School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

Final Report

Analysis of Reinforcement Learning Algorithms in the Unity Engine

Samuel Poirier

Submitted in accordance with the requirements for the degree of
BSc Computer Science & Digital Technology Solutions

2022/23

COMP3932 Synoptic Project

Summary

The use of Artificial Intelligence (AI) in modern computer games has become increasingly prevalent in recent years. However, these AI systems were often implemented without utilizing modern AI technologies. This gap between AI research and implementation in the game development industry motivated part of this project.

The project aimed to explore the possibility of training two different reinforcement learning agents that could adapt to new environments without requiring retraining. The study utilized the Unity game engine and two different RL algorithms, Proximal Policy Optimisation (PPO) and Soft Actor-Critic (SAC), to train the agents.

Overall, this project successfully met its objectives and provided useful insight into the benefits and drawbacks of the two algorithms.

Acknowledgements

I would like to thank my supervisor, Dr Abdulrahman Altahhan for his guidance and support, especially during the early stages of the project. I'd also like to thank my assessor, Dr Toni Lassila for his advice and encouragement during our mid-project meeting.

And thanks to my flatmates, friends and parents for their support during both this project and the various COVID-19 lockdowns that happened throughout my degree.

Contents

Chapter 1

Introduction and Background Research

1.1 Introduction

Most modern Computer Games contain some form of 'Artificial Intelligence'. Usually, these 'AI' act as either an enemy, non-player character or perform some other automatic function within the game. Despite the name, they are usually implemented without any use of modern AI technologies, instead making use of a variety of manually defined conditions and rules that are time-consuming to create, require multiple versions for different scenarios and are difficult to make unpredictable or realistic. As noted in Yannakakis,[1] there is a distinct disconnect between academic research into AI and its implementation within the games development industry. This project aims to help bridge this gap, investigating the possibility of training a reinforcement learning agent that is capable of adapting to a new environment without needing to be retrained, allowing for reuse across multiple scenarios.

1.2 Aims and Objectives

1.2.1 Project Aims

This project will investigate the applications of reinforcement learning agents within the Unity game engine.[2,3] The aim is to develop and train two reinforcement learning (RL) agents using two different RL algorithms; Proximal Policy Optimisation (PPO)[4] and Soft Actor-Critic (SAC),[5] then evaluate the abilities of the agents trained with the specific algorithms. These algorithms were chosen as they are two of the best performing RL algorithms currently available.[6,7] While developing a custom variant of these algorithms would likely yield useful results, it would add a layer of complexity that is out of the scope of this project. Each agent will then be placed in a series of new environments that they have not trained in which include environmental features they have not encountered before. Their ability to adapt to these unknown scenarios will then be recorded, assessed and analysed.

1.2.2 Objectives

1. Build a simple environment for the agents to train in
2. Design RL rewards, actions and observations for the agent and environment
3. Train an agent to navigate an environment using the PPO algorithm
4. Train an agent to navigate an environment using the SAC algorithm
5. Evaluate the process of training each algorithm
6. Modify the training environment
7. Evaluate the success of each agent when introduced to a novel environment

1.3 Machine Learning Platform Investigation

There are several frameworks, toolkits, environments and libraries that are available to use for machine learning or reinforcement learning projects. Even when just looking at options designed to support real-time physics simulations, many viable technologies must be considered. It is important to evaluate some of the major options in order to determine which one will be the most effective option for completing the project objectives.

1.3.1 OpenAI Robot OS

> *"OpenAI provides a complete Reinforcement Learning set of libraries that allow [the developer] to train software agents on tasks, so the agents can learn by themselves how to best do the task. [The] main type of agents are software agents, like this example where the OpenAI team trained an agent to play Dota 2."*[8,9]

OpenAI ROS heavily relies upon industry standard Robotics simulation software; Robot OS[10] and Gazebo.[11] Robot OS (ROS) acts as an 'Operating System' for simulating a robot, allowing the abstraction of its different systems, while Gazebo acts as the simulation environment. They are both used extensively in robotics research and industry, including machine and reinforcement learning. OpenAI ROS is a version of ROS that provides several features that are useful for implementing various RL algorithms. This initially seemed like the ideal option to use in this project. OpenAI is a huge name in the field of AI research, and this library is well-designed for simulating the kinds of agents and environments that will be used in this project.

Unfortunately, even though standard ROS is well-supported across a number of device architectures, OpenAI ROS is not as well-supported. We were unable to get the software to run properly on the AMD architecture CPUs and GPUs we had available, so were forced to abandon this technology stack as an option.

1.3.2 Gymnasium

> *"Gymnasium is an open source Python library for developing and comparing reinforcement learning algorithms by providing a standard API to communicate between learning algorithms and environments, as well as a standard set of environments compliant with that API. This is a fork of OpenAI's Gym library by it's maintainers (OpenAI handed over maintenance a few years ago to an outside team), and is where future maintenance will occur going forward."*[12]

Like OpenAI ROS, Gymnasium was originally a project run by the OpenAI team. This means it supports a lot of the same cutting-edge RL implementations that are supported by OpenAI ROS, but better supports development across a range of platforms. It supports several different environments, such as MuJoCo,[13] an environment designed for simulating heavily jointed agents or Atari,[14] a set of environments which simulate classic games from the Atari 2600 - see figure 1.1. Both of these (and others not mentioned) would be suitable for this project, though the simulation environments are not as complex as Gazebo. There are also

many third-party environments supported, some of which are much more complex than the ones mentioned here, the most complex of these being the Unity ML-Agents toolkit.

Figure 1.1: Some MuJoCo and Atari Gymnasium environments

1.3.3 Unity Machine Learning Agents

> *"The Unity Machine Learning Agents Toolkit (ML-Agents) is an open-source project that enables games and simulations to serve as environments for training intelligent agents. We provide implementations (based on PyTorch) of state-of-the-art algorithms to enable game developers and hobbyists to easily train intelligent agents for 2D, 3D and VR/AR games. Researchers can also use the provided simple-to-use Python API to train Agents using reinforcement learning, imitation learning, neuroevolution, or any other methods. These trained agents can be used for multiple purposes, including controlling NPC behavior (in a variety of settings such as multi-agent and adversarial), automated testing of game builds and evaluating different game design decisions pre-release. The ML-Agents Toolkit is mutually beneficial for both game developers and AI researchers as it provides a central platform where advances in AI can be evaluated on Unity's rich environments and then made accessible to the wider research and game developer communities."*[2]

Unity is one of the most complex and feature-rich freely available 3D simulation engines. Its primary purpose is for Game Development, where it is an industry-standard (used by roughly 45% of independent game developers[15]), and is also widely used in research involving 3D simulation (including RL research). It is easy to utilise to build complex environments, well supported by the developers across a range of platforms and well documented. It also easily provides a way for the trained agents to be leveraged in an existing scenario, for example, to act as an AI enemy in a game. These factors make it the obvious choice for this project.

1.4 Existing Research

There is a variety of existing research into the use of RL algorithms within games and into the comparison between PPO and SAC within this use case.

Wurman et al.[16] make use of an SAC variant to train an agent that is capable of beating the world's best simulated Gran Tourismo drivers; their research clearly shows that these kinds of algorithms can be used to great success in order to produce an agent easily capable of driving around a track. Their method differs from this project's in that the agents were trained simultaneously against each other in order to simulate race conditions, while this one will focus on an agent that is trained on its own.

Muzahid et al.[17] also make use of multiple agents training against each other. Like ours, their research investigates the performance of PPO and SAC, however, the focus is different. Comparing the results of their collision-avoidance focused research with our analysis of one agent in novel environments should allow for a more detailed understanding of the benefits and/or weaknesses of the two algorithms.

The research of Vohra et al.[18] is also relevant. They make use of the Unity ML-Agents toolkit but focus on seeking a specific object and avoiding others, rather than navigating around a track. They also make use of default environments provided by Unity and do not investigate the effect of changing the environment, which is a useful insight when comparing the algorithms that will be investigated by this research. Creating a series of their own environments would likely have given them more concrete results.

As this project varies significantly from the previously mentioned ones, it is important to find some research that is similar enough to be comparable in order to confirm the results gathered. Xu et al.[19] also make use of a simulated kart and track within the Unity ML-Agents environment and compare PPO with SAC and their performance in new environments (though their research focuses more on PPO variations than SAC versus PPO). If the results gathered by this project are similar to the results of that research, it will be clear that the investigation was successful. If they differ, it will allow both teams to analyse and compare their process in order to more easily identify any errors in their method or results gathering.

Because of the quantity of research that uses a car or racetrack for their environment or has self-driving or navigation as the problem to be solved, it makes sense to use a version of this as the problem and environment for this project in order to build on that previous research.

Chapter 2

Algorithm Concepts and Design

Reinforcement learning is a subfield of machine learning. While there is some overlap between the two, RL operates using different objectives and methods than traditional ML and can be applied to different scenarios.

2.1 Machine Learning

Machine learning is a broad term that refers to the process of training algorithms to make predictions or decisions based on data. The goal of ML is usually to create a model that can accurately classify or predict new data points based on patterns observed in an input training dataset and algorithms generally fall under two main categories; supervised and unsupervised learning.

Supervised learning makes use of a labelled input dataset, where each data point is mapped to a known output or label. The algorithm aims to learn a method that maps inputs to outputs in such a way that it can be applied to new unseen and unlabelled data. It is commonly used for image classification, speech recognition, and natural language processing.[20]

Unsupervised learning, on the other hand, uses an unlabeled dataset and aims to identify patterns or structures in the input data. This usually includes techniques such as clustering, dimensionality reduction, and anomaly detection.

2.2 Reinforcement Learning

In contrast to machine learning, reinforcement learning involves learning through trial and error. In reinforcement learning, an agent interacts with an environment and learns to take actions that maximize a cumulative reward signal (or minimise a cumulative punishment signal) over time.[21]

The basic idea of reinforcement learning is that the agent learns to associate certain actions with certain outcomes or rewards. The agent starts by taking pseudo-random actions in the environment and observing the resulting state and reward. Based on this feedback, the agent updates its policy or strategy for taking actions. Over time, the agent learns which actions lead to higher rewards and adjusts its policy accordingly.

The reward signal is the key aspect of RL. The reward signal is a scalar value that the agent receives after each action, and it represents how good or bad the action was. The agent can then use this to determine the kind of actions that result in positive rewards and should be encouraged, and also the inverse; what actions result in negative rewards and should be avoided. It is also important for the agent to balance immediate rewards with long-term goals, as some actions may have immediate rewards but lead to lower rewards in the future, while

others may only lead to a small immediate reward (or even punishment) but create the opportunity for a much greater reward in the future.[22]

The trade-off between this short-term and long-term reward is often referred to as the 'Explore-Exploit dilemma'; to what extent should an agent exploit its current environment for reward versus exploring potential other actions that could lead to higher rewards? It applies not only to RL, but also to animals in nature exploring their environment for a place to make a nest, or for humans deciding where to eat; this is explored in detail in chapter 2 of 'Algorithms to Live by' - *"Do you go to the Italian restaurant that you know and love [exploit], or the new Thai place that just opened up [explore]?"*.[23] The dilemma is an area of study in many different fields including biology[24] and psychology.[25] The different approaches to solving this problem are reflected in the differing implementations used in differing RL algorithms.

Reinforcement learning has been successfully applied to a wide range of problems, such as game playing, robotics, and control systems. It is particularly useful in situations where the environment is complex, dynamic, and uncertain, as it allows the agent to adapt to changing conditions and learn from experience. There are many different algorithms that are used to determine what actions to take next, based on the current environment state and current cumulative reward value. This project will investigate two popular versions; Proximal Policy Optimisation and Soft Actor-Critic. While they can both be used to train agents with similar goals, their implementation and design are quite different.

2.3 Proximal Policy Optimisation (PPO)

Proximal Policy Optimisation (PPO) is a popular Reinforcement Learning Algorithm which is used to train agents to perform tasks in environments with continuous action spaces. It is a policy gradient algorithm that focuses on keeping the policy updates small while optimising the policy; this reduces the likelihood of unstable behaviour.

Keeping the policy updates small is known as clipping the policy updates, and is the main way PPO differs from other algorithms. This technique works by constraining the policy update to be within a specific range of the current policy (hence the 'proximal' in PPO). These smaller steps help improve the stability of the agent's behaviour, and reduce the risk of 'falling off a cliff', which is where an agent gets into a state where it is no longer able to properly learn.

This project will make use of the implementation packaged with Unity ML-Agents.[26] Pseudocode for the algorithm (as described in Schulman et al.[4]) can be found in Appendix C as Algorithm 1.

2.3.1 PPO Process

1. The first step is to collect the sequence of states, actions and rewards that the agent experiences in this iteration (or episode) of the algorithm. This sequence is referred to as the trajectory.
2. Next, a neural network is used to predict the expected future rewards from the given state, for each state in the trajectory. This is known as computing the advantage

estimates.

3. PPO aims to keep the policy changes small, whilst maximising the reward value. This is done by using a surrogate objective function (essentially a ratio of probabilities between the new and old policies), which is clipped to a small value to remain proximal.
4. Finally, the policy is updated based on the surrogate objective. A gradient ascent is calculated from the objective and used to update the policy.
5. This process is then repeated for many iterations, with each iteration collecting new data to further improve the policy until convergence is reached.

2.4 Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) is an RL algorithm which is also used for environments with continuous action spaces. It operates by maximizing the expected cumulative reward that an agent can receive over a long-term time horizon. Essentially, the idea is to learn a policy function while simultaneously learning a value function. The value function estimates the predicted cumulative reward starting from a given state, and the policy function defines the probability distribution over actions with a specific observation of the environment.

SAC differs from other stochastic actor-critic methods by making use of entropy regularisation. The entropy (i.e. randomness) is applied to the policy and value functions, encouraging their results to be uncertain. This reduces the risk of overfitting to a specific problem, and encourages exploration which helps the agent avoid getting stuck in a local maximum; a suboptimal policy.

This project will make use of the implementation packaged with Unity ML-Agents.[27] Pseudocode for the algorithm (as described in Haarnoja et al.[5]) can be found in Appendix C as Algorithm 2.

2.4.1 SAC Process

1. Like PPO, the first step is to collect the trajectories based on the agent's behaviour.
2. SAC makes use of two neural networks to predict the outcome; one is used to represent the policy function which maps states to actions, while the other represents the value function which estimates the future rewards based on the given state.
3. Next, SAC makes use of a function known as the 'Q-function' which is used to estimate the expected cumulative reward of taking an action in a given state and following the current policy after that. The values from this function (Q-values) are then used to compute the advantage function, which is a measure of how much better the current policy is than the previous one.
4. The policy is then updated, and entropy is applied to it in order to encourage exploration.
5. This process is then repeated over many iterations until a convergence in reward is reached.

2.5 In the context of this project

As this project is making use of pre-written implementations of the algorithms, their actual implementation is less relevant to the specifics of this project; what is important is to understand their overall approach to the ‘Explore-Exploit’ dilemma. This project will focus on implementing or defining four things:

1. Observations: These represent what the agent is aware of within its environment. They can be very simple, such as a location or speed, or more complex such as abstract ‘score’ values related to the current scenario.
2. Actions: These are any of the interactions that the agent is capable of taking within its environment, usually some form of movement of itself or another object.
3. Reward: This is the reward signal that tells the agent how well it is completing the task, and is usually made up of several different factors including time, goal proximity/count or punishment for certain actions such as collision.
4. Environment: The agent requires an environment to operate in. This should contain reward objects or areas if these are used within the reward calculation and a model of the task that the agent is attempting to solve.

The success of the trained agent largely depends on how well these values have been chosen and implemented; useful observations and reward signals will allow the agent to learn quickly whilst poor observations and rewards could lead to the agent prioritising behaviour which is not what the developer is hoping to teach to the agent. Details of the actual implementation used in this project are discussed in chapter 3.

Chapter 3

Method

3.1 Development Tools and Process

The large majority of this project was spent using Visual Studio for programming and Unity for the in-engine configuration and setup. The choice of Unity has already been justified within section 1.3.3. C# was used for the Unity scripts as that is the main language supported by Unity. Visual Studio was used for C# programming as the team has a large amount of experience using it, it is the industry standard tool, and much more powerful than a popular alternative, Visual Studio Code. However, Visual Studio Code was used for programming the simple data processing script written in Python, as Python is much more lightweight than C#, and benefits from a simpler programming environment. The specific implementations of PPO and SAC used are also written in Python.

Code was tracked using git for version control and regularly pushed to a GitHub repository (appendix C.7) to ensure the code could be easily rolled back, was safely backed up and was available on other devices. Again, these were used as they are generally considered the standard within the industry, the team have experience with them and they offer all the features required.

Graphs of the data gathered during training were created using the LaTeX plugin 'pgfplots'.[28] Use of this tool meant that the graphs could be built programmatically which increases the ease of modification once they are created. Pgfplots also creates graphs that are in a vector image format, meaning that they can be easily scaled to the correct size for being embedded into different mediums without any loss of clarity.

In order to organise the project and ensure it met the goals required within the timeframe, the team decided to adopt an agile methodology.[29] Work was divided into small components that could be completed independently which were then organised according to the time available and completed over a series of sprints according to current project status, the remaining time and the remaining work (see Gantt chart, appendix C.5 for the sprint details; the report will not discuss individual sprints as they aren't particularly relevant to the outcome). Due to the team size, only one component was worked on during each sprint. An agile approach to this project made sense due to both the size and the exploratory nature; having an approach that was capable of responding to change was crucial to ensure that as unexpected issues appeared it was easy to adapt and keep moving without derailing the project due to needing to redefine requirements.

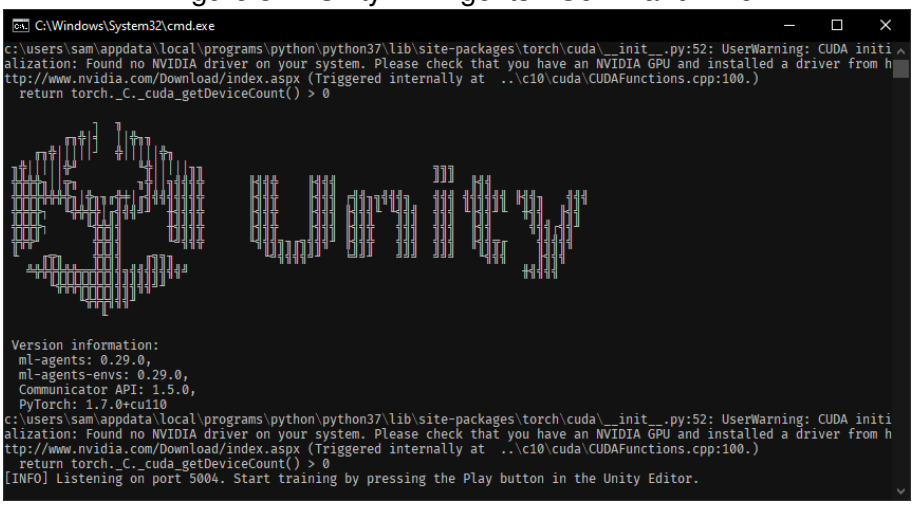
3.2 Unity ML-Agents Overview

Unity ML-Agents functions as a plugin to the Unity game engine that sits on top of a traditional Unity project. When building the environment and agent, the initial process is the same as developing a game or any kind of simulation. It is possible (and not difficult) to take an existing project, and have the agent simulate a human input; the original project has no idea it is being controlled by an AI model rather than a human. This helps to keep the project loosely coupled - it is easy to make changes to the game without having to update the model, and vice versa. However, you might want to add certain features to the game or simulation in order to aid the training of the AI model. For example, a human naturally knows what direction it should drive in around a track, whereas an AI has no idea initially, so might require some kind of extra information such as adding checkpoints around the track that it knows the location of. ML-Agents gives the developer the freedom to have their RL project and game/simulation project as loosely or tightly coupled as they wish.

3.3 Editor Extension

Normally, in order for the ML-Agents extension to run, it must be started from the command line (see figure 3.1). In order to avoid having to repeatedly type the same commands with subtle modifications in order to start training, we wrote a simple custom editor extension to Unity (see figure 3.2). This extension acted as a GUI that allows the user to enter some configuration values, and then automatically start the ML-Agents command line application with those parameters. This massively streamlined the process of training the two agents, in particular making it easy to pause and resume training when needed to avoid having to leave the agents training continuously for large quantities of time.

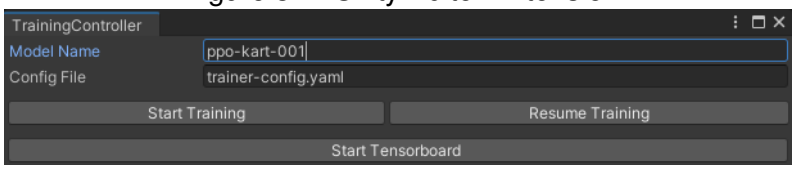
Figure 3.1: Unity ML-Agents - Command Line



3.4 Training Process

Each algorithm was trained separately, starting from an agent with absolutely no knowledge. Initially, the agent was observed during training, and various configuration values (including

Figure 3.2: Unity Editor Extension



kart physics values, RL hyperparameters and reward weighting) were iteratively tweaked to improve training performance. Once an optimal configuration was found, the agents were run while supervised in order to determine the number of episodes required for the agents to converge on a solution. We found that roughly 3,000 episodes were sufficient. Both agents were then left to run unsupervised for this number of episodes, with the simulation speed turned up to 200 times the standard real-time value (the maximum supported) in order to speed up the process. Other than these settings, the heuristics used were the same as the defaults recommended by the Unity ML-Agents project. The team did experiment with changing other settings, but did not discover any major benefits, so kept the stock settings to ensure stability. After training, the agents were then evaluated briefly to check that the training was indeed successful and the data gathered during training was processed as described below.

In order to improve the analysis process, a feature was added to the agent that would cause it to leave markers periodically as it drove around the track. This made it possible to easily see the path that the agent took when it drove around the track.

3.5 Data Processing

The raw JSON data from training was cleaned and processed using a Python script (found in the same GitHub repository as the rest of the project). The output data was stored as a CSV file for each algorithm in order to be graphed by pgfplots. As part of this process, some extra fields were calculated. These were all cumulative or averages of all the data until that entry; the cumulative iteration time, the average reward, the average velocity and the average iteration time. These extra fields were created as they will allow for a more complex analysis of results than would have been possible from the initial raw data and would be tedious to calculate individually for each graph that would use them. An example of the JSON data can be found in snippet C.1 - the actual data is too long to include in this report but can be found in the GitHub repository (appendix C.7).

Chapter 4

Implementation

4.1 Preliminary Tests

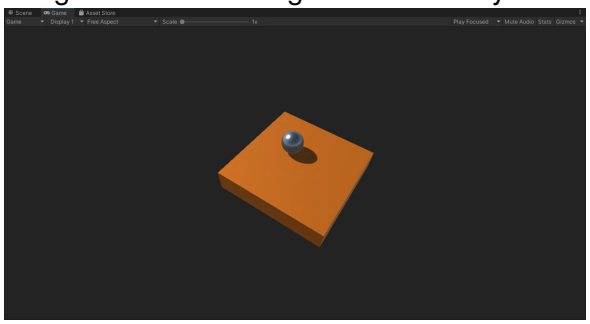
Before jumping into designing the agent and environment that will be used for the bulk of the project, we decided to build a much simpler problem in order to gauge the abilities and specific idiosyncrasies of the toolkit. One of the example Unity projects for ML-Agents is a simple roller-ball problem; rotate a plane in two axes in order to roll a ball to a specific location. This seemed like a suitable idea for a project that we could build ourselves for this test.

4.1.1 Preliminary Implementation

The first iteration of the rollerball was simple; a flat plane with a spherical ball on it. The ball's starting position was randomised, and the agent was able to rotate the plane in two axes and observe the rotation of its plane, and the velocity and position of the ball (see figure 4.1). Its reward simply increased over time, with a drastic negative reward if it dropped the ball, which also caused the episode to end.

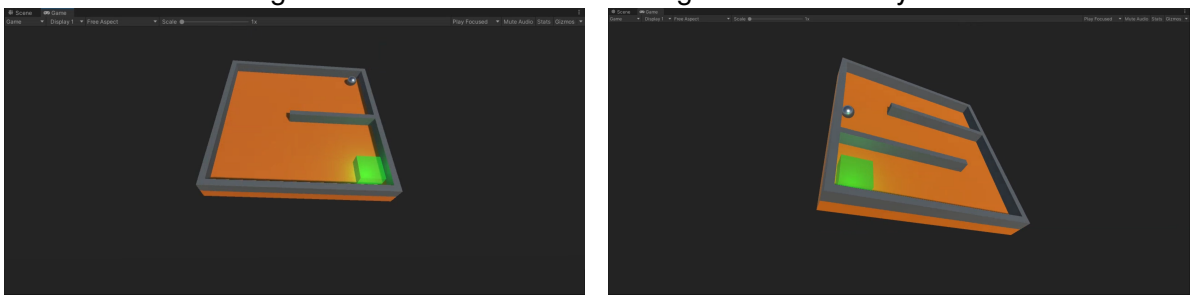
Only PPO was used for the preliminary tests, as switching the algorithm used is trivial, and comparing the two is not relevant for the preliminary investigation. Very few episodes were needed for the agent to create a model that was capable of keeping the ball on the platform indefinitely (see appendix C.1).

Figure 4.1: First Stage of Preliminary Test



The next two iterations adjusted the problem slightly; they involved adding a wall around the platform along with a goal for the ball. One or two walls between the ball and the goal respectively were also added to increase the challenge (see figure 4.2). The agent was given an additional observation; the position of the goal and the reward was updated to apply a small negative reward over time but with a large positive reward for reaching the goal state. The purpose of the negative time-based reward was to encourage the agent to try and solve the problem quickly and was largely successful. This problem required many more episodes to train (as expected) but was able to reach a solution that would solve the problem the large majority of the time (see appendix C.2).

Figure 4.2: Second and Third Stages of Preliminary Test



4.1.2 Key Findings from Preliminary Test

The main finding from the preliminary investigation was the apparent ease of producing a solution using the ML-Agents framework. Under 200 lines of code were needed to create an agent and environment capable of solving a reasonably complex problem after very little training. Increasing the complexity of the environment was trivial, and didn't require any more programming - the Unity editor could be used to update the maze to include more complex obstacles and the same logic could be used to train the agent. This was promising as it indicated that the main investigation was feasible and the toolkit chosen was an ideal choice for the problem.

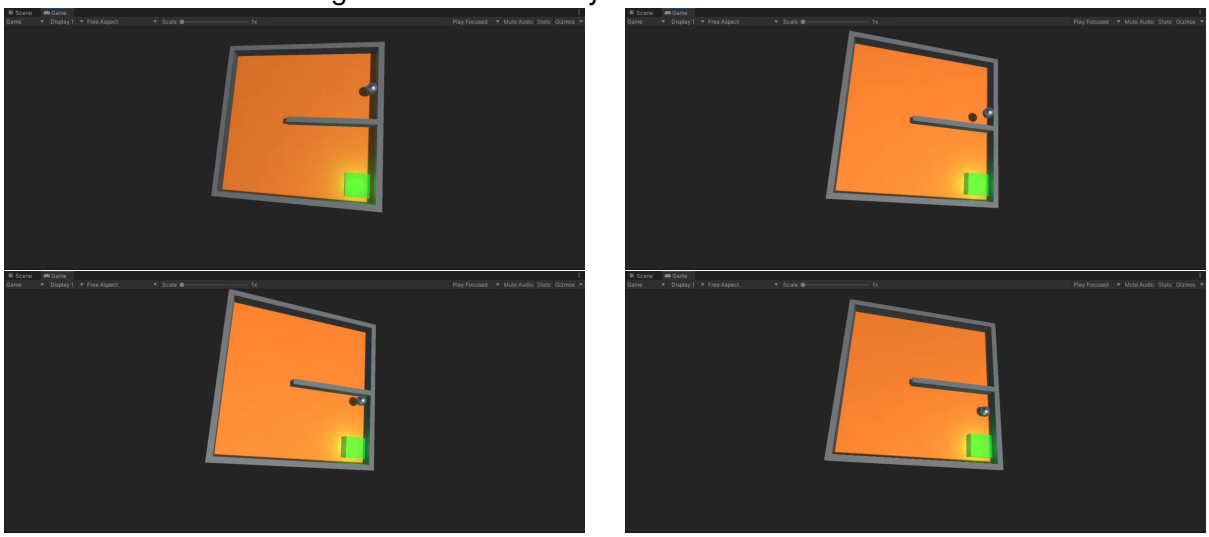
Another observation was the inherent 'creativity' of the training algorithm. When initially training with one wall, the agent worked out it could 'flick' the ball over the wall which was much faster than rotating the platform so that the ball travelled around the wall (see figure 4.3 and appendix C.3). It was interesting to see the agent come up with a solution that the average human might not discover or be skilled enough to pull off regularly like the agent was able to. A transparent collider was added to the top of the maze to stop this behaviour, but it indicates an important piece of information - any rules that we want the agent to follow must be defined, either by limiting the actions it can take by modifying the environment so they are impossible to perform (as done here) or ideally, updating the reward so that the AI knows it's meant to avoid certain behaviour. If these undesirable strategies aren't defined but are an optimal solution then the agent is likely to discover and make use of them as it has no inherent knowledge of 'cheating'.

These results provided a good basis and insight into the problems we are likely to encounter when working on the main portion of the project and outlined some approaches that are likely to yield good results. Discovering these as part of the simple preliminary tests validated the advantages of running an initial preliminary investigation before starting the implementation of the main project.

4.2 Agent Design

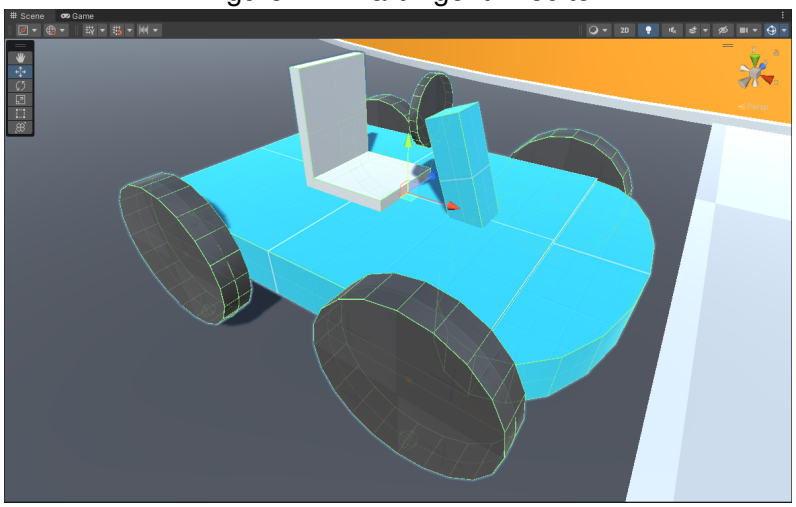
The kart agent is physically comprised of two parts; the body (plus some minor aesthetic additions) and the wheels (see figure 4.4). The body is constructed from several meshes that are combined into one rigid body. The four wheels make use of the built-in Unity wheel colliders which accurately simulate wheel rotation (and the corresponding forces) within the

Figure 4.3: Preliminary Test - Creative Solution



physics engine. To make the kart easier to follow, we made use of free Creative-Commons prototype textures developed by Kenny[30] (see figure 4.5). Physics values for the agent were determined by research into the properties of real-world karts,[31] combined with some in-engine experimentation to result in values that performed in a manner that seemed realistic. They can be inspected in appendix table C.1.

Figure 4.4: Kart Agent in editor

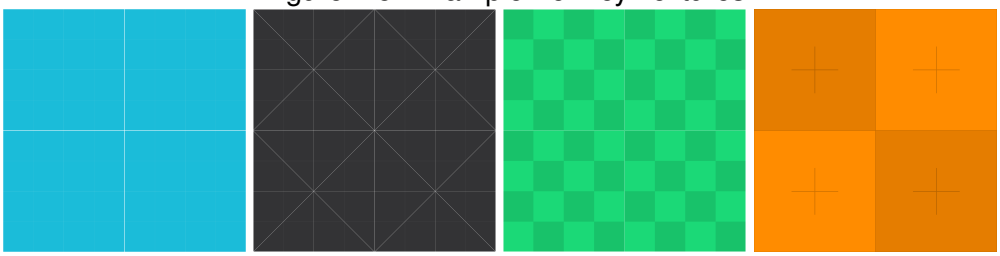


4.2.1 Observations

The agent has a number of observations that it is able to make within the environment. The most important being the 13 rays that it uses to measure the distance to colliders (see figure 4.6). These are arranged evenly between $-90°$ and $90°$ (where $0°$ is directly forward). These rays can tell the agent how close it is to colliding with objects before it hits them and replaces the vision element of a human driver. The rays are limited to a max distance of 20 arbitrary Unity units to stop the agent from being able to see too far ahead and are colour coded from green to red according to how long the ray travels before it collides with an object.

The agent is also able to observe the magnitude of its current velocity. This allows the agent to

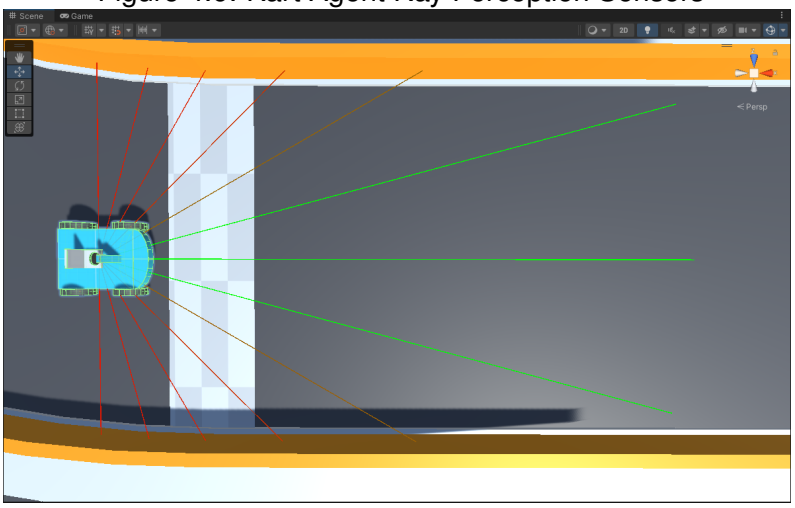
Figure 4.5: Example Kenney Textures



be aware of how fast it is travelling within its environment, which will allow the agent to adjust its speed to be faster or slower depending on what it determines is most suitable for its current situation. Without this observation, the agent would have to learn through trial and error that increasing the wheel force makes it move around the environment faster, as it would have no concept of velocity. This would drastically increase the training time. With this observation, the agent has a simple scalar value it can correlate with the reward signal it receives.

The final observation that the agent can measure is the distance between its current position and the next checkpoint (for more information about checkpoints, see section 4.3.3). This helps the agent learn where it is meant to go next.

Figure 4.6: Kart Agent Ray Perception Sensors



4.2.2 Actions

The agent is only able to make two simple actions; the torque (i.e. the turning force) it can apply to the wheels and the steering angle of rotation of the two front wheels. Values for these can be found in appendix table C.1. Initially, the agent was able to apply a positive and negative torque to the wheels, but this was adjusted to be only positive - similarly to the findings in section 4.1.2 the agent discovered it could increase its reward signal with low risk of failure by driving forwards then backwards repeatedly rather than traversing the track, which was not behaviour we wanted.

The steering angle of the wheels was clamped between two values (appendix table C.1) to limit the turning circle of the kart to something realistic. The agent was able to freely adjust the angle between these values as it saw fit.

4.2.3 Rewards

The reward of the agent is cumulative over the episode. The value for a given action step within the episode can be calculated from some constant reward values and some variable measurements taken at that step. As the value for the reward is an arbitrary value that does not represent anything other than an abstract 'score', we're able to discard all units of the input data. This means it is appropriate to perform certain operations that might not make much sense in other contexts, such as subtracting an angle from the magnitude of a velocity.

The reward constants are defined as:

- s: The scalar step modifier ($s << 1$).
- p: The failure penalty for colliding with a wall.
- r: The reward for reaching a checkpoint.

The variable values for each action step are defined as:

- v: The current velocity of the kart.
- θ: The current angle of the wheels.
- w: Whether the agent is in collision with a wall, where $w = [0, 1]$.
- c: Whether the agent is in collision with a checkpoint, where $c = [0, 1]$.

The reward function, R, for a given step, i, can be written as:

$$R(i) = (||v_i|| - |\theta_i|) \times s + (p \times w_i) + (r \times c_i)$$

Therefore, the cumulative reward over the episode (where the action step count is defined as a) can be defined as:

$$\sum_{i=0}^{a} R(i) = \sum_{i=0}^{a} [(||v_i|| - |\theta_i|) \times s + (p \times w_i) + (r \times c_i)]$$

These components aim to encourage the agent to drive quickly, as the larger the value of $||v_i||$, the greater the reward. It also encourages it to drive in a straight line, as the larger the value of $|\theta|$, the lower the reward. The scalar step modifier, s, is used to reduce the impact of the first component of the function, $(||v_i|| - |\theta_i|)$; w_i and c_i are almost always equal to zero, so would become negligible compared to the first component without s. Reward values used in training can be seen in appendix table C.2. The reward signal is structured as a summation which means that the kart's reward value increases over time as long as it is successfully driving around, encouraging it to learn to keep driving without collision. As soon as the kart collides with the terrain, the negative penalty, p, is applied and the episode is ended. This helps the agent learn to avoid collisions.

The $p \times w_i$ and $r \times c_i$ components act as a form of reward shaping.[32] Reward shaping is a RL technique that works by providing supplemental rewards in order to make a problem easier to learn. Reward shaping works in addition to the natural reward signal, and is applied when an

agent makes progress towards a good solution. Essentially, the idea is to indicate to the agent some form of progress towards the desired end goal state which is exactly what the checkpoints function as. The wall collision works as a form of inverse reward shaping where the collision indicates progress away from the desired state.

The reward function described here is the final reward function that was used during training; it was developed iteratively, starting with a simple function consisting of just the v component with the other variables and constants being added to improve the performance of the agent during testing.

4.2.4 Logging

While ML-Agents comes with some built-in logging functionality it is more about general RL training progress, rather than specific to the observations and rewards in this project. Because of this, we built our own logging functionality that outputs per-episode training results as JSON data that can then be easily processed later (for example, see appendix snippet C.1). This data was then processed in Python to produce extra insights, for example, averages, maximums and minimums for later analysis and graphing.

4.3 Environment Design

The environment is much simpler than the agent, comprised of multiple tracks for testing the agent and some basic lighting and default (realistic) physics. The tracks are designed such that the agent can collide with them and trigger some logic to end the episode if it hits the edges. There were multiple ways the track could be designed due to the variety of tools available within the engine. The initial training track is a simple oval as this is the simplest shape that contains both straight and curved sections. In theory, if the agent learns to navigate both of these types of track section independently, it will be able to handle most tracks.

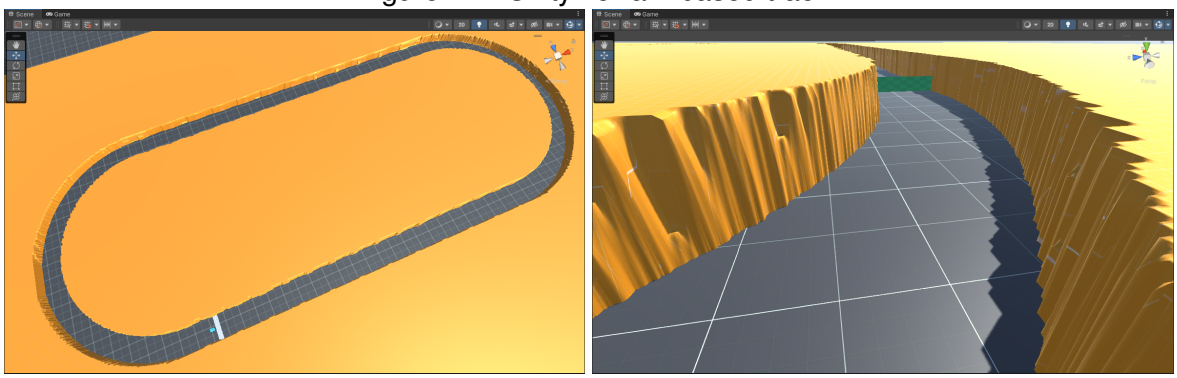
4.3.1 Initial Terrain

The first tool used to build a track was the Unity terrain tool. This tool functions by having a simple flat square mesh that can be deformed using tools within the editor. We deformed this mesh to create an oval-shaped track (see figure 4.7). The issue with this tool was that it was time-consuming to create new tracks; the tracks created were natural looking (as the tool was designed to emulate natural terrain) but not an ideal simulation of a real-world racetrack. Creating sloped sections in particular was difficult. As our team did not contain any 3D artists, we decided that using a different approach would yield better results (though it was possible to successfully train an agent on this track during testing).

4.3.2 Final Track Pieces

While looking for an alternative to the Unity terrain tool, we found the free Unity add-on ‘Simple Roads’.[33] The add-on licence agreement permits it to be modified and used in commercial or non-commercial projects for free, so it seemed like an ideal choice. It contains several track components that can be combined to create a variety of different track shapes. We

Figure 4.7: Unity Terrain-based track

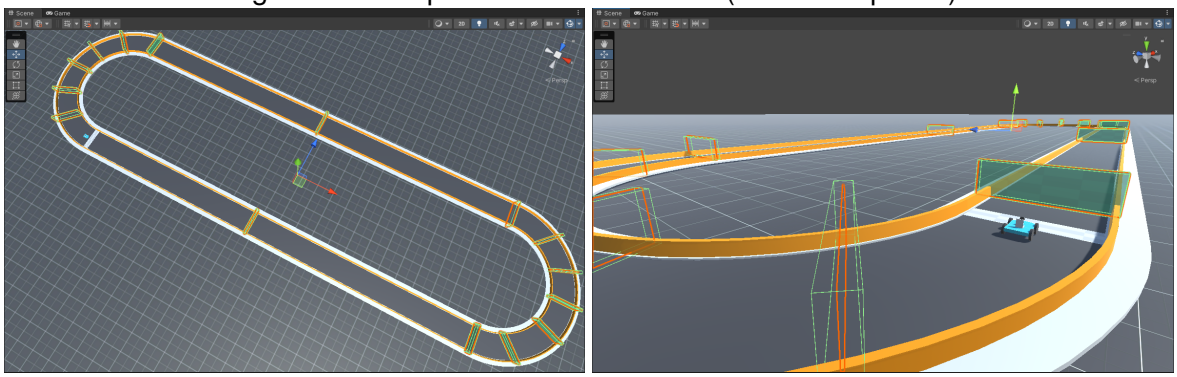


customised the pieces to add proper collision meshes and add materials to make the course visually easier to follow. A similar oval shape was constructed which was used for agent training (see figure 4.8). More complex tracks were also created for post-training tests, and are discussed in chapter 5.

4.3.3 Checkpoints

As mentioned in section 4.2.1, we discovered that for the first few steps of training, it is important that the agent is aware of the distance to the next checkpoint so that it can learn the way it is meant to navigate the track. The track was given 16 checkpoints; 3 in each straight section and 5 in each curved section. These values were determined through trial and error - we discovered that the curved sections need more checkpoints than the straight ones as they are more difficult for the agent to navigate. We also aimed to keep the checkpoint count as low as possible in order to reduce the reliance of the agent on the checkpoints. They can be seen in figure 4.8. The checkpoints are designed such that there are no physics collisions (the agent is able to pass straight through them). As the agent drives through a checkpoint, it is disabled, and the next checkpoint is enabled causing the agent's distance observation to update to indicate the next checkpoint. As checkpoints are only used for training, this track is the only one that requires checkpoints to be created.

Figure 4.8: Simple Roads-based track (with checkpoints)



Chapter 5

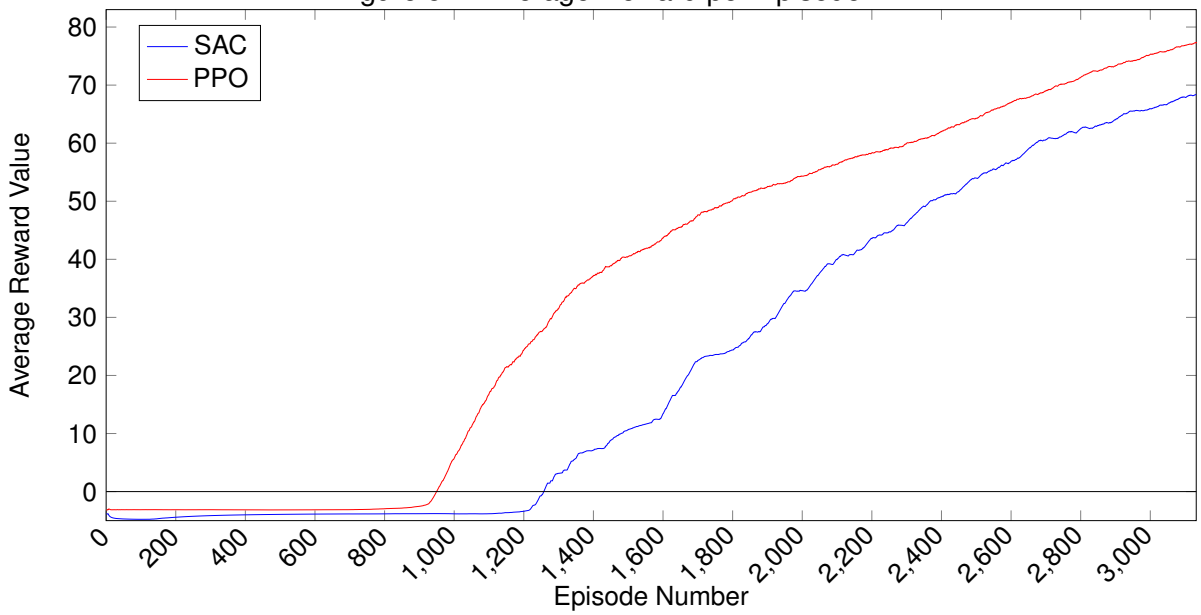
Training and Results

In this chapter, the agent training and outcomes of the agents' training will be discussed. Video evidence of the agents running on the different tracks can be seen in appendix C.4.

5.1 Training

Training both the PPO and SAC algorithms resulted in Agents that were able to complete the training task successfully. The resulting agents were similar in ability though PPO was able to reach an average reward value ~10% higher than SAC, and was able to do so over ~300 episodes earlier (see figure 5.1). This indicates that if the training speed of the agent is important, then PPO is likely a better choice - though the difference between the two is not very significant so other factors, such as the agent's performance, should be considered over the training time unless the training time is crucial.

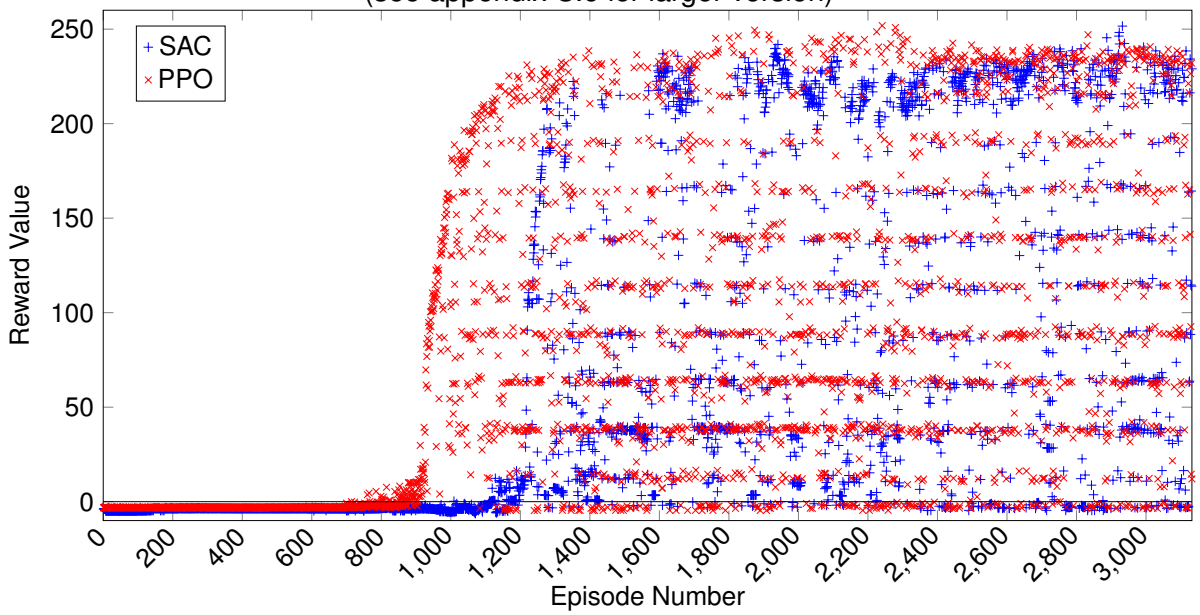
Figure 5.1: Average Reward per Episode



When comparing the average rewards (figure 5.1) with the actual reward values (figure 5.2), PPO is still the faster of the two. However, compared to the average reward values, the highest actual reward values appear pretty similar for both algorithms. Both reach these maximums pretty quickly, but SAC is slower to converge on producing them consistently. This likely is a result of the different natures of their implementations; off-policy exploration is a much larger part of SAC than it is for PPO which means that SAC is more likely to have episodes that perform poorly which will grant a lower reward so reduce the average more than if the algorithm had stayed on policy like PPO. This has a benefit though, which will be

discussed later in the chapter; this greater level of exploration results in an agent that is more flexible in its decisions than if it were trained using PPO.

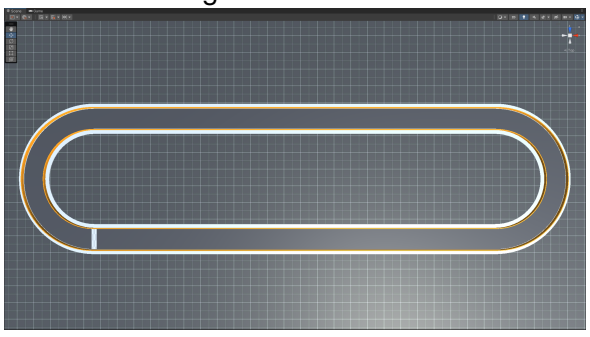
Figure 5.2: Reward Over Episodes
(see appendix C.6 for larger version)



5.2 Environment Specific Performance

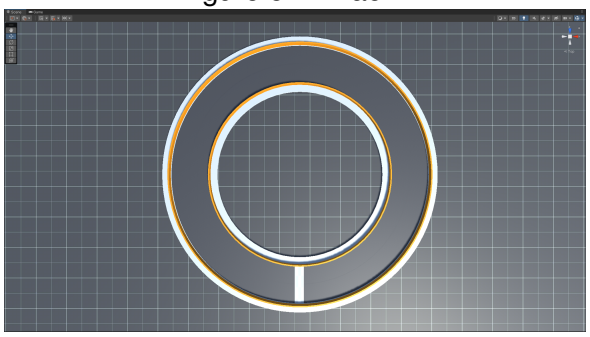
5.2.1 Track 1

Figure 5.3: Track 1



The first track was designed to train the agents on two simple sections; straight lines and curves. Unsurprisingly, both agents are easily able to complete the track as it is the one they were trained on. There is little deviation between the two, due to the simplicity of this environment. They can be run indefinitely without colliding with the edge, and adjust their speed so that they slow down when approaching corners and speed up in straighter sections of the track, though it is only a subtle deviation in speed (see appendix C.4 00:00 - 02:31).

Figure 5.4: Track 2

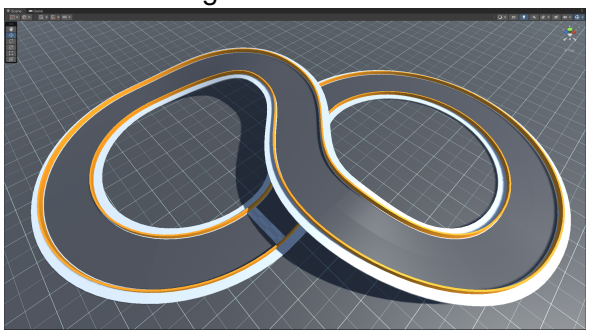


5.2.2 Track 2

Track 2 was designed to be similar to track 1, but with a simple change to check that the agents can make decisions based upon their environment, rather than just learning a solution to solve the specific problem that they were trained on. Both agents were able to easily complete the track and could run indefinitely (see appendix C.4 02:31 - 03:51). Despite the changed conditions, they are still not being exposed to track layouts that they have not seen before so more complex tests are needed.

5.2.3 Track 3

Figure 5.5: Track 3

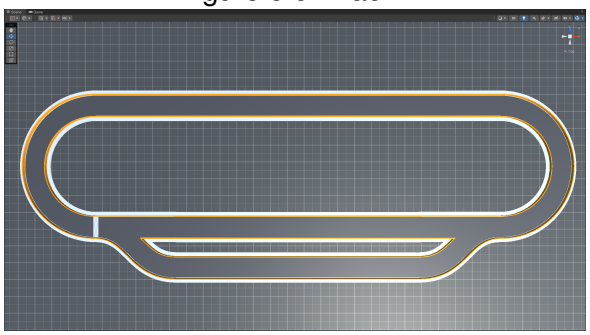


Track 3 features three new features that the agents have not seen before; an ‘s’ shaped curve section in the middle, and both upwards and downwards-sloped sections. As can be seen in appendix C.4 (03:51 - 05:26), PPO is able to cope with the new features well - it adjusts its speed when it encounters the sections it is unfamiliar with which is a good strategy for reducing the risk of collision. When it encounters extra velocity applied to itself due to gravity when moving downhill, it is even able to ‘drift’ around the corner and avoid a collision. While it does sometimes collide with the track edges, it recovers from this and keeps driving. It uses the same strategy continuously and overall is successful.

Conversely, SAC is not able to complete this track. In the standard starting position, it consistently veers off to the side and collides head-on with the wall to the point that is unable to recover and keep driving. The kart was then repositioned to a variety of other starting positions, where it was still unable to complete the track. This indicates that the training track was not complex enough for SAC to learn a solution that it could apply to these problems. It’s also possible that there was an undiscovered technical issue with this track and SAC.

5.2.4 Track 4

Figure 5.6: Track 4



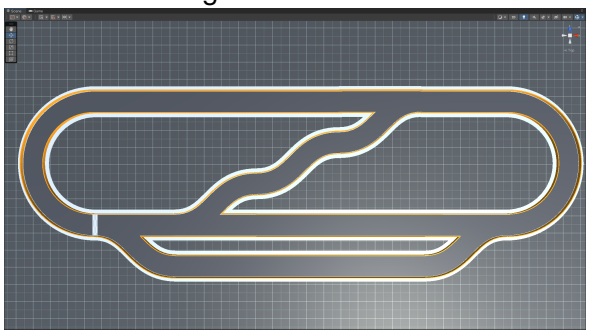
Track 4 introduces another (different) new concept for the agents to traverse; a choice in which portion of the track to take. Other than the 'pit stop' branch, the track is the same as the one that the agents were trained on. This is intentional to ensure that the only aspect that is being evaluated is the decision - if there were other changes it would be harder to analyse the effect of this change specifically (for video evidence, see appendix C.4 05:26 - 08:47).

PPO is able to detect this choice, but its actual behaviour does not differ from the standard plan of track 1. As it drives past the entrance to the alternative path, it does move towards it slightly but decides to continue with the standard route. This fits the design of PPO; it favours actions that are close to the current policy (which does not account for branches) so stays consistent with the original route.

SAC initially chooses to take the new alternative path, then continues on the standard route around the track when the path rejoins the main loop. On the second lap, it then takes the standard route around. This reflects the generally observed behaviour during more extended running, where the agent chose each path on a roughly 50:50 basis. The SAC implementation is designed to focus more on exploration, so this behaviour is consistent with the algorithm's intended design.

5.2.5 Track 5

Figure 5.7: Track 5



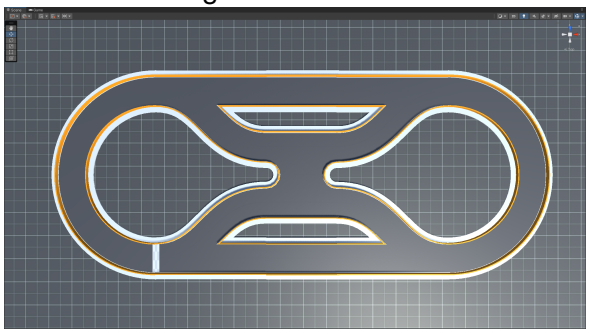
Track 5 (see appendix C.4 08:47 - 11:20) adds to the environment of track 4 with a new branch after the first that cuts the track in half. This change allows us to determine if the agents have different behaviour when making turns on their left as opposed to the right-hand turn in track 4.

This change does make a difference for PPO. Like track 4, it initially ignores the first turn, but then does decide to take the second turn. The training track involves only left turns, so it makes sense that the agent will take one when given the choice, but will not take a right turn. However, at the end of the alternative path it does take a right turn. This is likely due to the angle that the track joins the original path; the agent may not be able to detect the choice due to the angle or may deem it impossible to navigate. The agent then continues with 'normal' behaviour but in the opposite direction. It is then presented with the same choice as track 4, but with the 'pit-stop' now located on the left of the agent. This time it does take it, which reinforces the theory that the agent has been trained to handle left turns, but not right turns. The agent then continues with this behaviour consistently but does not take the middle track anymore (likely due to the fact it now requires performing a right-hand turn).

The behaviour of SAC is largely consistent with its behaviour in track 4. It takes the initial turn, which results in it missing the new turn that has been introduced in track 5. While it does then turn left when rejoining the main track, this is likely due to the fact that a minor collision prior to this results in it meeting the track at a right angle. This adjusts the orientation of the kart so that the angle between the left and right turns are roughly even, so it unsurprisingly prioritises the left turn that it is used to from its training.

5.2.6 Track 6

Figure 5.8: Track 6



Track 6 (appendix C.4 11:20 - 15:15) is the most complex of the 6 tracks. Agents are presented with multiple decisions, which can result in a large variety of paths around the track and the branches have overlapping paths resulting in multiple decisions needing to be made in quick succession.

PPO was consistent in the decisions it made. Whenever given a choice of path, it always took the left branch regardless of whether it represented a deviation from the main path or not and it did this regardless of starting position or orientation.

SAC was much more irregular; like on track 4, it makes different decisions on what branch to take multiple times, resulting in unpredictable behaviour. It is however successful in that it is able to recover from partial collisions multiple times and keep driving around the track despite the variety of routes it takes.

Chapter 6

Evaluation and Future Improvements

The objectives of this project were to build a simple environment for the agents to train in, design RL rewards, actions and observations for the agent and environment, train an agent to navigate an environment using both the PPO and SAC algorithms, evaluate the process of training each algorithm, modify the training environment and evaluate the success of each agent when introduced to a novel environment. All of these objectives have been covered or completed, so the project was an overall success.

Apart from one instance, the agents were able to navigate around a variety of unseen environments with no or only minor issues, and when presented with decisions they were consistent with the behaviour you would expect when considering the implementation of their specific algorithms. They were both able to recover and continue from minor collisions without any assistance and could run independently and indefinitely in almost all cases. However, it is clear from chapter 5 that there are a few issues with both agents and the process used to train them.

6.1 Environment

The most obvious issue is the agents' bias towards turning left. The track they were trained on only featured left turns, which is likely the cause of this issue, resulting in overfitting. Overfitting is a concept in reinforcement learning where an agent becomes too specialised to the specific training data, rather than learning to address the general problem.

The problem of overfitting could be addressed in this project in a few ways, such as by improving the training process to make use of a track with a greater variety of geometry, to randomly adjust the layout of the track and to start the kart at a randomly chosen position and orientation. This would train the agents to be more versatile as their agents are already familiar with a variety of scenarios which would help avoid overfitting.[34] This would also likely help avoid the issue that SAC had with track 3, as it is more likely to have learnt how to encounter those kinds of environments.

To test the agents further, more complex test tracks could be designed. There are certain features not tested in this project that it would be good to include, such as moving sections, sections with different physics, dead-ends, jumps and more. The more versatile the agent, the more useful it will be for any real-world applications (see section 6.7).

6.2 Reward Function

The reward function, $\sum_{i=0}^{a} R(i) = \sum_{i=0}^{a}[(||v_i|| - |\theta_i|) \times s + (p \times w_i) + (r \times c_i)]$, was designed well as the agents were able to use it to train successfully. As is visible in figure 6.1, the

average reward of both PPO and SAC is roughly proportional to the average velocity. This also holds for the average iteration time in figure 6.2 - where it is clear that the average time running and average reward are directly proportional. This indicates that the reward function models the problem correctly, as the two factors that were the most important (i.e. the length of time that the agent runs for, and the speed that it drives at) are major contributors to the result.

Figure 6.1: Average Velocity and Average Reward Over Episodes

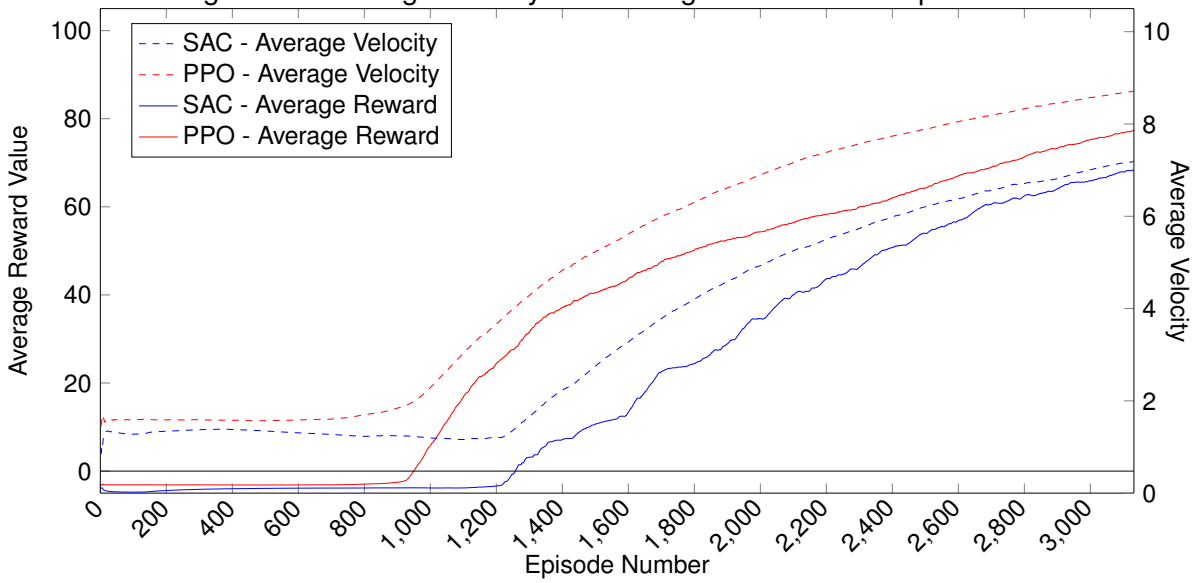
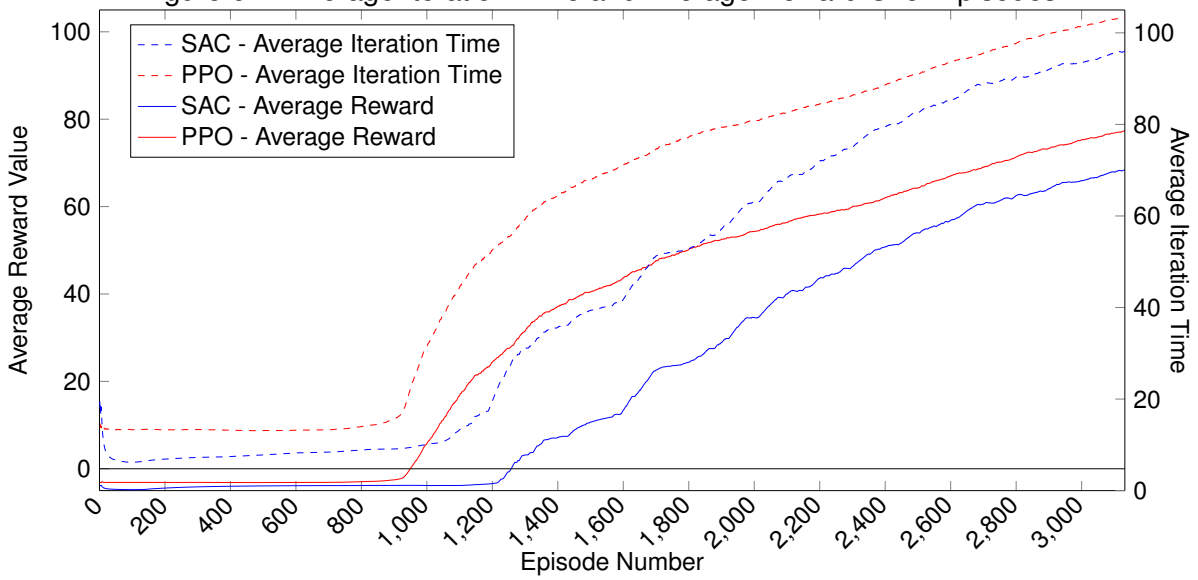


Figure 6.2: Average Iteration Time and Average Reward Over Episodes



The success of the project does not mean that there aren't improvements that could be made. Currently, the current velocity of the kart, v_i and the wheel rotation θ_i are both modified by the same scalar modifier value, s. They should each have individual values to adjust their impact on the overall reward at that step which would allow for finer tuning of the agents' priorities.

Another adjustment that could be made would be to instead of tracking iteration time reward

through increases in the velocity, the function could provide a constant reward each time a lap is finished. This would encourage the agent to complete laps quickly to increase its reward without directly tieing the score to the velocity. This would be an improvement as it more accurately models the problem presented to the agent, rather than the velocity which is just one factor of a successful solution to the problem. It would also mean that if the agent is trained on tracks with multiple paths, it could learn to prioritise paths which will result in a shorter lap time. Another side-effect of this change would be that the agent could be allowed to reverse once more, as driving backwards and forwards would no longer result in increasing the reward value.

6.3 Observations

The observations that the kart is able to make worked well to solve the problem presented by the project. For future investigations, the length of the rays should be increased - the value chosen is relatively short when compared with how far a real-world driver would be able to see. This would result in the agent having a better awareness of its environment, and therefore being able to make better decisions resulting in behaviour more accurate to the real world.

It would also be interesting to see if the agent would find it useful to be able to 'look backwards'. An agent could be trained with the ability to temporarily adjust its rays to point backwards instead of forwards to simulate a driver looking behind themselves. This functionality could be combined with the ability for the kart to reverse, allowing for more complex navigation, which would help for dead-end scenarios as mentioned in section 6.1.

6.4 Actions

Currently, the actions that the agent is able to take are directly tied to the wheel forces applied to the kart. If this project was to be continued, the agent could be updated to instead provide a steering wheel rotation, an accelerator pedal value and a brake pedal value which would then be automatically converted by the kart controller to the in-engine physics values. This would mean that the agent would act more like a real driver would, who only has control over the kart controls, rather than the physics of the kart itself. This could result in behaviour that is more accurate to the real world but could cause the agents to take longer to train due to the increase in complexity of the problem they must solve. It would also allow the agents to be more easily compared to human participants as they would use the same control scheme.

6.5 Evaluation of PPO versus SAC

While both PPO and SAC are successful at completing the tasks, they do not perform in the same manner once trained. Our results do align with Xu et al.;[19] in general, PPO is more successful at completing the task. However, success at completing the task is not the only important factor. Though harder to quantify, it is important to consider the feel of the behaviour of the agents. In particular, the nature of SAC and its focus on exploration results in an agent that is much more likely to vary its decisions under the same conditions when compared to

PPO, which generally sticks with the same decisions when presented with the same scenario.

The slightly higher reward value of PPO after training is mirrored in its ability to navigate the various tracks consistently and successfully - it is much less likely to collide with the edges or have other issues than SAC. However, PPO seems to suffer from overfitting much more than SAC does as evident from its reluctance to take actions that it would not have had to during its training. SAC is more resistant to this overfitting, which is likely a result of the entropy that is involved in its design; this randomness means that it is forced to deviate from previous actions which is what helps to avoid the problems from overfitting. It is hard to tell how much of the behaviour of PPO is due to true overfitting, and how much is due to its design to stay proximal to previous policies - it seems that the design of PPO has a flaw that it is prone to overfitting and requires more care than other algorithms during training to ensure that it does not occur.

In terms of actual use cases, the results indicate that neither PPO nor SAC is better - they both have merits for different scenarios. Training with PPO results in a predictable agent that is accurate and consistent; it is well suited for a task that requires actions that do not appear particularly human-like while SAC's behaviour appears more similar to how a human would perform.

6.6 Further Improvements

There are many other extensions to this project that have enough merit to them to be worth investigating. It would be easy to train other RL algorithms such as MA-POCA[35] and then evaluate those agents against the same tests as PPO and SAC. Evaluating many RL algorithms in the same way, on the same task and tests would provide a thorough and fair comparison which could provide useful insight into what each algorithm is well suited for, and what tasks they are not well suited for.

One of the issues with the agents (particularly SAC) is recovering from collisions. This makes sense considering how the training in this project was set up. When the kart crashes, the episode is ended. In order for the agent to learn how to continue from crashes, it must be able to continue training after the crash occurs. Building a system that would let the agent attempt to recover from the collision for a certain amount of time before ending the episode would enable the agents to learn to recover from failure.

It would also be good to use the environments built to train agents using other kinds of ML algorithms, for example, imitation learning algorithms such as BCO.[36] Imitation Learning is a subset of ML where agents learn from actual humans providing the actions, and attempt to mimic their actions in order to complete the defined task. Similarly to imitation learning, Inverse RL[37] is a RL technique that works by attempting to extract a reward function from observed optimal behaviour which could be used to build successful, human-like agents. Comparing the results of these algorithms with more traditional RL algorithms such as PPO and SAC would allow for a detailed comparison of the behaviours of agents left to learn on their own, versus ones which learn from humans. This style of algorithm could also be able to mimic human behaviour more accurately, which may be desirable depending on their application - see section 6.7.

Another improvement could be to train the agent simultaneously against other instances of the agent (known as Multi-Agent Reinforcement Learning[38]). This is well supported in the Unity ML-Agents package and would allow for agents to train in a more complex race scenario, similar to Wurman et al.[16] The agents could also then have a reward value based on their current position in the race, further encouraging faster lap times without having to reward based on velocity as mentioned previously. This could be extended further, and make use of a real-world racing game to train agents rather than the simple simulation developed for this project.

6.7 Applications for future work

There are two major applications for the kind of reinforcement learning agents trained in this project; self-driving cars and videogame AI. While self-driving cars do need to learn a lot of the same skills as a game AI, they are a less appropriate use case for the agents in this project as they need to operate safely in the real world. This means they must be trained in an environment with very accurate physics intended for simulating the real world, which is less of a focus of Unity due to its design for use in games. They also need to make use of much more complex observations, including computer vision and are focused on safely navigating environments rather than completing laps which was not a focus of this project.

In contrast, this project is well suited for forming part of a game AI. The agent is already part of a popular engine, so would be easy to integrate. If certain improvements mentioned previously are implemented, such as using a steering wheel, accelerator and brake pedals for the actions instead of physics values, it would be easy to map to existing game controls. It would also be possible to design a system that starts with a simple ‘easy’ agent, that is then allowed to train against the user as they play using imitation learning. This would result in an AI agent that is capable of learning from the player; improving as the player improves and mimicking any clever tactics that the player uses. It also means that if the player struggles with the game, the AI agent will learn ‘bad’ actions from them, reducing the difficulty of the AI opponents to mimic the player’s abilities essentially creating an automatic difficulty balancing system for the game.

When considering the specific behaviour of the two algorithms, as determined by this project, PPO would be best suited for an enemy with a complex series and patterns of regular attacks that the player can slowly learn and predict in order to beat; this functionality would be similar to a more classic game ‘AI’ but has the advantage that the developer does not need to manually create these patterns and strategies.

SAC would be more suited for an AI that should appear ‘human-like’. Its preference for exploration and making different decisions is a natural advantage over a traditional game ‘AI’ as this behaviour is typically difficult to program. An SAC agent would be well suited for acting as a computer opponent in multiplayer games due to this more natural behaviour, in particular, its ability to make decisions that appear unpredictable, which is a difficult thing to include in a traditional game AI.

The difference in the behaviour of agents depending on the algorithm used implies that specific variations of these algorithms could be written to encourage certain kinds of

behaviour. Currently, most RL research is into producing ‘optimal’ agents that are as good at completing a task as possible. Game developers or researchers could instead focus on creating training algorithms that produce agents with specific behaviour that they want to see in their game AI, or with a focus on appearing to act like a human.

Regardless of the algorithm used, RL has the potential to save developers large amounts of time, allowing them to work on other features of the game while the agents are training, instead of having to manually program the AI. It could also give the ability to easily create multiple difficulties of AI by saving the agents at different stages of training and providing the ability to easily create variants by simply altering the rewards and leaving the agent to train once more.

A piece of research into the use of different reinforcement learning algorithms for game AI playing against real players, similar to Thurau et al.,[39] and an evaluation of their experience would be a good topic for another project that builds on this one.

6.8 Conclusion

In conclusion, this project indicates that there is significant merit in the idea of applying reinforcement learning techniques to the field of game development. Our results indicate that both SAC and PPO are promising algorithms for this use case with different advantages to each. While some algorithms are quantifiably better at certain tasks than others the results indicate that in this case, rather than one being strictly ‘better’ than the other, it is important to consider the intended use case and desired behaviour of the trained agent. Assuming that there are no sudden developments that change the current state of RL, finding and tailoring an algorithm that will fit the specific problem is more important than using the current best ‘jack of all trades’ algorithm. This project also indicates that further research into building a game with RL agents built-in, or into RL algorithms specifically for this use are likely to yield promising and informative results.

References

[1] G. N. Yannakakis, “Game AI Revisited,” p. 285–292, 2012. [Online]. Available: https://doi.org/10.1145/2212908.2212954

[2] Unity Technologies, “Unity ML-Agents Toolkit.” [Online]. Available: https://github.com/Unity-Technologies/ml-agents

[3] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, “Unity: A General Platform for Intelligent Agents,” 2018. [Online]. Available: https://arxiv.org/abs/1809.02627

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[5] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, “Soft Actor-Critic Algorithms and Applications,” *CoRR*, vol. abs/1812.05905, 2018. [Online]. Available: http://arxiv.org/abs/1812.05905

[6] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor,” *CoRR*, vol. abs/1801.01290, 2018. [Online]. Available: http://arxiv.org/abs/1801.01290

[7] S. Fujimoto, H. van Hoof, and D. Meger, “Addressing Function Approximation Error in Actor-Critic Methods,” *CoRR*, vol. abs/1802.09477, 2018. [Online]. Available: http://arxiv.org/abs/1802.09477

[8] OpenAI, “Dota 2,” https://openai.com/blog/dota-2/.

[9] A. Ezquerro, M. A. Rodriguez, and R. Tellez, “OpenAI ROS.” [Online]. Available: http://wiki.ros.org/openai_ros

[10] Stanford Artificial Intelligence Laboratory et al., “Robotic Operating System.” [Online]. Available: https://www.ros.org

[11] N. Koenig and A. Howard, “Design and use paradigms for Gazebo, an open-source multi-robot simulator,” vol. 3, pp. 2149–2154 vol.3, 2004. [Online]. Available: https://ieeexplore.ieee.org/document/1389727

[12] OpenAI and Farama Foundation, “Gymnasium.” [Online]. Available: https://github.com/Farama-Foundation/Gymnasium

[13] “MuJoCo.” [Online]. Available: https://gymnasium.farama.org/environments/mujoco/

[14] “Atari.” [Online]. Available: https://gymnasium.farama.org/environments/atari/

[15] M. Foxman, “United we stand: Platforms, tools and innovation with the unity game engine,” *Social Media+ Society*, vol. 5, no. 4, p. 2056305119880177, 2019.

[16] P. R. Wurman, S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. J. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs, L. Gilpin, P. Khandelwal, V. Kompella, H. Lin, P. MacAlpine, D. Oller, T. Seno, C. Sherstan, M. D. Thomure, H. Aghabozorgi, L. Barrett, R. Douglas, D. Whitehead, P. Dürr, P. Stone, M. Spranger, and H. Kitano, "Outracing champion Gran Turismo drivers with deep reinforcement learning," *Nature*, vol. 602, no. 7896, pp. 223–228, Feb 2022. [Online]. Available: https://doi.org/10.1038/s41586-021-04357-7

[17] A. J. M. Muzahid, S. F. Kamarulzaman, and M. A. Rahman, "Comparison of PPO and SAC Algorithms Towards Decision Making Strategies for Collision Avoidance Among Multiple Autonomous Vehicles," pp. 200–205, 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9537063

[18] I. Vohra, S. Uttrani, A. K. Rao, and V. Dutt, "Evaluating the Efficacy of Different Neural Network Deep Reinforcement Algorithms in Complex Search-and-Retrieve Virtual Simulations," pp. 348–361, 2022. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-95502-1_27

[19] C. Xu, R. Zhu, and D. Yang, "Karting racing: A revisit to PPO and SAC algorithm," pp. 310–316, 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9718819

[20] J. Bell, *What Is Machine Learning?* John Wiley & Sons, Ltd, 2022, ch. 9, pp. 207–216. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119815075.ch18

[21] F. Woergoetter and B. Porr, "Reinforcement learning," *Scholarpedia*, vol. 3, no. 3, p. 1448, 2008.

[22] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, Cambridge MA, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[23] B. Christian and T. Griffiths, *Algorithms to Live by: The Computer Science of Human Decisions*. 1 London Bridge Street, London, SE1 9GF: William Collins, 2016.

[24] P. M. Todd and G. F. Miller, "The Evolutionary Psychology of Extraterrestrial Intelligence: Are There Universal Adaptations in Search, Aversion, and Signaling?" *Biological Theory*, vol. 13, no. 2, pp. 131–141, Jun 2018. [Online]. Available: https://doi.org/10.1007/s13752-017-0290-6

[25] R. C. Wilson, A. Geana, J. M. White, E. A. Ludvig, and J. D. Cohen, "Humans use directed and random exploration to solve the explore–exploit dilemma." *Journal of Experimental Psychology: General*, vol. 143, pp. 2074–2081, 2014. [Online]. Available: https://doi.org/10.1037/a0038199

[26] Unity Technologies, "Unity ML-Agents PPO Implementation." [Online]. Available: https://github.com/Unity-Technologies/ml-agents/tree/develop/ml-agents/mlagents/trainers/ppo

[27] Unity Technologies, "Unity ML-Agents SAC Implementation." [Online]. Available: https://github.com/Unity-Technologies/ml-agents/tree/develop/ml-agents/mlagents/trainers/sac

[28] Feuersänger, Christian, “pgfplots.” [Online]. Available: https://tikz.dev/pgfplots/

[29] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, “Manifesto for Agile Software Development,” 2001. [Online]. Available: https://agilemanifesto.org/

[30] Kenney, “Prototype Textures.” [Online]. Available: https://www.kenney.nl/assets/prototype-textures

[31] A. Chauhan, L. Naagar, S. Chawla *et al.*, “Design and Analysis of a Go-kart,” *International Journal of Aerospace and Mechanical Engineering*, vol. 3, no. 5, pp. 29–37, 2016.

[32] Y. Hu, W. Wang, H. Jia, Y. Wang, Y. Chen, J. Hao, F. Wu, and C. Fan, “Learning to utilize shaping rewards: A new approach of reward shaping,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 15 931–15 941, 2020. [Online]. Available: https://arxiv.org/abs/2011.02669

[33] Drunks, Stepan, “Simple Roads.” [Online]. Available: https://assetstore.unity.com/packages/3d/environments/roadways/simple-roads-212360

[34] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, “A Study on Overfitting in Deep Reinforcement Learning,” *CoRR*, vol. abs/1804.06893, 2018. [Online]. Available: http://arxiv.org/abs/1804.06893

[35] A. Cohen, E. Teng, V. Berges, R. Dong, H. Henry, M. Mattar, A. Zook, and S. Ganguly, “On the Use and Misuse of Absorbing States in Multi-agent Reinforcement Learning,” *CoRR*, vol. abs/2111.05992, 2021. [Online]. Available: https://arxiv.org/abs/2111.05992

[36] F. Torabi, G. Warnell, and P. Stone, “Behavioral Cloning from Observation,” *CoRR*, vol. abs/1805.01954, 2018. [Online]. Available: http://arxiv.org/abs/1805.01954

[37] A. Y. Ng, S. Russell *et al.*, “Algorithms for inverse reinforcement learning.” in *Icml*, vol. 1, 2000, p. 2. [Online]. Available: http://www.datascienceassn.org/sites/default/files/Algorithms%20for%20Inverse%20Reinforcement%20Learning.pdf

[38] K. Zhang, Z. Yang, and T. Başar, *Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms*. Cham: Springer International Publishing, 2021, pp. 321–384. [Online]. Available: https://doi.org/10.1007/978-3-030-60990-0_12

[39] C. Thurau, C. Bauckhage, and G. Sagerer, “Imitation learning at all levels of game-AI,” in *Proceedings of the international conference on computer games, artificial intelligence, design and education*, vol. 5, 2004.

[40] OpenAI, “GPT-4 Technical Report,” *ArXiv*, vol. abs/2303.08774, 2023. [Online]. Available: https://arxiv.org/abs/2303.08774

[41] L. Clarke, “Call for AI pause highlights potential dangers,” *Science (New York, NY)*, vol. 380, no. 6641, pp. 120–121, 2023.

Appendix A

Self-appraisal

A.1 Critical self-evaluation

As I was allocated a machine learning-based project, but was not taking the machine learning university module, I knew that the first step to this project would be to do some personal research into the field. I made sure to read a textbook recommended to me by my supervisor on the subject.[22] This was a great place to start and ensured I had a solid basis to use for the rest of the project. I also made sure that I had enough time to complete the project by taking most of my final year modules in the first term, leaving plenty of free time to work on it in the second term.

Overall, I am pleased with the outcome of this project. It met its objectives, producing results that yielded some interesting findings and had the potential for further research. One of the reasons for this is likely due to the care I took in planning the project, ensuring each section was completed on time to keep the project moving. Performing the initial preliminary research was also crucial in the successful outcome of the main project. Leveraging Unity for the simulation environment and C# for the programming meant that I could make use of technologies I was already familiar with, which meant I could skip the extensive learning curve associated with other tools available.

It was difficult to work around training the AI; while the agents were training it became virtually impossible to work on other aspects of the project. Luckily, the generous timeframe left for each task meant this was not an issue, but it was frustrating having to wait for a process like that to complete before being able to work on other aspects. It would be good to be able to use a remote, powerful machine for training in order to speed this process up and allow me to work on other tasks while it was running.

It would also have been nice to have more time available to fix some of the minor issues, such as the difficulties SAC had with track 3, or to complete some of the extensions. However, the project was a success without these so it was not a major issue. It would also be great to have more concrete data for the results section of the agents running on the alternative tracks in order to back up the findings with quantifiable data that could be discussed in more detail.

A.2 Personal reflection and lessons learned

I am pleased with both the outcome and the approach I took to this project. I learnt a lot about ML, particularly RL as a result of doing it which was very interesting. I was also really glad to be able to relate it to the field of game development, as it is a personal interest of mine and an area with relatively little research in comparison to other RL fields such as robotics or autonomous driving. However, due to being ‘a game’ it was easy to get distracted working on

things like the appearance of the project, which was not a factor in the actual results or outcome. It did however help with motivation to continue the work, which would have been harder to find if it was in a different field. It would be nice to extend the research further in a more freelance manner, being able to work on parts that seemed interesting without the focus on specific parts and the timeline associated with a final university project.

A.3 Legal, social, ethical and professional issues

While there are not many legal, social ethical or professional issues with this project, there are some to consider if the project was extended, and lots that apply to AI in general.

A.3.1 Legal issues

This project makes use of some code and assets that are developed by third parties. While care was taken to ensure that they were all used according to their specific licenses, it is possible that the license listed was incorrect. In this case, then we must be ready to update the project so that it no longer uses these assets or is in compliance with the correct license.

Another potential issue would be if the research was extended to make use of a specific existing game. In this case, the game developers should be contacted to ensure that they consent for their game to be used for the research. Another project development that could cause issues would be if the project was extended to train using a player's actions. In this case, the developers must consider who owns the right to this AI; if it was trained on a player's actions, do they have a legal claim to the resulting AI model? A final issue is the aspect of responsibility; if an AI model makes a decision that causes some kind of legal issue, who is responsible for it? Should the developers, the operator, the project owner or someone else be blamed? There is no clear answer to these questions, so they should be considered carefully if future research is carried out.

A.3.2 Social issues

If AI is used to save time in a business setting, it could result in a loss of employment or responsibility for humans who were previously assigned to this task. It is important to ensure that these people will be assigned appropriate alternative work if their existing work is replaced by an AI, or if no work is available that they are looked after to ensure that they will not suffer from this use of AI. Another social factor to consider is a reduction in human contact; if AI is used to replace a personal assistant, or to replace other humans in a multiplayer game the reduced amount of human contact and the impact this has on society must be investigated to ensure that it does not lead to any kind of social ostracization or other negative outcomes that outweigh the positives or have other severe negative impacts.

A.3.3 Ethical issues

There are lots of potential ethical issues with AI. Due to the 'black-box' nature of most modern AI, there is a lack of transparency about why most decisions are made by AI. This leads to the issue of responsibility (as mentioned in the legal issues section); if there is no way to

understand why a decision was made, how is it possible to ethically assign responsibility or blame for that decision? Considering the idea of training the AI models against a player, is it ethical to have them unwittingly teach an AI? Does the player 'own' their own strategies, meaning that the AI learning from them is a form of plagarism? These ethical dilemmas do not have a clear answer but must be considered before a project is undertaken that makes use of AI.

A.3.4 Professional issues

Due to the relatively recent emergence of powerful AI, there is a general lack of professional standards and guidelines that companies or researchers can follow to ensure that their use of AI is professionally sound. This means that any time AI is used in business or for research it is up to the individuals to assess their use. This is very a large responsibility, especially considering that these people might not be professionally qualified to make these decisions. This also raises the question of who should be responsible for writing and maintaining these guidelines, and who should be responsible for educating people about this. It is a huge undertaking that requires much more research before an informed decision and regulations can be made. Sadly, most current research is into expanding the abilities of AI causing regulations and proper impact studies to lag behind research significantly. This is especially relevant to the current state of AI, with recent developments in AI (particularly natural language processing with GPT-4[40]) causing many experts to call for a pause in AI development to give researchers and governments time to write these guidelines and regulations.[41]

Appendix B

External Material

B.1 Visual Studio and Visual Studio Code

Visual Studio is a comprehensive IDE developed by Microsoft. Used for programming C# Unity scripts. https://visualstudio.microsoft.com/

Visual Studio Code is a lightweight IDE developed by Microsoft. Used for editing Python scripts used for data processing. https://code.visualstudio.com/

B.2 Pgfplots

Plotting library for producing high-quality graphs in LaTeX. Used to produce graphs used in the report. https://ctan.org/pkg/pgfplots.

B.3 Git

Free and open-source distributed version control system. Used for version control and backup of the project. https://git-scm.com/

B.4 Unity and Unity ML-Agents

Unity is a cross-platform game engine for developing 2D and 3D applications. Used to run simulations and build agent environment. https://unity.com/

Unity ML-Agents is an open-source toolkit for Unity that enables modern Reinforcement Learning development within the Unity Game engine.
https://github.com/Unity-Technologies/ml-agents

B.4.1 PPO Implementation

Specific implementation of PPO algorithm packaged with Unity ML-Agents described by Shulman et al.[4] Used for training one of the RL agents.
https://github.com/Unity-Technologies/ml-agents/tree/develop/ml-agents/mlagents/trainers/ppo

B.4.2 SAC Implementation

Specific implementation of SAC algorithm packaged with Unity ML-Agents described by Haarnoja et al.[5] Used for training one of the RL agents.
https://github.com/Unity-Technologies/ml-agents/tree/develop/ml-agents/mlagents/trainers/sac

B.5 Prototype Textures

Textures for 3D prototyping developed by Kenney. Used for texturing various objects in the simulation in order to aid visibility. https://www.kenney.nl/assets/prototype-textures

B.6 Simple Roads

Free 3D set of models for designing race tracks or city junctions. Used to build tracks for the agent to train and test on.
https://assetstore.unity.com/packages/3d/environments/roadways/simple-roads-212360

Appendix C

Supplementary Material

Algorithm 1 PPO Pseudocode - Schulman et al.[4]

for iteration = 1,2, ... **do**
  for iteration = 1,2, ..., N **do**
    Run policy $\pi_{\theta_{old}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1$, ..., $\hat{A}_T$
  end for
  Optimise surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{old} \leftarrow \theta$
end for

Algorithm 2 SAC Pseudocode - Haarnoja et al.[5]

Input: θ_1, θ_2, ϕ ▷ Initial parameters
$\theta_1 \leftarrow \theta_1, \theta_2 \leftarrow \theta_2$ ▷ Initialise target network weights
$D \leftarrow \emptyset$ ▷ Initialise an empty replay pool
for each iteration **do**
  for each environment step **do**
    $a_t \sim \pi_\phi(a_t|s_t)$ ▷ Sample action from the policy
    $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$ ▷ Sample transition from the environment
    $D \leftarrow D \cup (s_t, a_t, r(s_t, a_t), s_{t+1})$ ▷ Store the transition in the replay pool
  end for
  for each gradient step **do**
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update policy weights
    $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$ ▷ Adjust temperature
    $\bar{\theta}_i \leftarrow T\theta_i + (1 - T)\bar{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target network weights
  end for
end for
Output: θ_1, θ_2, ϕ ▷ Optimised parameters

Snippet C.1: Example JSON Data

```
[
    {
        "episodeNumber":0,
        "iterationTime":240.01445,
        "reward":0.004649742,
        "checkpointCount":0,
        "averageVelocity":0.0003874767,
        "timedOut":true
    },
]
```

Table C.1: Kart Agent Physics Properties (Unity Arbitrary units)

Property		Value
Body Mass		2000
Drag		0
Angular Drag		0.05
Speed		700
Steer Speed		50
Max Steer Angle		30
Wheel Mass		80
Wheel Radius		4
Wheel Damping Rate		0.25
Suspension Distance		0.3
Wheel Suspension Spring	Spring	35000
	Damper	4500
	Target Position	0.5
Wheel Forward Friction	Extremum Slip	2
	Extremum Value	5
	Asymptote Slip	5
	Asymptote Value	2
	Stiffness	1
Wheel Sideways Friction	Extremum Slip	0.4
	Extremum Value	1
	Asymptote Slip	0.5
	Asymptote Value	0.75
	Stiffness	1

Table C.2: Kart Agent Reward Values

Factor	Value
s	0.001
p	-5
r	1
v	Variable depending on episode context
θ	Variable depending on episode context, ($-30° \leq \theta \leq 30°$)
w	Variable depending on episode context, $w = [0, 1]$
c	Variable depending on episode context, $c = [0, 1]$

Figure C.1: First Stage Preliminary - YouTube (00:30) - https://youtu.be/S_hxA_rSakc

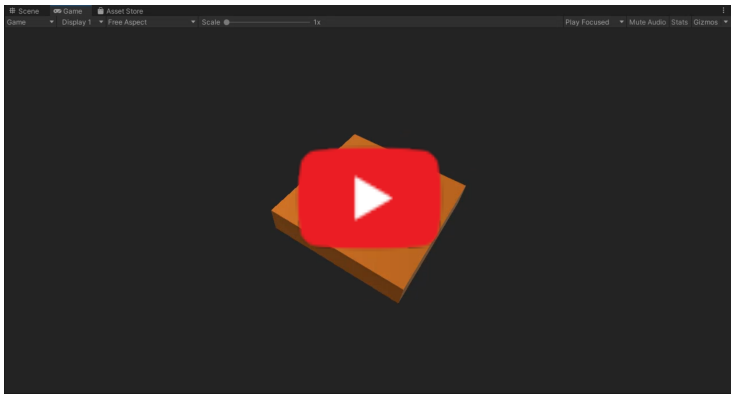


Figure C.2: Third Stage Preliminary - YouTube (00:31) - https://youtu.be/3_sh7YbSX38

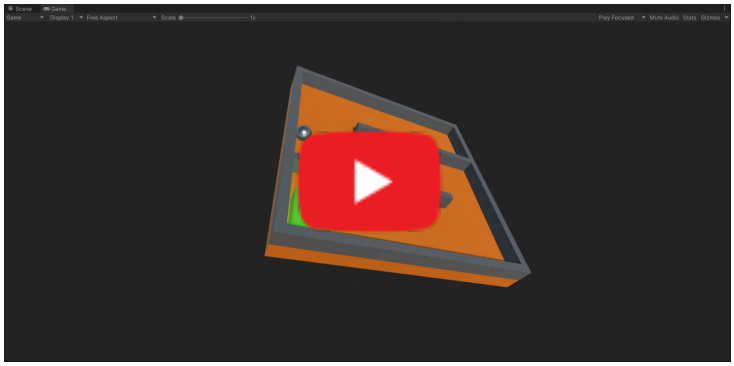


Figure C.3: Preliminary Creative Solution - YouTube (00:34) - https://youtu.be/JVnJLgiuVGs

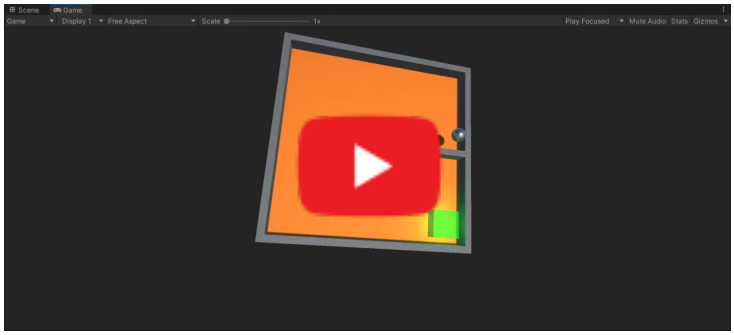


Figure C.4: Full Agent Demonstration (15:15) - https://youtu.be/F5yxp3WKO-o

Figure C.5: Project Gantt Chart

#	Task	Duration	Start	End
01	Initial Investigation & Learning how to use toolkit	3 Weeks	Jan 25th	Feb 15th
02	Build Environment for Agent Training	2 Weeks	Feb 16th	March 1st
03	Training and Tuning of Agent	3 Weeks	March 2nd	March 22nd
04	Introducing Agents to New Environments	1.5 Weeks	March 23rd	April 1st
05	Final Writeup and Evaluation	3.5 Weeks	April 2nd	April 30th

Timeline												
January	February				March				April			
Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4

Figure C.6: Reward Over Episodes (Large)

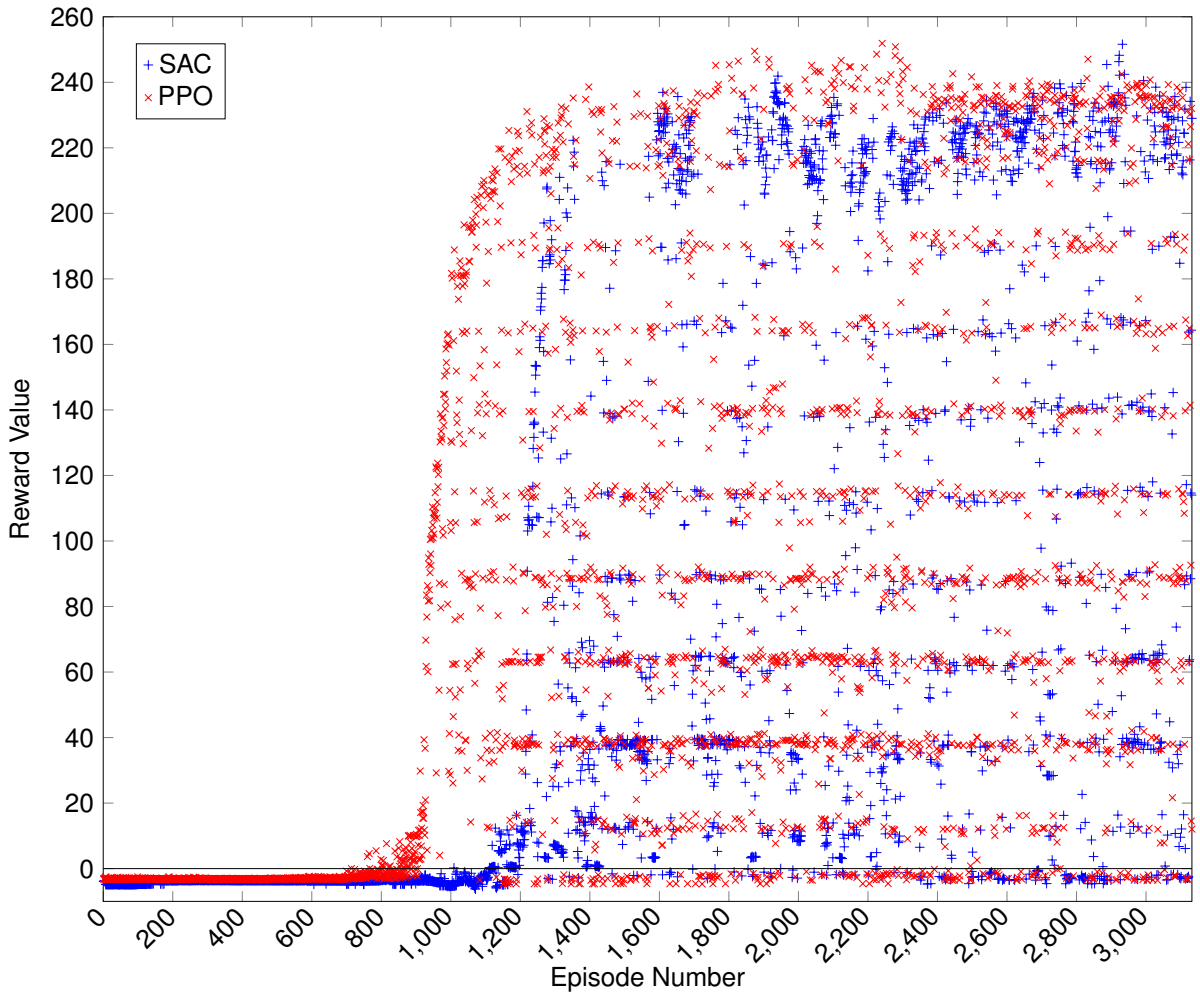


Figure C.7: GitHub Repository
https://github.com/darthmorf/ohmu-ml