Final Summer Dissertation

LK274 Master of Science in Artificial Intelligence and Machine Learning

Comparative Analysis of Reinforcement Learning and Imitation Learning: Evaluating learning rates and processing time across varied task complexities.

Department of Computer Science & Information Systems

Maciej Augustynek – 18222757

Supervisor: Dr Malachy Eaton

25th August 2024

Declaration

This dissertation is presented in fulfilment of the requirements for the degree LK274 Master of Science in Artificial Intelligence and Machine Learning project. This is entirely work completed by me and has not been previously submitted to any other University or Higher Education Institution. All work and finding done by other have been fully acknowledged and referenced.

Name : Maciej Augustynek

Date : 25/08/2024

Data Source and Generative AI

All data used for creating this document has been collected and prepared by me. I have not used generative AI to conduct my research or to help in writing this document.

Abstract

The training methods of Reinforcement Learning and Imitation Learning have been studied individually where different algorithms for training have been compared inside their respective learning methods. However, throughout my research I have observed there is scarcely any research done in comparing the efficacy of both Imitation and Reinforcement learning collectively. In this dissertation I have analysed and contrasted these two methods of training AI agents across both the performance of agents trained and also how time and resource intensive each respective method is. To keep the comparison consistent I have trained Reinforcement Learning agents using the Proximal Policy Optimisation(PPO) algorithm while the Imitation Learning is done using Generative Adversarial Imitation Learning(GAIL). The methods are tested in multiple environments of increasing task complexity created using the Unity ML-Agents package. This dissertation also focuses on comparing the benefits of using each method. The results of the experiments reject the idea that task complexity should be considered when deciding on which learning method should be used.

Table of Contents

Table of figures

1. Introduction

1.1 Background and Motivation

Reinforcement Learning (RL) and Imitation Learning (IL) are common methods used to train AI agents, however it is not always beneficial to join these two methodologies as each come with their own challenges (Nair et al., 2018). While investigating this topic, I have noticed that while both methods are researched independently inside their own respective domains, I couldn't find instances of them being compared simultaneously. RL and IL both have great comparison studies done regarding the effectiveness of different algorithms in various environments, but despite that, it is difficult to find any research comparing the use of both methods for the same problems and adequately measuring the benefits and demerits of using them for training agents in different tasks. The main motivation for me to take up this project is that by conducting further research into comparing RL and IL, we can get a better insight into when we should use one approach over the other. There are unique advantages for each method, and it would be beneficial to narrow down the specifics of when a programmer should consider each approach to speed up training time of agents.

1.2 Goals and Objectives

The goal of this dissertation is to create a set of increasingly complex environments inside Unity Machine-Learning Agents with trainable agents. In each of these environments the agents will be trained in 3 main ways: one will be trained using RL only, second one will be trained using only ILand the last agent will be trained by applying both Learning methods simultaneously. During the training process the following metrics will be tracked:

Convergence Speed – The rate at which the agent learns the best policy for a task and cannot improve its results any further. We want the agents to arrive at the optimal solutions with the minimal amount of training steps taken.

Maximum total reward – The average maximum reward obtained by an agent in an environment. The rewards are placed to influence the process of reinforcement learning where the agent strives to maximise this reward value.

Real time taken – Similar to convergence speed however with the difference of being measured in real time compared to training steps. This measurement is taken to be able

to compare the processing time differences between the different RL and IL approaches. While one algorithm can possibly achieve a better result at a smaller amount of training steps it wouldn't be very beneficial if the real training time was much longer than the other approaches.

The first two metrics are common methods of measuring RL performance (Kaelbling et al., 1996) while the last metric is used for me to track the learning time with real-time instead of using steps and episodes. Real-time performance can be affected by algorithm complexity since different approaches can have higher processing requirements. By tracking these specific metrics and how they are affected by the complexity of each task I hope to establish a link between the environment complexity and best suitable method of learning. Each method will be compared to quantify the performance impact of applying the methods which will enable a more accurate insight into when it is recommended to use which training algorithm.

1.3 Overview of the Dissertation

The overall goal of the dissertation is to investigate the benefits of utilising Reinforcement Learning with Imitation Learning while also investigating potential demerits by comparing their performance across different task complexities. The focus will be on trying to qualify which algorithm is most suited depending on the training environment. By researching whether task complexity is a notable factor in choosing the learning algorithm it can help other programmers and researchers to justify their approach when tackling new problems.

1.4 Dissertation Outline

This report will consist of 5 chapters.

Inside the literature review there is a brief discussion about the current reinforcement and imitation learning methods, the theory of how they function as well as outlining the methods that will be used for the experiments further in the paper.

In section 3 will be outlining multiple environments created for agents to benchmark the different algorithms. I will also be listing the engine that will be used to run the experiments and the machine specifications to give readers a reference to the processing time needed depending on their processing capacity.

Section 4 will focus on compiling the results from running training inside the different environments with focus on the comparing the metrics between each setup. It also discusses the implications of the results achieved.

Finally section 5 will focus on summarising the findings from the experiments and how it relates to the goal of the dissertation. It will also outline potential improvements that could be made in future development.

2. Literature Review

2.1 Reinforcement Learning

Reinforcement Learning dates back to early work in psychology, neuroscience, statistics, and computer science (Kaelbling et al., 1996). It offers an attractive way to create solutions by allowing to program agents by rewarding them for actions which aid in completing the task and punishing them if they are showing unfavourable behaviour(Li, 2017). The main reason behind the attractiveness of such approach is the fact that programmers do not exactly define *how* the task will be achieved, and this gives a rise to varied training results where each training can possibly lead to different emerging behaviours which all try to maximise the reward. In Reinforcement Learning agent learns behaviours through trial-and-error when it interacts with its environment to try and come up with an optimal policy to decide its actions given their current state(Li, 2017).

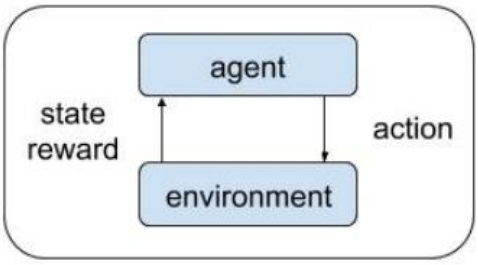

Fig. 1 RL State-Action-Reward model(Li. 2017)

Figure 1 shows a basic overview of how reinforcement learning allows the agent to learn through trial-and-error. An agent placed in an environment performs and action given a state after which the environment returns a reward for the action that the agent took. The reward can be either positive or negative. In order to train an agent, it is crucial to carefully select what information should be visible to the agent during each step also known as the observation-space of the agent. Defining a suitable observation space to allow the agent to infer underlying states can significantly affect how the agent learns (Kim & Ha, 2021). In order to optimise learning an agent should receive the minimal amount of information necessary to solve the problem. This is because each additional variable adds new states and correlations between data points which greatly impacts the learning time for an agent. Similarly to the observation-space, an

action-space has to be defined which corresponds to all possible actions an agent can take for a given state. The action space is more clearly defined as for example in robotics your action space is defined and limited by the hardware of the robot just corresponds to all of the actions that your agent can perform. Lastly the reward structure for the agent has to be defined which is the most important to set up correctly as RL uses these rewards to provide feedback when updating the policy for the agent(Li, 2017). One of the most impressive accomplishments of reinforcement learning that motivated more people to start working on RL is the development of Googles AlphaGo(Chen, 2016) which was able to reliably beat human professional players in the old board game of Go.

2.1.1 Exploration vs Exploitation

One major problem in reinforcement learning is the issue of exploration vs exploitation. A reinforcement learning agent must explicitly explore its environment in order to find good solutions however how much should an agent explore before it settles on a solution? Ideally the agent starts off stochastic and gradually perform less random actions over time as it explores more states. This problem is perfectly showcased in the simple k-armed bandit problem which has been thoroughly studied in mathematics and statistics (Berry & Fristedt, 1985). The problem consists of an agent placed in a room of *k* one-armed bandit gambling machines and the agent is only allowed a set amount of pulls *x*. Each step an agent can play any of the machines however each machine has a different unknown probability of paying out. This problem directly showcases the trade-off between exploration and exploitation. Choosing to prioritise exploration of all machines which gives the most accurate estimate of each probability at the cost of the final score, or instead if we choose to prioritize exploitation, we could pull a machine which pays out and just keep using that machine. In that scenario we are potentially losing more points as there could be a machine with a higher payout rate than the one we got lucky on initially. There are various solutions to the k-armed bandit but the point of it is to showcase one of the major problems inside reinforcement learning in order to optimise training and convergence time.

2.1.3 Rewards

What fundamentally guides an agent towards making the right decisions are the reward signals(Li, 2017). Setting up a correct reward feedback is crucial to obtaining desired results from an agent as even small oversights can lead to the agent refusing to explore or prioritising

undesired behaviours as a result of the rewards they receive. There are many strategies for setting up the reward structure and each have their own unique benefits. A sparse reward structure with rewards only being given at the point of winning or losing a game is an option for rewards shaping. In this environment the agent would converge quite slow as it rarely receives any feedback on what actions it should be performing. Even though the learning would be slow it could lead to the development of new unexpected methods of solving the environment. In comparison in spaces where rewards are given to an agent during every step using some kind of formula or according to some specific metric the agent is less likely to develop unique approaches to a problem. This can be a double-edged sword as it allows the agent to converge and learn much faster but at the same time the reward structure might restrict new better solutions that could be available to solve the problem. It is important to not overly incentivise an agent to not stifle naturally occurring solutions unless that is what you are aiming for (Sutton & Barto, 2018).

2.1.2 Q-Learning and Deep Q-Networks(DQN)

2.1.2.1 Q-Learning

Q-Learning and Deep Q-Networks are both used as methods of training RL agents and updating which actions an agent should take dependent on the current states visible to the agent. The Q-Learning algorithm is a value based approach which attempts to fill a policy table called the Q-Table(Loeber, 2022). The Q-Table stores all expected reward values for each action at every state possible in the environment. The table values are updated using the Bellmans Equation after each step by the agent exploring the environment until no further improvement is made. After training the Q-Table it explicitly defines the best possible move at that state which allows the agent to optimally complete its task. The drawback of this method is that it can struggle when trying to evaluate large state spaces which extends to tasks where observations made by the agents are in the continuous domain instead of a discrete one (Luu, 2023). In continuous state environments the usual Q-Table would grow to an immeasurable size because of the nature of the states. In an environment where for example the agent walks and takes in observations about its head's absolute height and angles of legs the traditional Q-Learning methods would require the algorithm to create states for every possible height and angle combination of the agent. In less complex scenarios it is possible to apply Q-Learning to continuous spaces by discretizing the states as an attempt to reduce the state-action space (Seyde et al., 2023).

2.1.2.2 Deep Q-Networks

The principle behind the implementation of DQNs is identical to Q-Learning where the Q-Table is replaced by a neural network. DQNs are used as a way to mitigate continuous state issues of the Q-Learning algorithm by instead of mapping Q-Values for each state in a Q-Table it takes in the observations and passes them through a neural network which in turn gives back the predicted probabilities (or Q-Values) for each action at that state (Xiong et al., 2018).

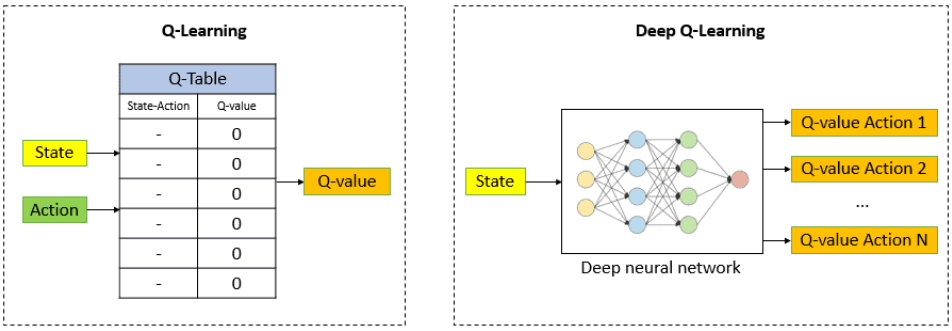

Fig. 2 Q-Learning vs Deep Q-Learning Diagram (Luu, 2023)

This project will be utilizing solely a variation of DQNs as some of the environments will contain both observations and actions inside the continuous domain. The continuous value aspect of the tasks will allow to create more complex environments to further test the algorithms and DQNs are suited perfectly for this task. Additionally, by solely using DQNs it keeps down the number of variables that can affect the results.

2.1.3 Deep Reinforcement Learning State-of-the-Art

Reinforcement Learning methods have a multitude of variants which tend to perform better in different applications. To narrow them down, common algorithms which perform well with continuous domains, robotics and game AI will be discussed as the environments that will be created fall under these requirements. Commonly used methods for this are policy gradient methods such as Proximal Policy Optimisation and actor-critic methods such as Soft-Actor Critic (Williams, 1992; Mnih et al., 2016).

2.1.3.1 Proximal Policy Optimisation (PPO)

Proximal Policy Optimization got introduced in 2017 as method to simplify the first attempt at a policy gradient method of Trust Region Policy Optimisation (TRPO) (Williams, 1992).

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

PPO update function (Simonini, n.d.)

PPO uses a policy gradient method to help with speeding up and fixing the instability issues of traditional DQN's. The "min" function is crucial in its function as it uses the lower bound viable update to change the policy (Schulman et al., 2017). By using the minimum, it tries to stabilise the learning of the network and ensures the changes made are not major to prevent sudden changes. The clip function in the algorithm also focuses on this issue as it is defined to only capture changes in policy in between (1-epsilon, 1+epsilon) with epsilon being variable.

These features of the update function effectively limit the update function and in cases where the model improves the model is rewarded by a small margin while in the case of the action being worse the model is penalised heavily.

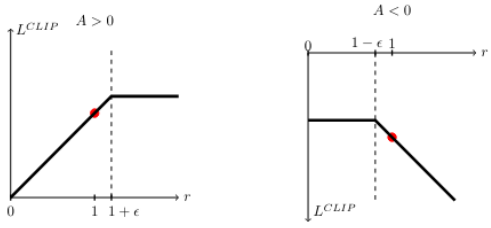

Fig. 3 PPO Effective update function when action is good (left) and when the action is bad(right). (Schulman et al., 2017)

2.1.3.2 Soft-Actor Critic (SAC)

Soft-Actor Critic introduced in 2018 as a method to combine the advantages of PPO and TD3 into a single update policy. The advantage of PPO is its stability but low sample efficiency while TD3 is more unstable but provides higher sample efficiency. SAC aims to maximise reward while also maximising entropy, which means that it tries to take the best

action while also acting as randomly as possible(Haarnoja et al., 2018). It adds an additional parameter called the entropy regularisation coefficient which controls the exploration and exploitation rates.

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta \left(\tilde{a}_\theta(s) | \, s \right) \right),$$

Fig. 4 SAC Update formula (Soft Actor-Critic — Spinning Up documentation, n.d.)

To justify a choice of algorithm I have found 2 papers which focus on comparing PPO and SAC performance in the gait generation domain. Both papers ultimately concluded that SAC outperforms PPO in almost all scenarios with the exception of scenarios with relatively small state action spaces(Mock, 2023). However PPO shows much greater stability throughout learning.

While subsequent tasks will be more challenging, an algorithm that is capable of giving more stable results is preferrable in this instance. Additionally changing the algorithm in between the environments will not be done as the focus is not on investigating SAC vs PPO performances and the update algorithm will remain a constant variable in the experiments. With more time SAC could be considered as the update algorithm for even more complex environments. As a result of these factors these PPO will be used as the update method for the experiments, as it will be sufficient for the complexity scope of this dissertation.

2.2 Imitation Learning (IL)

Imitation Learning is similar to RL as both often solve Markov Decision Process(MDP) problems and use Trust Region Policy Optimisation (TRPO) to improve its policy(Zheng et al., 2024). IL methods are an attempt to teach agents to mimic specific actions through providing demonstrations to the agents, where they can subsequently learn the mapping between the current state and what actions should be taken(Hussein et al., 2017). The main draw and advantage of this method is that it facilitates the training of intricate tasks without requiring full knowledge of the tasks and how they could be solved using an algorithm or designing reward structures unlike RL. For example, people are able to learn actions such as

swimming from demonstrations instead of through following mathematical functions. This makes it an attractive solution to solve complex problems such as training humanoid robot behaviours and self-driving vehicles (Hussein et al., 2017).

Similar to RL the examples are represented as pairs of observations and actions. The difference being that the desired actions are known from the supplied demonstrations during the training. Before any learning can be done state-actions pairs are recorded from sampling an expert performing the task, after which these demonstrations are used to train and optimise the policy. To prevent issues with the co-dependency between previous and future actions IL attempts to learn *trajectories*. Trajectories help IL by focusing on trying to imitate multiple steps which influence the environment instead of only direct state-action mappings (Hussein et al., 2017).

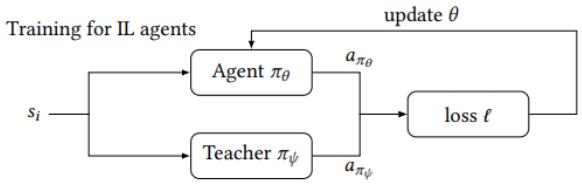

Fig. 5 General Approach to IL (Gavenski et al., 2024)

A major advantage of IL can be seen specifically in tasks where the goal is to directly imitate and not to optimise an action. This could be desired in many domains and especially robotics to prevent the robots from developing unhuman like behaviours. In those scenarios mostly the appearance of the developed actions are valued and not the other possible valid paths of accomplishing the tasks. This advantage is also the biggest drawback of this method of teaching agents as the agent can only perform as well as the best demonstrations that are fed while learning(Zheng et al., 2024). Other factors such as suboptimal and insufficient demonstrations prove to be a major challenge when training agents.

2.2.2 Imitation Learning State-of-the-Art

Currently the most used branches of IL include behavioral cloning, inverse reinforcement learning and adversarial IL(Zheng et al., 2024). All of these have different variations of training, two of the widely used algorithms will be discussed. Behavioural cloning

focuses on direct imitation of supplied behaviours, inverse reinforcement learning attempts to extract an optimal policy to fit the demonstrations and adversarial IL is used to tackle the challenge of limited demonstration data by generating new data that is used during training.

2.2.2.1 Behavioural Cloning

Behavioural Cloning is the earliest approach to conduct IL (Pomerleau, 1989), in which the agent learns to predict the most probable action in a state based on the expert dataset. BC attempts to estimate the agent's trajectory to match the expert trajectory. This method becomes very intensive for more complex tasks(Gavenski et al., 2024) where it requires a huge number of samples to learn about the possible states of the agent and how actions influence future states. For example, problems involving a more robust action space such as actions in the continuous domain require roughly a hundred times more samples than classic control tasks(Torabi et al., 2018). This happens as BC policies fall short when trying to estimate unseen states to known trajectories. This issue is not major as long as we wish to only evaluate on how good the agent is at imitating the expert demonstrations and not evaluate it based on how it solves the task(Gavenski et al., 2024). Currently BC is often used as a way to bootstrap other algorithms where it is used initially to fully imitate a behaviour after which a different approach such as RL methods are used to fully hone the agents performance.

2.2.3.2 General Adversarial Imitation Learning (GAIL)

GAIL was proposed in 2016 by Jonathan Ho et al. which combines imitation learning and generative adversarial networks to learn policies directly from data(Ho & Ermon, 2016). GAIL leverages Generative Adversarial Networks (GAN) by using them to generate state-action pairs that are then used in the IL process. The generated state-action pairs are then examined by GAIL's *discriminator* which tries to determine whether the pairs have been generated or if they come from the expert demonstration. During the learning, the discriminator attempts to update the policy to make the generated pairs match the expert pairs. As a result of the additional data produced by GANs GAIL is very sample efficient with demonstration data(Ho & Ermon, 2016). GAIL has proven to work much better in highly complex tasks even with continuous values and requires a much smaller demonstration set compared to conventional IL methods (Kinose & Taniguchi, 2019).

Additionally, the demonstrations don't have to be optimal for the agent to achieve competent results and a variant of GAIL called Task-achievement Reward GAIL (TRGAIL) is

even capable of learning with an incomplete expert trajectory. Due to the more robust nature and performance of GAIL in complex and sparse environments, it will be used in this dissertation as the chosen IL method.

2.3 Reinforcement Learning and Imitation Learning

2.3.1 Bootstrapping

A common way to combine the use of RL and IL is by using IL such as behavioural cloning to initialise the training after which the training is further improved by using RL methods(Gavenski et al., 2024). This is especially effective in complex tasks which use sparse rewards to motivate the agent. By teaching the agent to replicate expert trajectories using IL initially where it isn't concerned by rewards, it speeds up the time it would usually take for an RL agent to explore enough states to receive initial rewards. Since IL methods aren't explicitly suited for optimising for maximum rewards from the environment, after this initial bootstrapping RL algorithms take over the training process. This approach has been proven to work in the work of Daftry et al. 2017 where the researchers used this exact approach to speed up the initial learning rate of MAV drones.

2.3.2 Hybrid Update models

Currently there are many new solutions proposed as to how we should merge RL and IL to optimise the learning process. One solution is to run both methods simultaneously and apply different strengths to the updates applied by each method (Liu et al., 2023). When RL update strength is higher this method allows the RL algorithm to try to optimise the environment rewards while the IL algorithm nudges the agent towards replicating the expert behaviours. There are also various other propositions which are variants of this idea where the policy update is a combination of RL and IL such as TRGAIL (Kinose & Taniguchi, 2019) and Truncated Horizon Policy Search (THOR) (Sun et al., 2018) with varied success across different tasks but usually improving results compared to standard RL methods. This leveraging of information from IL to train RL agents is what will be a metric to test how task complexity impacts this variance in results in this dissertation.

2.5 Conclusion

To my knowledge, while there is a plethora of research done on the efficacy of solitary RL and IL, there aren't many sources that benchmark the performance of the combination of both methods. This section has identified suitable state-of-the-art RL (PPO) and IL (GAIL) algorithms which are capable of performing in complicated environments which will be used to train agents.

3. Implementation

3.1 Chosen Development Environment

I have decided to choose Unity as the engine for developing the agent environments. I have chosen Unity as I have previous experience with the engine, as well as the machine learning ML-Agents package offers support for some of the most used state of the art RL and IL algorithms (Juliani et al., 2018).

ML-Agents supports three Reinforcement Learning algorithms:

- Proximal Policy Optimisation
- Soft-Actor Critic
- Multi Agent Posthumous Credit Assignment (MA-POCA)

As for Imitation Learning, it supports the use of:

- Behavioural Cloning
- Generative Adversarial Imitation Learning

This selection of algorithms is perfect for this since I will be focusing on comparing PPO and GAIL throughout my project. Being comfortable with the main Unity engine will also allow me to focus less on learning how the main engine works and just focus on the machine learning aspect of ML-Agents, hastening my ability to produce different complex environments which are required for testing. The Unity toolkit also does not limit the control over the environment's design as it can be exported and interacted through regular Python API (Juliani et al., 2018), which allows for complete control over the training process if necessary. All parts of the Neural Network can be modified and, additionally the Unity Engine provides an interface for creating files which can be used to train agents via Imitation Learning without the need for external tools. This is done with a demonstration recorder, which records the actions of an agent over a multitude of episodes and exports it into a file which can be used by both Behavioural Cloning and GAIL. This engine will allow me to complete the goal of this dissertation without compromising on any aspect of the results and my previous experience will let me work through the tasks faster than any other engine currently available.

3.2 Agent Environments

This section will outline the four different environments which are be used to compare the performance of the RL and IL learning methods. The environments were all created and tested inside the Unity Engine.

3.2.1 Grid World

For the first environment it is a recreation of the simplest RL problems, the grid world environment. The grid world problem is an easy game where the agent must move through a grid to get to a specific point on the game board while avoiding a bad marker on the grid. The environment has 3 entities which include the agent, a target and a penalty space. Before the beginning of each episode, the positions of all entities on the board are randomised to ensure that the agent does not learn a behaviour pattern by memory and instead learns to take the observations given into consideration when performing actions.

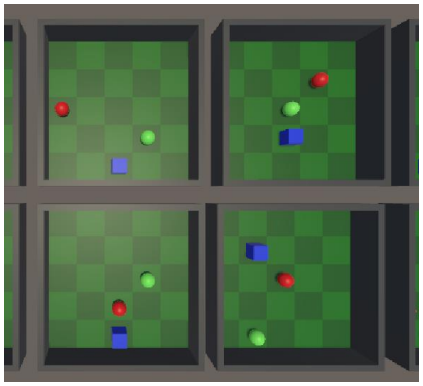

Fig. 6 Grid World Environment (Agent = blue , Target = green, Enemy = red)

3.2.1.1 Observation Space

The observations space has been configured to relay the following information to the agent.

- Int Agent_x, Agent_z: contains information about x and z positions of the agent
- Int Target_x, Target_z: contains information about x and z positions of the target
- Int Enemy_x, Enemy_z: contains information about x and z positions of the enemy

3.2.1.2 Action Space

The agent can only perform 1 action at a time choosing from the following 5 discrete actions:

- Do nothing
- Move up 1 grid space
- Move down 1 grid space
- Move left 1 grid space
- Move right 1 grid space

3.2.1.3 Reward Structure

The agent receives rewards for the following actions:

- Agent moves onto the target: +1
- Agent moves onto the enemy: -1
- Every step the agent receives: -0.1

The small negative penalty of -0.1 incentivises the agent to look for faster solutions over time as with every step the total reward received gets decreased. Without a negative time penalty agents will determine that any solution which eventually leads to the target is just as desirable, which is not correct as we want to prioritise getting to the target quickly.

3.2.1.4 Design Rationale

This problem was chosen as it makes use of only discrete data for both the observations and actions of the agent. As it is discrete and rather small in scale, the environment complexity is relatively small where even a few lines of code would be capable of solving this problem without the need of RL. The rewards are also only applied sparsely as the possible exploration space is small and the agent is able to converge to an optimal solution even without continually assessing its actions.

3.2.2 Find Exit Agent

This environment was designed to be more complicated compared to Grid World. The observation space has increased significantly due to the fact that multiple sensors are used to

convey information to the agent about its surroundings. Sensors inside Unity provide information about the entity hit by using one hot encoding a list of potential tags that can be detected. This environment consists of 5 entities which are visible to the agent, walls, an enemy, a key and the door, with all of them being detectable except for the agent. This means each sensor delivers 5 discrete observations, 1 observation for every detectable tag or if it didn't hit anything and it also obtains 1 continuous observation which is the distance to the hit entity. The agent has a total of 11 sensors which allow it to collect observations within a 180° angle of its forward direction.

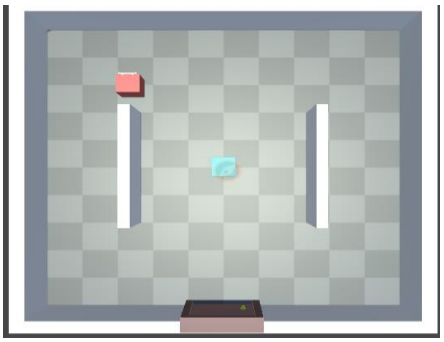

Fig. 7 Top down environment view

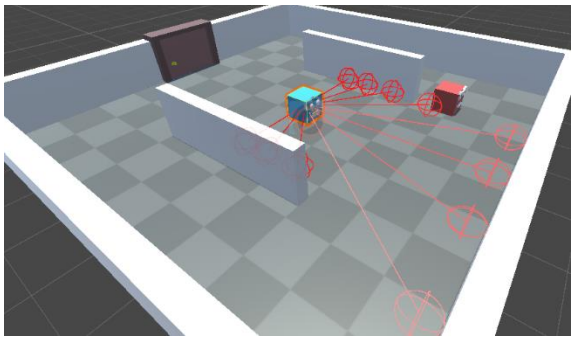

Fig. 8 Find Exit Agent with sensors

To complete this task, the agent must perform a few tasks in the correct order to succeed.

1. Initially the agent must locate the enemy which can appear anywhere inside the environment. The enemy entity constantly moves towards the agent and if the agent collides with the enemy the episode ends, and a penalty is applied. The agent must approach the enemy and use an action to attack it within range before it makes physical contact.
2. After the enemy is attacked it drops a key entity. The agent has to use a different pickup action within range in order to collect the key.
3. After the key is picked up, the agent has to exit the environment using the door.

At the start of each episode, the agent rotation, enemy position, door location and wall location are randomised.

3.2.2.1 Observation Space

The observations space has been configured to relay the following information to the agent.

- 11x Boolean[5]: containing sensor information of what objects were hit and not hit.
- 11x Float: containing sensor information of the distance to the hit object.
- Boolean KeyObtained: informs the agent if it is currently holding the key.

These observations are limited but allow the agent to fully complete its task. The sensors allow it to navigate throughout the environment. Meanwhile KeyObtained enables it to discern what step it should take to proceed.

3.2.2.2 Action Space

The agent can only take 1 action at a time and has 7 possible discrete actions. The agent can perform the following actions.

- Do nothing
- Move forward: Done by applying forward velocity
- Move backward: Done by applying backward velocity
- Rotate left: Rotates left by 1°
- Rotate right: Rotates right by 1°
- Attack: Attacks a small area ahead of the agent
- Pick up: Attempts to pick up key in a small area ahead of the agent

3.2.2.3 Reward Structure

The agent receives rewards for the following behaviours.

- Agent collides with enemy: -0.4
- Agent attacks enemy: +0.15
- Agent picks up key: 0.15
- Agent enters the door while holding the key: +1
- Every environment step: -1/3000

Additionally, the environment tracks the current target of the agent and tries to further motivate it to find the solution faster. This is done by conditionally rewarding it depending on its current target.

If enemy is present:

- Once enemy becomes visible: +0.1
- Once enemy becomes obscured: -0.1

This conditional reward incentivises the agent to explore the environment more actively in instances where the enemy is hidden behind any walls by rewarding it whenever they have a clear sight of the target. After the enemy is attacked the next target for the agent is to retrieve the key.

If key is present:

- Once key becomes visible: +0.1
- Once key becomes obscured: -0.1

Once the agent is holding the key, we prompt it to the direction of the exit by applying a small reward depending on its current distance to the door, its distance during the previous step (delta distance) and divide it by the starting distance to the door from the moment the key is picked up. From this the reward is equal to:

If key is obtained:

- (last distance - new distance) / starting distance

3.2.2.4 Design Rationale

This environment is designed to be more complicated than the grid world environment. The agent receives more observations which include continuous observations from the sensors. The action space of the agent is also larger; however, all actions are still discrete. The task is also much more robust, requiring the agent to complete different several consecutive tasks while tracking if it has picked up the key. While this task is more complex the observations are still limited to the x and z-axis, which is why this is a suitable second stage for measuring task complexity.

3.2.3 3D Space Shooter

This environment on the base level is similar to the exit agent environment with the difference being that for the Space shooter environment, the goal was to finally utilise the 3rd dimension when it comes to its observations and actions. Another step-up in complexity comes from allowing the agent to use continuous values to control its movement. The agent is placed in a cube where it has a constant forward velocity, pushing it in the range of 0.02 – 0.3 units depending on the action value which is now continuous. The aim was for the agent to imitate a spaceship with constant forward movement. The agent is not allowed to hit any walls of the environment except for the door - only after all the enemies are defeated. If it hits the door before enemies are attacked or hits the walls at any point, it ends the episode. Alongside adding the 3rd dimension to the observations, I have also allowed the agent to tilt which allows it to move anywhere inside the cube. The agent now is also capable of performing multiple compound actions per step unlike the previous tasks. In this environment the sensors can only detect 4 entities as I have removed the key from the previous environment. This leaves the one-hot encoded space to be: nothing, wall, enemy, door.

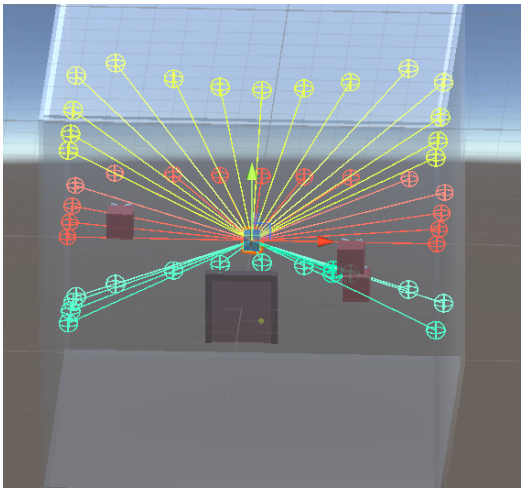

Fig. 9 Space shooter Agent with sensors visible

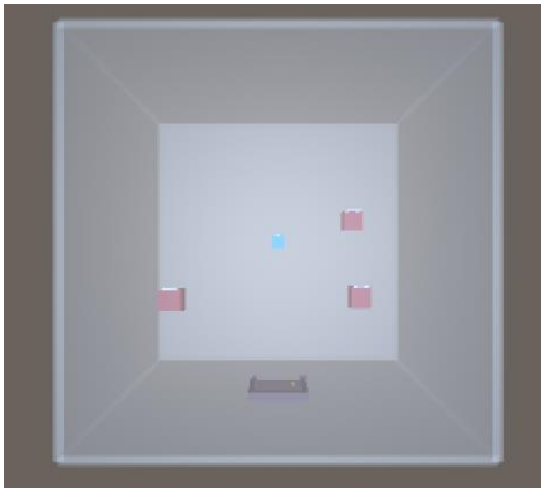

Fig. 10 Top down view

To complete this task, the agent must perform the following tasks.

1. Avoid collision with walls, as collision immediately ends the episode and applies a penalty.
2. Attack all agents present in the environment.
3. Exit through the door after all the enemies are cleared.

At the start of each episode the agent rotation, agent tilt, enemy position and door location are randomised.

3.2.3.1 Observation Space

The observations space has been configured to relay the following information to the agent.

- 45x Boolean[4]: containing sensor information
- 45x Float: containing sensor information of the distance to the hit object
- Int EnemyCount: informs the agent of the number of enemies left in the stage
- Float velocityX, velocityY, velocityZ: informs the agent of its current velocity in all directions.

The number of sensors have been expanded to allow the agent to also see below and above of its faced direction.

3.2.3.2 Action Space

The agent can now take 4 actions at once and has 2 discrete and 3 continuous actions. Continuous actions inside the Unity environment are always within the limit of [-1,1] . Using this I normalised these continuous values to represent the velocity, rotation and tilt of the agent for each action. The agent can perform the following actions.

Discrete - (and only 1 of these occur during each step)

- Do nothing
- Attack: Attacks using a ray-cast ahead of the agent within range

Continuous - (and all of the following actions happen each step)

- Adjust Speed: sets the current speed of the agent to any value between [0.02, 0.30] units
- Rotate: Rotate the agent around the y axis by any value between [-1, 1] °

- Tilt: Tilt the agent around the x axis by any value between [-1, 1] ° with a maximum tilt of 80° and -80°

3.2.3.3 Reward Structure

The agent receives rewards for the following behaviours.

- Agent collides with wall: -0.5
- Agent attacks an enemy: 0.2
- Agent exits using the door after all enemies are cleared: 1
- Once agent is facing an enemy: 0.05
- Once agent loses sight of an enemy: -0.05
- Every environment step: -1/5000

3.2.3.4 Design Rationale

The 3D space shooter environment was designed to be more complex than the previous environment by utilising the 3rd dimension alongside introducing continuous actions to the agent. Continuous actions substantially increase the complexity of the environment. This is because instead of an agent learning to perform an action in a binary fashion (agent performs the action or doesn't), it instead tries to hone in on optimal values from an uncountable number of decimal values between -1 and 1 depending on its state. Furthermore, the agent is now capable of performing multiple actions simultaneously which makes the task even more complex.

3.2.4 Crawler

The Crawler Environment is not developed by me and has been completed by the team at Unity as a showcase of what the ML-Agents Toolkit is capable of. None of the scripts, environment or neural network configurations were done by me but are available online and open source. The agent consists of 4 arms and forearms which work together to push the agent towards the goal.

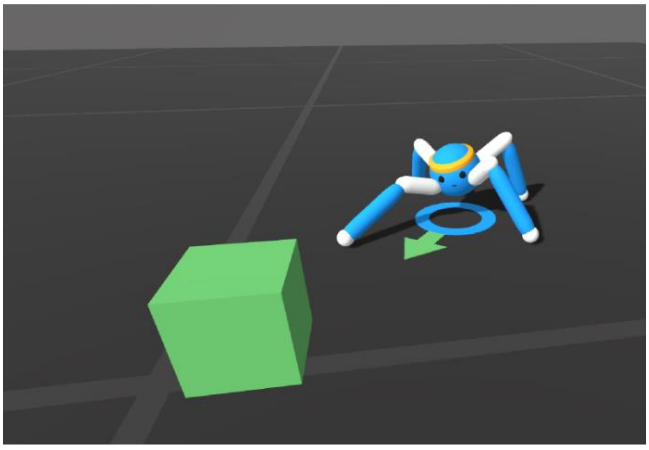

Fig. 11 Crawler Environment

The goal of this agent is to move towards a target without falling while maintaining a desired speed and keeping its head pointed towards the target. The agent is physics-based so in order to move the agent must learn to coordinate 20 different joint rotations in its body.

3.2.3.1 Observation Space

The observations space has been configured to relay the following information to the agent.

- 172x Float: the variables are responsible for storing the position, rotation, velocity and angular velocities of each limb with additional variables for the acceleration and angular acceleration of the body.

3.2.3.2 Action Space

The agent in this environment performs 20 simultaneous continuous actions. All actions are the same between all of the agent's limbs The agent can perform the following actions.

- Set lower arm target x_rotation
- Set upper arm target x and y_rotation
- Set the strength of the arm rotation action

3.2.3.3 Reward Structure

The agent receives rewards for the following behaviours.

- Agent collides with target: 1

- Every environment step: ((Agent velocity normalised to [0,1]) * ((Head Alignment with target) normalised [0,1])

3.2.3.4 Design Rationale

This environment was chosen as it is by far the most complex environment with both the biggest observation and action space. The initial idea for the last environment was to create an agent capable of walking, however with IL it would require motion capture data to train the agent. Increasing the action space further than the 3D space shooter would make it difficult to reliably collect adequate demonstration data from a human. To circumvent this, Unity ML-agents developers used a system of creating animations and translating them into actions to create a demonstration, which allows IL agents to learn in this environment.

3.3 Neural Network Configurations

The neural network configurations used to train RL, IL and combined RL IL are all similar, with each remaining constant within the same environments. This enables a better contrast point between the performance of each method, as the neural network configuration remains a constant. The majority of variables are also kept constant with the exception of: the number of hidden units per layer, the number of layers, the batch size and buffer size. I have chosen these parameters to be variable as some environments require a bigger neural network to be able to develop more intricate relationships between the data which is required to complete the task. For example, the grid world would not require a large network as the relationships between the data points are simple and not as robust as the later 3D space shooter and Crawler environment. Other variables such as learning rate have been tested before settling on a constant rate throughout the experiment.

```yaml
FindExitAgent:
  trainer_type: ppo
  hyperparameters:
    batch_size: 2024
    buffer_size: 40480
    learning_rate: 0.0003
    beta: 0.005
    epsilon: 0.2
    lambd: 0.95
    num_epoch: 3
    learning_rate_schedule: linear
  network_settings:
    normalize: true
    hidden_units: 512
    num_layers: 3
    vis_encode_type: simple
  reward_signals:
    extrinsic:
      gamma: 0.99
      strength: 1.0
  keep_checkpoints: 5
  max_steps: 10000000
  time_horizon: 1000
  summary_freq: 10000
  threaded: true
```

Fig. 12 RL agent configuration

The configuration shown in figure 12 shows the neural network configuration used to train the Find Exit Agent. All of the remaining configuration files for each environment are located in the appendix of this document.

3.3 Demonstration files

The demonstration files needed for IL training have all been collected using a demonstration recorder where human input was used to provide data for the agents to learn behaviours from. Data was collected during the recording of the demonstrations to ensure no outliers were present and that no subpar episodes are gathered which could impact the learning of the IL agent.

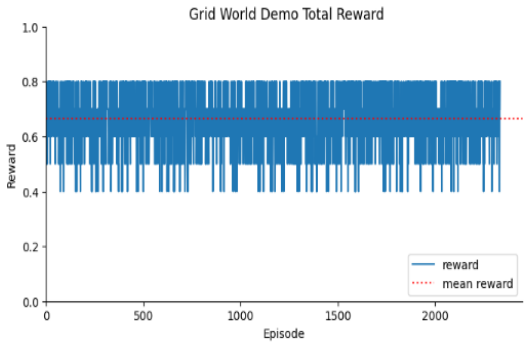

Fig. 13 Grid World Demonstration Data

Fig. 14 Find Exit Agent Demonstration Data

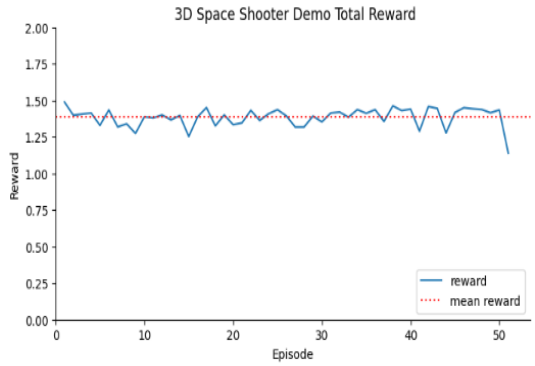

Fig. 15 3D Space Shooter Demonstration Data

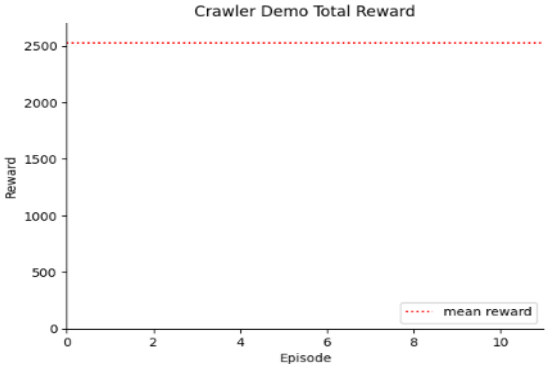

Fig. 16 Crawler Demonstration Data

Environment	Total Episodes	Total Steps	Mean Reward	Std Dev of Reward
Grid World	2334	7831	0.664	0.107
Find Exit Agent	70	11314	1.445	0.152
3D Space Shooter	52	26942	1.385	0.064
Crawler	11	10011	2527.892	N/A

Unfortunately, due to the fact that the demonstration for the Crawler was created by the Unity team it is not possible to get insight into the exact rewards obtained by the agent during each recorded episode. However, the demonstration provided for the Crawler environment has been previously tested and achieved good results when trained using the GAIL algorithm. Some of the variance in rewards obtained during each episode is a result of the randomisation of entities in the environments, which affects how quickly the task could be completed. This is particularly visible inside the grid world environment, as placing the agent 1 grid further from the target results in an immediate decrease of -0.1 to the maximum reward possible.

4. Results and Discussion

This section provides an analysis of the agent's performance in each environment. The results are gathered from running RL, IL and combined RL+IL on all of the environments. They will be compared on the three primary metrics outlined in the literature review: convergence speed, maximum total reward, and real time taken to converge. Figures 19-24 show an additional plotted line of averaged points to allow better visibility of the data.

4.1 Results

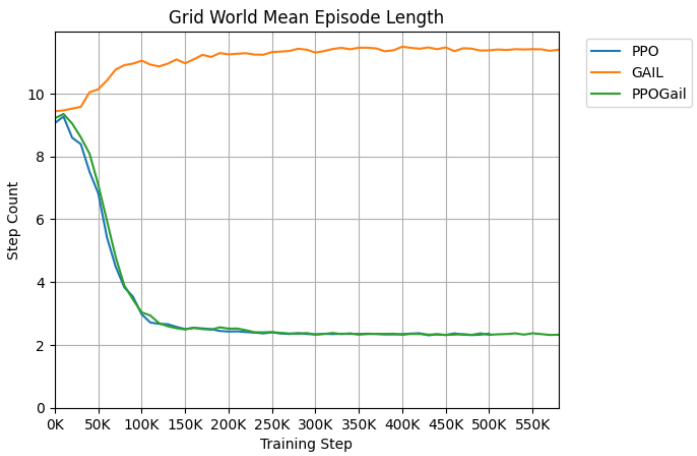

Fig. 17 Grid World Mean Episode Length

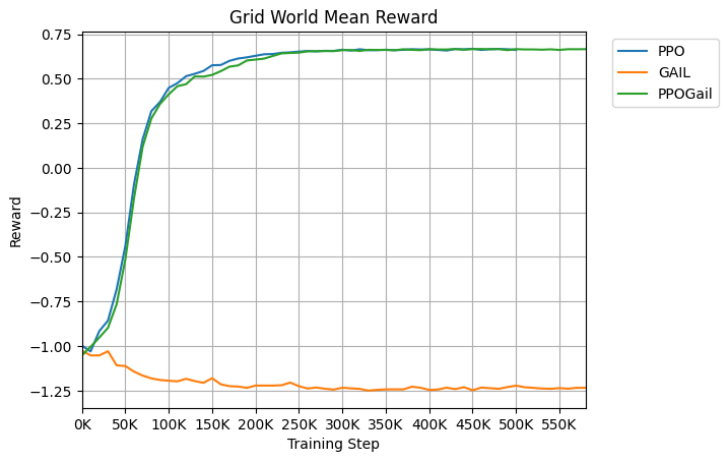

Fig. 18 Grid World Mean Reward

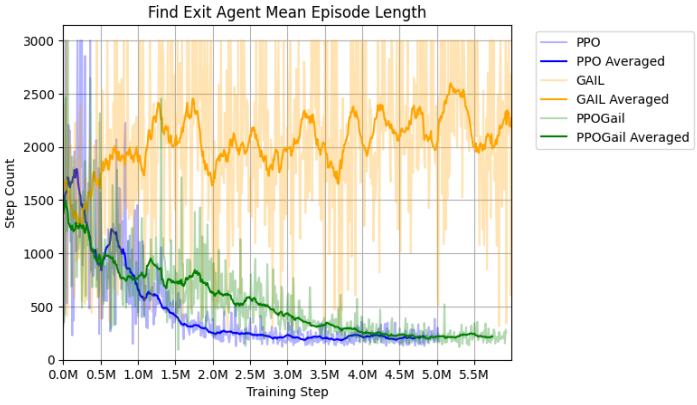

Fig. 19 Find Exit Agent Mean Episode Length

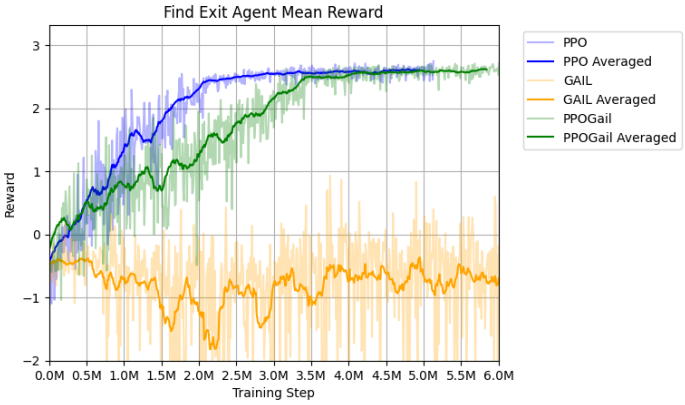

Fig. 20 Find Exit Agent Mean Reward

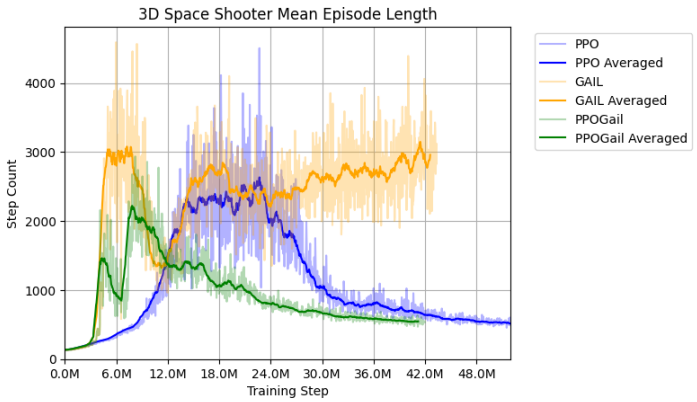

Fig. 21 3D Space Shooter Mean Episode Length

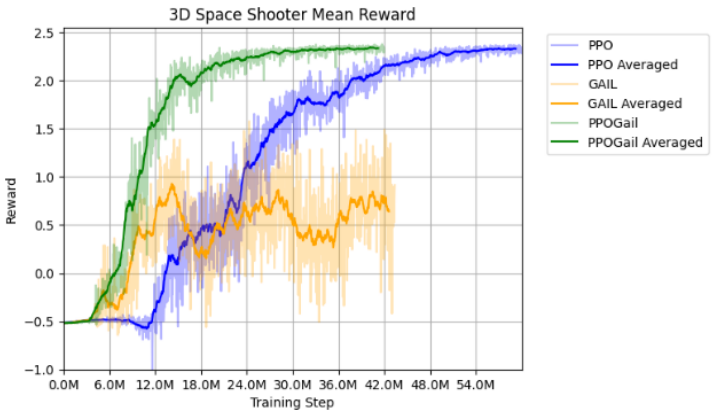

Fig. 22 3D Space Shooter Mean Reward

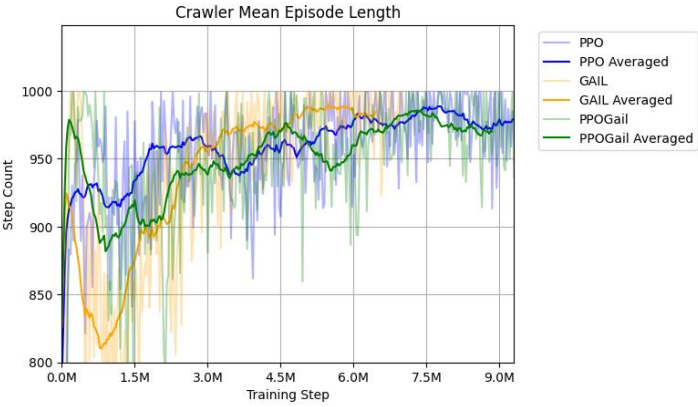

Fig. 23 Crawler Mean Episode Length

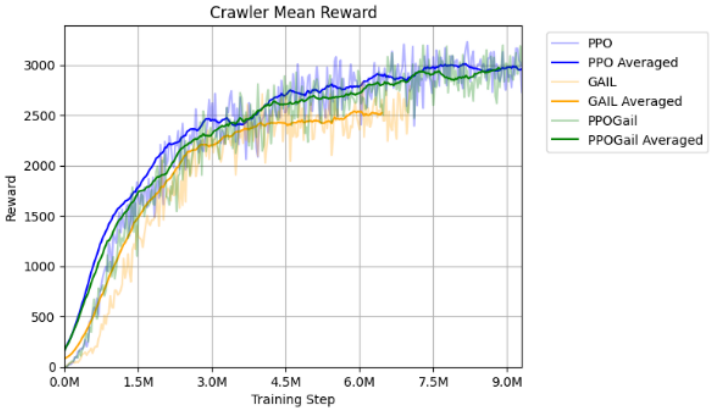

Fig. 24 Crawler Mean Reward

Table 1. Environment Maximum Total Reward

Environment	PPO	GAIL	PPO+GAIL
Grid World	0.6677	-1.02534	0.6666
Find Exit Agent	2.7462	2.1444	2.7192
3D Space Shooter	2.3686	1.5758	2.3773
Crawler	3228.82	2828.49	3194.41

Table 2. Maximum reward reached at step

Environment	PPO	GAIL	PPO+GAIL
Grid World	450K	0	460K
Find Exit Agent	5.13M	8.06M	6.01M
3D Space Shooter	52.5M	24.3M	41.7M
Crawler	8.19M	6.06M	7.29M

Table 3. Steps to converge

Environment	PPO	GAIL	PPO+GAIL
Grid World	300K	N/A	300K
Find Exit Agent	3.2M	N/A	4.1M
3D Space Shooter	54.2M	N/A	35.4M
Crawler	8M	6.2M	8.7M

Table 4. Real Time taken to converge:

Environment	PPO	GAIL	PPO+GAIL
Grid World	6m 25s	N/A	6m 27s
Find Exit Agent	54m 23s	N/A	1h 9m 36s
3D Space Shooter	25h 16m 12s	N/A	16h 06m 18s
Crawler	4h 11m 47s	3h 51m 20s	5h 01m 4s

All tests have been done on a machine with the following specifications:

Processor: Ryzen 7 3700X

GPU: Nvidia GeForce 3070ti

Operating System: Windows 10

Ram: 32GB DDR4 3200MhZ

Unity Version: Unity Engine 2023.2.13.1f

ML-Agents Version: 1.0.0

4.2 Results Discussion

4.2.1 Maximum total Reward

The "maximum total reward" results showcase that both RL and RL+IL both reliably manage to obtain similar scores, albeit reaching them at different rates depending on the complexity of the environment. IL on the other hand consistently struggles to maximise its reward especially in the less robust environments. In the later more complex environments, it learns sufficiently to complete the tasks, but without receiving rewards from the environment it fails to improve in completing the task faster. This leads IL to stagnate after a certain point where it shows no trend of further improvement or becoming more stable. This can be best seen from the results presented in Table 1 where it consistently scores the lowest of the 3 learning methods. Through these observations we can note that both RL and RL+IL are both viable when teaching an agent to maximise rewards regardless of the task complexity. We can also note that IL is very unstable given the demonstrations. IL failed to outmatch even the performance of fully stochastic runs in the Grid World and Find Exit Agent environments when given thousands of demonstration episode samples used in Grid World. Once the environments utilised continuous actions like the other two environments, its performance considerably improved.

4.2.2 Convergence Speed and Time

In terms of convergence speed, RL has resulted in the most stable results when working with a discrete action space in the simpler environments where it outperformed the other learning methods. This can be best seen in figure 20, where the RL agents learning rate is consistently higher than RL+IL, resulting in convergence in 78% less steps and 78.2% less of the time it took the RL+IL agent to converge, saving 15 minutes. However, once the environment grew more complex and utilised continuous actions like presented in the 3D Space Shooter agent, the RL struggles to start developing new behaviours at the beginning of the learning process. In figure 22 we can notice that the RL agent struggles to improve its score at all until a breakthrough after almost 12 million steps. This is considerably slower compared to both IL and RL+IL where they both started improving in under 4 million steps. Even after the RL agent is able to start improving, its rate of learning is a lot lower than the one of combined RL+IL. In this environment the improvement made from utilising RL+IL is quite significant

as the agent manages to converge in only 65.3% of the steps and 63.7% of the time it takes the RL agent. In this instance the real time saved comes out to over 9 hours of training time.

So far, the results have shown that as the complexity of the environment increases, utilising RL+IL becomes more beneficial as it greatly improves the initial learning rate. Despite that, the Crawler environment breaks this premise as even with 20 simultaneous continuous actions that must be optimised, the RL method manages to outperform RL+IL throughout the entire learning process as shown in figure 24. The difference between them is not very significant, yet RL manages to outperform other methods even at the very beginning of the learning process where IL would usually have the advantage. The runs of the Find Exit Agent and 3D Space Shooter RL+IL have shown an initial bump of learning in the earlier stages of learning which could be potentially used to bootstrap a pure RL agent to speed up the initial exploration process of the agent(Hu et al., 2023). The Crawler agent does not show this initial bump at all, while being the most robust agent. It manages to learn more through only RL with a slight advantage over RL+IL. This makes the possibility that RL+IL should be utilised depending on task complexity less likely, as these results directly contradict this notion. This means that there must be a different variable which greatly affects the learning rate of the RL agent. One noticeable difference between the Crawler and the 3 other environments is the reward structure. All other environments have a rather sparse reward structure, only giving out positive rewards sparingly in contrast to the Crawler which calculates the rewards for the agent during every step of the process. This is likely to result in a much higher learning rate for RL as the agent receives feedback on its actions during every step which lead it to outperform RL+IL.

4.2.3 Summary of results

From these findings we can deduce that it would not be correct to justify choosing RL+IL over IL and vice versa only based on task complexity, with the exception for very simple tasks, where RL performs the best. Instead, these results could possibly point to the belief that RL+IL is preferable over RL in complex environments with sparse rewards. New experiments would need to be designed comparing the effects of differing reward sparsities on the learning rates of each method inside identical environments to get further insight into when each method is most beneficial.

5. Conclusions and Future Work

5.1 Future Improvements

The first immediate improvement that could be done is to greatly expand the experiment set of the dissertation. By adding more complex experiments we can gain further points of comparison which could showcase more evidence of a link between task complexity and performance of the different learning methods. This would, however, require more time and better equipment as further expanding of the action space would require tools to collect action data for demonstrations such as motion-tracking and motion-to-action translation tools. Furthermore, since PPO begins to struggle with very complex scenarios, SAC could prove a viable RL alternative for further experimentation.

One big challenge I faced when conducting experiments were the results obtained from running IL where it struggled to converge on simpler environments. While the demonstrations gathered were checked and analysed to not contain outliers, agents still failed to imitate and match the results of the demonstrations correctly. To rectify this, more thorough tests of IL and an expanded demonstration set could be gathered to ensure each agent is able to reach satisfactory results on every environment.

Lastly, since the results show a weak link between task complexity and performance of the different learning environments, this project could also be expanded to verify an existence of a larger link. The biggest possible pitfall of the project was the difference between the reward structure of the environments. In future works if this variable was to be standardized so that every environment gets rewarded similarly, there is a possibility of finding correlations and gaining a better understanding of when to use which learning algorithm. Additionally, instead of only standardising reward structures, it would be possible to expand the environment set to develop multiple reward structures within the same environment to observe a possible link. However, developing multiple viable reward structures might prove challenging, especially for simpler tasks as they can't be substantially broken down or expanded upon.

5.2 Conclusion

Overall, the results gathered from this project show that there isn't a direct link between the task complexity and the performance of each learning method. There isn't enough evidence from the experiment results to justify choosing one learning method over another based on task

complexity. While there is a slight correlation with RL performing the best in simple tasks, once more complex task results are taken into consideration, any correlation between task complexities and the performance of different algorithms dissipates. There is a different major factor which affects the learning rates of RL, IL and RL+IL other than how complex its learning environment is, which could be explored.

All of the

6. References

Berry, D. A., & Fristedt, B. (1985). *Bandit problems: Sequential allocation of experiments* [PDF]. Springer. https://doi.org/10.1007/978-94-015-3711-7

Chen, J. X. (July-Aug 2016). The Evolution of Computing: AlphaGo. *Computing in Science & Engineering*, *18*(4), 4–7.

Daftry, S., Bagnell, J. A., & Hebert, M. (2017). Learning Transferable Policies for Monocular Reactive MAV Control. In D. Kulić, Y. Nakamura, O. Khatib, & G. Venture (Eds.), *2016 International Symposium on Experimental Robotics* (pp. 3–11). Springer International Publishing.

Gavenski, N., Meneguzzi, F., Luck, M., & Rodrigues, O. (2024). A Survey of Imitation Learning Methods, Environments and Metrics. In *arXiv [cs.LG]*. arXiv. http://arxiv.org/abs/2404.19456

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *arXiv [cs.LG]*. arXiv. http://arxiv.org/abs/1801.01290

Ho, J., & Ermon, S. (2016). Generative adversarial imitation learning. *Advances in Neural Information Processing Systems*, *29*.

Hu, H., Mirchandani, S., & Sadigh, D. (2023). Imitation Bootstrapped Reinforcement Learning. In *arXiv [cs.LG]*. arXiv. http://arxiv.org/abs/2311.02198

Hussein, A., Gaber, M. M., Elyan, E., & Jayne, C. (2017). Imitation Learning: A Survey of Learning Methods. *ACM Comput. Surv.*, *50*(2), 1–35.

Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., & Lange, D. (2018). Unity: A general platform for intelligent agents. In *arXiv [cs.LG]*. arXiv. http://arxiv.org/abs/1809.02627

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kim, J. T., & Ha, S. (2021). Observation Space Matters: Benchmark and Optimization Algorithm. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 1527–1534.

Kinose, A., & Taniguchi, T. (2019). Integration of Imitation Learning using GAIL and Reinforcement Learning using Task-achievement Rewards via Probabilistic Graphical Model. In *arXiv [cs.LG]*. arXiv. http://arxiv.org/abs/1907.02140

Li, Y. (2017). Deep Reinforcement Learning: An Overview. In *arXiv [cs.LG]*. arXiv. http://arxiv.org/abs/1701.07274

Liu, Z., Liu, L., Wu, B., Li, L., Wang, X., Yuan, B., & Zhao, P. (2023). Dynamics Adapted Imitation Learning. In *Transactions on Machine Learning Research*. https://openreview.net/pdf?id=w36pqfaJ4t

Loeber, P. (2022, February 2). *Reinforcement Learning with (Deep) Q-Learning explained*. News, Tutorials, AI Research. https://www.assemblyai.com/blog/reinforcement-learning-with-deep-q-learning-explained/

Luu, Q. T. (2023, April 10). *Q-Learning vs. Deep Q-Learning vs. Deep Q-Network*. Baeldung on Computer Science. https://www.baeldung.com/cs/q-learning-vs-deep-q-learning-vs-deep-q-network

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In *arXiv [cs.LG]*. arXiv. http://arxiv.org/abs/1602.01783

Mock, J. W. (2023). *A Comparison of PPO, TD3, and SAC Reinforcement Algorithms for Quadruped Walking Gait Generation and Transfer Learning to a Physical Robot*. University of Wyoming.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., & Abbeel, P. (2018). *Overcoming Exploration in Reinforcement Learning with Demonstrations*. In *arXiv [cs.LG]*. arXiv. http://arxiv.org/abs/1709.10089

Pomerleau, D. A. (1989). *ALVINN: An Autonomous Land Vehicle in a Neural Network*. Carnegie Mellon University, Computer Science Department.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. In *arXiv [cs.LG]*. arXiv. http://arxiv.org/abs/1707.06347

Seyde, T., Werner, P., Schwarting, W., Gilitschenski, I., Riedmiller, M., Rus, D., & Wulfmeier, M. (2023). Solving Continuous Control via Q-learning. In *arXiv [cs.LG]*. arXiv. http://arxiv.org/abs/2210.12566

Simonini, T. (n.d.). *Proximal Policy Optimization (PPO)*. Retrieved August 8, 2024, from https://huggingface.co/blog/deep-rl-ppo

Soft Actor-Critic — Spinning Up documentation. (n.d.). Retrieved August 8, 2024, from https://spinningup.openai.com/en/latest/algorithms/sac.html

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning, second edition: An Introduction*. MIT Press.

Sun, W., Andrew Bagnell, J., & Boots, B. (2018). Truncated Horizon Policy Search: Combining Reinforcement Learning & Imitation Learning. In *arXiv [cs.LG]*. arXiv. http://arxiv.org/abs/1805.11240

Torabi, F., Warnell, G., & Stone, P. (2018). Behavioral Cloning from Observation. In *arXiv [cs.AI]*. arXiv. http://arxiv.org/abs/1805.01954

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3), 229–256.

Hussein, A., Gaber, M. M., Elyan, E., & Jayne, C. (2017). Imitation Learning: A Survey of Learning Methods. *ACM Comput. Surv.*, *50*(2), 1–35.

Xiong, J., Wang, Q., Yang, Z., Sun, P., Han, L., Zheng, Y., Fu, H., Zhang, T., Liu, J., & Liu, H. (2018). Parametrized Deep Q-Networks Learning: Reinforcement Learning with Discrete-Continuous Hybrid Action Space. In *arXiv [cs.LG]*. arXiv. http://arxiv.org/abs/1810.06394

Zheng, B., Verma, S., Zhou, J., Tsang, I. W., & Chen, F. (2024). Imitation Learning: Progress, Taxonomies and Challenges. *IEEE Transactions on Neural Networks and Learning Systems*, *35*(5), 6322–6337.

7. Acknowledgements

First, I would like to thank my supervisor, Eaton Malachy for helping me decide on the scope of this project and for providing support which guided me through the initial steps for the project.

I would also like to extend my thanks to my friends and family that gave me feedback on some aspects of the project as well as supported me on the way to completing this dissertation.

8. Appendices

All of the code used and the results of this dissertations are available on my personal GitHub repository:

https://github.com/MrBaconMajster/18222757_supporting_material

```yaml
hyperparameters:
  batch_size: 1024
  buffer_size: 2048
  learning_rate: 0.0003
  beta: 0.005
  epsilon: 0.2
  lambd: 0.95
  num_epoch: 3
  learning_rate_schedule: linear
network_settings:
  normalize: true
  hidden_units: 64
  num_layers: 2
  vis_encode_type: simple
reward_signals:
  extrinsic:
    gamma: 0.99
    strength: 1.0
keep_checkpoints: 5
max_steps: 10000000
time_horizon: 5
summary_freq: 10000
threaded: true
```

Figure 1 Grid World PPO Hyperparameters

```yaml
hyperparameters:
  batch_size: 1024
  buffer_size: 2048
  learning_rate: 0.0003
  beta: 0.005
  epsilon: 0.2
  lambd: 0.95
  num_epoch: 3
  learning_rate_schedule: linear
network_settings:
  normalize: true
  hidden_units: 64
  num_layers: 2
  vis_encode_type: simple
reward_signals:
  extrinsic:
    gamma: 0.99
    strength: 1.0
  gail:
    gamma: 0.99
    strength: 1.00
    learning_rate: 0.0003
    use_actions: false
    use_vail: false
    demo_path: Demos/GridWorldDemo.demo
keep_checkpoints: 5
max_steps: 10000000
time_horizon: 5
summary_freq: 10000
threaded: true
```

Figure 2 Grid World PPO+GAIL Hyperparameters

```yaml
hyperparameters:
  batch_size: 1024
  buffer_size: 2048
  learning_rate: 0.0003
  beta: 0.005
  epsilon: 0.2
  lambd: 0.95
  num_epoch: 3
  learning_rate_schedule: linear
network_settings:
  normalize: true
  hidden_units: 64
  num_layers: 2
  vis_encode_type: simple
reward_signals:
  gail:
    gamma: 0.99
    strength: 1.00
    learning_rate: 0.0003
    use_actions: false
    use_vail: false
    demo_path: Demos/GridWorldDemo.demo
keep_checkpoints: 5
max_steps: 10000000
time_horizon: 5
summary_freq: 10000
threaded: true
```

Figure 3 Grid World GAIL Hyperparameters

```yaml
hyperparameters:
  batch_size: 2024
  buffer_size: 40480
  learning_rate: 0.0003
  beta: 0.005
  epsilon: 0.2
  lambd: 0.95
  num_epoch: 3
  learning_rate_schedule: linear
network_settings:
  normalize: true
  hidden_units: 512
  num_layers: 3
  vis_encode_type: simple
reward_signals:
  extrinsic:
    gamma: 0.99
    strength: 1.0
keep_checkpoints: 5
max_steps: 10000000
time_horizon: 1000
summary_freq: 30000
```

Figure 4 Find Exit Agent PPO Hyperparameters

```yaml
hyperparameters:
  batch_size: 2024
  buffer_size: 40480
  learning_rate: 0.0003
  beta: 0.005
  epsilon: 0.2
  lambd: 0.95
  num_epoch: 3
  learning_rate_schedule: linear
network_settings:
  normalize: true
  hidden_units: 512
  num_layers: 3
  vis_encode_type: simple
reward_signals:
  extrinsic:
    gamma: 0.99
    strength: 1.0
  gail:
    gamma: 0.99
    strength: 0.01
    network_settings:
      normalize: true
      goal_conditioning_type: none
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
    learning_rate: 0.0003
    use_actions: false
    use_vail: false
    demo_path: Demos/MediumDemo.demo
keep_checkpoints: 5
max_steps: 10000000
time_horizon: 1000
summary_freq: 30000
```

Figure 5 Find Exit Agent PPO+GAIL Hyperparameters

```
  hyperparameters:
    batch_size: 2024
    buffer_size: 40480
    learning_rate: 0.0003
    beta: 0.005
    epsilon: 0.2
    lambd: 0.95
    num_epoch: 3
    learning_rate_schedule: linear
  network_settings:
    normalize: true
    hidden_units: 512
    num_layers: 3
    vis_encode_type: simple
  reward_signals:
    gail:
      gamma: 0.99
      strength: 1.00
      demo_path: Demos/MediumDemo.demo
  keep_checkpoints: 5
  max_steps: 10000000
  time_horizon: 1000
  summary_freq: 10000
  threaded: true
```

Figure 6 Find Exit Agent GAIL Hyperparameters

```
hyperparameters:
  batch_size: 2024
  buffer_size: 409600
  learning_rate: 0.0003
  beta: 0.005
  epsilon: 0.2
  lambd: 0.95
  learning_rate_schedule: linear
network_settings:
  normalize: true
  hidden_units: 512
  num_layers: 3
  vis_encode_type: simple
reward_signals:
  extrinsic:
    gamma: 0.99
    strength: 1.0
keep_checkpoints: 10
max_steps: 42000000
time_horizon: 2048
summary_freq: 40000
threaded: true
```

Figure 7 3D Space Shooter PPO Hyperparameters

```yaml
hyperparameters:
  batch_size: 2024
  buffer_size: 409600
  learning_rate: 0.0003
  beta: 0.005
  epsilon: 0.2
  lambd: 0.95
  learning_rate_schedule: linear
network_settings:
  normalize: true
  hidden_units: 512
  num_layers: 3
  vis_encode_type: simple
reward_signals:
  extrinsic:
    gamma: 0.99
    strength: 1.0

  gail:
    gamma: 0.99
    strength: 0.01
    network_settings:
      normalize: true
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
    learning_rate: 0.0003
    use_actions: false
    use_vail: false
    demo_path: Demos/ComplexDemo.demo
keep_checkpoints: 10
max_steps: 60000000
time_horizon: 2048
summary_freq: 40000
threaded: true
```

Figure 8 3D Space Shooter PPO+GAIL Hyperparameters

```yaml
hyperparameters:
  batch_size: 2024
  buffer_size: 409600
  learning_rate: 0.0003
  beta: 0.005
  epsilon: 0.2
  lambd: 0.95
  learning_rate_schedule: linear
network_settings:
  normalize: true
  hidden_units: 512
  num_layers: 3
  vis_encode_type: simple
reward_signals:
  gail:
    gamma: 0.99
    strength: 1.0
    network_settings:
      normalize: true
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
    learning_rate: 0.0003
    use_actions: false
    use_vail: false
    demo_path: Demos/ComplexDemo.demo
keep_checkpoints: 10
max_steps: 60000000
time_horizon: 2048
summary_freq: 40000
threaded: true
```

Figure 9 3D Space Shooter GAIL Hyperparameters

```yaml
hyperparameters:
  batch_size: 2048
  buffer_size: 20480
  learning_rate: 0.0003
  beta: 0.005
  epsilon: 0.2
  lambd: 0.95
  num_epoch: 3
  learning_rate_schedule: linear
network_settings:
  normalize: true
  hidden_units: 512
  num_layers: 3
  vis_encode_type: simple
reward_signals:
  extrinsic:
    gamma: 0.995
    strength: 1.0
keep_checkpoints: 5
max_steps: 10000000
time_horizon: 1000
summary_freq: 30000
```

Figure 10 Crawler PPO Hyperparameters

```yaml
hyperparameters:
  batch_size: 2048
  buffer_size: 20480
  learning_rate: 0.0003
  beta: 0.005
  epsilon: 0.2
  lambd: 0.95
  num_epoch: 3
  learning_rate_schedule: linear
network_settings:
  normalize: true
  hidden_units: 512
  num_layers: 3
  vis_encode_type: simple
reward_signals:
  extrinsic:
    gamma: 0.995
    strength: 1.0
  gail:
    gamma: 0.99
    strength: 0.01
    network_settings:
      normalize: true
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
    learning_rate: 0.0003
    use_actions: false
    use_vail: false
    demo_path: Assets/ML-Agents/Examples/Crawler/Demos/ExpertCrawler.demo
keep_checkpoints: 5
max_steps: 10000000
time_horizon: 1000
summary_freq: 30000
```

Figure 11 Crawler PPO+GAIL Hyperparameters

```yaml
hyperparameters:
  batch_size: 2024
  buffer_size: 20240
  learning_rate: 0.0003
  beta: 0.005
  epsilon: 0.2
  lambd: 0.95
  num_epoch: 3
  learning_rate_schedule: linear
network_settings:
  normalize: true
  hidden_units: 512
  num_layers: 3
  vis_encode_type: simple
reward_signals:
  gail:
    gamma: 0.99
    strength: 1.0
    network_settings:
      normalize: true
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
    learning_rate: 0.0003
    use_actions: false
    use_vail: false
    demo_path: Assets/ML-Agents/Examples/Crawler/Demos/ExpertCrawler.demo
keep_checkpoints: 5
max_steps: 10000000
time_horizon: 1000
summary_freq: 30000
```

Figure 12 Crawler GAIL Hyperparameters