Diploma in Engineering Science

93ESFYP

<u>Final</u> Report

Project ID: **PD-08**

Title: **Reinforcement Learning for Search and Patrol**

By:

Wee Yen Zhe (S10170390K)

Wee Yu Shuen (S10170355C)

Supervisor:

Poh Chun Siong

Defence Science & Technology Agency (DSTA)

Supervisor:

Lim Ching Yang

School of Engineering

Ngee Ann Polytechnic

1. Abstract

This report explores the use of reinforcement learning to create a collaborative model for autonomous search and patrol. Collaboration is not a qualitative variable and can be expressed with a variety of methods. This project uses Unity ML-Agents as a platform for training and implementing reinforcement learning. Multiple methods of collaboration had been explored – a crowding penalty (to spread agents out), zoning techniques (to delegate agents to various parts of the search area), and a target selection method (to allow agents to communicate intent).

These techniques were compared with each other and with a non-collaborative agent in an enlarged environment to determine the effectiveness of collaboration. To quantify the best performing agent, the episode length (number of steps taken to solve the task) was recorded and compared. From this, it is determined that 'target selection' is the best way for agents to collaborate. Agents performed best when they can read other agents' intent and hence would make better decisions. Other methods of collaboration performed similarly, or slightly better than the non-collaborative model.

Future extensions to this project include the use of curriculum learning and self-play to enhance the performance of the agents and to explore new ways of collaboration. Furthermore, other use cases such as robotics can be explored further.

Table of Contents

2. Introduction

Singapore is a country defined by trade; given its strategic position and the lack of its hinterland, trade is the only option. Singapore has been an important maritime port since its inception [1]. In modern Singapore, its maritime shipping is not a mere economic boon, but an essential enabler that allows the rest of the economy to operate. Furthermore, Singapore's ports collectively make up the world's second-busiest port, shipping over 36 metric tons annually [2]. Hence, Singapore must have the capabilities to protect its maritime activities and defend its port.

Today, multiple divisions protect and petrol Singapore's waters – some of these include the Police Coast Guard, the Immigration and Checkpoint Authority and the Singapore Navy. Traditionally, patrol operations are conducted manually with manned ships. However, with over a thousand commercial vessels travelling through Singapore's waters each day, it is difficult to patrol water vessels in an area due to the manpower and logistical challenges involved. Hence, in this final year project, we aim to use machine learning techniques to develop a model to simulate an autonomous vehicle in patrol tasks.

3. Project Objectives

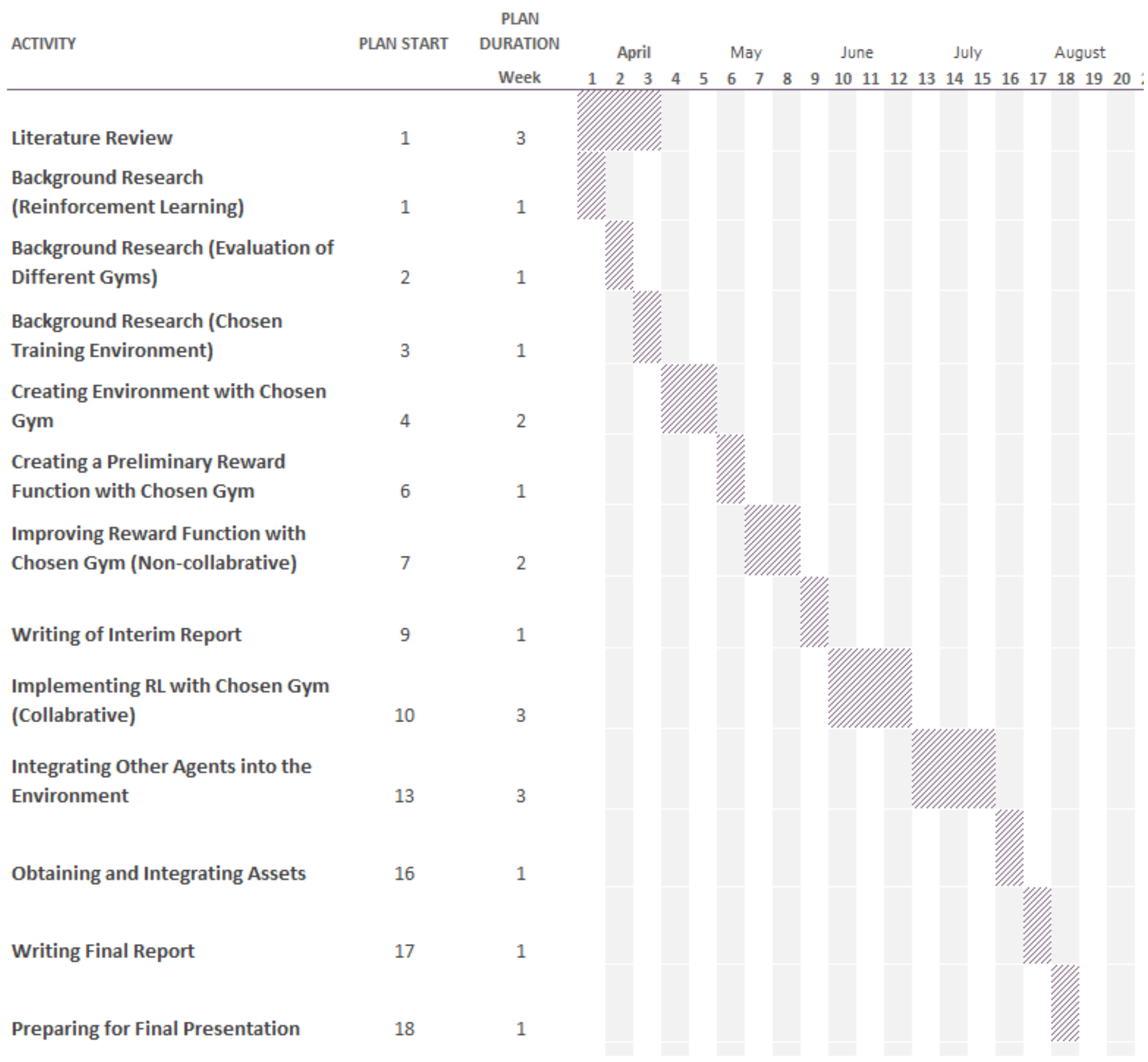
In this Final Year Project, the goals for this project are as follows:

- Have a deep understanding of the theories and concepts underlying reinforcement learning (RL)
- Create a practical and realistic environment for training agents
- Implement RL in the chosen environment
- Explore different methods of collaboration and implement these methods in the chosen environment
- Compare different methods of collaboration and determine the most appropriate method to use

4. Week by Week Schedule

For this Final Year Project (FYP), there are 21 weeks for us to achieve the objectives stated in Section 3. To ensure we keep on task throughout this FYP, we have decided to set several milestones and checkpoints at certain junctions.

To do this, we used a Gantt Chart.



ACTIVITY	PLAN START	PLAN DURATION		April				May				June				July				August		
		Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Literature Review	1	3	▨	▨	▨																	
Background Research (Reinforcement Learning)	1	1	▨																			
Background Research (Evaluation of Different Gyms)	2	1		▨																		
Background Research (Chosen Training Environment)	3	1			▨																	
Creating Environment with Chosen Gym	4	2				▨	▨															
Creating a Preliminary Reward Function with Chosen Gym	6	1						▨														
Improving Reward Function with Chosen Gym (Non-collabrative)	7	2							▨	▨												
Writing of Interim Report	9	1									▨											
Implementing RL with Chosen Gym (Collabrative)	10	3										▨	▨	▨								
Integrating Other Agents into the Environment	13	3													▨	▨	▨					
Obtaining and Integrating Assets	16	1																▨				
Writing Final Report	17	1																	▨			
Preparing for Final Presentation	18	1																		▨		

5. Literature Review

5.1 Reinforcement Learning

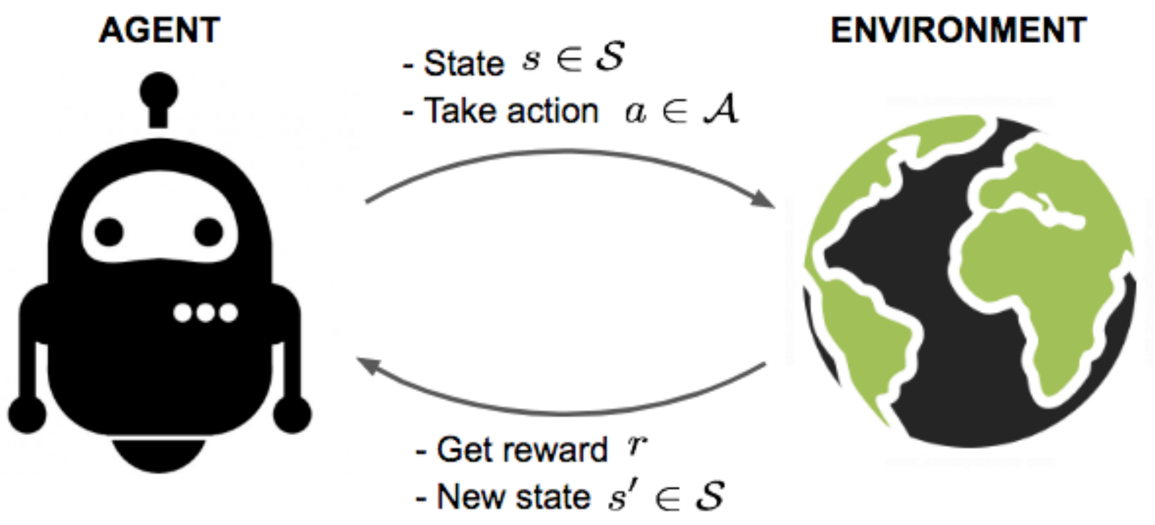

AGENT
ENVIRONMENT
- State $s \in \mathcal{S}$
- Take action $a \in \mathcal{A}$
- Get reward r
- New state $s' \in \mathcal{S}$

Figure 1 Illustration of Reinforcement Learning. Retrieved from https://lilianweng.github.io/lil-log/assets/images/RL_illustration.png

Reinforcement learning is a subset of machine learning where agents discover which sequence of actions would perform the best in the environment it is in. This is achieved by the agent enacting an action *a* at a time step *t* given the state of the environment *s.* Upon reaching the environment, a reward *r* is given to the agent, and a new state of the environment *s′* is issued to the agent. This process of 'action and reward' is repeated until a terminal state is reached, in which the whole process restarts [3].

To predict the performance of the agent at future timesteps, a **return function** is utilized, which is a representation the discounted sum of reward values from the current timestep to a timestep in the future with a discount factor $\gamma \in (0,1]$, to account for both short-term and long-term usages,

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

To determine the performance of a given state s in each environment, a **state value** is used, which is the expected return value for following a given policy π from a given state *s,*

$$v_\pi(s) = \mathrm{E}[G_t | s_t = s]$$

In each state the agent encounters, various actions are available for the agent to choose from. An **action-value** determines the return value for enacting an action a in state *s,* and after which adhering to policy π,

$$q_\pi(s, a) = \mathrm{E}[G_t | s_t = s, a_t = a]$$

To determine the effectiveness of an action in each state, an **advantage function** can be utilised, where the difference between the action value and the state value is found,

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$$

There are two ways in which reinforcement learning problems can be approached from model-based methods, where a model is explicitly created to simulate the environment in which optimal state value and action value can be found, and model-free methods, where the problem is solved by a series of trial-and-error procedures, and solutions obtained by the algorithm may not be the best in the given environment [4].

5.1.1 Model-based methods

In model-based methods, the agent utilises a predictive model to predict various outcomes from different actions and to then choose the best action in each situation. This allows for the optimal value function and optimal policy to be found, leading to it being accurate in cases where the environment has definite states for the agent to consider. It also allows for the explicit planning and decision making of various options by the agent itself. One such example of model-based methods is Dynamic Programming [3].

5.1.1.1 Dynamic Programming

In reinforcement learning, dynamic programming refers to a procedure which iteratively improves a given policy through a **series of policy improvement and policy evaluation**. In the pursuit of finding a better policy in the environment, we utilise the state value to search for a policy which performs better than the current one [5].

Policy evaluation refers to the determination of the current policy's performance through the computation of the state value $v_\pi(s)$ for a state s when following a given policy π. This evaluation is possible due to the known domain of the states in the given environment and is done through the following update rule, in which π(a|s) refers to the probability of action a enacted at state s governed by policy π, and where γ refers to the discount factor applied on reward values obtained at later time-steps,

$$\begin{aligned}
v_\pi(s) &= \mathrm{E}[G_t|s_t = s] \\
&= \mathrm{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |s_t = s] \\
&= \mathrm{E}[R_{t+1} + \gamma G_{t+1} \,|S_t = s] \\
&= \mathrm{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\
&= \sum_{\mathrm{a}} \pi(\mathrm{a}|\mathrm{s}) \sum_{s',r} P(s',r|s,a)(r + \gamma V_k(s'))
\end{aligned}$$

In **Policy Improvement,** we find new actions in which the current policy would benefit visiting as compared to the previous actions they have visited at a certain state of the environment [6]. This determination of new action is done by finding the action with the highest expected action-value at a certain state,

$$q_\pi(s,a) = \mathrm{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$$

$$= \sum_{s',r} P(s',r|s,a)[r + \gamma v_\pi(s')]$$

When an action has the highest action-value amongst all the possible actions in a particular state *s*, then the following is true,

$$q_\pi\big(s,\pi'(s)\big) \geq v_\pi(s)$$

This is because if this new action is better than or as good as the current action a, as dictated by the action value of the action, then the new policy π' must be better or as good as the current policy π, hence leading to a performance improvement of the policy itself [5]. This can be shown in the following formulae, where new policy π' chooses actions greedily, or actions which have the highest action value amongst the domain of actions available at various states,

$$\pi'(s) = \max_a q_\pi(s,a)$$

$$= \max_a \mathrm{E}[\mathrm{R}_{t+1} + \gamma \mathrm{v}_\pi(\mathrm{S}_{t+1}) | \mathrm{S}_t = \mathrm{s}, \mathrm{A}_t = \mathrm{a}]$$

$$= \max_{\mathrm{a}} \textstyle\sum_{\mathrm{s}',\mathrm{r}} \mathrm{P}(\mathrm{s}',\mathrm{r}|\mathrm{s},\mathrm{a})[\mathrm{r} + \gamma \mathrm{v}_\pi(\mathrm{s}')]$$

Both policy evaluation and improvement play a part in the improvement of the policy, in an algorithm termed ‘Generalised Policy Iteration’, in which the policy reaches the optimal one, attribute to a cycle of policy evaluation and policy improvement.

5.1.2 Model-free methods

In model-free methods, the agent generalizes the learning it has encountered in the past states and tries to apply it to the new states. This is normally applied in situations where the state space of the environment is large, making it impractical to simulate each and possible scenario which the agent could encounter. Hence, in a givens state *s,* there is a need for the agent to enact actions done in similar states encountered in the past, which had previously resulted in a high reward signal from the environment. This 'emulation' is achieved through attempts to approximating the optimal policy of the agent. In this sub-section, algorithms relating to this emulation, including proximal policy optimization and soft actor-critic, will be explained.

5.1.2.1 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a type of model-free method which utilises the policy gradient method to improve policy in a given environment. Policy gradient methods are methods which do not consider action value or state value directly; instead, the policy learns directly from the actions it does through the maximization of a performance function $J(\theta)$ [4]. In PPO, we update the policy through a clipped probability ratio between the old policy and a new policy. The probability ratio between the two policies can be defined as follows,

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

With this, an objective function can be derived using a Trust Region Policy Optimization (TRPO), which aims to keep changes within a certain range [7]. However, TRPO involves complicated calculations, including finding whether the distance between the two policies, as determined by Kullback-Leibler-divergence, a measure of the difference between two probability distribution [8], is within a given parameter [7]. The objective function which utilises TRPO uses the following formula, which uses the probability ratio and the previous advantage value of the action,

$$J^{TRPO}(\theta) = E\left[r(\theta)\hat{A}(s,a)\right]$$

Coupled with its difficulty, there is also a need to ensure the change to the policy with the TRPO function does not deviate too far from the original policy [7]. Hence, PPO aims to solve the issue by imposing a limit to the deviation, through a hyperparameter ε,

$$J^{CLIP}(\theta) = E\left[\min\left(r_t(\theta)\hat{A}(s,a), clip(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon)\hat{A}(s,a)\right]$$

The clip function ensures that for a positive advantage value, the change in policy will be limited to 1 + ε, whereas in a negative advantage value, the change in policy is subject to a minimum change of 1 – ε, as shown in the figure below,

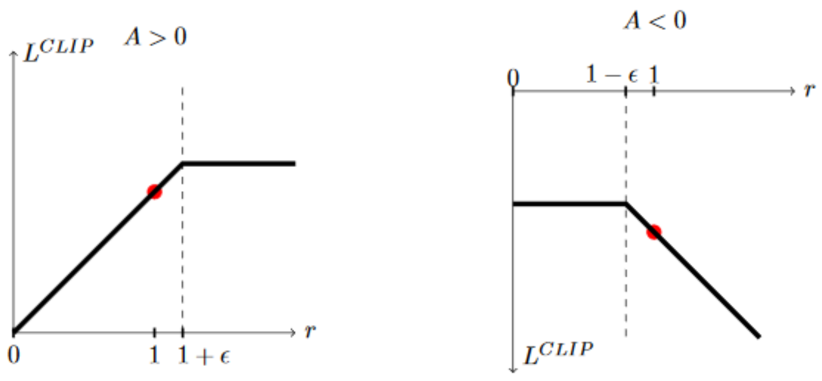


Figure 1: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. Note that L^{CLIP} sums many of these terms.

Figure 2 Effect of Clipping on the advantage function of PPO Source: https://arxiv.org/pdf/1707.06347.pdf

5.1.2.2 Soft Actor-Critic

Soft Actor-Critic (SAC) is an actor-critic algorithm which uses an actor-critic architecture to learn, with an entropy regularisation to ensure randomness in its policy [9]. In an actor-critic architecture, there are two models; the critic model, which estimates the state value function and the action-value function of the policy, and the actor model, which updates the policy $\pi_\theta(a|s)$ based on the update function. In SAC, this update is done by stochastic gradient descent on the state value and action value [9]. Furthermore, to ensure the stability of the policy, two state value functions are compared and learn from each other through the MSBE loss function [10].

In an **entropy-regularized reinforcement learning** such as SAC, a supplemental reward relating to the entropy of the policy at a particular time-step is added to the policy. As a result, the value function would consider both the rewards obtained and the value of the entropy of the actions enacted by the policy, in which *H* represents the entropy function and where α represents a parameter to alter the weightage of the entropy function [10],

$$H(P) = E_{x\sim P}[-\log P(x)]$$

$$V^{\pi}(s) = E_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}(R(s_t,a_t,s_{t+1}) + \alpha H(\pi(.|s_t)))\mid s_0 = s\right]$$

The state function would have similar components as the value function, except for the entropy component for the first time-step,

$$Q^{\pi}(s,a) = E_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}(R(s_t,a_t,s_{t+1}) + \alpha\sum_{t=1}^{\infty}\gamma^{t}\, H(\pi(.|s_t)))\mid s_0 = s, a_0 = a\right]$$

Soft Action-Critic aims to maximise the action-value function of the entropy-regularized reinforcement learning, which is done by maximising the rewards obtained in each policy, and the entropy in decisions made by the policy [10].

5.2 Curriculum Learning

Curriculum learning stems from a training strategy, to begin with easy tasks, and to then build up to more difficult tasks gradually. This is especially applicable to deep learning, where methods of which use lower-level features to build up to higher-level features. In real life, one example of curriculum learning is the building up of mathematical concepts, where one learns how to manipulate with numbers before learning algebra or even calculus [11].

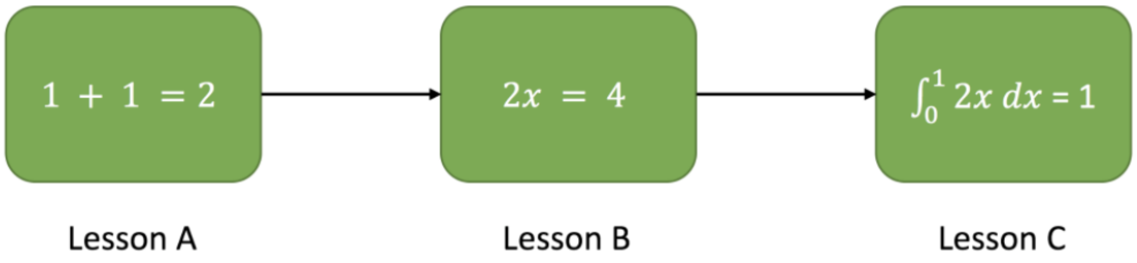


Figure 3 Illustration of reinforcement learning. Source: https://medium.com/@pprocks/curriculum-learning-654aa6423abd

One type of reinforcement learning is known as “Task-specific curriculum”, which uses a curriculum which gradually increases in difficulty in the tasks assigned for the given agents in an environment. It is also known as a “teacher-student setup”. One approach in quantifying the difficulty level of curriculum learning would be to analyse the loss incurred by the various tasks assigned and to then arrange these tasks in sequential order, from the easiest task to the hardest task [12].

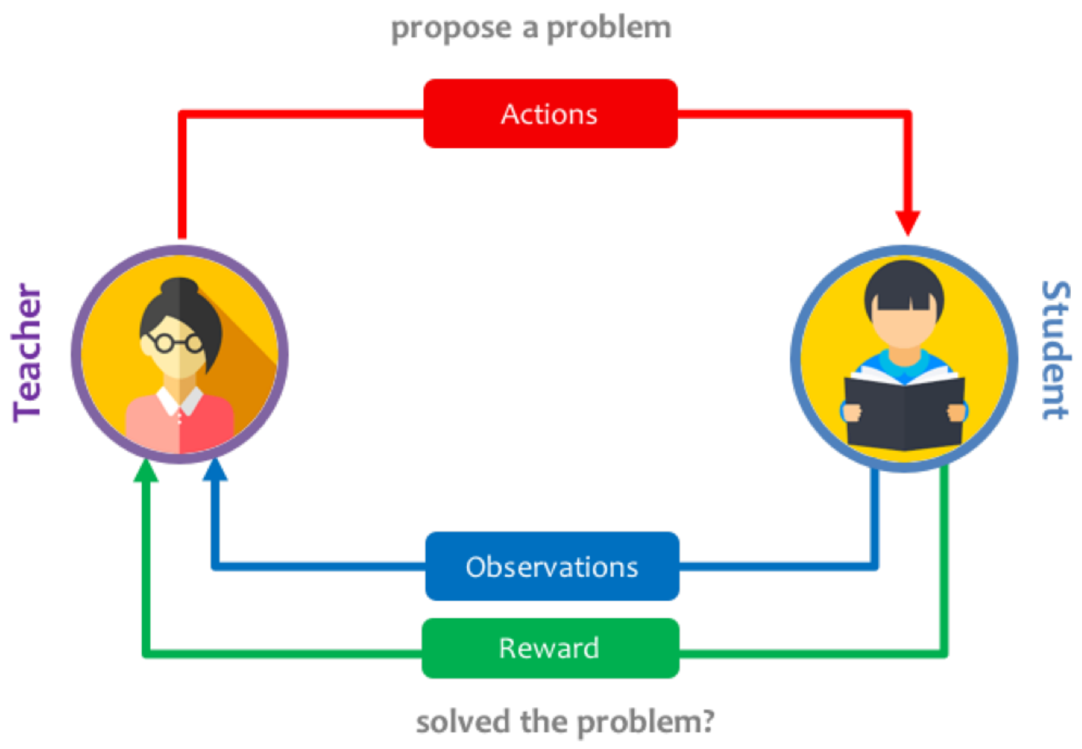


Figure 4 Illustration of the "teacher-student" approach

5.3 Data Processing

For a neural network to function best, there are some best practices to follow.

5.3.1 Data Normalization

Data normalization is important because it allows a neural network to converge to a solution faster. Different data points can have drastically different numerical values (e.g. when comparing sales in monetary value and number of customers) which can make it difficult for a model to find a satisfactory pattern. Normalisation negates this issue by ensuring the data points have a similar scale which can allow the machine learning algorithm to function better [13].

Min-Max Normalization

Min-max normalization is one of the more common ways to normalize data. It aggregates all data to values between 0 and 1, or in some cases -1 and 1. This is done using the following formula:

$$\frac{value - min}{max - min}$$

Min-max normalization is simple to use and works well for most cases. However, min-max normalization struggles with extreme outliers. Using the figure below as reference, outliers drastically askew datapoints.

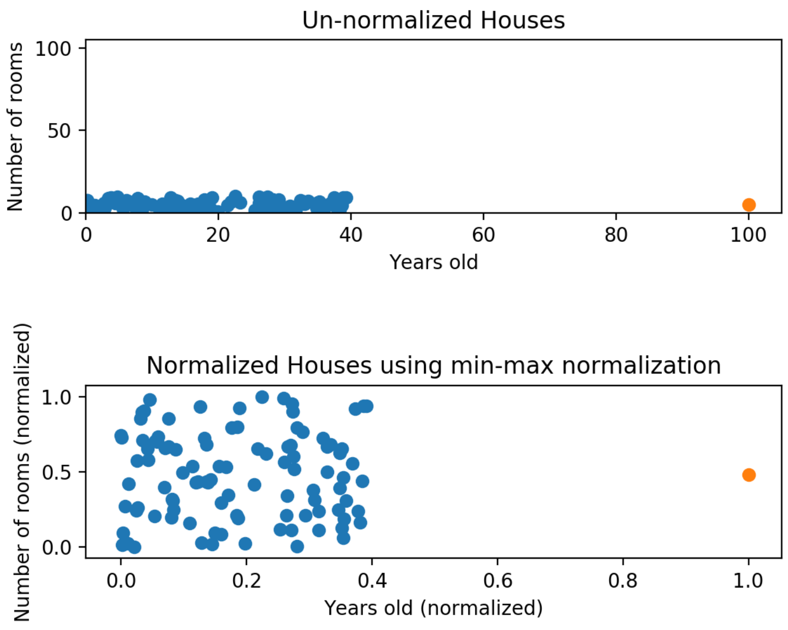


Figure 5 Normalization with min-max normalization. Image courtesy of https://www.codecademy.com/articles/normalization

In this case, the "years old" factor has an outlier of 100 years old while most other datapoints being from 0 to 40 years old. This will, in turn, result in the normalised values being skewed, with most values only being from 0 to 0.4. This is bad for the model as there will be a significant bias towards datapoints with a lower "years old" value [14].

Z-Score Normalization (Standardization):

Z-score normalization is another type of normalization that is often used in datasets to account for outliers. The Z-score can be calculated using the following formula:

$$z = \frac{x - \bar{x}}{\sigma}$$

where:

$\bar{x}$ represents the mean of the dataset

σ represents the standard deviation of the dataset

As the Z score is calculated via the difference between the data value and the mean, then divided by the standard deviation, outliers are much less pronounced than min-max normalisation [15]. This is good for uses with datasets with significant outliers [13].

5.3.2 One-Hot Encoding

Neural networks use numerical values as inputs, which makes observing qualitative values difficult. One-hot encoding "labels" each value with an "encoded" number. Take, for example, the different colours – red, green, and blue. Red can be assigned the first value, green the second and blue the third. This is better represented in a table:

	Red	Green	Blue
Red	1	0	0
Green	0	1	0
Blue	0	0	1

Hence, should red be a data point, it can be represented via one-hot encoding as "{1,0,0}" [16]. This is a better solution than giving the qualitative values a numerical value – if red is labelled as "1" and green "2", this will be incorrect as green and red are not consecutive values, hence affecting learning.

5.4 Training Environment

For the models to learn, it must be able to function in a training environment. This segment will look at various training environments available for use.

5.4.1 OpenAI Gym

Gym by OpenAI is one of the more versatile and commonly used training platforms. It boasts a wide variety of training environments, from classic control theory problems and robotics simulations to Atari 2600 games. It also allows for one to develop their environment with their common API. Furthermore, there are many reinforcement learning algorithms to implement the environment with [15].

5.4.2 Unity ML-Agents

Unity’s ML-Agents is a plugin for the popular game development software. This allows one to easily create their training environment or modify an existing game environment. ML-Agents supports commonly used reinforcement learning algorithms, Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) built-in, making it relatively simple to use [16]. However, to implement other algorithms, other ways would need to be utilised.

5.4.3 Project Malmo

Project Malmo is a platform built upon Minecraft and developed by Microsoft to train models in a first-person, 3D space. It allows close human-AI collaboration as the agent runs in Minecraft, allowing human players and other models to freely interact with it. Moreover, owning to the boundless variety of actions made possible in Minecraft, there are many possibilities in designing a suitable environment for reinforcement learning [17].

5.4.4 MazeBase (Facebook)

MazeBase is an environment developed by Facebook to emulate and train models on simple 2D games. It has 10 different games built-in, with the option to create more. However, MazeBase runs on Lua rather than Python and is generally less flexible than the other options. Furthermore, it is limited by its lack of options in designing environments, which makes it unsuitable of utilising reinforcement learning to learn problems situated in various environments [18].

6. Implementation of Environment

Throughout the project, the environment went through many different iterations. In this segment, we will be documenting how the final environment (at the time of writing) is written. The following segment will be going through the major versions, their training results, and an analysis of said results.

6.1 Training Overview

Unity ML-Agents was chosen as the platform, as it supports Unity, a game development software, allowing us to easily develop 3D models through built-in tools such as in-game physics and rigid bodies. Furthermore, it has support for Tensorboard, a graphical user interface which plots the performance of various algorithms through various metrics like cumulative rewards and entropy, allowing for easy and quantitative comparisons between different configurations. Finally, the language in which Unity works well with is C#, a popular programming language. Hence, future support for this program would be easier, as there would be no need to pick up the inner workings of a new programming language, reducing the learning curve in contributing to this project.

To simulate the context of a Search and Rescue environment, a 100 by 100 walled grid was created in Unity with three agents, simulated with small cubes, and target objects, represented by rectangular blocks. In each episode of the environment, the target would be spawned at random positions on the grid itself, preventing the agent from overfitting.

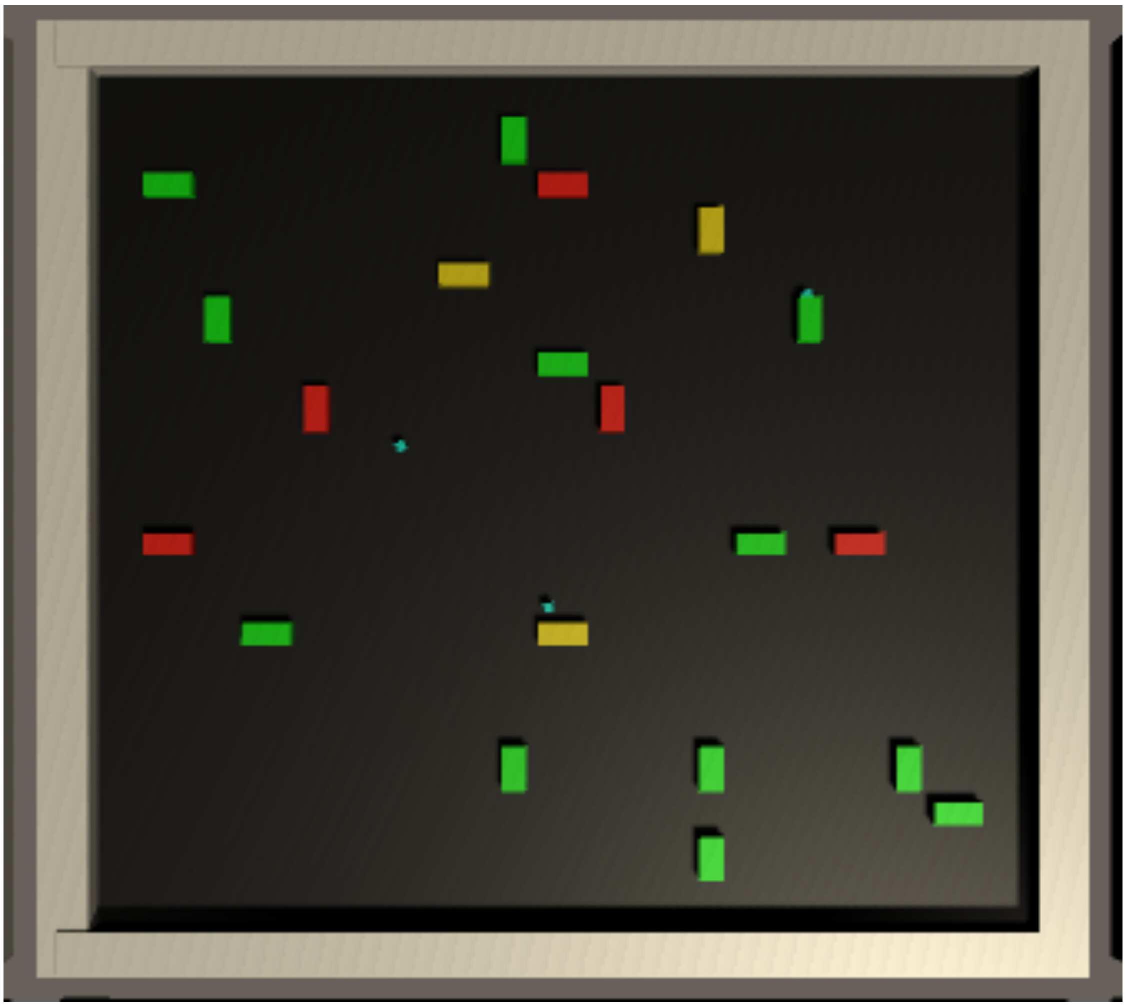


Figure 6 Simulated environment on Unity

To represent the different status of the target objects, different colours are used: green represents a target object agent should make their way towards, yellow represents a target object which had already been in contact with an agent, red represents an obstacle– an “anti-target” which should be avoided by agents. This can be summarised in the figure below.

Legend

🟩	Target (not searched)
🟨	Target (searched)
🟥	Obstacle
🟦	Agent

Figure 7 Legend for colours used in the environment

After each episode, targets and obstacles would first be removed, after which agents would be randomly repositioned. Finally, targets and obstacles would be respawned at random locations on the environment grid itself. This can be represented in a flowchart.

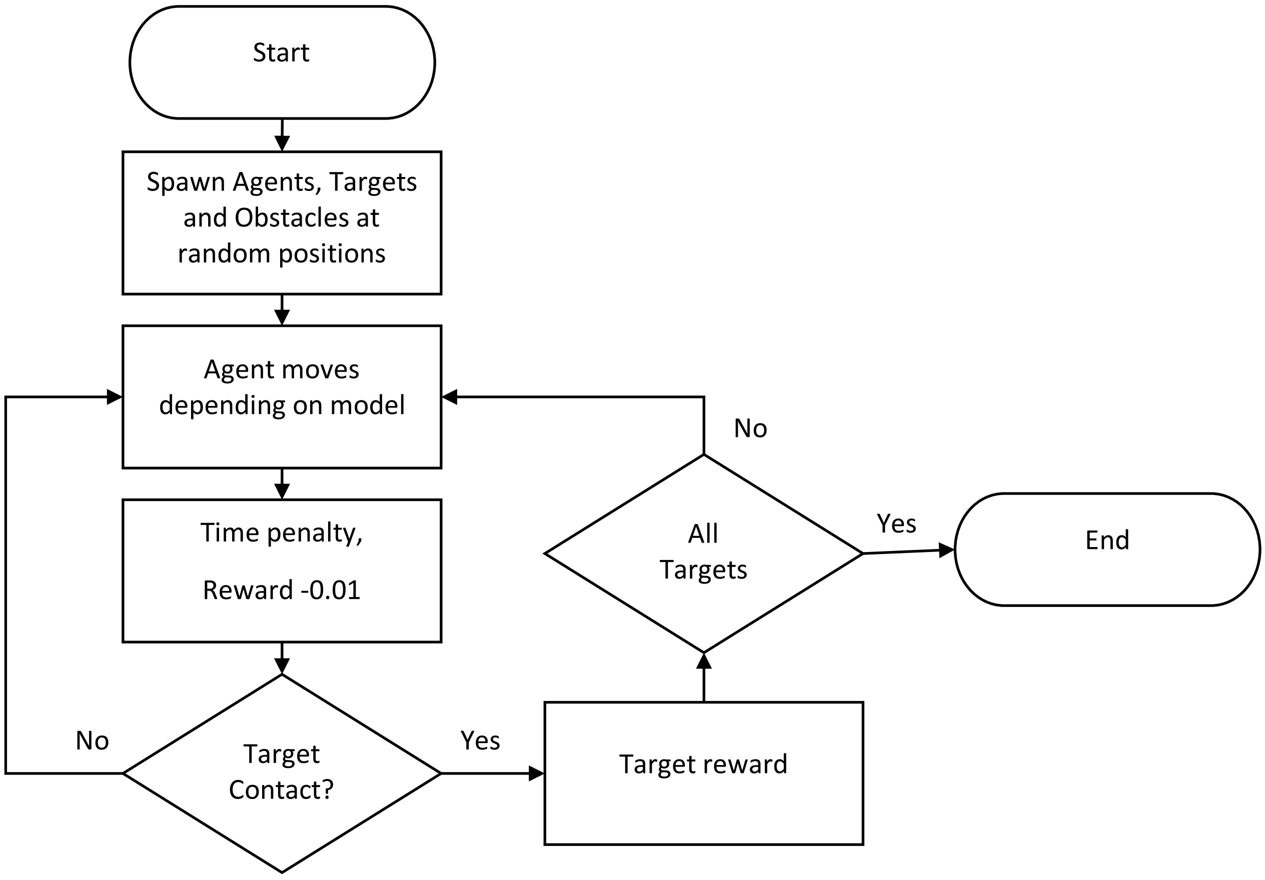

Start
Spawn Agents, Targets and Obstacles at random positions
Agent moves depending on model
Time penalty, Reward -0.01
Target Contact?
No
Yes
Target reward
All Targets
No
Yes
End

Figure 8 Episode flowchart

6.1.1 Models

During training, the models used were basic cuboid meshes, which are Unity’s default. For the final product, a 3D model ship is used for aesthetics.

6.1.2 Agent

In a reinforcement agent, agents require both a reward structure and an observation space. This changes from version to version. All versions do follow a general reward and observation feature set.

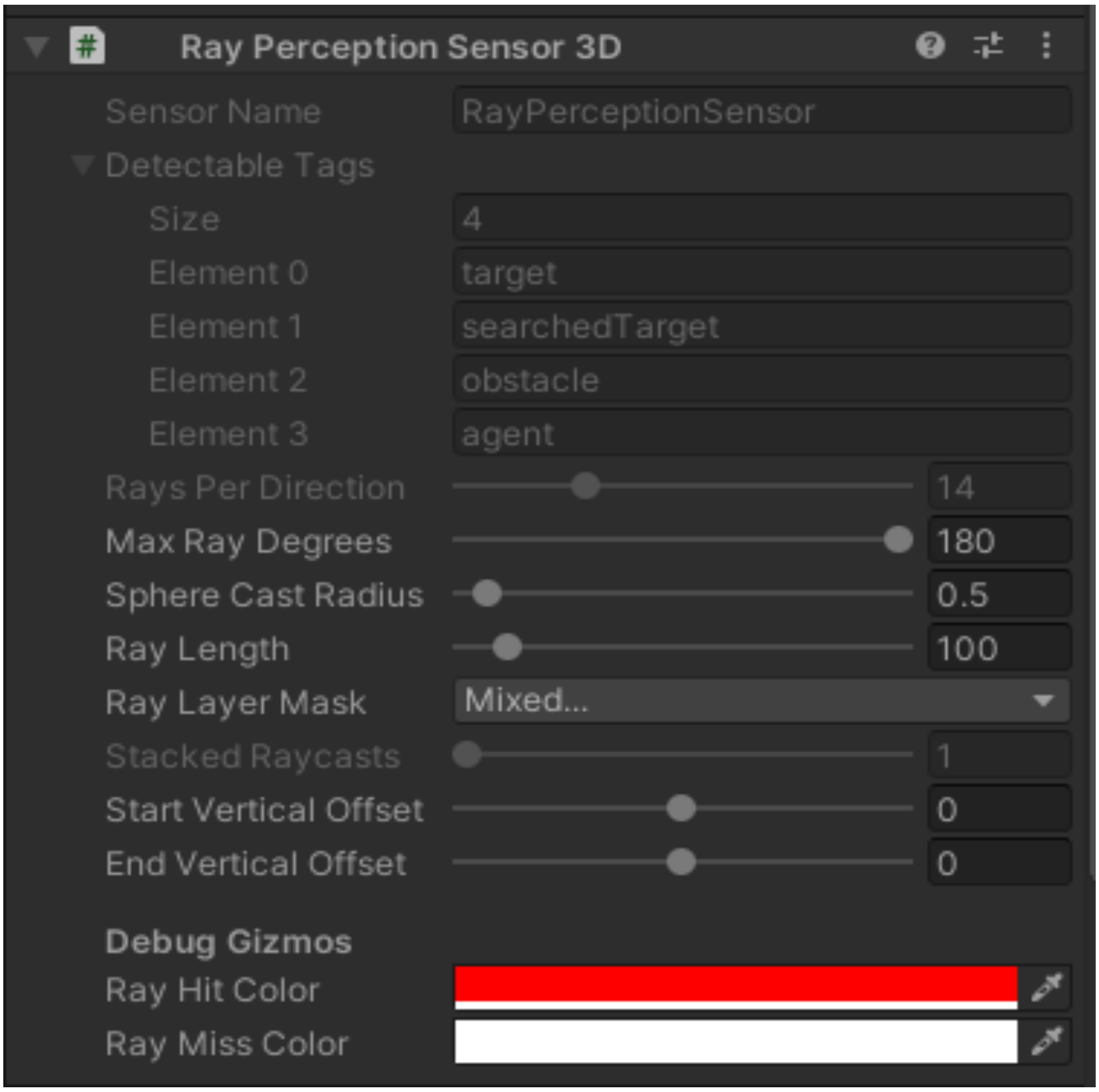
Reward structure:

- Reward for searching a target
- Penalty for each action taken (time penalty)
- Miscellaneous rewards/penalties (dependent on the version)

Observation space:

- Ray casts (using tags of objects in the environment)
- Agent velocity (x and z dimensions)
- Miscellaneous observation spaces (dependent on the version)

-


Ray Perception Sensor 3D

Sensor Name	RayPerceptionSensor
Detectable Tags	
Size	4
Element 0	target
Element 1	searchedTarget
Element 2	obstacle
Element 3	agent
Rays Per Direction	14
Max Ray Degrees	180
Sphere Cast Radius	0.5
Ray Length	100
Ray Layer Mask	Mixed...
Stacked Raycasts	1
Start Vertical Offset	0
End Vertical Offset	0
Debug Gizmos	
Ray Hit Color	🟥
Ray Miss Color	⬜

Figure 9 Configuration of the Agent's RayPerceptionSensor

For ray-casts based observations, Ray Perception Sensors are used. For this sensor, ray casts are emitted from the target object. If these ray casts encounter an object whose tag is included in "Detectable Tags", the ray cast will turn red, signifying it has detected an object of interest. On the back end side of things, an array with the size of the detectable tags is updated with the detection of objects by the ray cast, with a '1' at the index of the object if the ray cast detects an object which is of interest, and a '0' otherwise.

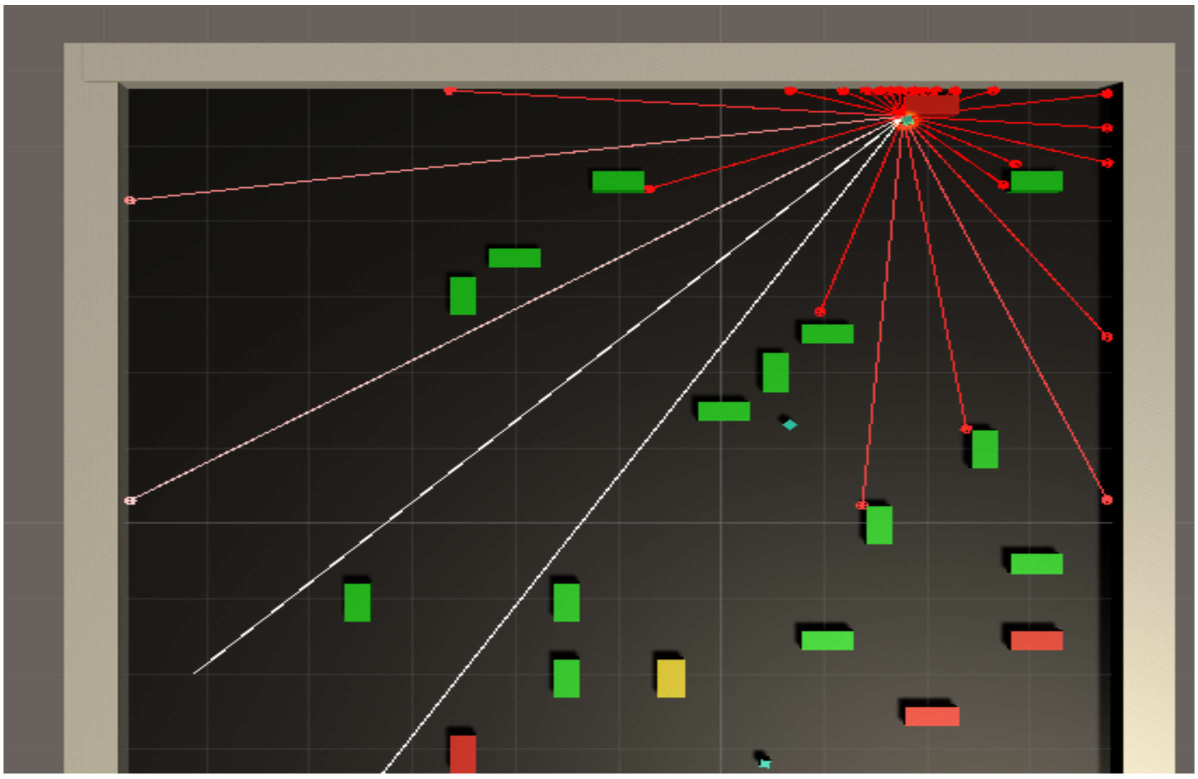


Figure 10 Workings of the Ray Cast Perception Sensor in the environment

For the agent to learn the way it was intended to, a good movement system has been implemented in the agent. It relies on the method *MovePosition* from class *Rigidbody*, which allows the agent to move towards a given position in a smooth manner. Through Reinforcement Learning, the agent would be able to learn which direction would be most suitable in any given situation through control of the continuous variables *hAxis* and *vAxis.* The direction would then be multiplied with the base speed of the agent, and the unit quantity of time to determine an offset position of the agent for *MovePosition* to act on.

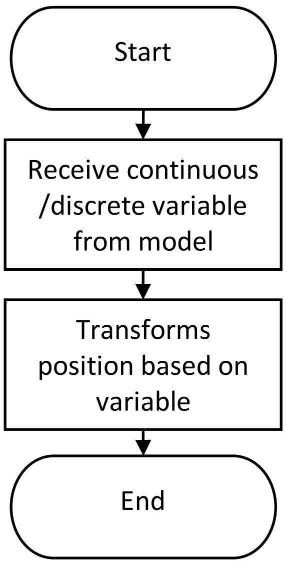

Start
Receive continuous /discrete variable from model
Transforms position based on variable
End

Figure 11 Agent movement flowchart

The agent transforms its position using the following formula.

```
Vector3(hAxis, 0, vAxis) * moveSpeed * Time.deltaTime;
    m_AgentRb.MovePosition(transform.position + movemvent);
```

To ensure the functionality of the agent having a functional movement system, Heuristics can be used to test the agent manually. In this heuristics code, up and down arrows represent the vertical axis, nudging the agent upwards or downwards, while the left and right arrows represent the horizontal axis, nudging the agent to the left or the right. This heuristics method allows for manual testing of the environment mechanics, including amongst others, respawning of agents and targets, and ensuring the environment has a proper reward system.

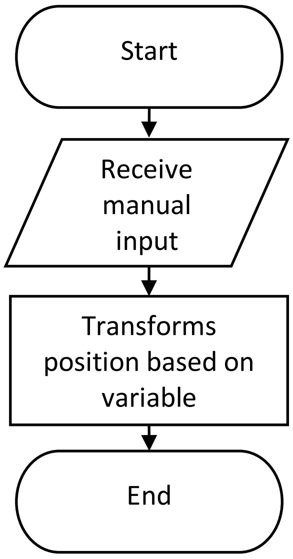


Figure 12 Heuristics Flowchart

Agents need to be able to know when they collide with a target, and a system must be devised to detect and consider the approach undertaken in dealing with these collisions. To that end, we have devised a reward system, where agents are rewarded whenever they collide with a target which has not been collided with before. To do this, we can use Unity's OnCollisionEnter() method.

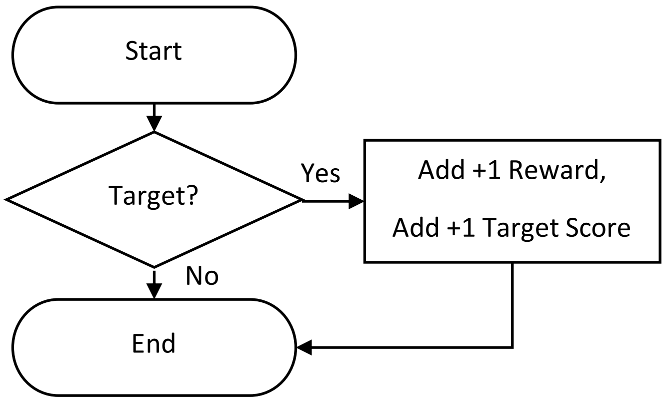


Figure 13 Agent OnCollisionEnter() flowchart

6.1.3 Environment

To ensure proper training and to support the reinforcement learning algorithms, the environment needs to work well, to support the training of the reinforcement learning algorithms. The environment would need to be able to spawn agents and targets and to reset them when necessary.

Firstly, the environment needs to be reset to a new state. This is done using a series of methods, which functions can be represented in pseudocode.

```
Calculate the total area where targets and obstacles can spawn
Generate locations where targets and obstacles can spawn
Remove all targets
Remove all obstacles
Respawn all agents at random locations
Spawn a predetermined number of targets from generated locations
Spawn a predetermined number of obstacles from generated locations
```

6.1.4 Target and Obstacles

Targets and obstacles are very similar. Obstacles contain no additional code and act as a neutral object that the agent can interact with, can collide with, and cannot see through.

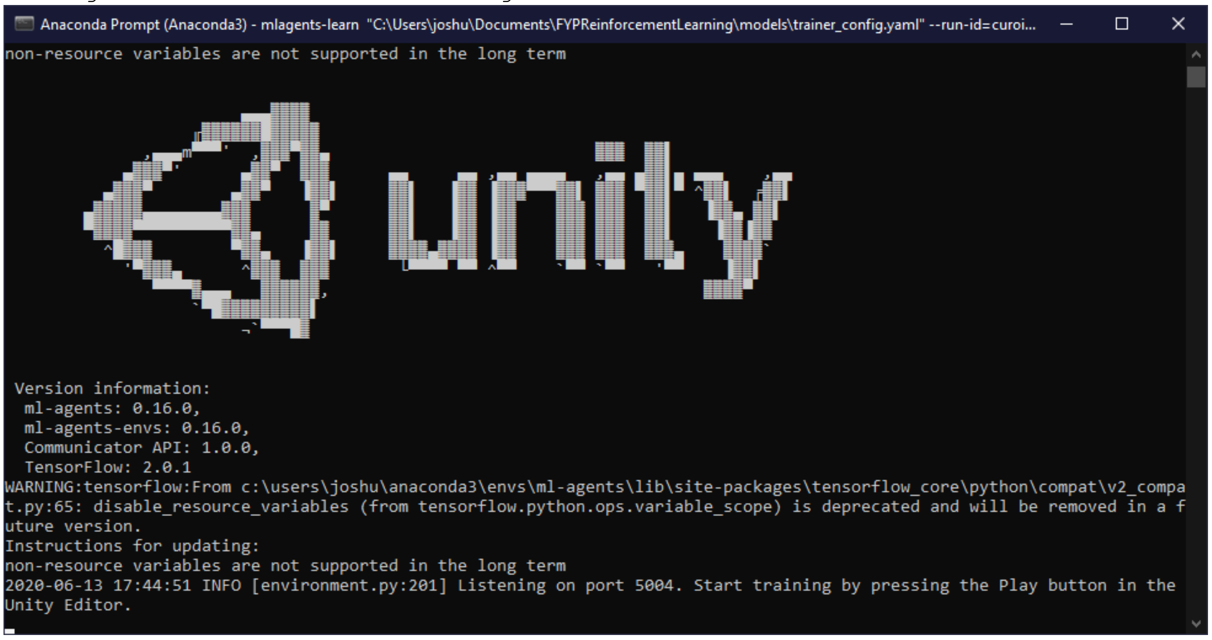
Targets are similar but contain a method that changes the colour of the target after it is ‘searched’ by the agent. When the object is instantiated, the object’s ‘state’ – or colour – is set as unsearched, or green. This occurs at the start of each episode when targets are instantiated.

To generate the object location, objects choose from a list of pre-generated locations. When an object chooses a location, it gets removed from the list. This prevents objects from spawning in the same location. Objects are also rotated horizontally or vertically at random.

6.2 Training in ML-Agents

Training in Unity's ML-Agents is simple. After loading the environment in unity, the one can run the following command to train.

```
mlagents-learn <trainer-config-file> --run-id=<run-identifier>
```



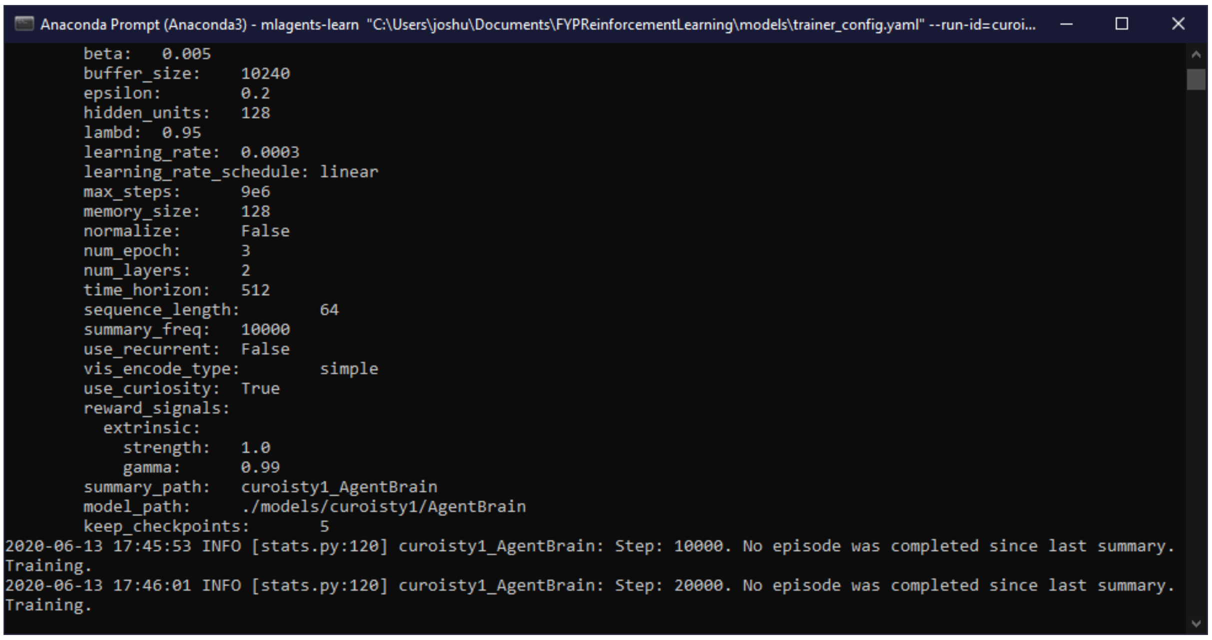
Anaconda Prompt (Anaconda3) - mlagents-learn "C:\Users\joshu\Documents\FYPReinforcementLearning\models\trainer_config.yaml" --run-id=curoi...

```
non-resource variables are not supported in the long term

 Version information:
  ml-agents: 0.16.0,
  ml-agents-envs: 0.16.0,
  Communicator API: 1.0.0,
  TensorFlow: 2.0.1
WARNING:tensorflow:From c:\users\joshu\anaconda3\envs\ml-agents\lib\site-packages\tensorflow_core\python\compat\v2_compa
t.py:65: disable_resource_variables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed in a f
uture version.
Instructions for updating:
non-resource variables are not supported in the long term
2020-06-13 17:44:51 INFO [environment.py:201] Listening on port 5004. Start training by pressing the Play button in the
Unity Editor.
```

Figure 14 Screenshot of command prompt when training

Follow the instructions and press the play button in the Unity Editor and training will automatically commence.



```
Anaconda Prompt (Anaconda3) - mlagents-learn  "C:\Users\joshu\Documents\FYPReinforcementLearning\models\trainer_config.yaml" --run-id=curoi...
        beta:   0.005
        buffer_size:    10240
        epsilon:        0.2
        hidden_units:   128
        lambd:  0.95
        learning_rate:  0.0003
        learning_rate_schedule: linear
        max_steps:      9e6
        memory_size:    128
        normalize:      False
        num_epoch:      3
        num_layers:     2
        time_horizon:   512
        sequence_length:        64
        summary_freq:   10000
        use_recurrent:  False
        vis_encode_type:        simple
        use_curiosity:  True
        reward_signals:
          extrinsic:
            strength:   1.0
            gamma:      0.99
        summary_path:   curoisty1_AgentBrain
        model_path:     ./models/curoisty1/AgentBrain
        keep_checkpoints:       5
2020-06-13 17:45:53 INFO [stats.py:120] curoisty1_AgentBrain: Step: 10000. No episode was completed since last summary.
Training.
2020-06-13 17:46:01 INFO [stats.py:120] curoisty1_AgentBrain: Step: 20000. No episode was completed since last summary.
Training.
```

Figure 15 Screenshot of command prompt after starting training

Training perimeters will be quickly shown on screen for reference and training will start. As shown in the perimeters, training summaries will be shown every 10 000 steps.

6.2.1 Hyperparameters

Hyperparameters are values we can utilise to tinker with the operation of the agent. Notable general hyperparameters include

1) **batch size**, which dictates the number of experiences in one iteration of gradient descent
2) **learning rate,** corresponding to the extent of each update in gradient descent, or how much the model changes after each step
3) **max step**, which refers to the maximum steps the model would take before ending training
4) **time_horizon,** which determines the number of steps to collect from an agent before adding it to an experience buffer for algorithms to learn from

For Proximal Policy Optimization (PPO), the following hyperparameters can be used:

1) **beta,** which controls the strength of the entropy regularisation, ensuring the agent explores as much of the action space during a training step
2) **epsilon,** which influences the speed of change of the policy during training

For Soft Actor-Critic (SAC), the following parameters can be used:

1) **buffer_init_steps,** which specifies the number of experiences to collect in an experience buffer before updating the policy
2) **init_entcoef**, which indicates the amount of exploration an agent should do at the beginning of training

In curiosity, certain hyperparameters can be used to force the agent to explore the area space:

1) **gamma,** the discount factor variable for predicted future rewards
2) **strength,** referring to the magnitude of the curiosity reward generated by intrinsic reward

6.3 Results

6.3.1 Analysing Results

Results can be conveniently analysed using TensorBoard, TensorFlow's visualization toolkit. We can access TensorBoard by running the following command:

```
tensorboard --logdir=summaries
```

This can be accessed from a browser via http://localhost:6006/.

Through this project, we trained many different models with varying perimeters and reward structures. We will go through this in more detail in a later segment; this segment will be used to go through how we analyse and understand the training results.

Although TensorBoard can be customised to display particular values, Unity's ML-Agents come up with a set of pre-defined variables.

Cumulative Reward

Cumulative reward is, as its name suggests, the total reward an agent obtains at the end of the episode. This can vary depending on the reward structure of the environment and can be taken as a direct measure of the agent's performance. In general, the higher the reward, the better the performance. However, different reward structures will result in different total cumulative rewards.

The total cumulative reward is often the metric first analysed. The shape of the graph plays an important role in determining the effectiveness of the training, reward structure etc.

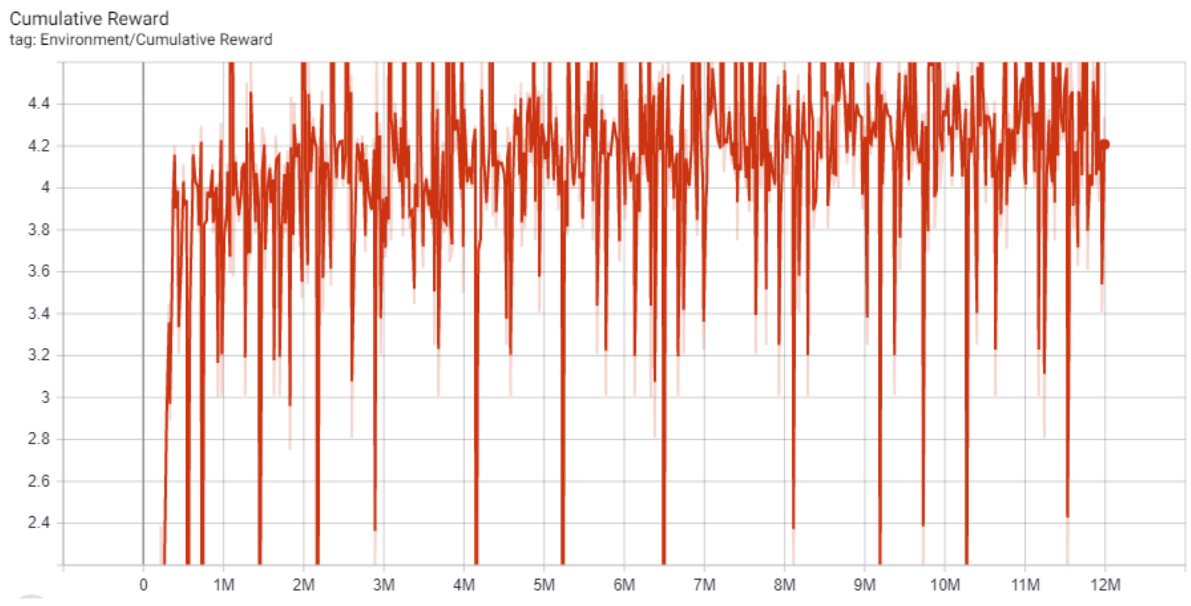


Figure 16 Cumulative Reward Graph with high standard deviations

Take, for example, the figure above. It shows very extreme results, with a high standard deviation. This shows that the results are more due to chance rather than actual learning, which is not ideal.

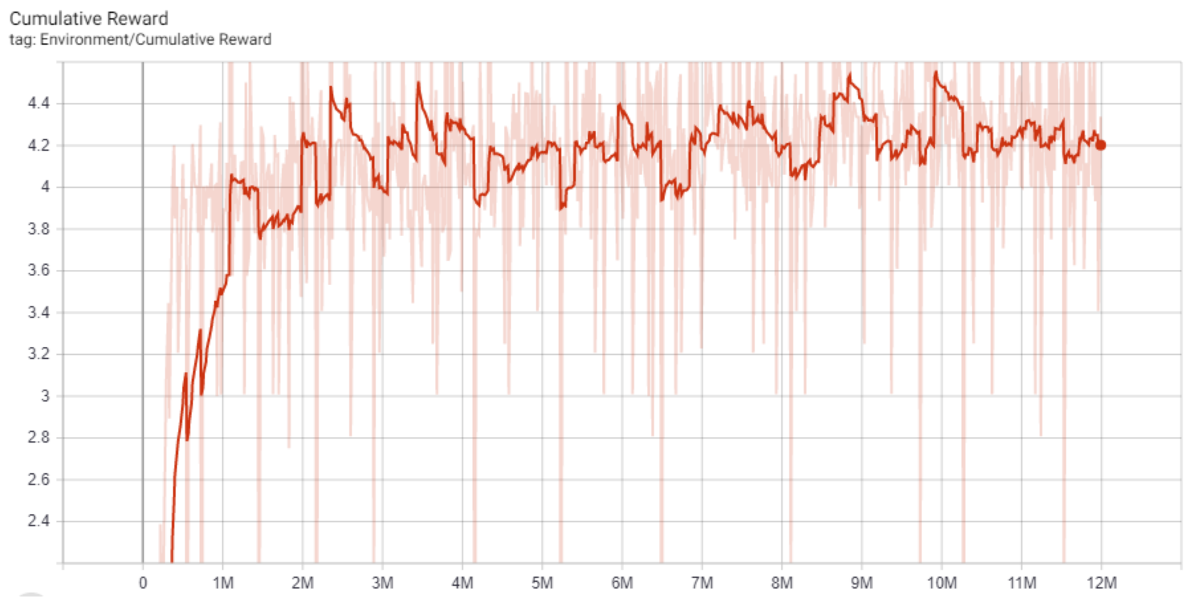


Figure 17 Smoothened cumulative reward graph with high standard deviations

With higher smoothing, it shows the reward gain experiencing diminishing returns after about 1 to 2 million episodes. This shows that the agent's learning after is not very effective thus can adjust accordingly.

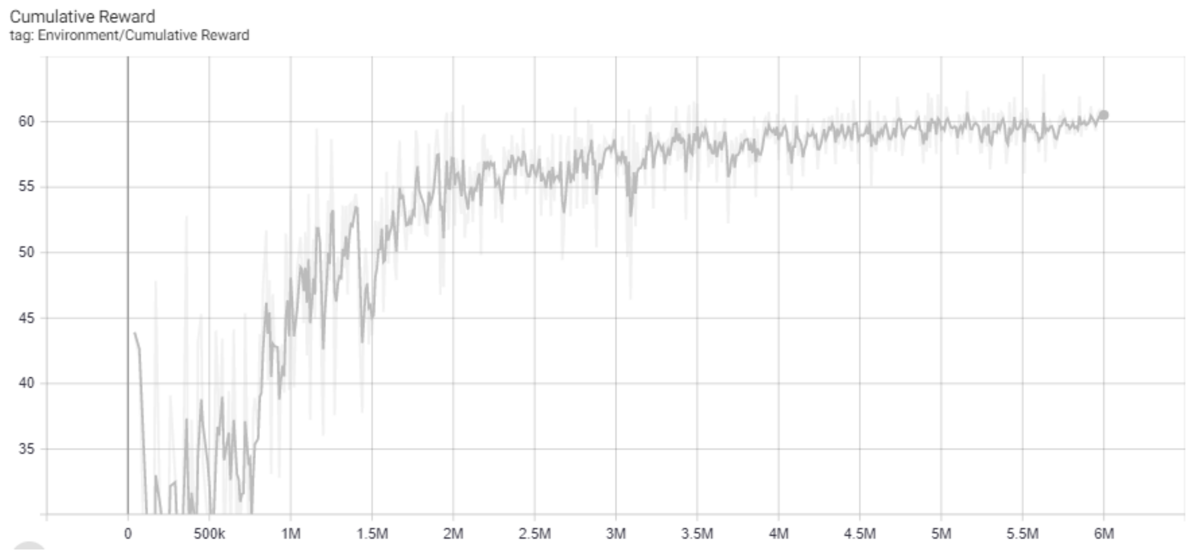


Figure 18 Smoothened cumulative reward graph with a lower standard deviation

The figure above shows a better cumulative reward graph. It has a lower standard deviation, which indicates the agent can complete the problem more reliably.

Episode Length

Another metric is the episode length. This is especially useful because of the nature of our environment – the episode ends when all the targets have been found or when the maximum number of epochs have been reached. Hence, as the agent learns, each episode gets shorter.

Episode length is a more accurate measure for the agent's effectiveness because of varying reward structures. Often, due to the nature of a time penalty, the episode length is inversely proportional to the cumulative reward. As demonstrated in the figure below, the episode length reduces directly proportional to increase in a cumulative reward.

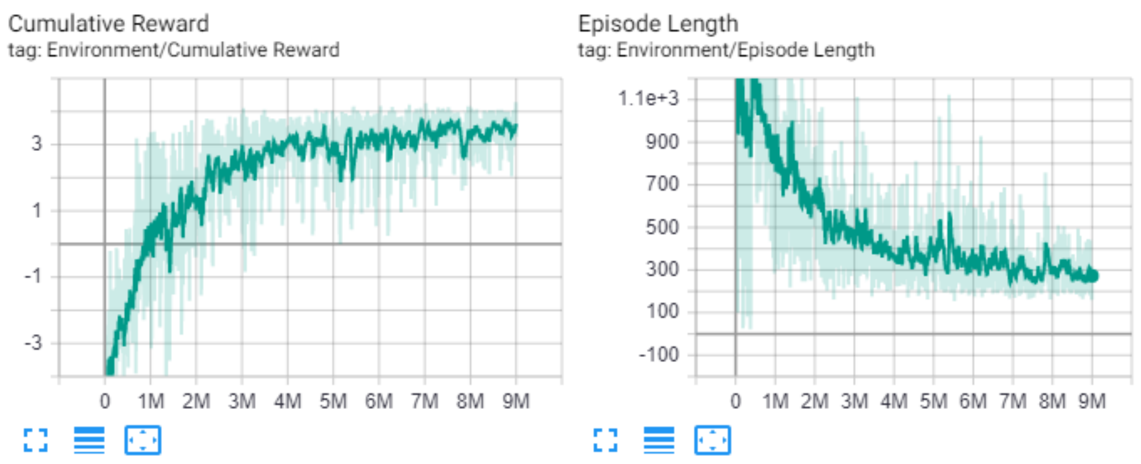


Figure 19 Cumulative reward and episode length graph

Is Training

The *Is Training* is a Boolean that simply checks if the agent is training over time. Usually, the agent is constantly training in our environment; this metric is not often used.

Policy Loss

This is the mean magnitude of the policy loss function. This correlates with how much the policy is changing and should decrease over time.

Value Loss

This represents the mean loss of the value function update and correlates to how well the model can predict the value of each state. This will increase while the agent is learning and decreases when the reward stabilizes.

Entropy

This value determines the relative importance of the entropy term and is automatically adjusted to ensure the agent retains some amounts of randomness during training. This is important for the agent to continue to learn.

PS08 Reinforcement Learning for Search and Patrol

Extrinsic Reward

The extrinsic reward corresponds to the mean cumulative reward received from the environment per episode. Due to the nature of our environment, this value is very similar – if not the same – as the cumulative reward.

Extrinsic Value Estimate

The extrinsic value estimate represents the mean value estimate for all states visited by the agent. As the agent learns, this value increases.

Learning Rate

The learning rate represents the percentage of random decisions versus learned decisions. Over time, this value goes down.

7. Implementation of Reinforcement Learning

This section focuses on the setting up of an environment for reinforcement learning and the training settings which we have implemented to ensure smooth training of agents. Throughout this project, the environment went through many changes. These can be classified into 'versions', which will be explored in this segment. To make it more concise, only some versions will be gone through in detail. All versions can be found in an attached appendix.

* represents agents with elements of collaboration

Version	Name	Description
1	Basic Environment	A basic environment with just targets and agents. Targets disappear when searched by agents. Agents receive a simple reward when searching for a target and a time penalty. Agents observe their velocity and ray casts.
1.1	Updated Basic Environment	Like version 1, but with larger targets for agents to search targets more easily.
2	Updated Movement Engine	Updated movement engine with the same environment as version 1.1
3	Obstacles	Added obstacles, an object that does not interact with an agent. Targets no longer disappear after searched, instead of changing 'status' from unsearched to searched. This is indicated with a colour change, from green to yellow.
4	Updated Observational Space (Single Target)	Observation feature set increased to include the distance between the agent and the nearest target to itself.
4.1	Updated Observational Space (All Targets)	Observation feature set increased to include the distance between the agent and all targets.
5	Target Selection System	Used a "target selection system", where agents select a target and move towards it. Agents are free to search any target but receive more rewards when searching the target, it had

		selected. The agent can also observe target status (if targets have been searched)
6*	Crowding Penalty	Allows agents to observe the distance between itself and other agents. Agents receive a penalty when getting too close to each other.
7*	Splitting Agents into Assigned Zones	Agents are assigned into 4 zones. Agents can observe which zones they are in. Rewards and penalties are given if agents are or are not in their assigned zones, respectively.
7.1*	Floating Zones	Agents are not assigned zones but receive rewards or penalties should there be other agents in the same zone as itself.
8*	Target Selection System (Collaborative)	Agents can observe other agents' target selection. Agents receive a reward when selecting a target which other agents have not selected.
8.1*	Target Selection System (Collaborative, Right of Way – First Agent)	Agents can observe other agents' target selection. Agents receive a reward when it is the first one to select a target which is unselected by other agents.
8.2*	Target Selection System (Collaborative, Right of Way – Nearer Agent)	Agents can observe other agents' target selection. Agents receive a reward when selecting a target which other agents have not selected. If it selects a target already selected by another agent, the nearer agent will get the reward.

7.1 Non-Collaborative Agents

7.1.1 Version 4.1: Updated Observational Space (All Targets)

This version uses a base set of observation space and has a basic reward structure of targets and the time penalty incurred. It can be used as a baseline to compare collaborative and non-collaborative models.

7.1.1.1 Environment

This environment uses a **continuous** action space with branches only for movement. Agents have the following feature set:

Observation Space

- Velocity of the agent (X and Z axis)
- Distance between the agent and all targets (normalised) (per target)
- The angle between the agent and all targets (normalised) (per target)

Reward Structure

- Search reward (+1 for locating a target)
- Time penalty (-0.0005 per time step)

7.1.1.2 Results

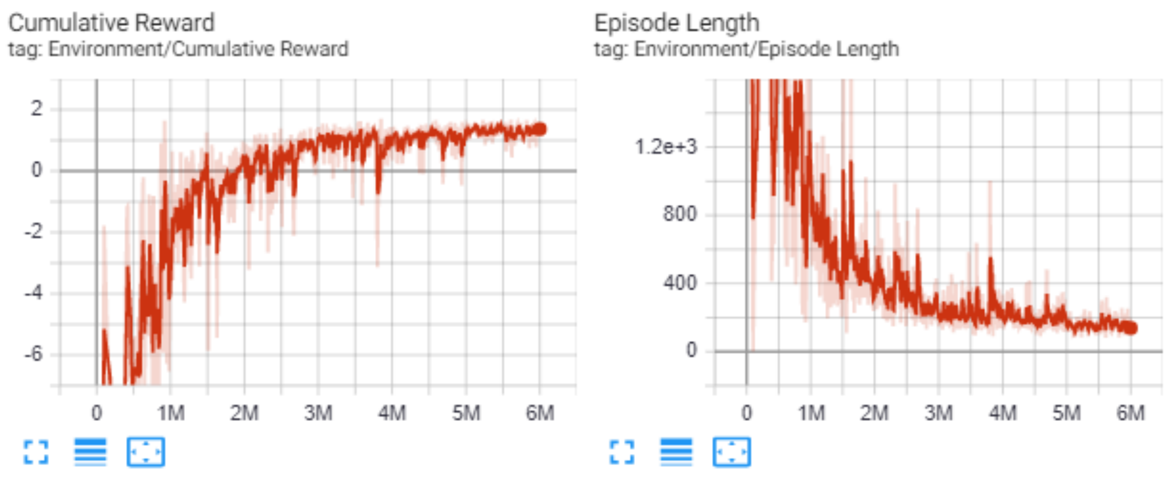


Figure 20 All distances and angles between all targets and the agent

The general shape of the graphs shows a good learning curve, with the learning plateauing at about 3 million steps.

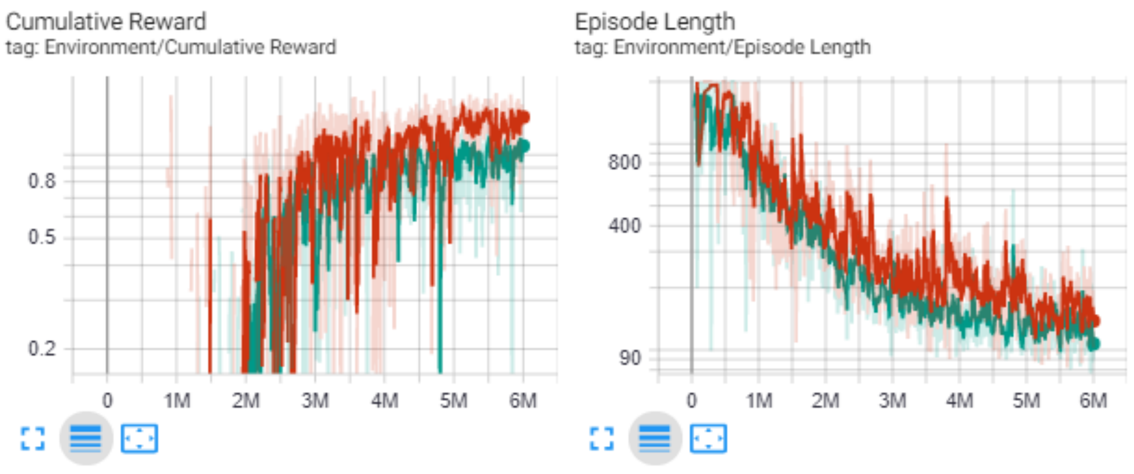


Figure 21 Log scale comparison of a single target (green) and all targets (brown)

Agents in this version head towards targets which are nearest to them in a “group-like” manner, where these agents would clump up together and go for targets which are near to them. Furthermore, there are some incidences where agents are unable to capture the last target due to them not being able to detect it through its ray casts.

7.2 Collaborative Agents

7.2.1 Version 5: Crowding Penalty

7.2.1.1 Environment

In this version, an alternative system of getting the agents to distance themselves from one another is explored. It is hoped that by distancing themselves, agents can cover a greater area and complete the task faster. Agents are given a negative reward for being too close with one another which incentivises the agents to spread out. The feature set is as follows:

Observation Space

- Velocity of the agent (X and Z axis)
- Distance between the agent and all targets (normalised) (per target)
- Angle between the agent and all targets (normalised) (per target)
- Distance between the agent and other agents (normalised) (per target)
- Angle between the agent and other agents (normalised) (per target)

Reward Structure

- Search reward (+1 for locating a target)
- Time penalty (-0.0005 per time step)
- Penalty for being within 10 distance units from another agent (-0.0005 per occurrence)

7.2.1.2 Results

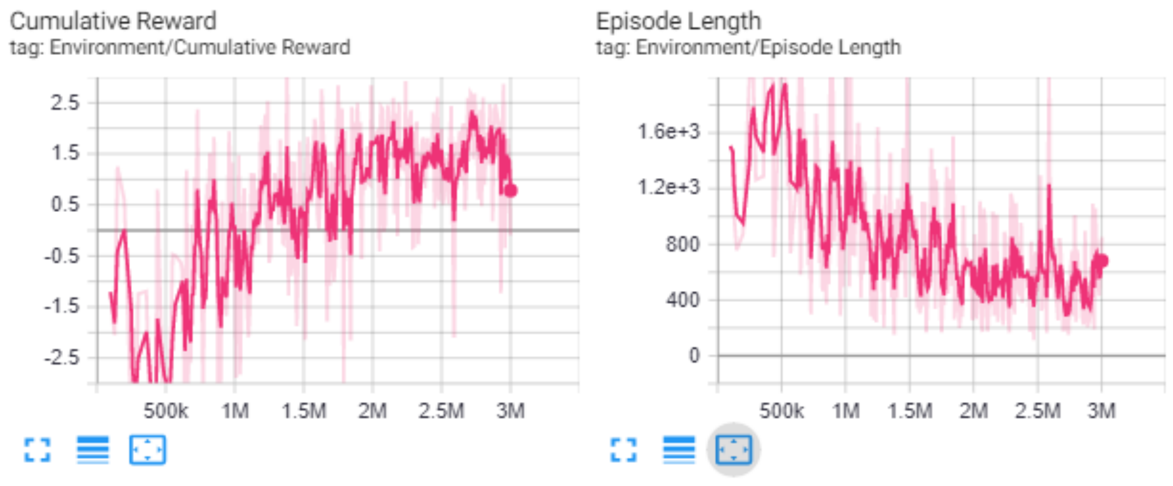


Figure 22 Results

In the figure, the implementation of a crowding model can achieve the task of retrieving the targets in 200-time steps. Furthermore, the learning curve for the cumulative reward increases gradually, with a corresponding gradual decrease in episode length.

In this model, agents tend to space out from one another and head for targets nearest to them. However, there is a tendency for these agents to head towards the same targets, increasing the overall episode length.

.

7.2.2 Version 6: Splitting Agents into Assigned Zones

7.2.2.1 Environment

In this version, the agents are explicitly split into distinct zones in the environment, as seen in the figure below. Although this instance is split into 4 zones, the number of zones could be changed to the desired number.

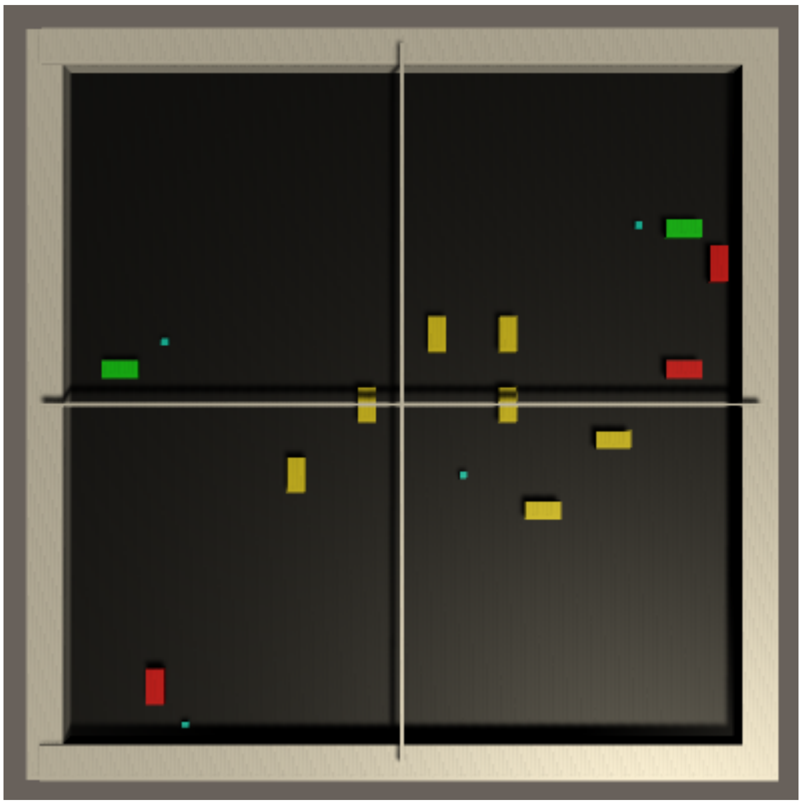


Figure 23 Agent zones environment

Agents are manually assigned to each zone. Agents are not fixed to individual zones programmatically and are free to move between zones. The feature space is as follows:

Observation Space

- Velocity of the agent (X and Z axis)
- Distance between the agent and all targets (normalised) (per target)
- Angle between the agent and all targets (normalised) (per target)
- Distance between the agent and other agents (normalised) (per target)
- Angle between the agent and other agents (normalised) (per target)
- Zone the agent is assigned to (one-hot encoded)
- Zone the agent is currently in (one-hot encoded)

Reward Structure

- Search reward (+1 for locating a target)
- Time penalty (-0.0005 per time step)

Unlike other versions where agents spawn randomly throughout the board, the agents were spawned in the middle of their assigned zone.

7.2.2.2 Results

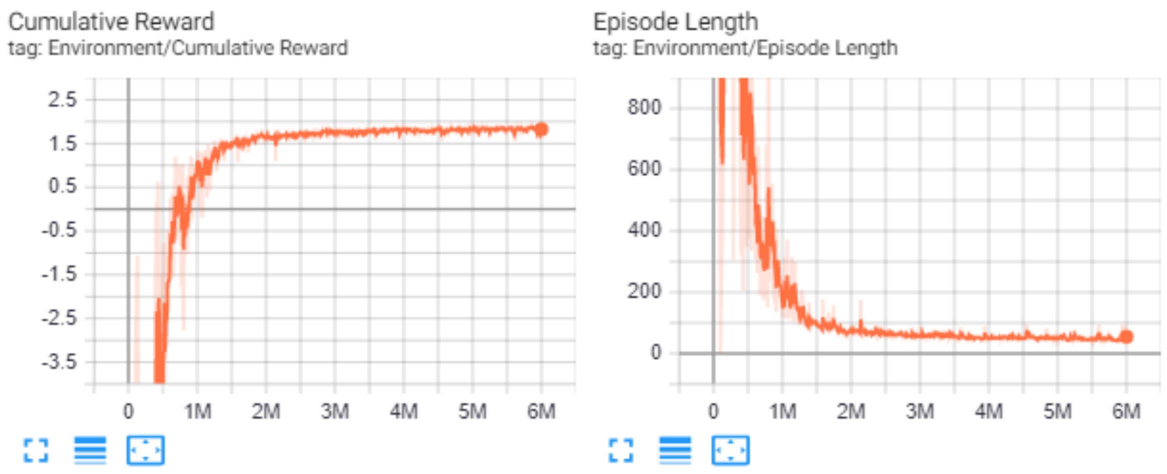


Figure 24 Splitting agents into zones

Compared to previous versions, the average number of steps taken per episode was less than 100 steps, about half of the average 200 steps of the others. The standard deviation is also noticeably less than the other versions. The training also converged faster, at about 1.5 million steps compared to 2 to 4 million steps in previous versions.

Upon visual observation, the agents mostly kept to their zones, with agents sometimes leaving all their zones after searching all the targets in their zones and ‘helping’ other agents search their targets. Moreover, although agents were trained with starting locations in their zones, agents will actively seek out their zone should they be spawned in a different one. For example, when all agents started in the centre of the board, they would actively spread out to their zone.

7.2.3 Version 7.2: Target Selection System (Collaboration, v3)

7.2.3.1 Environment

This version uses a 'selection system' where the agent was to choose a target and move towards it. The agent uses a **discrete** action space with 3 branches – 2 for movement (X and Z axis), and 1 for selecting a target. All targets and agents have numerical "IDs" -- integer variables assigned to each object. The agent can observe information about the targets, including the targets distance and angle relative to the agent and the targets search status (if the target has been searched). Agents have the following feature set:

Observation Space

- Agent's velocity in the x-axis
- Agent's velocity in the z-axis
- Agent's selected target (one-hot encoded)
- Distance between the agent and selected target (normalised)
- Angle between the agent and selected target (normalised)
- Status of the selected target (bool)
- Distance between the agent and all targets (normalised) (per target)
- Angle between the agent and all targets (normalised) (per target)
- Status of all targets (bool) (per target)
- Target ID (one-hot encoded) (per target)
- Other agents' selected targets (one-hot encoded)

Moreover, because of the stacked observations, agents can observe other agent's current targets and the previous targets for the last 10 steps.

Reward Structure

- A time penalty of <u>-0.0005</u> (per step)

- Generic target search reward of +4
- Selected target search reward of +13
- Target selection change penalty of -0.00001
- Selection of unsearched target reward of +0.0005 (per step)
- Reward for choosing a target not selected by other agents, or is the nearer agent (+0.00005) (per step)
- Variable reward dependent on the time taken for agents between targets (0.01 * (2000 – steps taken between targets)), with a minimum of 0.

In particular, the variable reward uses the formula 0.01 * (2000 – steps taken between targets). The number 2000 is largely arbitrary but has been chosen because it is the typical episode length midway during training.

7.2.3.2 Results

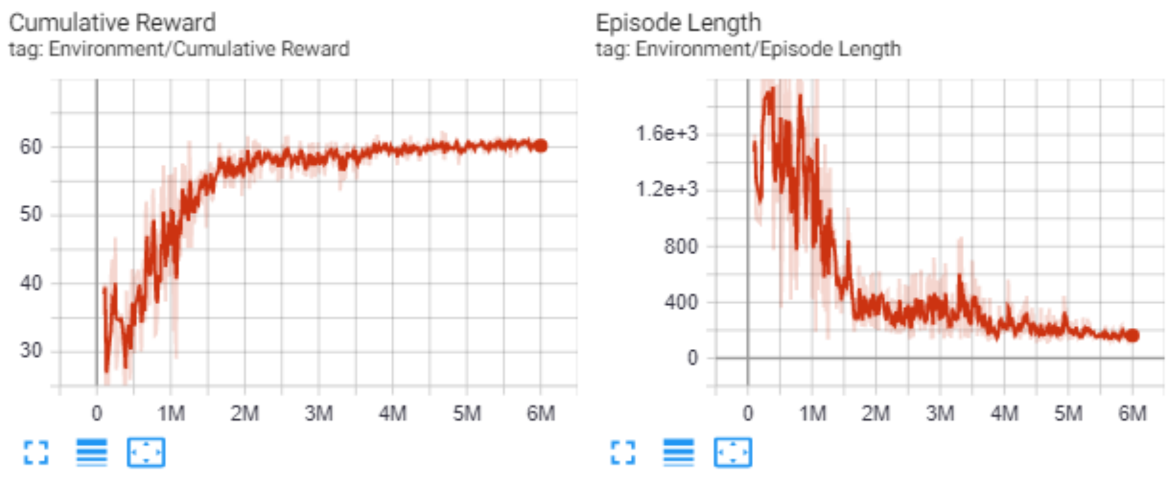


Figure 25 Target selection system, collaboration with the distance-based reserve system

The results were as expected, with a good learning curve and low standard deviation. The agent converges on a solution at around 1.5 million steps, which is markedly lower than the versions. The episode length is around 200 steps, like other versions.

Visual observation of the agent’s performance indicates improved collaboration between agents. Unlike other collaboration versions, agents do stay together when needed but do not search the same target at the same time. When two agents select the same target, the nearer agent will continue the target while the agent further away will select another target.

7.3 Overview of Approaches

In the baseline version, agents were given the information on the positions of the targets on the environment board. Even though agents in the baseline version performed well, a “greedy” approach was undertaken by the agents in this version, in which agents only seek for the closest target block with no consideration of the other agents’ movement.

In consideration of the issues encountered in the previous version, a crowding penalty was introduced, with a marked improvement of spacing between the agents themselves. However, it was noticed that even though agents are spaced out from one another, there were instances where agents were heading towards the same target. There is thus a need to ensure agents are efficient by making them head towards different targets.

A solution to ensure agents head towards individual targets is through zoning, where agents are split into four distinct zones and are encouraged to head for targets in their respective zones. While zoning provides good performance, there are many limits built into this version, one of which is that there are only 4 zones. Hence, hard-coded limits make it hard to expand to other use cases and thus undesirable.

Another approach to allow agents to head towards different targets is by allowing the agents to select the targets with relevant information of themselves and the targets around them. This version ran well, but there were instances where agents tend to erratically swap around their intended targets.

7.4 Final Implementation

To test out if the models can perform well in the real-world scenario, the environment was expanded to 250 units by 250 units. The environment was also given realistic features through appropriate colours and skins.

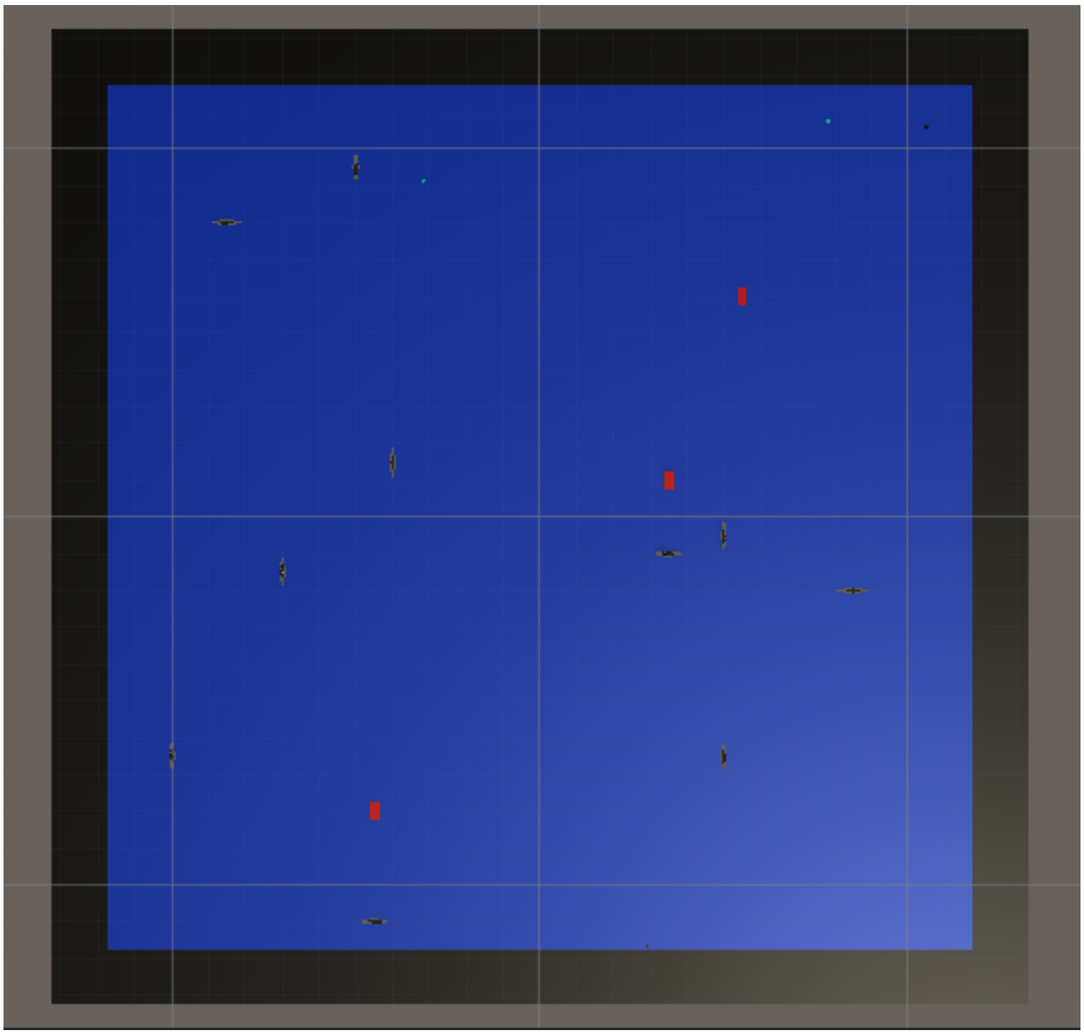


Figure 26 Figure of the test environment, with reskinned targets and colours for the environment

With the different model implementations stated previously, two types of simulations are then run with the agents: agents which spawn together at the centre of the environment, and agents which spawn at a corner of the environment.

PS08 Reinforcement Learning for Search and Patrol

The following table describes the results achieved and their respective standard deviation:

Agents spawn at the centre of the environment board

Model Method	**Mean Episode Length**	**Standard Deviation**	**Ranking**
Baseline	8883	2342.69	2
Zoning	9317	2047.5	3
Crowding	9453.6	1639.2	4
Target Selection	7372.8	3106.225	1

Agents spawn at the corner of the environment board

Target Selection	**Target Selection**	**Target Selection**	**Target Selection**
Baseline	8852	2528.48	3
Zoning	10000	0	4
Crowding	8334.3	2553.722	2
Target Selection	4731.2	3693.67	1

From the table of results, it can be observed agents perform best when trained with the target selection method, with it performing the best in both spawn patterns. Furthermore, other collaboration methods generally performed well when compared to the baseline method, with the collaboration methods performing close to the baseline method when these agents are spawned at the centre of the board, and the crowding method performing better than the baseline method when agents are spawned at the corner of the environment board.

8. Future Plans

Through this project, methods of inter-agent collaboration were explored and implemented.

One possible implementation to introduce inter-agent collaboration would be through a quantifiable variable to describe the "collaborative spirit" of the agents, and henceforth utilise curriculum learning to train the agents. This could allow agents to learn more effectively, as curriculum learning ensures agents can pick up concepts in basic operations before moving on to more complicated operations.

Another implementation would be through self-play, where agents are encouraged to improve itself by playing against itself. Through explicit means, agents would learn to split themselves to pick up targets in a more cohesive manner and would thus lead to further performance gains.

The application used in this project is Unity, a game development engine. Applications of the collaborative approach in a search and patrol context could include robotics, where agents would be tasked to generate appropriate waypoints to allow for effective pathfinding in each environment area.

9. Conclusion

In this FYP, we have conducted a literature review and identified a platform to simulate reinforcement learning environments. Concepts relating to reinforcement learning were used in understanding the training agents undergo. Unity ML-Agents was used to create the environment for training agents.

This report outlined various methods of collaboration and compared them with each other and with a non-collaborative model. Experimentation demonstrates that ‘target selection’ vastly outperforms other methods of collaboration and the baseline, with it taking significantly fewer decisions to complete the environment. In a sense, ‘target selection’ is a true collaboration – agents identify other agents’ intended target to search and can then act on that information. Other methods of collaboration do show some improvement in specific scenarios compared to a non-collaborative model but do not consistently perform better overall. Hence, the best method for agents to collaborate in this environment is to communicate their intent via selecting targets.

10.Bibliography

[1] BBC, “Singapore profile - Timeline,” BBC, 10 Msy 2018. [Online]. Available: https://www.bbc.com/news/world-asia-15971013. [Accessed 10 May 2020].

[2] World Shipping Council, “Top 50 World Container Ports,” World Shipping Council, [Online]. Available: http://www.worldshipping.org/about-the-ind. [Accessed 10 May 2020].

[3] Y. Li, “Deep Reinforcement Learning,” 15 October 2018. [Online]. Available: https://arxiv.org/pdf/1810.06339.pdf. [Accessed 4 April 2020].

[4] OpenAI, “Part 2: Kinds of RL Algorithms -- Spinning Up documentation,” OpenAI, 2018. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html. [Accessed 5 April 2020].

[5] R. S. Sutton and A. G. Barto, Reinforcement Learning - An Introduction (Second Edition), Cambridge, Massachusetts, United States of America: The MIT Press, 2018.

[6] L. Weng, “A (Long) Peek into Reinforcement Learning,” 19 February 2018. [Online]. Available: https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html.

[7] J. Schulman, F. Wolski and P. Dhariwal, “arXiv:1707.06347,” 28 July 2017. [Online]. Available: https://arxiv.org/pdf/1707.06347.pdf. [Accessed 6 April 2020].

[8] S. Kullback and R. Leibler, “On information and sufficiency,” *Annals of Mathematical Statistics,* pp. 79 - 86, 1951.

[9] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel and S. Levine, “arXiv:1812.05905v2: Soft Actor-Critic Algorithms and Applications,” 29 January 2019. [Online]. Available: https://arxiv.org/pdf/1812.05905.pdf. [Accessed 6 April 2020].

[10] OpenAI, “Soft Actor-Critic -- Spinning Up documentation,” OpenAI, 2018. [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/sac.html. [Accessed 10 April 2020].

[11] Y. Bengio, J. Louradour, R. Collobert and J. Weston, “Curriculum Learning,” in *International Conference on Machine Learning*, Montreal, 2009.

[12] L. Weng, “Curriculum for Reinforcement Learning,” Lil'Log, 29 January 2020. [Online]. Available: https://lilianweng.github.io/lil-log/2020/01/29/curriculum-for-reinforcement-learning.html#curriculum-through-distillation. [Accessed 26 July 2020].

[13] M. Shanker, M. Hu and M. Hung, “Effect of data standardization on neural network training,” *Omega,* vol. 24, no. 4, pp. 385 - 397, 1996.

[14] Codecademy, “Normalization,” Codecademy, [Online]. Available: https://www.codecademy.com/articles/normalization. [Accessed 29 May 2020].

[15] E. Kreyszig, Advanced Engineering Mathematics (Fourth ed.), Wiley, 1979.

[16] dansbecker, “Using Categorical Data with One-Hot Encoding,” Kaggle, [Online]. Available: https://www.kaggle.com/dansbecker/using-categorical-data-with-one-hot-encoding. [Accessed 29 May 2020].

[17] OpenAI, “Gym,” OpenAI, 2016. [Online]. Available: https://gym.openai.com/docs/. [Accessed 15 April 2020].

[18] Unity Technologies, “ML-Agents,” Unity Technologies, 8 April 2020. [Online]. Available: https://github.com/Unity-Technologies/ml-agents. [Accessed 15 April 2020].

[19] Microsoft, “Project Malmo,” Microsoft, July 2016. [Online]. Available: https://www.microsoft.com/en-us/research/project/project-malmo/. [Accessed 15 April 2020].

[20] Facebook, “Mazebase,” Facebook, 7 Jan 2017. [Online]. Available: https://github.com/facebookarchive/MazeBase. [Accessed 15 April 2020].

11. Acknowledgements

We would like to thank Mr Lim Ching Yang and Mr Poh Chun Siong in providing us guidance throughout this project, and for rendering us assistance. We would also like to give thanks to DSTA for giving us this unique opportunity to pursue an interesting project with real-world applications.

12. Appendix A: Project Materials

The project materials used in this report can be found on GitHub via this link:

https://github.com/randomwish/FYPReinforcementLearning

13.Appendix B: All Versions

13.1 Non-Collaborative Agents

13.1.1 Version 1: Basic Environment

13.1.1.1 Environment

Figure 27 First version of the environment

The first version of the environment was relatively feature-light. Agents were given a reward when they hit a target; the target then disappears, and the agents set off to find the next target. The key features are

- 1 agent

- 5 targets
- No obstacles

To spawn obstacles, we simply generated random coordinates relative to the board, without regard for any other target. This can lead to targets spawning in the same location as each other. However, due to the small number of targets and large space, this problem did not occur often.

13.1.1.2 Results

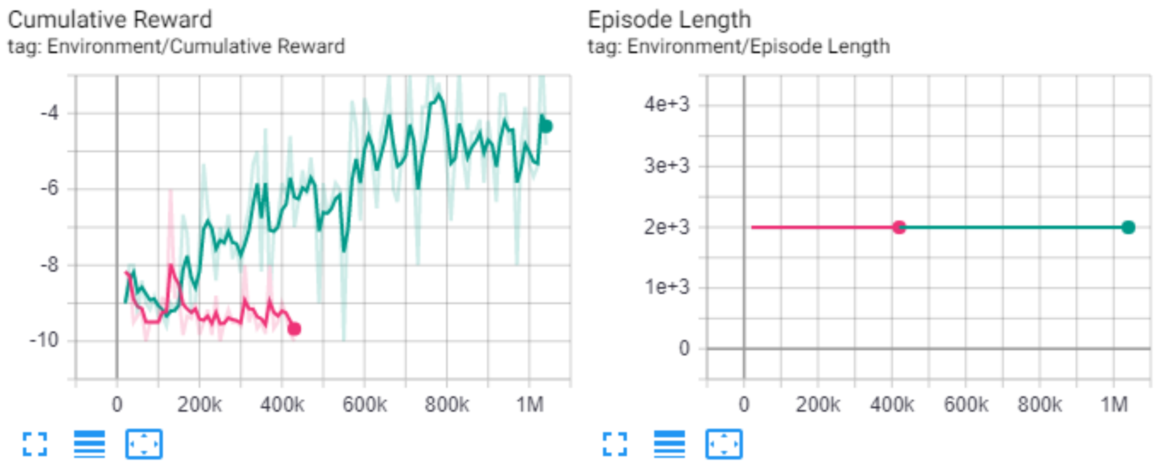


Figure 28 Training results from the first session

The figure above shows two training sessions ran on this environment. The first session (in pink) trained for 500k episodes and the second one, in green, for 1.04M episodes.

In the first training run, the agent did not appear to learn at all. The cumulative reward did not increase and the agent's behaviour in was very erratic. The agent tended to move towards the top right-hand corner of the board, which we have no explanation for.

Moreover, the episode length remained constant at 2k epochs. The episode will only end via 2 conditions – if all the targets have been found, or the maximum step has been reached. As shown in the figure above, the episode length did not decrease throughout both training sessions and remained at its maximum value, indicating that no episode was completed. This strongly implies that the agent is not able to complete the desired task.

13.1.1.3 Analysis

The agent had an extremely difficult time finding and navigating towards the targets. The agent often circled the target, trying to get towards the target but rarely reaching.

This is likely because agents were not reaching targets often enough. The targets were very small and sparse thus making it unlikely that the agent will randomly collide with a target and receive a reward. This makes it harder for the agent to learn as they do not receive sufficient rewards.

13.1.2 Version 1.1: Basic Environment, with Bugfixes

13.1.2.1 Environment

This version built on the observations we made in the previous version. In particular,

- Targets were made bigger, from a scale of 1 to 3
- The number of targets increased from 5 to 15
- The number of agents increased from 1 to 3
- Decreased the time penalty from 0.01 to 0.001

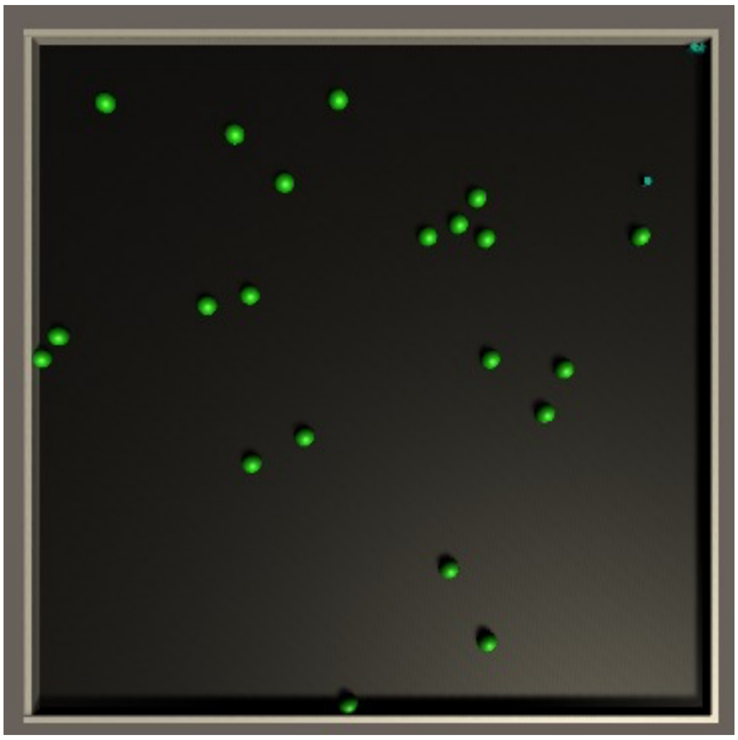


Figure 29 Version 1.1

Minor bug fixes notwithstanding, everything else remained the same.

13.1.2.2 Results

As expected, this went better than the previous version. The changes are immediately noticeable, as shown in the figure below.

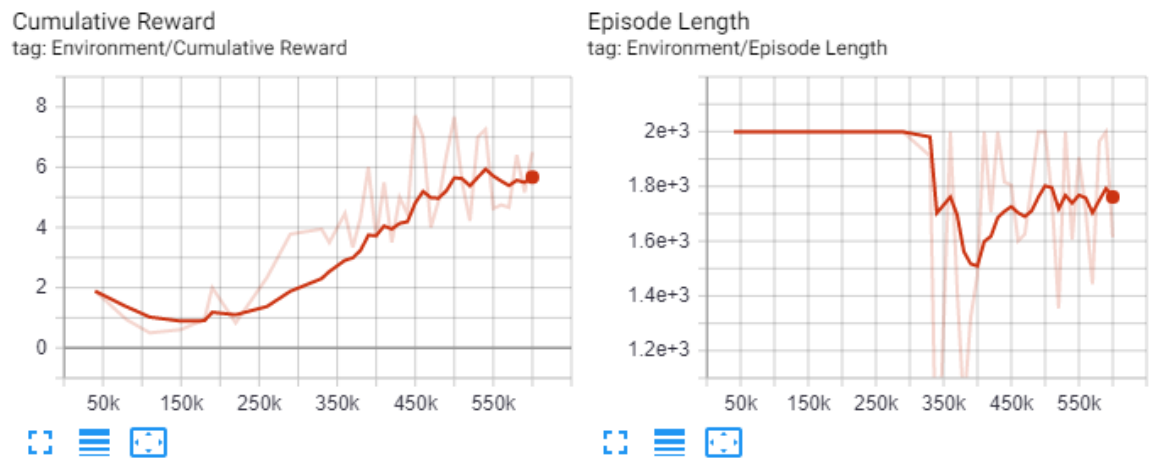


Figure 30 Training results from the first training session

The cumulative reward did increase continually, and the episode length decreases over time. This indicates that the agents are both learning and able to complete the episode before the max step.

Although the performance looked good on paper, the agents did not perform as well when compared to the application it is being developed for. Rather than learning to locate targets and moving towards targets, the agents learnt to perform a "sweeping" motion where they would move at extremely high speeds from fall to wall. Moreover, they would often do this without regard to the target's positions. The agents would repeat this behaviour until all the targets were cleared.

We originally thought that the model was simply not being trained enough, hence we tried training the model for drastically more episodes. Rather than 600k, we increased it to 3 million, and when that did not work again, 30 million then 90 million. As it can be observed in the figure below, the rewards more or less stagnated after 600k.

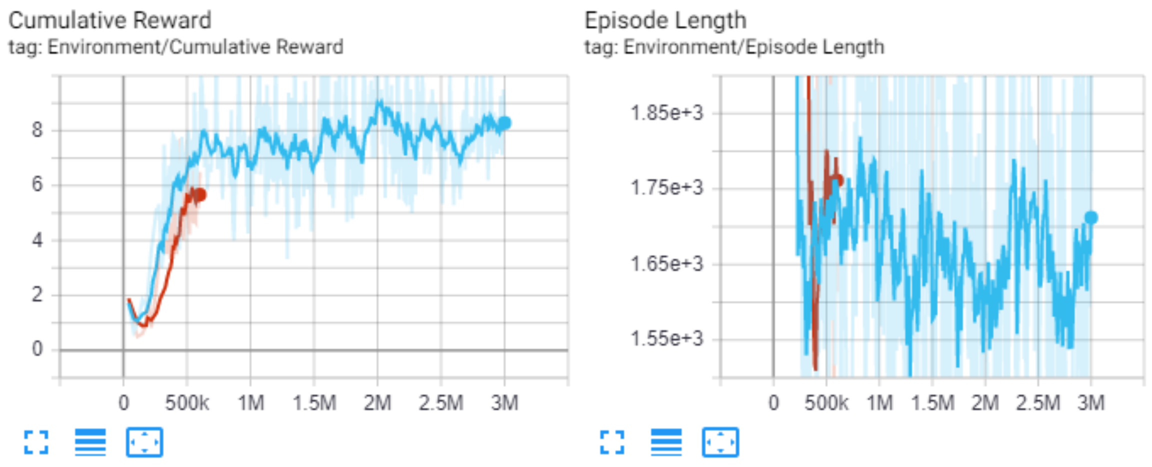


Figure 31 3 million episodes compared with 600k episodes

Next, we reduced the time penalty and decided to train longer. As it can be observed below, training to 30 million and 90 million did not affect the result drastically.

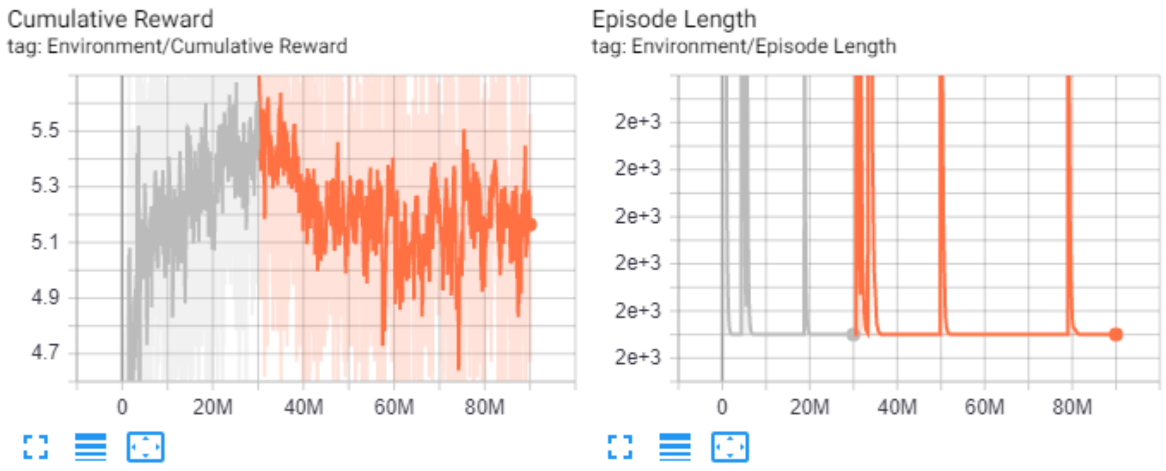


Figure 32 30 million and 90 million

It looked promising at around 30 million and we tried to train up to 90 million. However, it started overfitting the data and the performance of the model decreased. Furthermore, the standard deviation is still extremely high – this indicates that although the agent was learning marginally, but still not close to being reliable.

13.1.2.3 Analysis

As described in the results, although the agent learned, it did not produce an optimal result. The agent learned to increase their chances of finding targets by increasing their area covered (via sweeping) rather than being able to locate other agents

Eventually, the cause of this was determined to be from the movement engine. Originally, we used an AddForce function to add a force on the rigid-body. This works well in a small environment. However, our environment size is relatively large, and performing movement in this way causes the agents to move at exceptionally high speeds with very little control. This often results in the agent circling a target for a very long time and only reaching the target after great difficulty.

```
Vector3 controlSignal = Vector3.zero;
controlSignal.x = vectorAction[0];
controlSignal.z = vectorAction[1];
rBody.AddForce(controlSignal * speed);
```

In a nutshell, this applies a force to the agent's rigid body, which is decided by the model. This works well in a small environment, e.g. RollerBall, where the environment is about 1 by 1. When scaled up, however, the force is applied to the rigid body over an extended period. As described in Newton's second law of motion, the force results in acceleration which increases the agent's speed. In a large environment, this force can result in very high speeds and make it very difficult to control the agent. Moreover, since the force applied is constant, it would take the same amount of force, and therefore steps, to brake or change direction. In all, it makes it very difficult for the agent to control itself, much less navigate to targets.

13.1.3 Version 2: Updated Movement Engine

13.1.3.1 Environment

In this version, we changed the movement engine to give the agent more control over its movements. Rather than adding a force to the rigid body, the agent's position is transformed into the next position. This increases the agent's control of its movements and prevents the problems outlined above.

```
    // Version 2 movement engine
Vector3(hAxis, 0, vAxis) * moveSpeed * Time.deltaTime;
    m_AgentRb.MovePosition(transform.position + movemvent);
```

This functioned much better and allowed the agent to train as intended. Visually, it remained the same as before.

13.1.3.2 Results

As expected, the results for this version is a lot better.

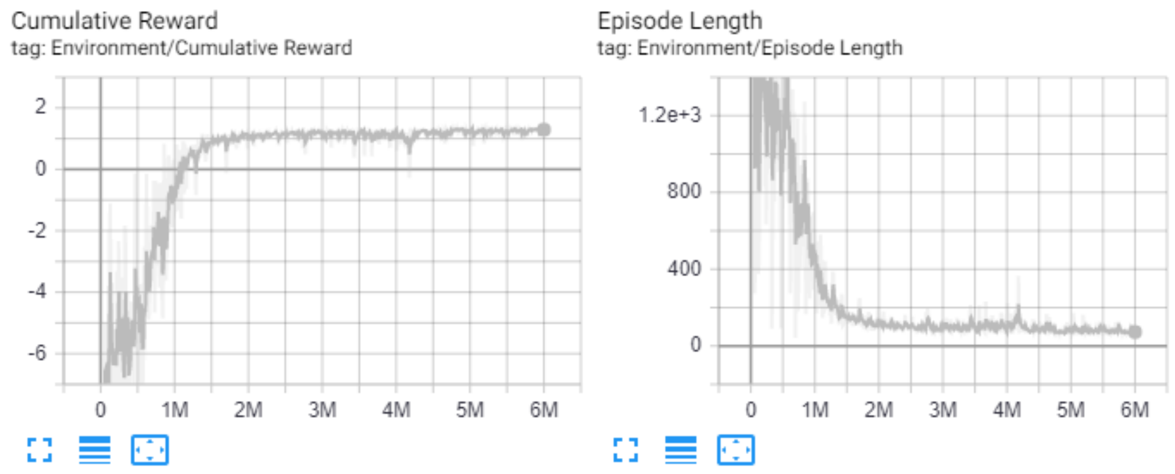


Figure 33 Version 2 training results

The training results in a much cleaner graph, with the agent learning substantially till about 1.6M episodes where it then flattens out. The cumulative reward and episode length graphs are directly related and indicate that the agents can find all targets and finish the episode consistently.

Via visual observations, the agent is seen as being able to find targets consistently using its Ray Perception Sensor and move towards the target.

13.1.3.3 Analysis

Overall, this version had been successful, and the agents were able to complete the assigned tasks without any problems. Admittedly, training could be much shorter and could have stopped at 2M rather than 6M.

13.1.4 Version 3: Added Obstacles

13.1.4.1 Environment

After the positive results in version 2, the environment was updated to include more elements in version 3. Obstacles were also added. The key features in this version are

- 3 agents
- 15 targets
- 5 obstacles

Additionally, it was designed such that the targets do not disappear after the agent collides with it but instead changes colour from green to yellow. The environment cannot spawn targets in the same location twice, hence preventing targets from spawning on top of each other.

The agent's hyperparameters were also changed to increase its performance.

- Increased Ray Perception Sensor range from 135 degrees to 180 degrees
- Increased Ray Perception Sensor rays per direction from 7 to 14
- Included Ray Perception Sensor detectable tags "agent", "obstacle", "searchedTarget" (originally was just "target")

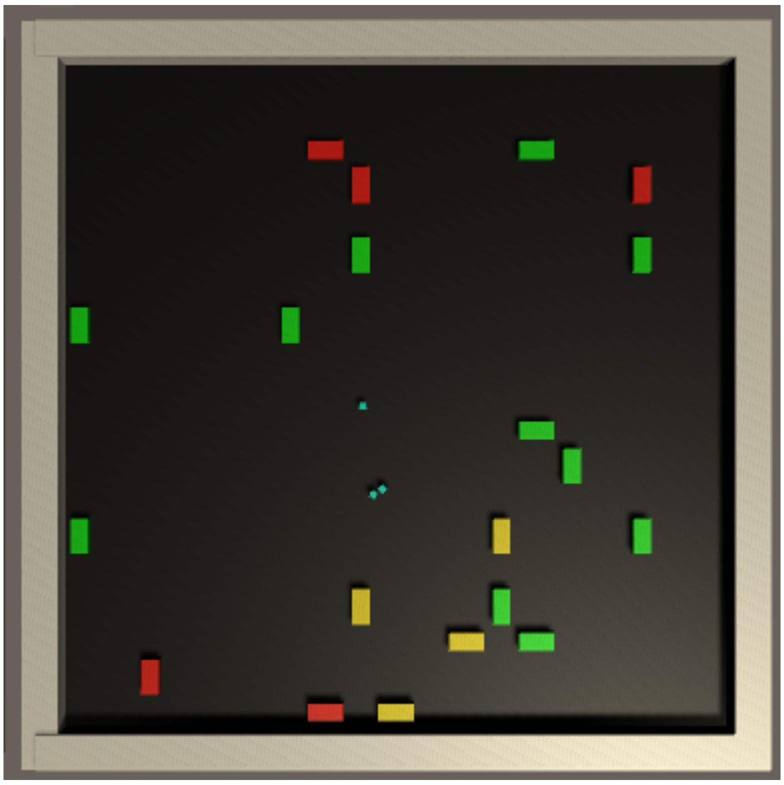
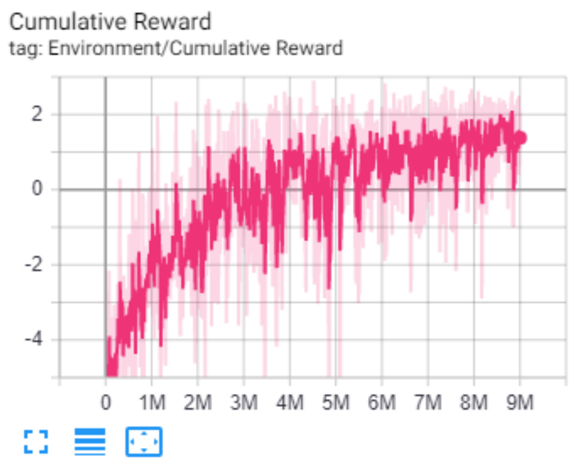


Figure 34 Version 3 environment

13.1.4.2 Results

For this version, we tried two different reward structures: with an obstacle penalty, and without. The reasoning behind the obstacle penalty was to incentivise the agent to avoid obstacles, hence saving time.

With obstacle penalty



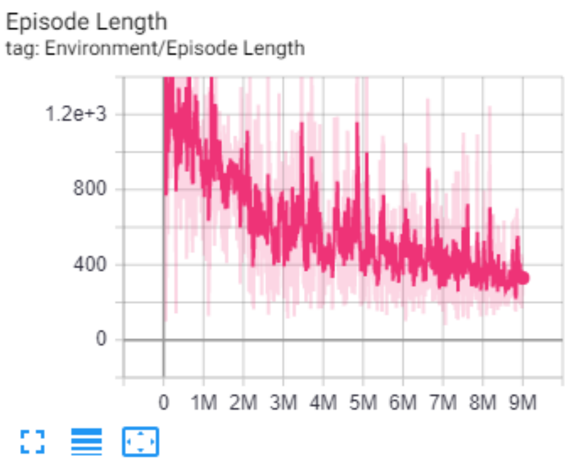


Figure 35 Version 2, with obstacle penalty

The obstacle penalty did not work as expected. The agents tended to do more to avoid hitting obstacles than actively seeking out targets. For example, if there was a target and an obstacle near each other, the agent would choose not to go close or take a long time to go close to the target. This also leads to inconsistent results, as the agent's performance is would be affected when agents spawn at unfavourable locations. Hence, we tried again but without the obstacle penalty.

Without obstacle penalty

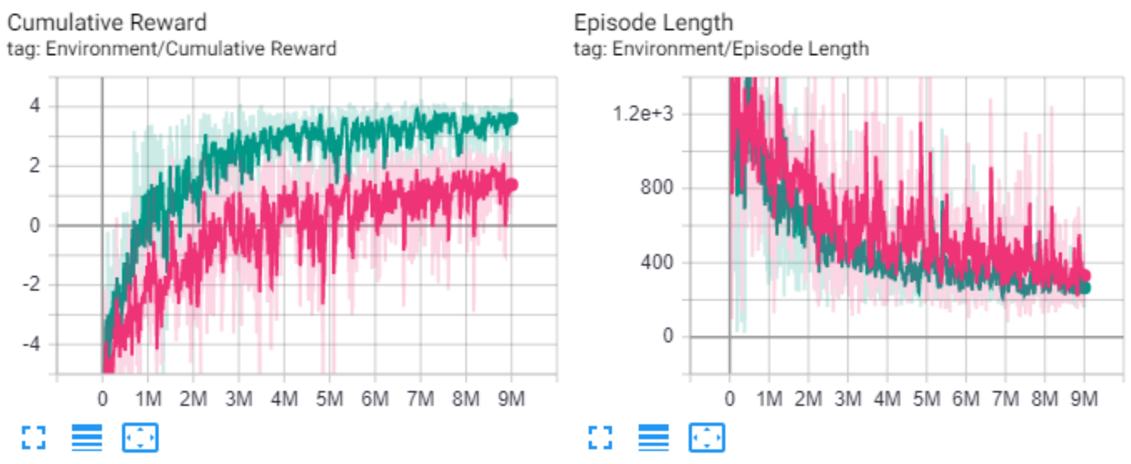


Figure 36 With obstacle penalty (pink) and without (green)

When comparing the graphs, some differences can be observed. The most noticeable one is the higher overall reward of the green line (without obstacle penalty) – this is because of the lack of obstacle penalty; hence the total reward of each episode is higher. This is expected.

Rather than directly comparing the cumulative reward, a better way to compare the effectiveness of the reward structure is the episode length. From the figure, it is evident that the training session without obstacle penalty takes fewer steps to complete its task thus proving that it is more efficient.

Another point to note is that the standard deviation of the green graph is noticeably less than the pink. This value is after smoothening of 0.65. This indicates that the performance of the agent is more consistent without the obstacle penalty.

13.1.4.3 Analysis

Overall, the model performs well on both counts. However, the difference of having a reward penalty for obstacles changes the agent's behaviour quite substantially. With the penalty, the agent acts more conservatively and attempts to avoid the obstacles at all costs. Without, the agents get more aggressive and ignores obstacles completely. This does make it faster, but sometimes crashing into obstacles is not realistic – or rather, optimal – behaviour for an AUV. For now, the more consistent model (without obstacle penalty) was used.

When running the model, the agents noticeably have trouble finding the last few targets, particularly when the targets were not in the agents' field of view, i.e. when hidden behind obstacles or searched targets. Hence, this can result in a high standard of deviation in the cumulative reward and episode length – because the agents may have difficulty during finding the last target, the episode length can vary greatly.

13.1.5 Version 4: Updated Observational Space (Single Target)

13.1.5.1 Environment

This version focused on increasing the agent's observational space by allowing the agent to observe the target's locations. This was done in two ways – allowing the agent to observe the distance and angle between the agent and the target and the coordinates of the target and the coordinates of the agent. In this version, the target which the agent observes is the target closest to it. This is chosen programmatically, then passed into the agent's observation space.

Moreover, the observations were collected first without normalisation, then with normalisation. In all, there are 3 different training runs with different observation spaces:

- Distance and angle between target and agent (not normalised)
- Coordinates of target and agent (not normalised)
- Distance and angle between target and agent (normalised)

The environment itself was not changed significantly from the previous version.

13.1.5.2 Results

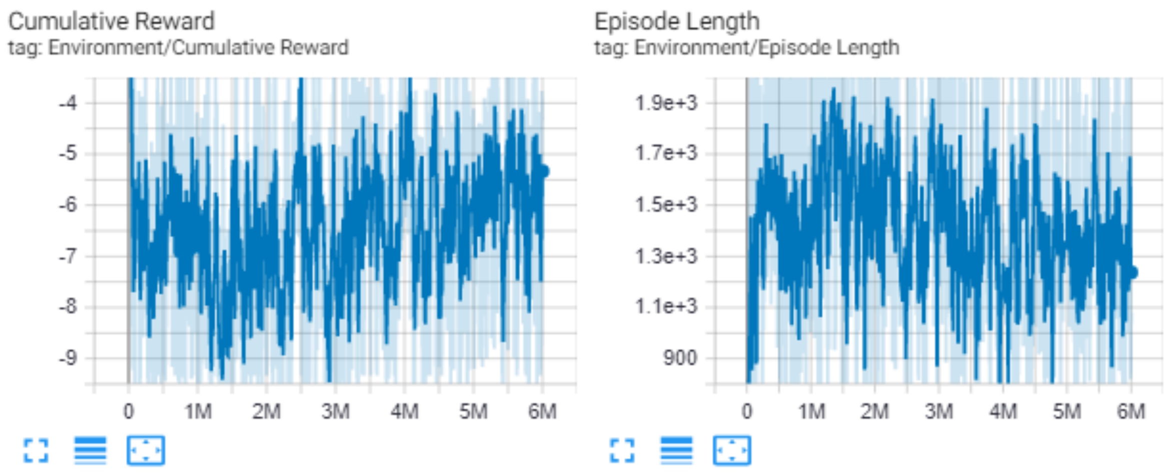


Figure 37 Distance and angle, without normalisation

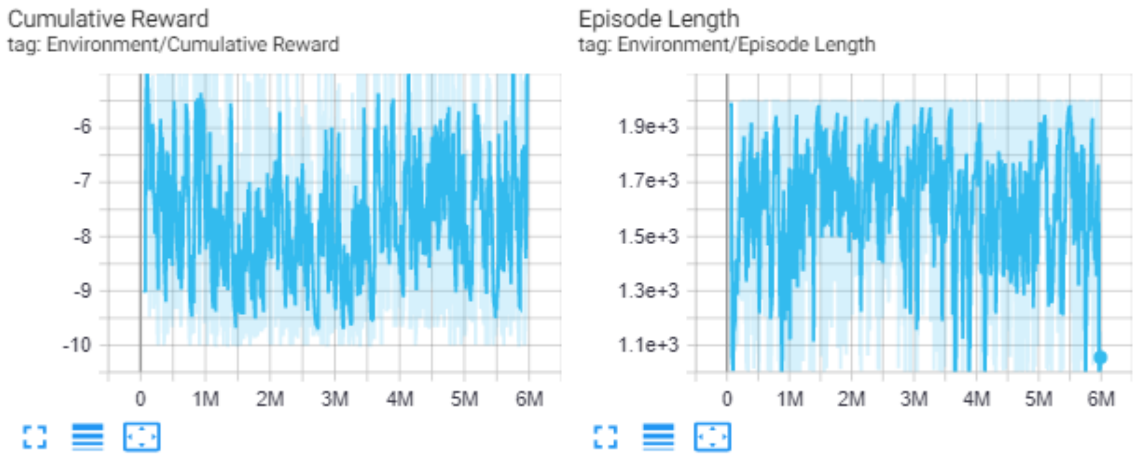


Figure 38 Target location, without normalisation

The cumulative reward and episode length graphs have a very high standard deviation with little learning exhibited. There was some indication of learning with the agent learning to move, but the agents were not able to navigate towards targets reliably. Moreover, the agents often appeared confused and moved about semi-randomly.

Both the models using target locations and the distance and angles performed similarly.

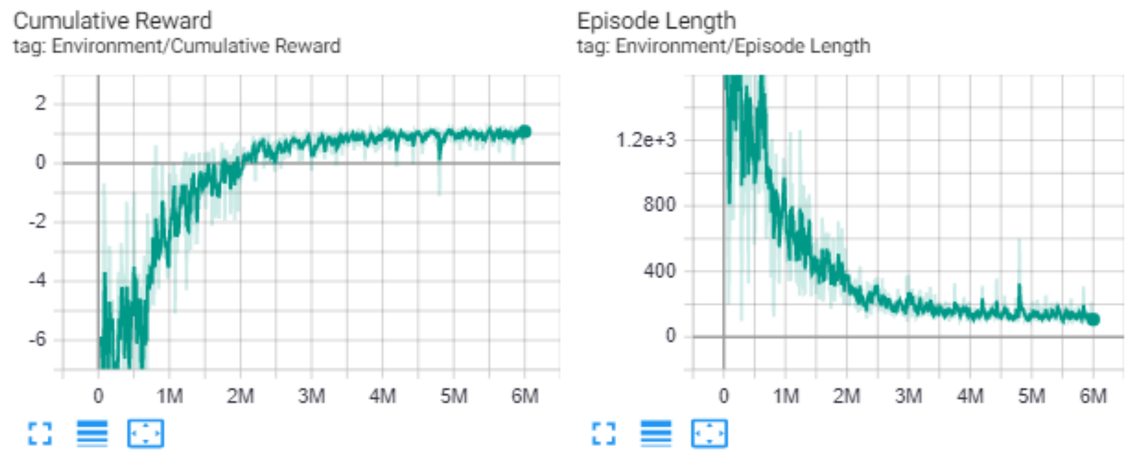


Figure 39 Distance and angle, with normalisation

After normalising the result, the agent performs remarkably better. After around 3 million steps, the reward and episode length plateaued, signifying that the agent was able to train sufficiently. The standard deviation is noticeably low, indicating that the agents can consistently find and search all the targets.

13.1.5.3 Analysis

Normalisation played an important part in the difference in performance between models; nothing else was changed between the runs using the distances and the angles. This is likely because the values were relatively high – up to 360 degrees, over 100 units of distance or coordinates. This is in stark comparison to other unnormalized observations, i.e. the agent's velocities were only single digits. This likely made it difficult for the neural network to learn hence made it difficult for the agent to learn.

The agent's performance was remarkably better than the previous version, particularly the large difference in standard deviation – when the agent can observe the normalised value of the nearest target to it, it can find and search all targets much more consistently. The high standard deviation in version 3 was mostly due to the agents being unable to find targets hidden behind searched targets or obstacles. In this version, the agents know the distance and the angle to the nearest unsearched target, allowing the agent to find the targets that it cannot see with raycasting more easily.

However, the target fed to the agent's observation space is always the nearest one. This is chosen programmatically. While it does not restrict the agent to move only towards the nearest target, the agent only gets the information of the nearest target. This only taught the agent to move towards the nearest target, rather than learning the best way to move to optimise target searching.

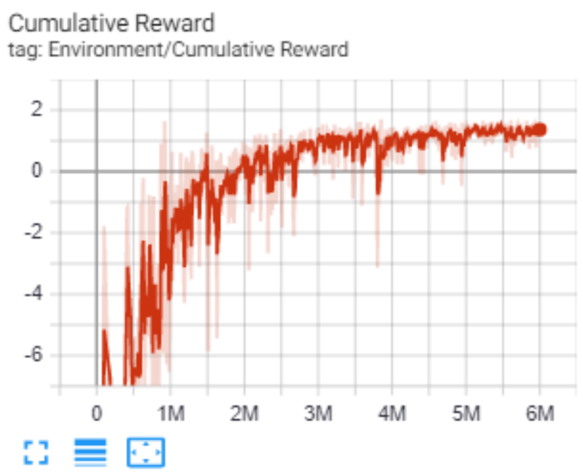
Although both the locations of the target and the distance and angle of the targets both had comparable performances without the normalisation, the observation space of the distance and angle between the target and the agent was chosen to because it was simpler for agents to learn and to code.

13.1.6 Version 4.1: Updated Observational Space (All Targets)

13.1.6.1 Environment

This version is similar to the previous version, but with the information of all targets added to the target observation space rather than just the nearest target. After the previous version, the normalised value of the distance and angle between the targets and the agent was used.

13.1.6.2 Results



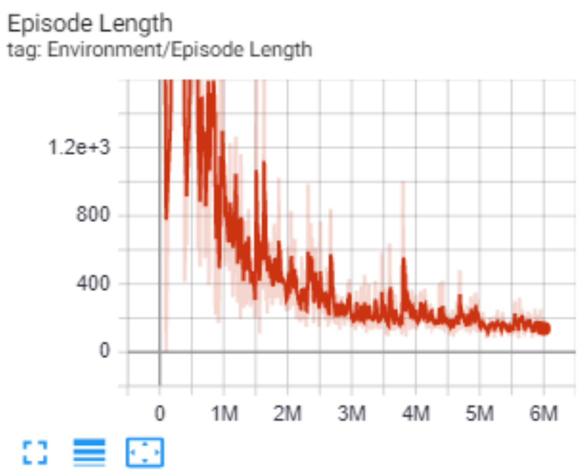


Figure 40 All distances and angles between all targets and the agent

The general shape of the graphs shows a good learning curve, with the learning plateauing at about 3 million steps.

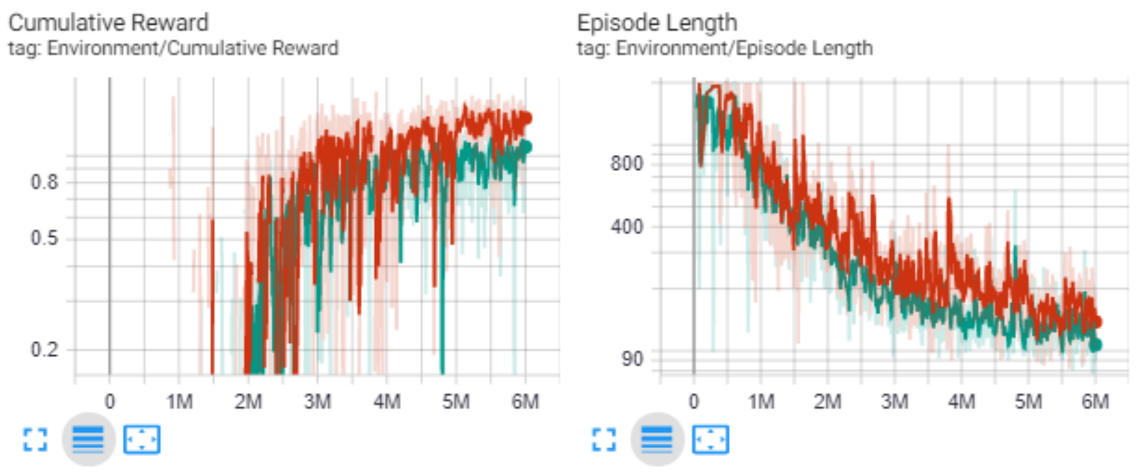


Figure 41 Log scale comparison of a single target (green) and all targets (brown)

When compared to the single target version, it is notable that the agent takes slightly longer to complete episodes when the agent observes all targets. Moreover, the standard deviation was slightly higher with all targets version compared to the single target version. This indicates that the agent's performance when observing all targets were slightly worse than when only observing one.

13.1.6.3 Analysis

This version has a lower performance than the previous version. This is likely due to more information the agent has access to, rather than the less information which the single target agent had access to. Moreover, the single target agent's target is the 'optimal' target, with the nearest target always being selected. However, when all targets are observed, the agent can choose whichever target. As there was no additional incentive but the time penalty for agents to choose the nearest target, the agent may not have placed as high importance in choosing the nearest target. As such, the agent may not have learnt the 'optimal' target to navigate to.

13.1.7 Version 7: Target Selection System (Non-Collaborative)

13.1.7.1 Environment

This version added a ‘selection system’ where the agent was to choose a target and move towards it. Unlike previous versions, the agent uses a **discrete** action space with 3 branches – 2 for movement (X and Z axis), and 1 for selecting a target. The speed of the agent was also decreased, from a multiple of 20 to 15 because of complications during training. The agent is not explicitly disallowed to search other targets but will have a significantly less of a reward than if it searches the selected target. All targets and agents have numerical “IDs” -- integer variables assigned to each object. The agent can observe information about the targets, including the targets distance and angle relative to the agent and the targets search status (if the target has been searched). In all, the observation space includes:

- Agent’s velocity in the x-axis
- Agent’s velocity in the z-axis
- Agent’s selected target (one-hot encoded)
- Distance between the agent and selected target (normalised)
- Angle between the agent and selected target (normalised)
- Status of the selected target (bool)
- Distance between the agent and all targets (normalised) (per target)
- Angle between the agent and all targets (normalised) (per target)
- Status of all targets (bool) (per target)

The reward structure is significantly different from previous versions. Before, the reward structure was quite simple – a reward for searching a target, a time penalty for every step. For this version, a more detailed reward structure was required because of the larger action space. It is as follows:

- A time penalty of <u>-0.0005</u> (per step)
- Generic target search reward of <u>+2</u>
- Selected target search reward of <u>+10</u>

- Target selection change penalty of -0.00001 (per step)
- Selection of unsearched target reward of +0.0005 (per step)
- Distance penalty for choosing targets (-0.00001 * distance between agent and target) (per step)

These rewards are designed to incentivise the agent to select nearby agents and move towards said agents. In particular, the variable reward uses the formula 0.01 * (2000 – steps taken between targets). The number 2000 is largely arbitrary but has been chosen because it is the typical episode length midway during training.

13.1.7.2 Results

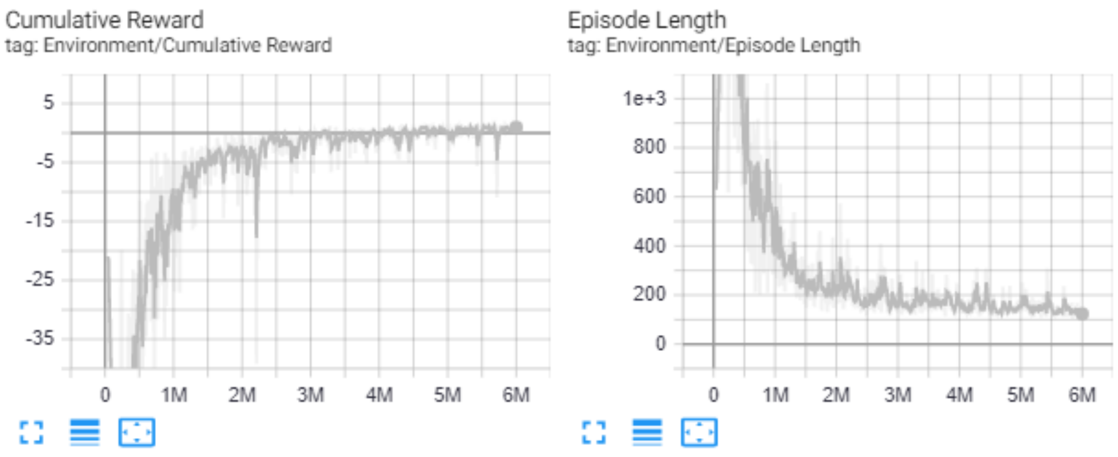


Figure 42 Selection, non-collaboration

The graphs show a good training curve, with the graphs plateauing at around 3 million episodes. The standard deviation is not large, with agents able to consistently able to find all targets. The average episode length is about 200 steps, which is similar to previous results.

When observed, the agents navigated targets around the board with relative ease, appearing more decisive than previous versions. Agents moved towards selected targets, sometimes ignoring targets nearer to itself to get to said selected target. Agents sometimes chose already searched targets, often when there were no unsearched targets near itself, and navigated towards it to attempt to search it before moving on to other targets. Agents often chose the same targets and moved towards them together. If two agents select the same target and one searches the target first, the other agent would give up and select another target.

13.1.7.3 Analysis

The results are comparable with other versions, with the average episode length around 200 steps and a relatively low standard deviation.

When observed in the Unity editor, agents often appeared more decisive and moved towards its intended target without getting 'confused' and deciding between other potential targets. This is likely due to the selection system, which incentivises agents to 'plan' which target to search rather than decide on the go. This can result in time saved, but the difference may be negligible in our current environment as agents move fast and the 'confusion' may be too little to have a significant impact on the episode length.

Agents tended to sometimes select already searched targets, especially when the targets nearby were already searched. This is likely due to a flaw in the reward structure. Selecting an unsearched provided a reward of +0.0005 units per step, while the distance penalty for choosing targets is calculated with -0.00001 * distance between agent and target, per step. Hence, should the agent choose a target that was more than 50 units away, the penalty of choosing targets more than 50 units away outweighs the reward of choosing an unsearched target and generated an overall penalty for that step, which the agent seeks to avoid. A possible solution to this problem is either increasing the reward for choosing unsearched targets to +0.071 or decreasing the penalty multiplier to -0.0.0000035 as the maximum distance across the board is 141.42 units. However, this presents a dilemma – increasing the reward may be too much and may overshadow other rewards, decreasing the penalty by such an extent may result in it becoming negligible. Although not changed in this version, future versions of the target selection method use a onetime reward for choosing nearer targets to prevent this problem.

In all, using a target selection system over a regular system presents a little benefit to the agent when used alone. Before, agents would move towards targets, often towards the targets nearest to itself. With the target selection system, agents often select targets which it would have moved towards even without the selection system; the selection's impact is negligible. Despite this, the selection could be used to communicate with other agents in future versions as a message of intent, which can allow an agent factor other agents' intended targets when choosing a target.

13.2 Collaborative Agents

Collaboration is fundamentally not a quantitative variable hence is difficult to directly quantify if collaboration is occurring. Hence, although curriculum learning would be suitable in the performance of the agents themselves, they would not be suitable in ensuring agents collaborate in accomplishing the task they are assigned.

This segment of the report will go through the four main approaches used to make the agents collaborate – allowing agents to check the distance between each other, penalizing the agents for staying close together, splitting the agents into zones, and a target selection system.

13.2.1 Version 5: Crowding Penalty

13.2.1.1 Environment

In this version, an alternative system of getting the agents to distance themselves from one another is explored. There are two versions which have been explored: an explicit approach in which agents are directly penalised for being too close with other agents, and an implicit approach in which agents are fed the locations of other agents.

In the explicit approach, agents are given a negative reward for being too close with one another. This incentivises the agents to spread out from one another. As part of our implementation, a negative reward of -0.001 is given to the agent if they are within 10 units from another agent.

In the implicit approach, a version where agents are fed the locations of other agents is also done, where normalised values of these locations are known to the current agent. This encourages the learning of agents to adapt to other agents' locations and is an indirect way of implementing the penalty itself.

13.2.1.2 Results

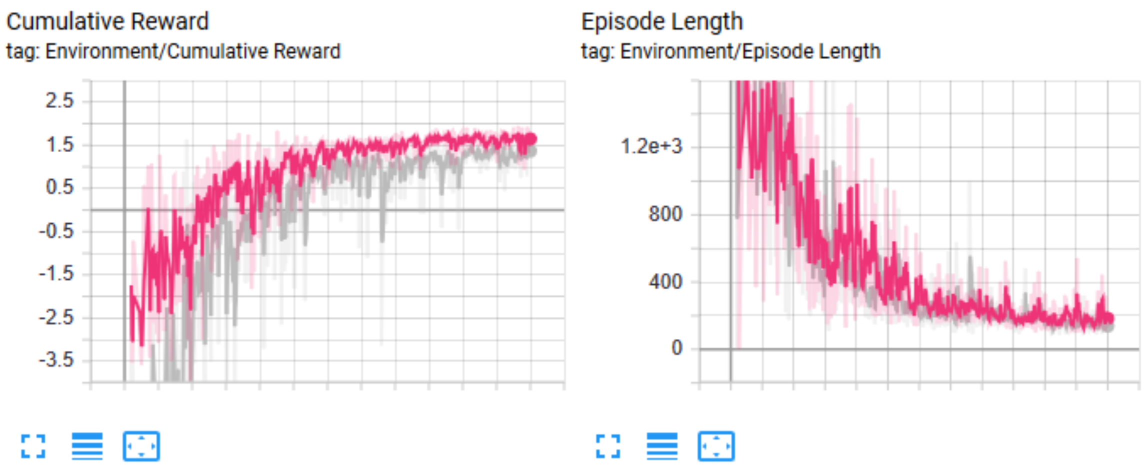


Figure 43 Results for explicit approach (in red) and implicit approach (in grey)

In the figure, both approaches can achieve the task of retrieving the targets in the environment to an episode length of 200-time steps. Furthermore, the rate at which both approaches reach the final episode length is similar.

However, the final cumulative reward for the explicit approach is marginally higher than that of the implicit approach, and this occurs throughout the training process. Moreover, there is a high standard deviation present in both cumulative reward and episode length, and that it occurs in the starting phase of training as shown by the graphs. But, as training progresses, the standard deviation decreases, showing the agent had a more consistent performance as time passes.

13.2.1.3 Analysis

Agents trained under both systems tend to spread out, with the agents trained under explicit approaches being more spread out. However, both systems result in multiple agents heading towards the same target, hence raising an issue with efficiency, as the time used for these agents to head towards a target together could have been better used to head towards other agents separately.

An explanation as to why the implicit approached fared worse than the explicit approach could be because agents are not taught that the given approach was intended, and therefore would not be inclined to distance themselves from one another. This lack of distancing means the agents took a longer period to reach targets which are further apart from one another, hence leading to a higher amount of time needed to complete an episode.

An approach to solve the issue of agents heading towards the same targets is to implement the inclusion of a "target selection system", where agents are explicitly instructed to head towards different targets.

13.2.2 Version 6: Splitting Agents into Assigned Zones

In this version, the agents are explicitly split into distinct zones in the environment, as seen in the figure below. Although this instance is split into 4 zones, the number of zones could be changed to whatever desired.

Agents are manually assigned to each zone. Agents are not fixed to individual zones programmatically and are free to move between zones. However, a reward of 0.001 and a penalty of -0.001 per step is applied if the agent is in and not in the zone assigned, respectively. Unlike other versions where agents spawn randomly throughout the board, the agents were spawned in the middle of their assigned zone.

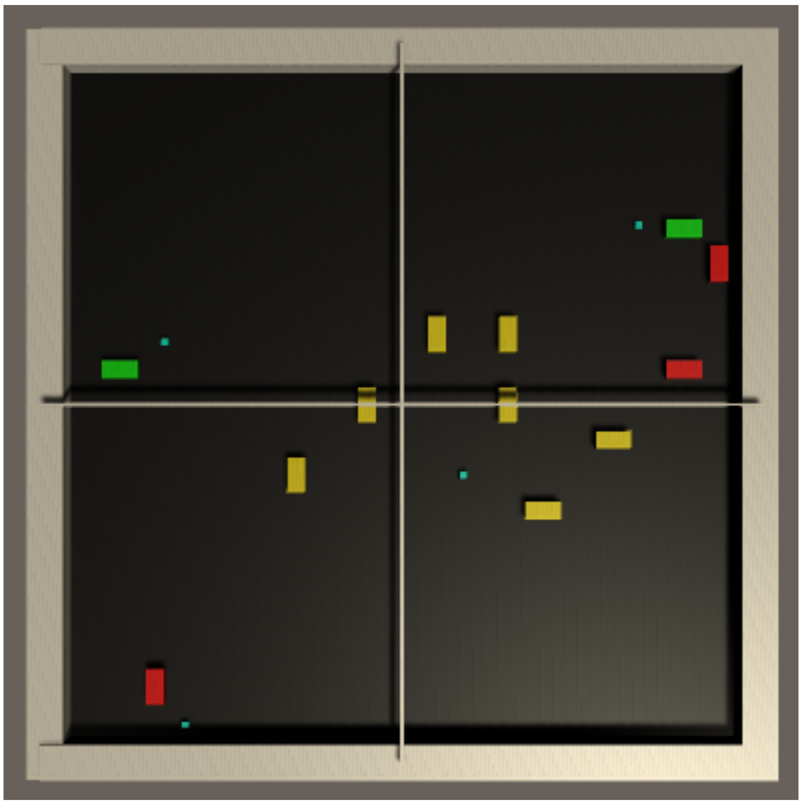


Figure 44 Agent zones environment

This version was tested both with the distance observation space from version 4.

13.2.2.1 Results

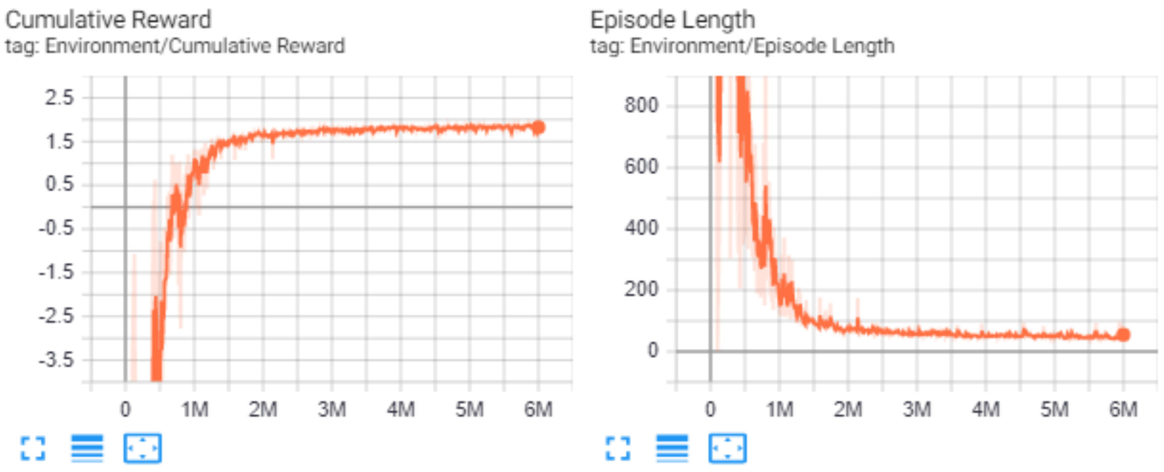


Figure 45 Splitting agents into zones

The results were remarkably clean, with a good learning curve plateauing at around 1.5 million steps and having very little standard deviation thereafter.

Compared to previous versions, the average number of steps taken per episode was less than 100 steps, about half of the average 200 steps of the others. The standard deviation is also noticeably less than the other versions. The training also converged faster, at about 1.5 million steps compared to 2 to 4 million steps in previous versions.

Upon visual observation, the agents mostly kept to their zones, with agents sometimes leaving all their zones after searching all the targets in their zones and ‘helping’ other agents search their targets. Moreover, although agents were trained with starting locations in their zones, agents will actively seek out their zone should they be spawned in a different one. For example, when all agents started in the centre of the board, they would actively spread out to their zone.

13.2.2.2 Analysis

In comparison to previous versions, both collaborative and non-collaborative, the trained agents performed much better than other agents. The episode lengths had a very low standard deviation indicating that the agents found all the targets consistently and reliably. Moreover, the significantly lower episode length of less than a hundred indicates that the agents completed the task quicker than other versions.

The lower episode length does show that the agents can search all the targets more efficiently. However, this is important to note that the number of agents in this version is 4, compared to 3 previously. Although this alone does not account for all of the change – the increase with an additional agent will not result in a performance increase of over 100% - it is important to keep the additional agent in mind when comparing the results directly.

Another point to note is the faster convergence, about 1.5 million episodes compared to previous versions where convergence learnt at about 3-4 million episodes. This indicates that the problem was fundamentally easier to solve. Hence, it is not surprising that the average episodes length and the standard deviation is much lower when compared to other versions.

Taking these factors into account, it is evident that spreading agents into zones and manually assigning them zones was beneficial for the problem. Moreover, the agents' ability to travel between zones to aid other agents demonstrated the agents were able to collaborate.

However, the manual placement and assignment of the agents make the environment much easier to solve. Hence, it is likely that this was specific to the situation and may not work well in others without heavy manual interference.

13.2.3 Version 6.1: Floating Zones

13.2.3.1 Environment

This version builds on the previous one, with the key difference being the lack of assignment for agents. As there are no fixes assignments, agents received rewards and penalties when in zones without and with agents, respectively.

Agents were also able to observe the distances and angles between themselves and other agents.

13.2.3.2 Results

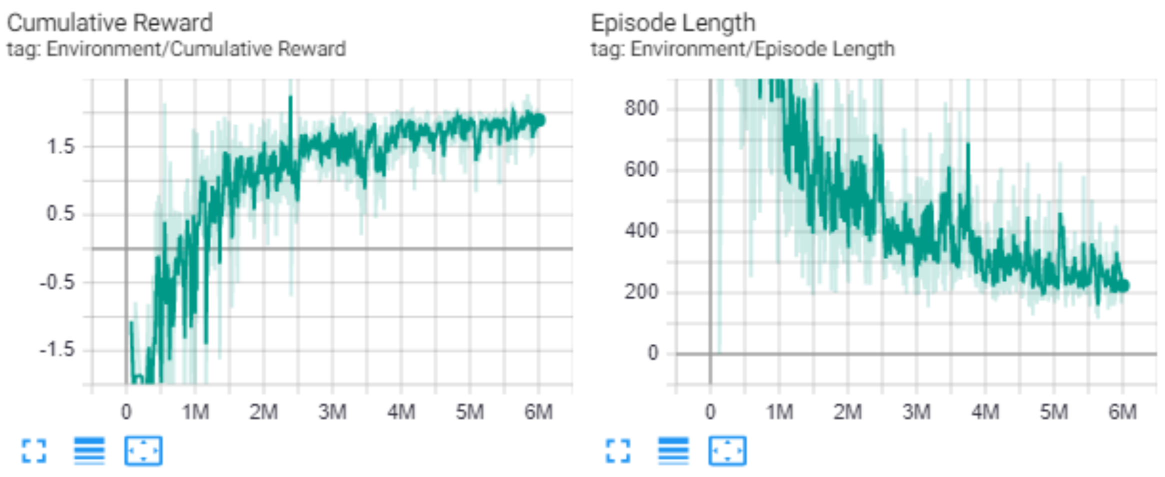


Figure 46 Floating zones

The learning curve was sufficient, with the episode length lowering quickly till about 3 million episodes, then more slowly afterwards. Moreover, the standard deviation for both cumulative rewards and episode lengths were quite high. This indicates the agent was able to complete the task, albeit inconsistently.

When observed, the agents tended to place a high priority in avoiding zones with other agents in it. Consequently, the agents often hovered around the edges of zones, switching between zones constantly. When another agent enters a zone with another agent already in it, the latter attempted to exit said zone rather than searching targets.

13.2.3.3 Analysis

Floating zones did not perform as well as assigned zones. Although requiring no manual interference, the agents were not able to sufficiently learn to optimally search targets, instead of learning to avoid being in the same zones as other agents. Although the agents have access to the distance and angle between itself and other agents, the agents could not relate the information to the zones. As a result, the agent could only make discrete observations, i.e. how many other agents are in the same quadrant as itself.

From visual observations, the agents often hovered around the edge of zones, moving between them. The high standard deviation and long episode time was directly a result of this behaviour. Agents focused on moving around zones rather than searching targets, especially when other agents moved into the zone that they were currently in, or if they moved into a zone that already had an agent.

The structure of the floating zones was not an ideal combination for collaboration. The discrete properties of zones forced agents to stay around the edges of zones, which allowed for agents to move between zones quickly but was not ideal for the agents to move further into the zones to search targets. This is in stark contrast to assigning agents zones, which encouraged them to search within their zones, then, if all targets were searched in their zone, agents could enter other zones to help search.

The difference in training could have affected the results in the zones. In the assigned zones version, the agents spawned in the middle of their zones while floating zones spawned agents randomly around the board. This could have affected the training significantly – the fixed spawns in the assigned zones encouraged agents to search all targets in their zone, but the random spawn in floating zones may have spawned multiple agents in same zones; agents may have been encouraged to escape the zones rather than search for targets.

13.2.4 Version 7.1: Target Selection System (Collaborative, v1)

13.2.4.1 Environment

This version is built on the non-collaborative target selection system. It uses the same observational space, but with the addition of observing other agents' selected targets to enable collaboration.

The reward structure was also modified for collaboration and to incentivise agents to choose optimal targets by adding rewarding the agent depending on the length of time the time between searching targets, with shorter times getting rewarded more. The penalty for choosing targets further from itself was removed. Rewards for searching targets were also increased to compensate for the overall increased reward.

- Reward for choosing a target not selected by other agents (<u>+0.00005</u>) (per step)
- Variable reward dependent on the time taken for agents between targets (<u>0.01 * (2000 – steps taken between targets)</u>), with a minimum of 0.
- Generic target search reward of <u>+4</u>
- Selected target search reward of <u>+13</u>

13.2.4.2 Results

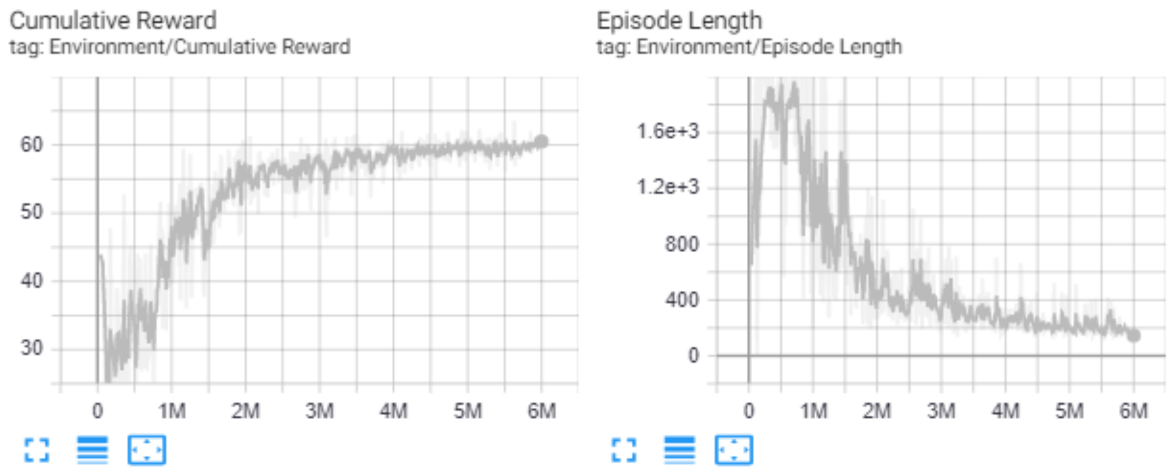


Figure 47 Target Selection System, Collaborative

Results are similar to other versions, with the episode length and cumulative rewards plateauing around 2 million steps but still steadily decreasing till 6 million steps. Training more may result in a shorter episode length. When comparing with the original target selection system, agents do not often choose already selected targets even if there are no unsearched targets near them.

Visual observation of the agent's performance indicates improved collaboration between agents. Unlike other collaboration versions, agents do stay together when needed but do not search the same target at the same time. Agents sometimes end up selecting the same target and immediately swap off. For example, if agent 1 selects a target and moves towards it and agent 2 suddenly selects the same target, both targets will select a different target and move away, leaving the target unsearched.

13.2.4.3 Analysis

Overall, agents can communicate with each other and can collaborate well. This form of collaboration is more flexible than previous forms of collaboration, with agents acutely aware of other agents' intent to search targets rather than just maintaining a distance between agents.

One of the main issues observed was a form of 'excessive' collaboration. In previous non-collaboration versions, agents often moved and searched the same target at the same time, especially when both agents were near the same target. In this case, agents displayed an extreme aversion to selecting the same target. It was often observed that when two agents select the same target, both would swap off and select a different target, leaving the original target unsearched. This is especially detrimental when one of the agents are already in the process of moving towards the target and changes its mind because another agent selects it. While this behaviour was the desired outcome to prevent two agents from selecting and searching one target, having both agents avoid the target was detrimental to performance.

13.2.5 Version 7.1: Target Selection System (Collaborative, v2)

13.2.5.1 Environment

This version is almost the same as the previous version, but with slight modifications in the reward structure. Rather than the original reward structure where agents got a +0.00005 reward for choosing a target which no other agent has chosen, only the first agent which selects the target gets the reward. For example, if agent 1 selects a target then agent 2 selects the same target, agent 1 will receive the reward because it selected the target first. Moreover, because of the stacked observations, agents can observe other agent's current targets and the previous targets for the last 10 steps. For easier reference, we call this condition as 'reserve', as the agent 'reserves' the target.

No other changes were made.

13.2.5.2 Results

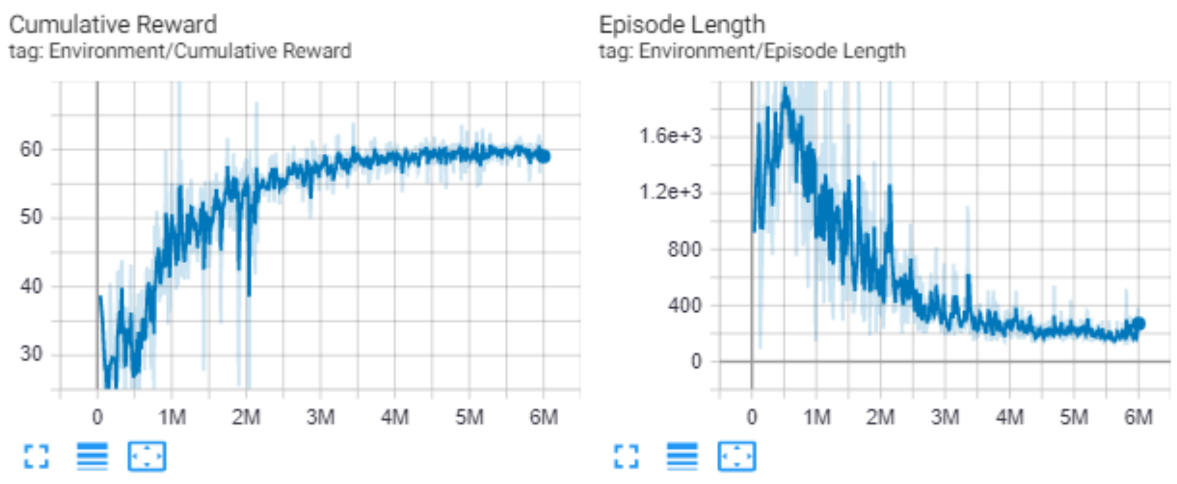


Figure 48 Target Selection System, collaborative with the time-based reserve system

The results are similar to other results, with the average steps per episode around 200 episodes and a low standard deviation. One thing to note is the rewards only plateauing at around 3 million steps, which is more than the convergence point of around 2 million steps.

One issue was when 2 agents selected the same target. The first agent will always continue moving towards the target, but should another agent select the target, the latter will then select another target. Should the second agent be nearer to the selected target than the first, the second will still move away and select another target.

Apart from this behaviour, the agents performed well and were able to search all targets consistently and reliably.

13.2.5.3 Analysis

This version was quite like the previous one but with only one change made to the reward structure. Hence, the results and performance are very similar to the previous version as well.

The major difference was the reserve system. With this version, the agents did not exhibit the 'overly-collaborative' behaviour as it did in the previous version. Should two agents both select the same target, only the agent which chose the target later would divert and choose another target. This did work most of the time, with agents that were nearer usually being the agent that selected a target first. This was ideal most of the time, but not all. When targets were sparse, e.g. only one or two left, all agents will select the remaining targets. This results in the first agent moving towards the target, and other agents selecting, then deselecting, the target. However, this does not mean the agent which chooses the target first was the one nearest to the target. Thus, the agent further away will move to search the target but the nearer agent will move away. This is not that ideal and results in some inefficiencies. Despite this, this inefficiency is quite minute and does not quite show up in the result graphs.

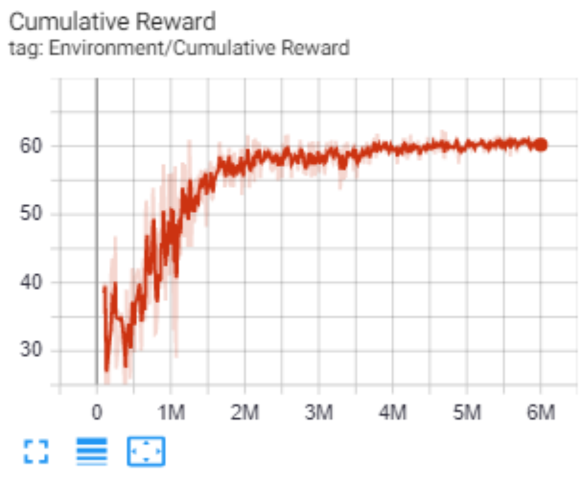
13.2.6 Version 7.2: Target Selection System (Collaboration, v3)

13.2.6.1 Environment

This version is almost the same as the previous, including the reserve system. The reserve system was tweaked slightly to change the agent that receives the reward to the one that is closest to it. For example, if two targets select the same target, the agent which is nearer to the target will get the reward.

No other changes were made.

13.2.6.2 Results



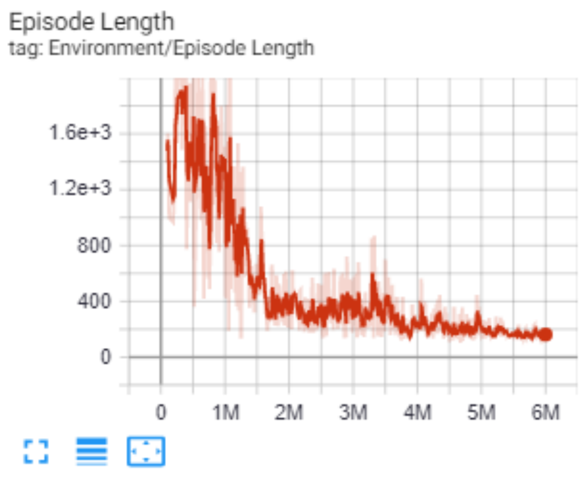


Figure 49 Target selection system, collaboration with the distance-based reserve system

The results were as expected, with a good learning curve and low standard deviation. The agent converges on a solution at around 1.5 million steps, which is markedly lower than the previous two target selection systems.

When two agents select the same target, the nearer agent will continue on the target while the agent further away will select another target.

13.2.6.3 Analysis

This version is very similar to the previous two, hence performance is very similar to previous versions. Agents demonstrate their ability to collaborate, move around and search targets consistently and reliably.

The only change in this version is the reserve system - the nearer agent receives the reward, rather than the agent that selected the target first. Before training, some concerns were raised that agents will not be able to train properly. The distance between targets is not a fixed value, unlike the agent that selects target first. Hence, it is theoretically possible that especially during early stages of training when agents move erratically, the nearest agent to the target will constantly change and will result in the reward agents receive being unreliable and therefore affect training. However, this did not happen as the agent converged to the solution quicker than previous versions, which indicate that the problem was easier for agents to solve with this particular feature set. After training, agents performed as expected, with agents searching targets appropriately and not often searching towards the same target, demonstrating collaboration.