially Observable MDPs. *AAAI fall symposia,* ; 2015. pp. 141. .

(52) Meng L, Gorbet R, Kulić D. Memory-based deep reinforcement learning for pomdps. *2021 IEEE/RSJ international conference on intelligent robots and systems (IROS),* : IEEE; 2021. pp. 5619–5626. .

(53) Chung J, Gulcehre C, Cho K, Bengio Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*. 2014; .

(54) Vassiliades V, Mouret J. Discovering the elite hypervolume by leveraging interspecies correlation. *Proceedings of the Genetic and Evolutionary Computation Conference,* ; 2018. pp. 149–156. .

(55) Pu Y, Wang S, Yang R, Yao X, Li B. Decomposed soft actor-critic method for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2104.06655*. 2021; .

(56) Liu J, Zhong Y, Hu S, Fu H, Fu Q, Chang X, et al. Maximum entropy heterogeneous-agent reinforcement learning. *arXiv preprint arXiv:2306.10715*. 2023; .

(57) European Parliament and Council. *Regulation (EU) 2024/1689 on artificial intelligence*. European Union; 2024. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32024R1689 .

(58) Lannelongue L, Grealey J, Inouye M. Green algorithms: quantifying the carbon footprint of computation. *Advanced science*. 2021; 8 (12): 2100707. .

Declarations

Use of GenAI

I declare that the project has been conducted with some use of GenAI. It has been used to aid with troubleshooting and bug fixing of code used for the final design and spelling and grammar checks for the final report.

Ethical Considerations

In terms of legislation there is the EU AI Act (57) which aims to regulate the deployment of AI. Since the goal of the project was to not deploy what was developed it was unlikely to apply, however in the situation that it is in the future, the system would likely be classified as low-risk as it is primarily intended for use in game-based research and entertainment contexts rather than critical infrastructure or decision-making domains affecting the safety of individuals.

There is a theoretical potential for misuse or dual use, particularly in the context of training autonomous agents that could be adapted for military robotics. However, such use is highly unlikely, given that the methods implemented are domain-specific to the SMAX environment and are not directly transferable to real-world robotics without significant adjustment and significant retraining for applications outside of the gaming context.

Another potential application is in military training or war-game simulations, where AI agents could hypothetically be used to simulate combat scenarios or operations. Even in this context, the risk of direct harm is minimal, as the simulation-based outputs are unlikely to influence physical actions without further human planning and intervention.

Data collection was also assessed for the project. Since it used reinforcement learning which relied on agent-environment interaction to generate its own training data, it was not a large concern except for the SMAX environment used. Permission under its license terms were obtained as SMAX was designed for research and is under the Apache licence. So, if proper accreditation and citation are attributed to the work done, there should be no ethical concerns. This reduced ethical concerns related to data privacy and informed consent, as no human-derived data was used.

Sustainability

The sustainability of the designed algorithm was considered during the design. The algorithm was designed to run within a reasonable time on limited hardware to minimise the amount of greenhouse emission from the compute and more accurately simulate the potential hardware a game developer may have access to. All experiments were run a single NVIDA A16 or NVIDIA Tesla T4 which both had access to 16GB of VRAM. The final designed algorithm took approximately 6 hours to run, and it is estimated that each run of the algorithm has a carbon footprint of 164.32 gCO2e and used 710.99 Wh of energy. These values were obtained according to (58) and their "Green Algorithms calculator".

Also, excessive hyperparameter tuning was avoided to minimise the amount of greenhouse emissions from compute used. This did affect the performance of the final algorithm as there was the chance that the best value for the hyperparameters were missed but the paper is more concerned with a proof of concept and further refinements can be made later if the algorithm shows promise.

Availability of Data and Materials

The link to the repository used during the project:

https://github.com/ED-W-W/Imperial-Masters-Project

The link to the SMAX environment from JaxMARL repository:

https://github.com/FLAIROx/JaxMARL

The link to QDax repository with the PGA-ME algorithm that was adapted for the final algorithm:

https://github.com/adaptive-intelligent-robotics/QDax

Appendices

A. Supplementary Results

Results from testing different recurrent layer size in the scenarios 3s_vs_5z (Figure 0.1), 5m_vs_6m (Figure 0.2) and 6h_vs_8z (Figure 0.3) for QMIX in SMAX.

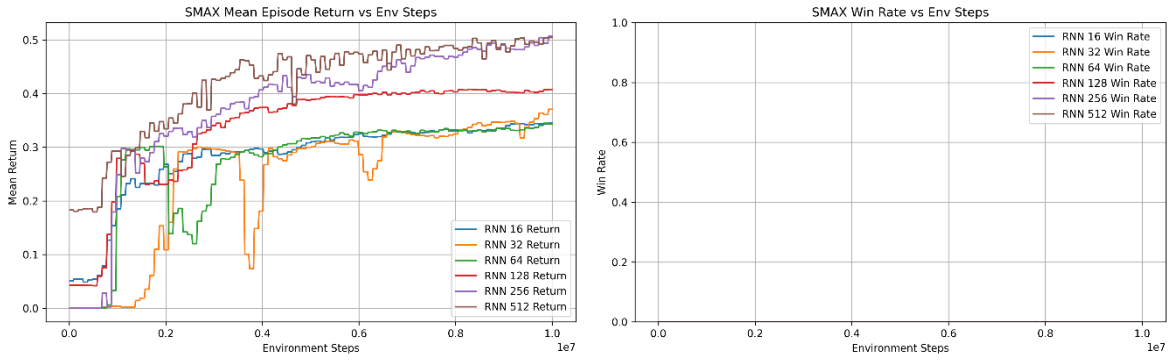


Figure 0.1: Graphs for the final mean return and win rate for the QMIX algorithm in the SMAX scenario 3s_vs_5z with different recurrent layer sizes.

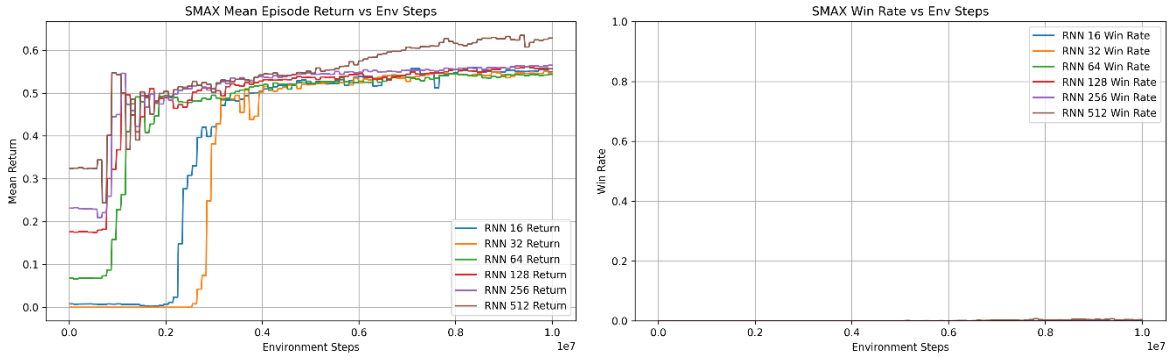


Figure 0.2: Graphs for the final mean return and win rate for the QMIX algorithm in the SMAX scenario 5m_vs_6m with different recurrent layer sizes.

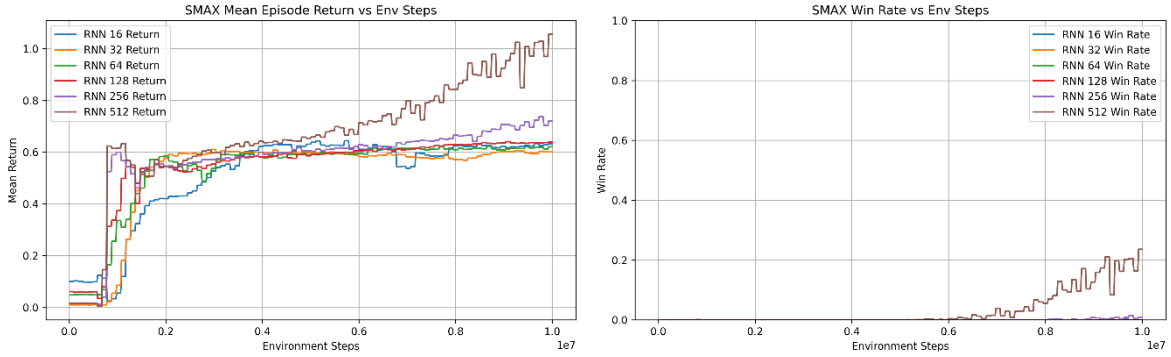


Figure 0.3: Graphs for the final mean return and win rate for the QMIX algorithm in the SMAX scenario 6h_vs_8z with different recurrent layer sizes.

Results from testing for different mixer network sizes for QMIX in the 5m_vs_6m scenario in SMAX.

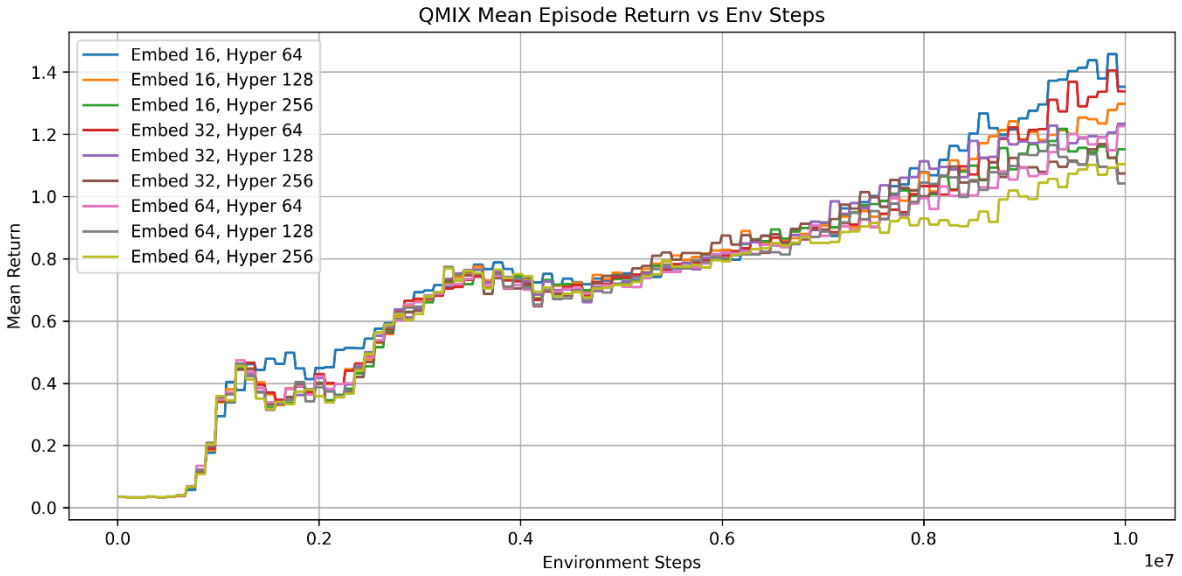


Figure 0.4: Graph of the final mean returns for different embedding layer size and hypernetwork hidden layer size for the QMIX algorithm in the 5m_vs_6m scenario.

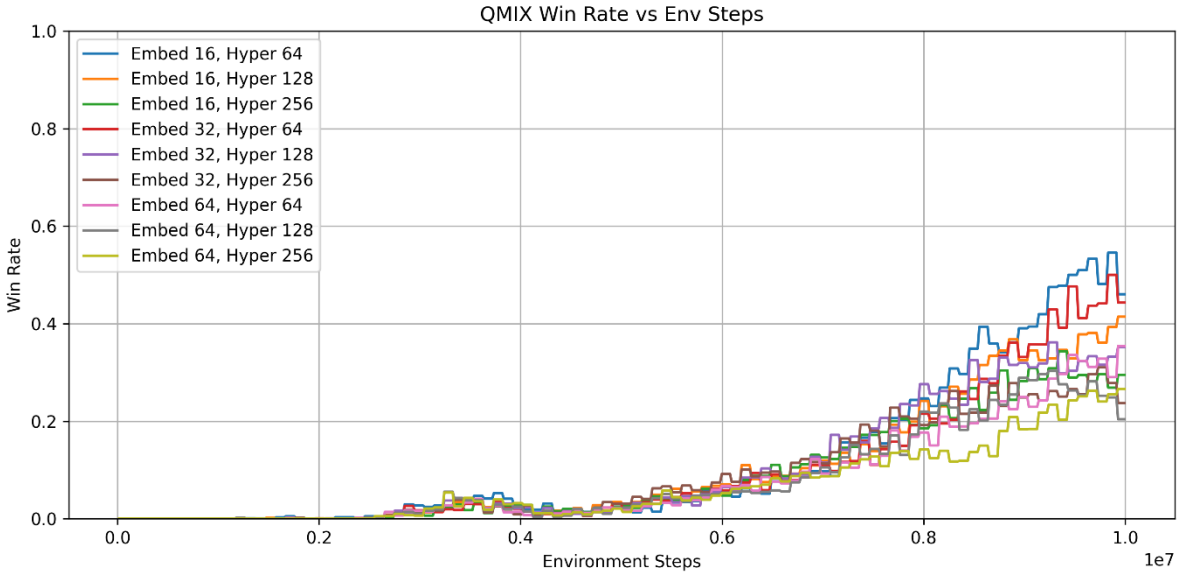


Figure 0.5: Graph of the win rates for different embedding layer size and hypernetwork hidden layer size for the QMIX algorithm in the 5m_vs_6m scenario.

Results from testing different mixer network sizes for QMIX with soft updates in the 2s3z scenario

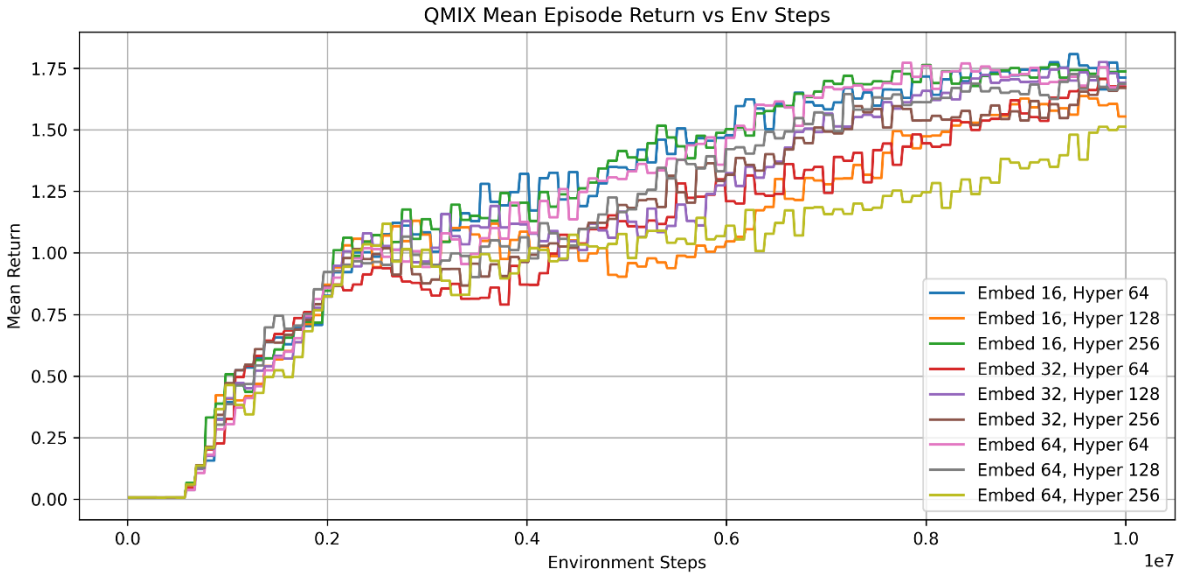


Figure 0.6: Graph of the final mean returns for different embedding layer size and hypernetwork hidden layer size for the QMIX algorithm in the 2s3z scenario using soft updates.

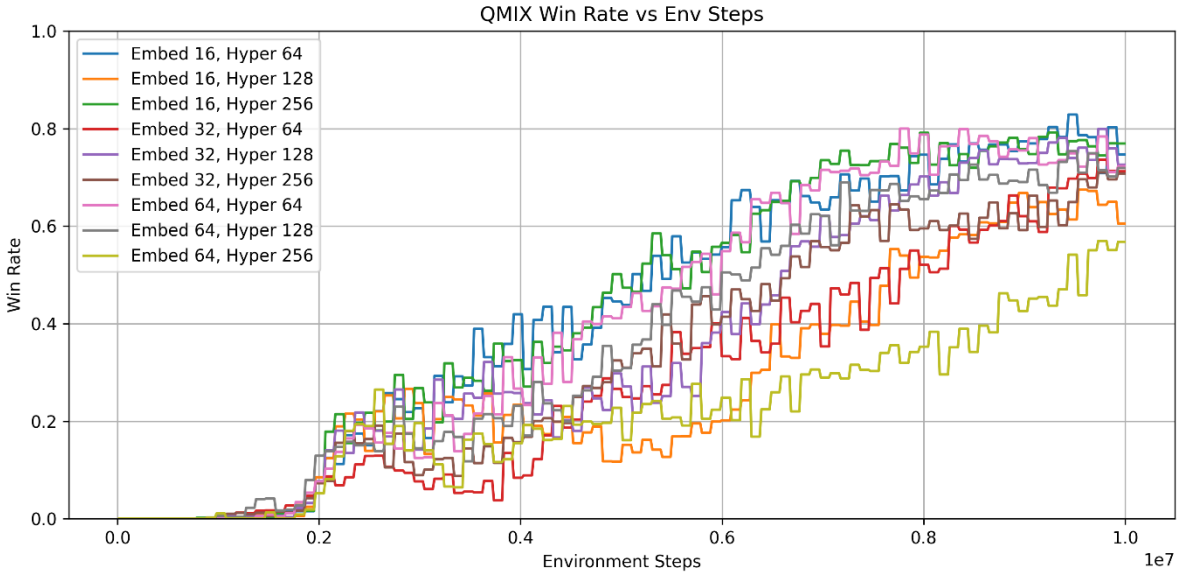


Figure 0.7: Graph of the win rates for different embedding layer size and hypernetwork hidden layer size for the QMIX algorithm in the 2s3z scenario using soft updates