TRAJECTORY PREDICTION THROUGH DEEP REINFORCEMENT LEARNING IN UNITY

A Thesis Presented

By

Tribhuwan Singh

to

The Department of College of Engineering

in partial fulfillment of the requirements

for the degree of

Master of Science

in the field of

Data Analytics Engineering

Northeastern University

Boston, Massachusetts

May 2021

ABSTRACT

The thesis was carried out to develop a 3D game that could be used to play. The case of the project was a trajectory prediction shooting game in which the player had to search the goal(star) inside a polytope and shoot the ball to the goal. There were a few obstacles inside the polytope, and they spawn randomly. The player must constantly hit the goal continuously as many as it can. If the player misses the goal, then the game is over. This thesis investigates how artificial intelligence techniques can be used to predict the trajectory of the ball.

The game contains two main game scenes, one is the Graphic User Interfaces (GUI) scene and another one is the game scene. The first part describes the practicalities of the game. The player could play, control the physics of the ball, get help with the Prediction line, and start or end this game on the GUI scene.

This thesis presents the, to our knowledge, second part for introducing the vast field of artificial intelligence. Topics introduced include Reinforcement Learning techniques, agent simulation, and Imitation Learning.

We believe our approach demonstrates the artificial agent that can learn a conditioned trajectory to model human motion inside a polytope using imitation learning.

ACKNOWLEDGEMENTS

I would like to thank wholeheartedly Dr. Tony Mullen and Dr. Sagar Kamarthi for mentoring me throughout the course of this thesis and providing me with invaluable suggestions, feedback, and encouragement without which, I would not have been able to complete this thesis. I am thankful to them for their constant help and support in light of this thesis.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS

GAIL Generative Adversarial Imitation Learning

PPO Proximal Policy Optimization

CNN Convolutional Neural Network

IL Imitation Learning

RL Reinforcement Learning

ML Machine Learning

IL Importance Weighted Actor-Linear Architecture

LSTM Long Short-Term Memory

TABLE OF CONTENTS

1. Introduction

The theory of reinforcement learning is to train an agent to learn to do some actions that maximize the expected reward with an environment. The objective is to train agents to optimize their control of an environment when they are confronted with a complex task. Reinforcement learning has been successful in a variety of domains and we want to extend that to a notch by developing an agent that can learn successful policies directly from sensory observations. To get a faster and a better agent we simply demonstrated the behavior we wanted the agent to perform by providing real-world examples from the game. Therefore, we used Generative Adversarial Imitation Learning (GAIL) to work parallelly with Proximal Policy Optimization (PPO) to ensure an optimal action.

We demonstrated that our agent is capable of learning a conditioned trajectory to model human motion inside a polytope using imitation learning.

2. Unity3D Asset Development

“Unity3D is a cross-platform game engine with a built-in IDE developed by Unity Technologies. It is generally used to develop video games for computer platforms such as web and desktop, consoles, and mobile devices, and is applied by several million developers in the world. Unity is primarily used to create mobile and web games, but there are various games to be developed for PC. The game engine was programmed in C/C++ and can support codes written in C#, JavaScript, or Boo. It grew from an OS X-supported game development tool in 2005 to the multi-platform game engine that it is today [1].”

“Unity is the perfect choice for small studios, indie developers, and those of us who have always wanted to make games. Its large user base and extremely active user community allow everyone from newbies to seasoned veterans to get answers and share information quickly [1].”

“Unity provides an excellent entry point into game development, balancing features, and functionality with a price point. The free version of Unity allows people to experiment, learn, develop, and sell games before committing any of their hard-earned cash. Unity is a very affordable, feature-packed Pro version that is royalty-free, allowing people to make and sell games with the very low overhead essential to the casual games market [1].”

2.1 Architecture

Unity3D is a commercial tool and a closed commercial engine. It integrates Animation Mechanics, Character Mechanics, Player Mechanics, Environment Mechanics, and Programming Developer. All the game projects of Unity follow the same model. Every game is component-based and contains at least a scene. The scene covers game objects. The components inside a game object can be scripts or predefined structures which are handled by the engine based on the game settings. A game can contain third-party libraries or toolkits which are located in the assets folder.

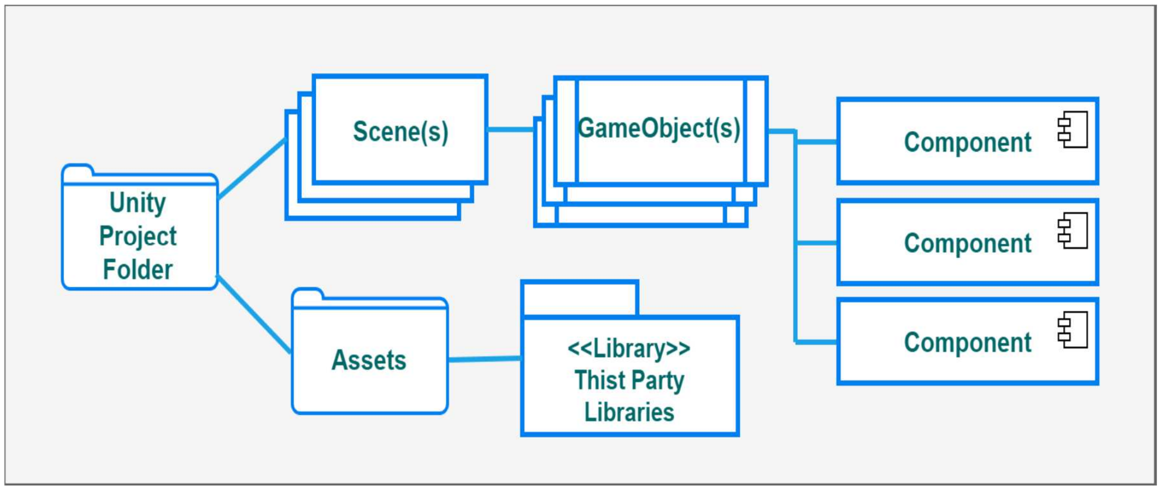


Figure 2.1: Unity Project Structure.

A folder represents a project and is recognized by the editor. Any changes made to the script keep the metadata updated in the script metafile. These metadata are constantly tracked by unity and the flow is automatic. Component-based design is supported by Game Loop pattern and Update method patterns.

2.1.1 High-Level Architecture

Unity game engine was programmed in C/C++, but the logic is managed using C# or UnityScript. The drawback of using two languages freely is that one cannot refer to another, therefore, requires one to use at a time to have a consistent workflow.

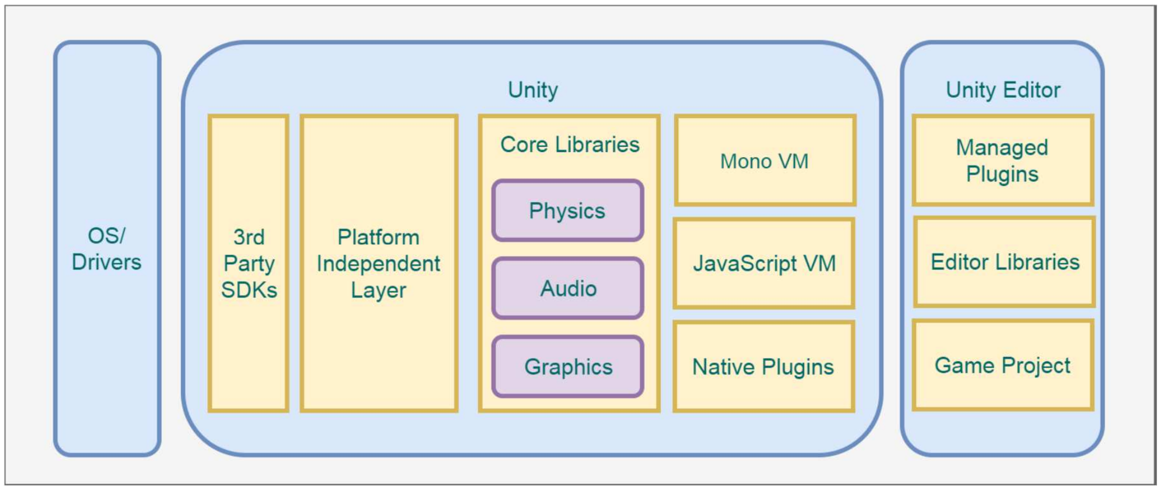


Figure 2.2: Unity High Level Architecture.

The code gets generated in the game assembly whenever a code is generated from an editor extension. This amount of level of detail regarding Unity's high-level architecture is enough for our thesis's goal.

2.1.2 Scene Management

The project in Unity organizes files by type with the game object forms a tree structure with a parent-child relationship between them. An object is the root class and all the things in the engine extend this class.

A GameObject is linked by every object in a scene. A game object contains components and has one-to-many relationships. While Component is attached to GameObject compositely, under the Component, many inheritance hierarchy levels exist. Transform is a special component that is mandatory for game objects.

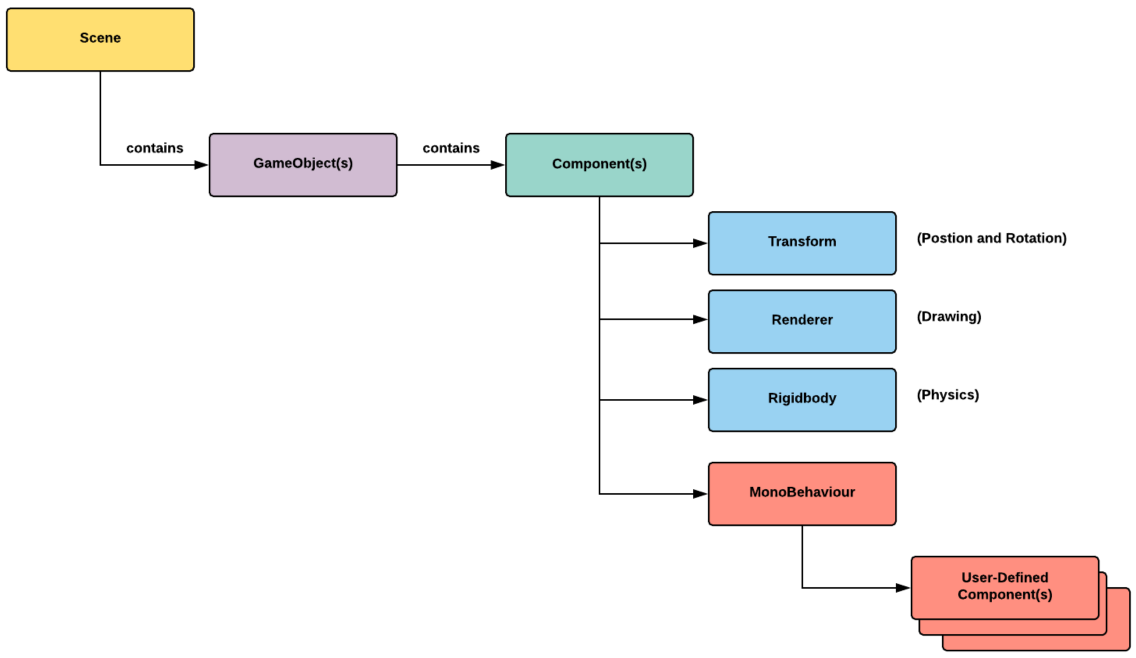


Figure 2.3: Unity Domain Model Diagram.

Unity is a component-based game engine, therefore, during a scene creation, the user can reach the GameObject and Transform objects. By default, each scene comes with a GameObject called Main Camera with transform component, the Camera component, and an Audio Listener to pick up audio in the application.

2.2 Unity Lifecycle

All the event functions of Unity are called in certain moments. Every script attached to the game objects extends MonoBehaviour can override these functions. MonoBehaviour inherits the Behaviour class, and Behaviour inherits the Component class.

There are few functions involved in Unity's Lifecycle.

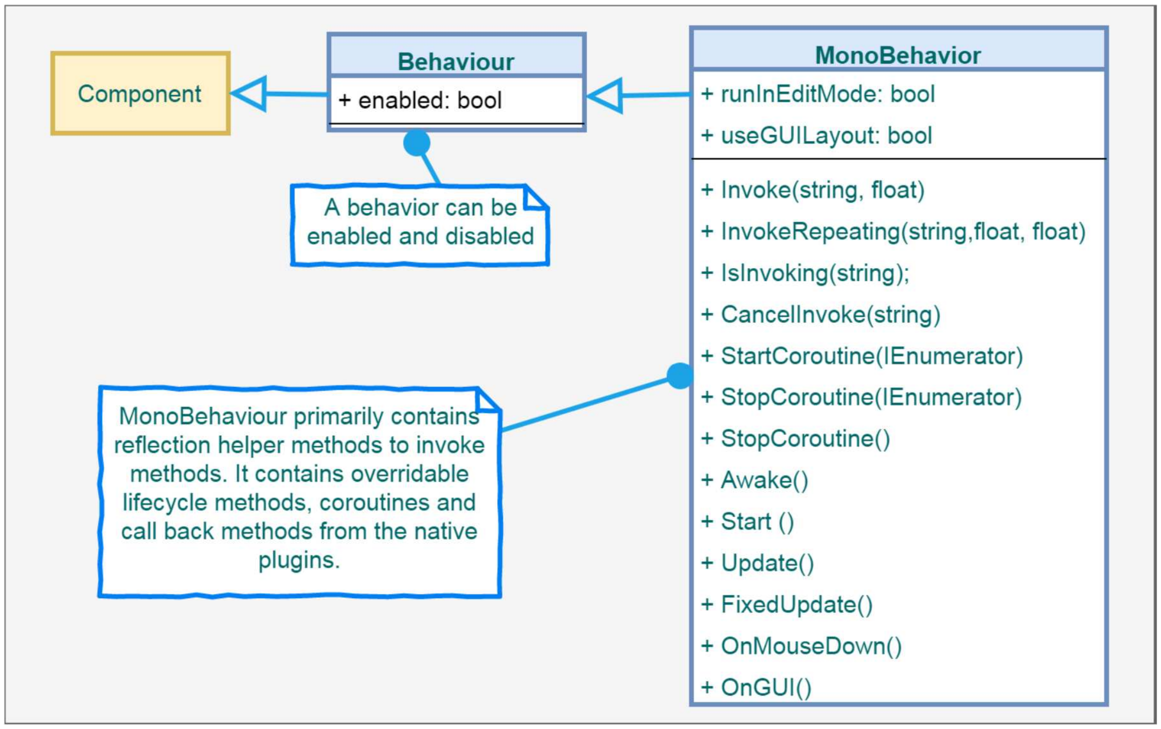

Component
Behaviour
+ enabled: bool
A behavior can be enabled and disabled
MonoBehavior
+ runInEditMode: bool
+ useGUILayout: bool
+ Invoke(string, float)
+ InvokeRepeating(string,float, float)
+ IsInvoking(string);
+ CancelInvoke(string)
+ StartCoroutine(IEnumerator)
+ StopCoroutine(IEnumerator)
+ StopCoroutine()
+ Awake()
+ Start ()
+ Update()
+ FixedUpdate()
+ OnMouseDown()
+ OnGUI()
MonoBehaviour primarily contains reflection helper methods to invoke methods. It contains overridable lifecycle methods, coroutines and call back methods from the native plugins.

Figure 2.4: Design of the Monobehaviour class.

The following functions are called when the first scene loads. This function is called one time per game object in the frame.

- Awake(): If a game object is active during startup, It is always called first for the object when the script is attached or reset.
- OnEnable(): If the object is active, it is called right after the awake method or when the script is enabled on desired events,

The following function is called before rendering or the first frame of the scene updates.

- Start(): It is called only once during the lifecycle and only if the script is enabled at initialization time.

The following functions are called while the frame updates.

- FixedUpdate(): It is called once during a physics cycle.

- Update(): It is called multiple times per cycle but called once per frame. It is used to regularly update the game logic.
- LateUpdate(): This is called after all the calculations in the Update are completed. It is called once and at the end of the game logic cycle.

The following function is called when the behaviour becomes disabled or inactive.

- OnDisable(): It is called for any cleanup code and called only once during the lifecycle. It is also called if the object is destroyed.

The following function is called when the object is destroyed.

- OnDestroy(): It is called in the last frame before the script is destroyed.

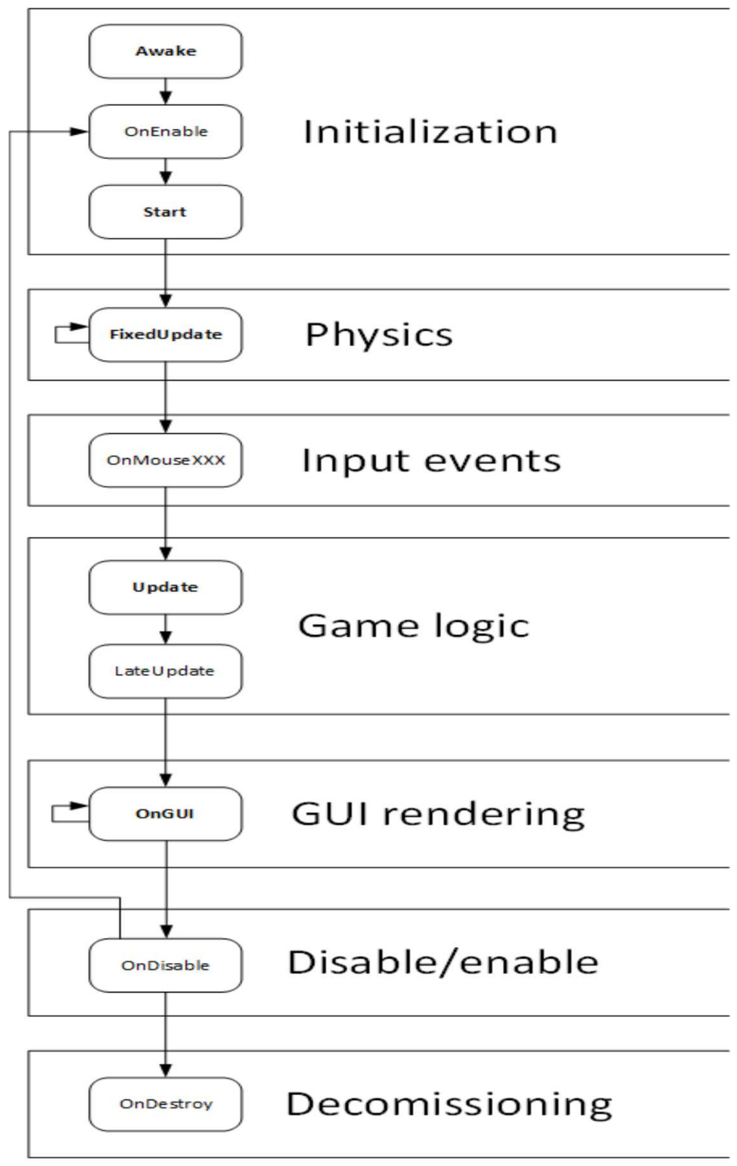


Figure 2.5: Execution order of Event Functions.

3. BACKGROUND

Unity is one of the most popular game engines because of its ability to deploy a single application for multiple platforms with minimal programming effort. The developer community is very active, and the developer forums are full of helpful threads. Unity3D provides a large variety of textures, game avatars, and plugins through its asset store which aids the development of applications.

In Unity, every action and event are required to be scripted, however, Unity provides a large collection of useful examples and segments to help anyone. The visual environment of making a game is scene view. Most scripts are bounded with objects or prefabs to object to work.

In this thesis, we introduced a field of artificial intelligence including Reinforcement Learning techniques, agent simulation, and Imitation Learning. These fields are implemented in the game to help our player learn a conditioned trajectory to model human motion inside a polytope.

3.1 ML-Agents

Unity offers tools not only to create virtual simulated environments with customizable physics, landscapes, and characters but also to train reinforcement learning (RL) agents against those environments. Unity provides a software development kit (SDK), ML-Agents as open-source software that allows us to train intelligent agents using Deep Reinforcement Learning, Imitation Learning, Neuro-Evolutionary Strategies, or other machine learning methods through a simple to use Python API.

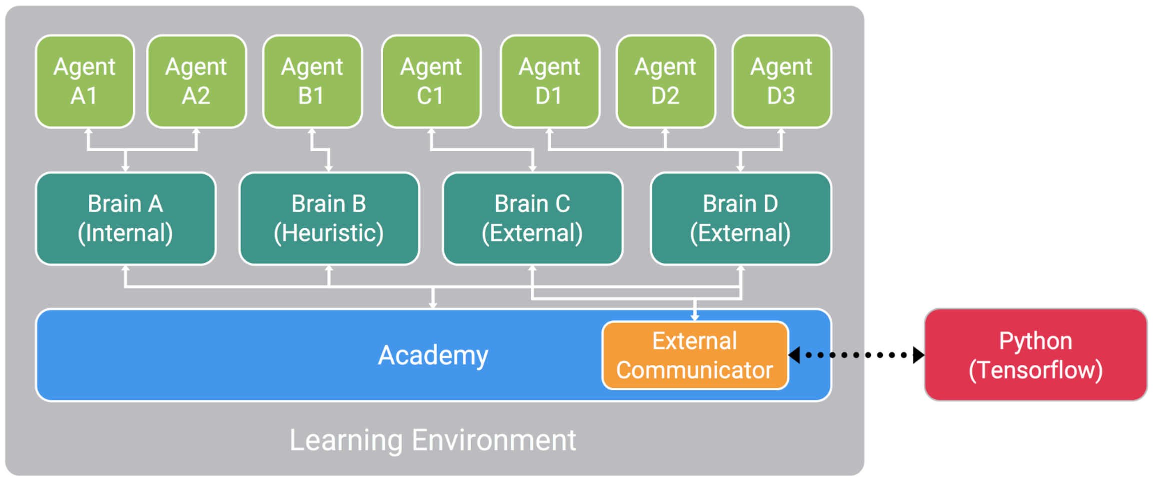


Figure 3.1: Visual Depiction of Learning Environment [2].

There are three main kinds of objects within any Learning Environment.

- Agent: An agent acts as an actor that has certain states and perform some actions within the environment based on some reward.
- Brain: This controls the decision regarding the state and action space of the linked agent. The modes vary as External (Decision made using ML Library with Python API), Internal (Decision made using a trained model), Player (Decision made using player input), or Heuristic (Decision made using as coded in the script).
- Academy: This controls the scope of a single environment. The academy defines Engine Configuration, Frameskip, and the Global episode length.

3.2 Reinforcement Learning

Reinforcement Learning (RL) [3] is an area of Machine Learning (ML) that allows us to train agents to take actions in an environment to maximize a long-term objective. The objective varies according to the game developed. In our thesis, we want the agent to navigate the ball to the goal as many times as it can.

RL is a type of unsupervised learning which assigns reward to the agent, thus, mapping observations to actions. The observations are what an agent measures from its environment using sensory inputs or ray tracing. The action varies as translating the position of the agent, rotating the agent, or controlling the power of the ball.

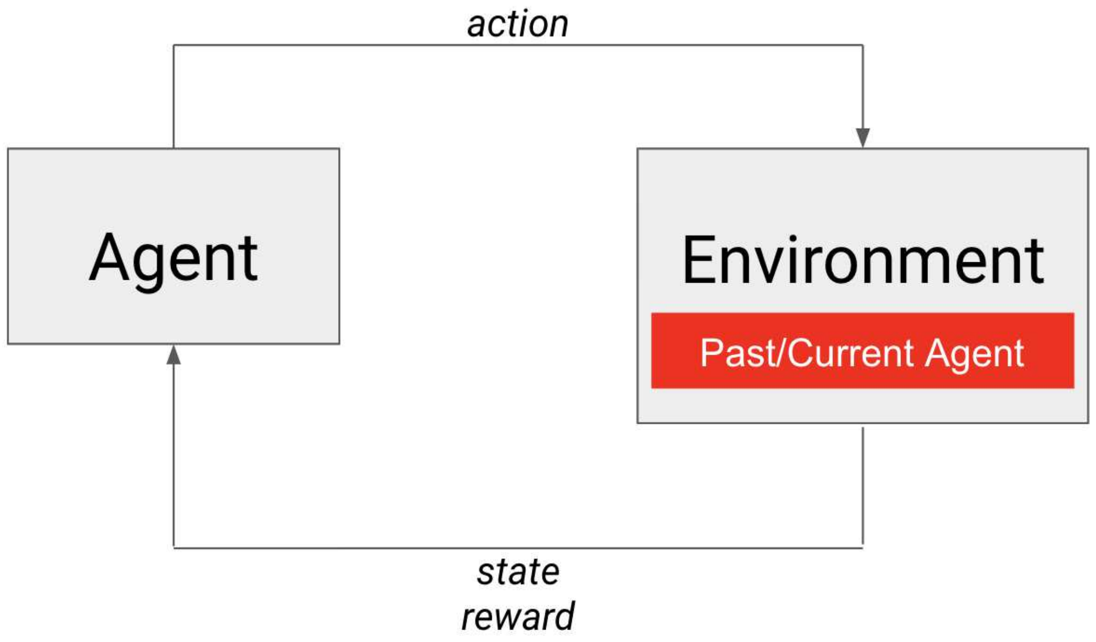


Figure 3.2: The reinforcement Learning cycle [4].

In a traditional RL problem, the agent is trained to learn a policy that maximized its cumulative reward. While training an agent, Rewards are provided to the agent indicating how good it has done to complete a task. The rewards can be hand-coded according to the overall result or can be internal. When training Unity agent a small negative reward is added at every step. The agent learns the objective because it receives a large positive reward when it completes an action favorable to the objective.

The agent's behavior is subject to the constraints of the environment [5]. These limit the viable agent behaviors and are the environmental forces the agent must learn to deal with to obtain a high reward.

3.3 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a new class of reinforcement learning algorithms released by OpenAI. It is a state-of-the-art approach to implement and tune hyperparameters. PPO makes use of the Policy gradient method by using multiple epochs of stochastic gradient ascent to perform minibatch updates. This optimization method is favored to be well balanced with complexity, ease of tuning, and simplicity.

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

Figure 3.3: Objective Function of PPO [5]

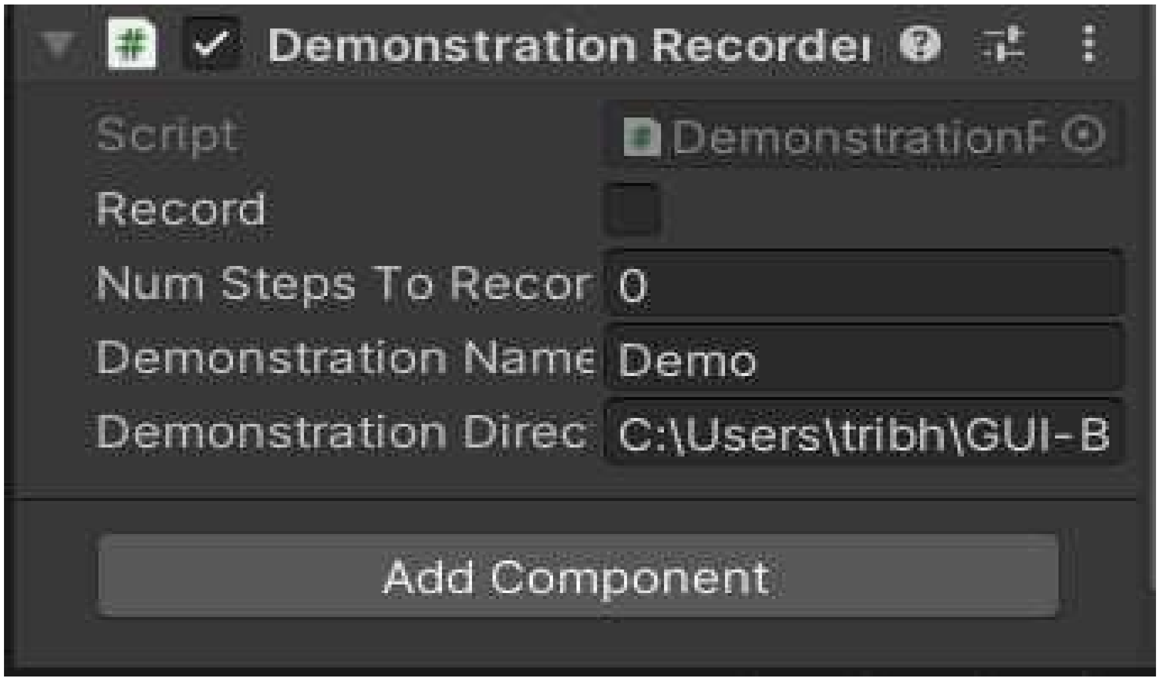
- Θ is the policy parameter.
- $E^{\wedge}{}_{t}$ denotes the empirical expectation over timesteps.
- r_t is the ratio of the probability under the new and old policies, respectively.
- $A^{\wedge}{}_{t}$ is the estimated advantage at time t.
- ε is a hyperparameter, usually 0.1 or 0.2.

The PPO algorithm is implemented in Tensorflow and communicates with the running Unity application over a socket.

3.4 Imitation Learning

It is sometimes very difficult for the Reinforcement Learning algorithm to teach a behavior to an agent via trial-and-error methods if the environment changes randomly with certain steps. Therefore, it is often used parallelly by simply demonstrating the behavior we wanted the agent to perform by providing real-world examples of observations and actions from the game. We use Generative Adversarial Imitation Learning (GAIL) to ensure an optimal action for the agent. Imitation Learning (IL) uses pairs of observations and actions from an expert demonstration to learn a policy [6].

Using GAIL, two neural networks are used simultaneously. One is an actual policy that the agent works on and the other is a network called the Discriminator. The role of the discriminator is to determine whether actions or observations taken by the agent came from the decision from RL or if it came from the demonstration. If the agent itself came up with the action or observation, the reward is assigned.


Demonstration Recorder

Script DemonstrationF

Record

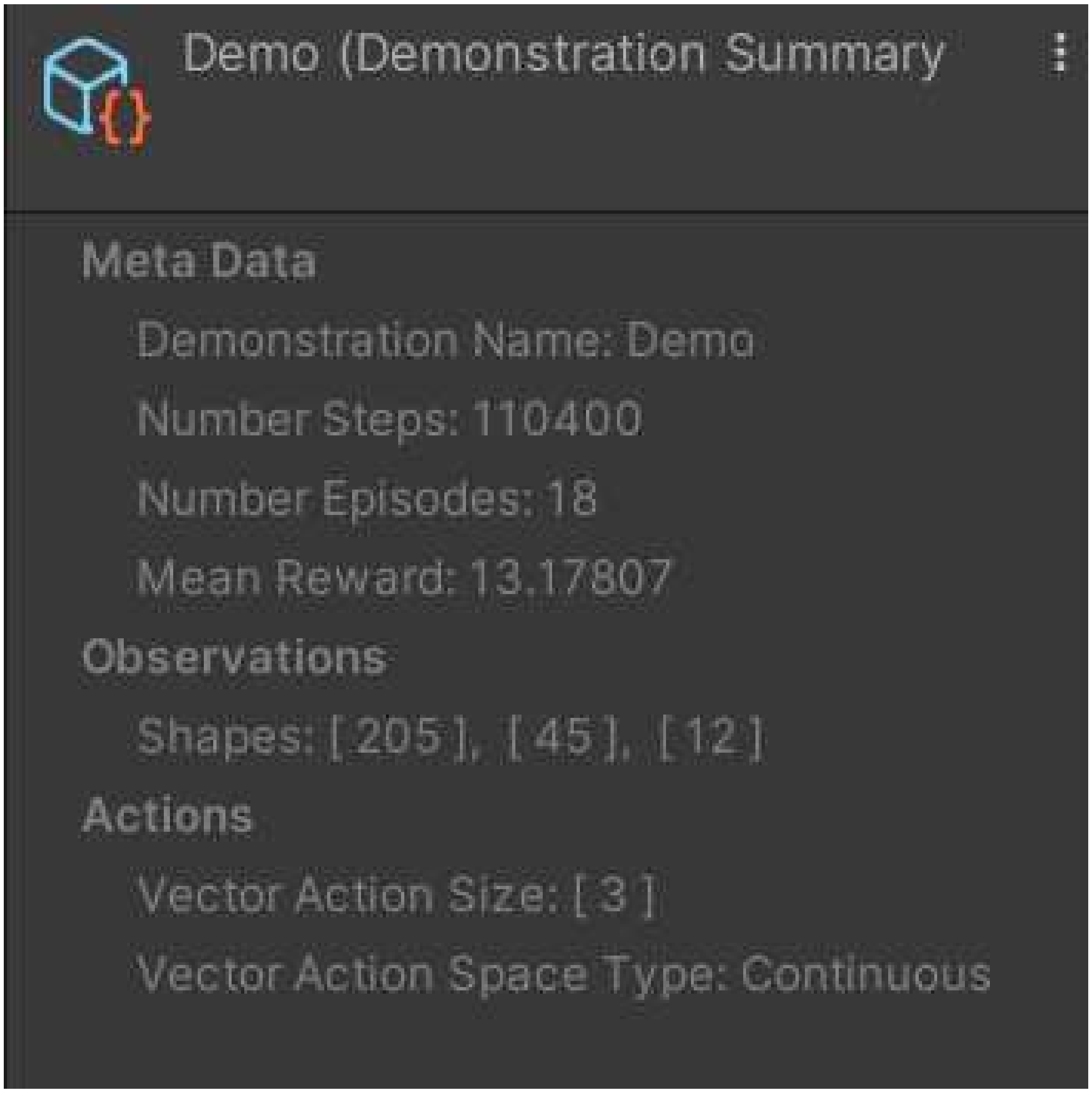
Num Steps To Recor 0

Demonstration Name Demo

Demonstration Direc C:\Users\tribh\GUI-B

Figure 3.4: Demonstration recorder component

Demonstration Recorder component is added to the agent in the scene. The scene is played depending upon the complexity of the game and all the data is recorded and saved as a .demo file.



Demo (Demonstration Summary

Meta Data

Demonstration Name: Demo

Number Steps: 110400

Number Episodes: 18

Mean Reward: 13.17807

Observations

Shapes: [205], [45], [12]

Actions

Vector Action Size: [3]

Vector Action Space Type: Continuous

Figure 3.5: Demonstration Inspector

The demonstration file contains information regarding observations, actions, and rewards given to the agent. It will show the number of episodes played and how much reward is amassed.

3.5 Visual Encoding

In a Unity scene, Agent can observe the environment around it in several ways. Adding more observations, on one hand, slow down the training process and make it complex, however, It may significantly improve the policy at the cost of wall time. Unity offers a solution by allowing developers and researchers to gather visual observations and encode them to add an extra stack of observational parameters. Convolutional Neural Networks (CNN) are implemented on the Images generated from the cameras attached to the agent.

There are several choices for Visual encoders.

- simple: This is the default encoder type for encoding visual observations. It consists of two convolution layers.
- nature_cnn: This implementation is proposed in an article by Nature. We have used this encoder type in our thesis. It consists of three convolution layers.
- resnet: This implementation uses the Importance Weighted Actor-Linear Architecture (IMPALA) Resnet that consists of three stacked layers, each with two residual blocks, making it a much larger network than the other two [6].
- match3: This implementation is optimized for games where content creation is sequential like Candy crush and can be used down to visual observation sizes of 5*5.

4. METHOD

4.1 Preprocessing

With the Unity ML-Agents toolkit, it is possible to train the behavior of an agent, however, we need to define few entities at every moment of the environment.

- Agents: Agent is a character attached to a Unity GameObject. All the actions are performed on the agent. It also handles receiving and assigning state rewards and collecting observations.
- Behavior: Behavior is like a function linked to an agent and identifies the attributes of the agent. It defines how many observations an agent can gather in a stacked vector. It also defines if the agent takes a continuous action or a discrete action. There are three types of behavior: Learning, Heuristic, or Inference. Learning behavior is one that is not, yet, defined but about to be trained [7]. A Heuristic behavior is hard-coded using code where the attributes of an agent are defined in a script along with events during episodes. An Inference behavior is one that includes a model already been trained and inferences from a Neural Network file. There can be multiple agents and multiple behaviors in an environment at the same time.
- Observations: Observations to an agent are their sense of the environment. These can either be numerical and/or visual. Visual observations are the images generated by the agent and represent what the agent is seeing at that point in time [7]. These are encoded and processed by the network using CNNs. Numeric observations, on the other hand, measure the attribute of the environment from the point of view of the agent. All the observations in a scene

are sent through a communicator to a Python API which processes the observations and sends back the actions appropriately.

- Actions: It defines the actions an agent can take, which can either be continuous or discrete. Continuous actions are more preferred on the agents which tend to move freely in the environment. There is, however, a huge drop back while using continuous actions as in a complex environment they may take a greater number of steps to complete an episode comparative to discrete actions which are often used in a simple grid world. To use a Long Short-Term Memory (LSTM) in the neural network, it is recommended to use discrete actions over continuous.
- Reward signals: Rewards are somewhat like incentives that are given to an agent indicating how good it has done the task. It need not be given every time an agent does some action, and whenever the agent does some tasks that is not preferred as an optimal behavior it is punished by a negative reward. The reward signal controls the actions of a state and needs to be implemented in a way that the cumulative reward maximizes and should be increasing.

4.2 Model Architecture

Our model consists of few parts.

4.2.1 Network Architecture

The main architecture of our model consists of the PPO Reinforcement learning model. The neural network consists of 512 hidden units with 4 hidden layers. The inputs are normalized beforehand.

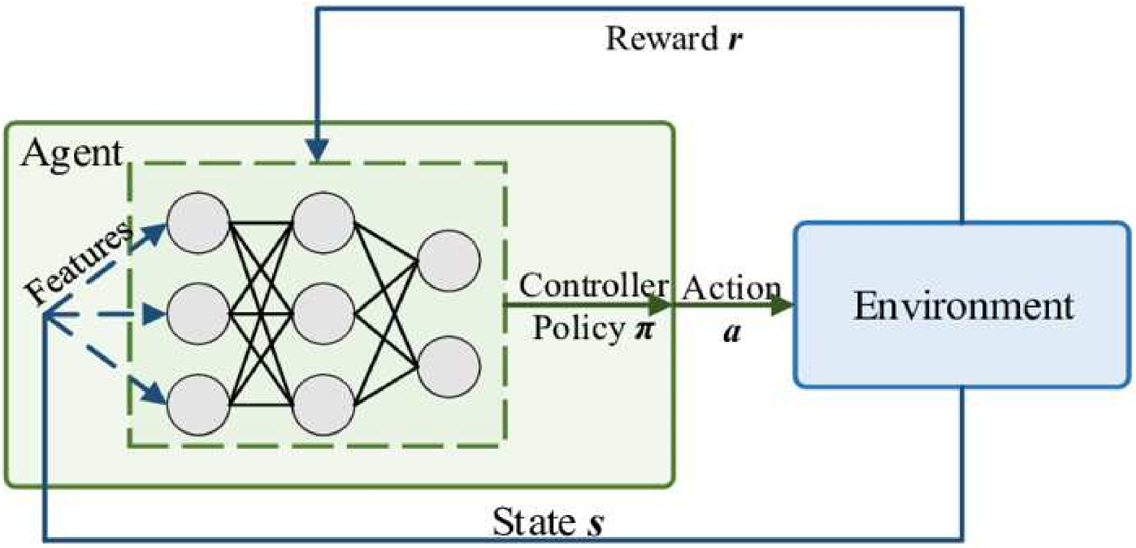


Figure 4.1: Illustration of a Reinforcement Learning Architecture

The maximum number of observation collection and the action that can be taken across all environments were set to be 10 million before ending the training process.

Hyperparameter	Value	Description
Batch size	2048	The number of experiences in each iteration of gradient descent.
Buffer size	20480	Number of experiences to collect before updating the policy model.
Learning rate	0.0003	The initial learning rate for gradient descent.

Beta	0.005	Strength of the entropy.
Epsilon	0.2	Determines how rapidly the policy can evolve.
Lambda	0.95	Regularization parameter.
Number of Epochs	3	Number of passes to make through the experience buffer when performing gradient descent optimization [6].
Learning rate schedule	linear	How learning rate changes over time.

Table 4.1: Data Table List of hyperparameters

4.2.2 Visual Encoder Architecture

The Visual Encoder consists of a Convolutional Neural Network first proposed in a Nature article. It consists of three convolutional layers.

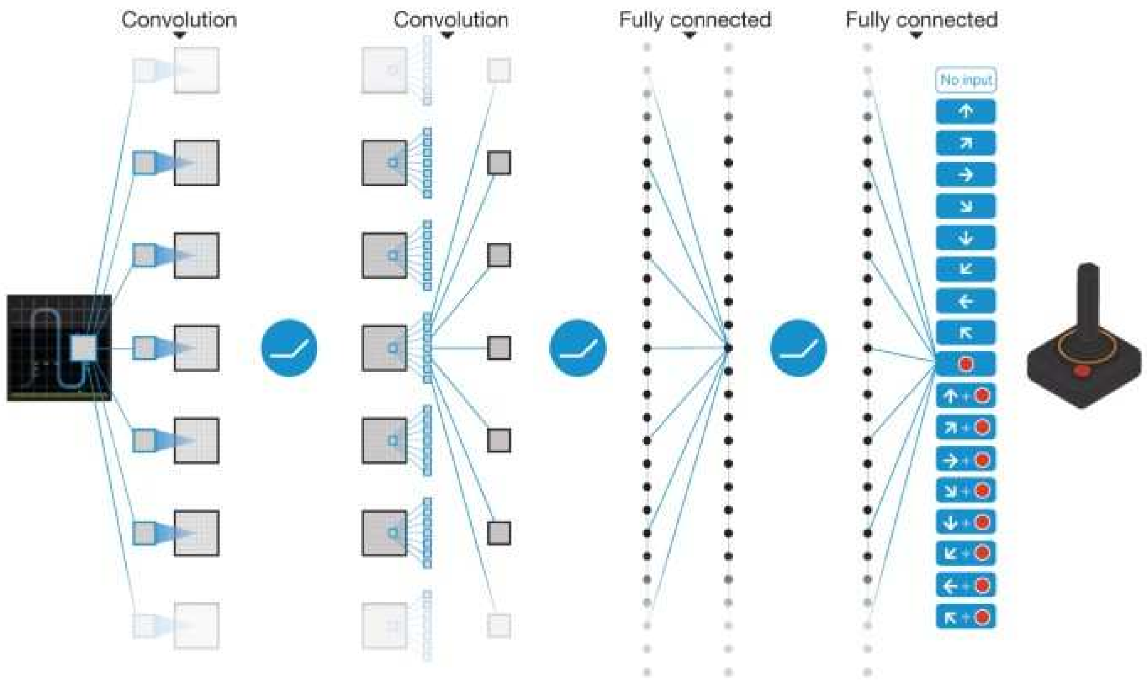


Figure 4.2: Illustration of the Convolutional neural Network

"The input to the neural network consists of an 84×84×4 image produced by the preprocessing map. The first hidden layer convolves 32 filters of 8×8 with a stride of 4 with the input image and applies a rectifier nonlinearity. The second hidden layer convolves 64 filters of 3×3 with stride 1 followed by a rectifier. The final hidden layer is fully connected and consists of 512 rectifier units. The output layer is a fully connected linear layer with a single output for each valid action [8]." The number of actions varied from 1 and 3 on the games we considered.

Network Settings	Value	Description
Normalize	True	Determines if normalization is to be applied to observation inputs.
Hidden Units	512	The number of units in the hidden layers of the neural network.
Number of Layers	4	The number of hidden layers in the neural network.
Visual Encode Type	nature_cnn	Encoder type for encoding visual observations.
memory	null	Whether to use memory for an agent.

Table 4.2: Data Table List of Network Setting

4.3. Training Details

We performed experiments on 76 games. The same model architecture was used across all the games.

Reward Signal	Value	Description
Trainer type	ppo	The type of trainer to use.
Extrinsic -> Gamma	0.99	A discount factor that determines how far into the future

		the agent should care about possible rewards.
Extrinsic -> Strength	1.0	Factor by which to multiply the reward given by the environment.
Gail -> Gamma	0.99	The discount factor for future rewards for the GAIL network.
Gail -> Strength	1.0	Factor by which to multiply the reward.
Gail -> Learning Rate	0.0003	Learning rate for the discriminator.
Gail -> Demo path	Demo.demo	The directory of the demonstration file.
Gail -> use actions	false	Determines if the discriminator wants the agent to mimic the actions from the demonstrations

Table 4.3: Data Table List of Reward Signal

4.4 Evaluation Procedure

The trained agents were evaluated by playing 76 games parallelly for up to 5 hours using GPU as an Inference device with different initial random conditions. The action was selected at 60 Hz and played under controlled conditions. The player was not allowed to pause, save, or reload the game.

Evaluation Settings	Value	Description
Maximum Steps	10000000	Total number of steps that can be taken across all environments.
Time Horizon	1000	Determines how many steps of experience to collect per-agent before adding it to the experience buffer [6].
Summary Frequency	50000	The number of experiences that needed to be collected before generating and displaying training statistics [6].
Capture frame rate	60	Instructs simulation to try to render at a specified frame rate [9].
Time scale	20	It defines the multiplier for the deltatime in the simulation.

Table 4.4: Data Table List of Evaluation Setting

5. RESULTS

To evaluate the training performance of our agent, we took advantage of the Tensorboard platform, which allows us to evaluate statistics while an environment is running. Tensorboard is a visualization tool provided by Tensorflow and is used to monitor training progress.

In a Unity environment, a cumulative reward for each step of the training session is calculated and later is averaged with the other environments. This mean cumulative reward is plotted on the Tensorboard, and each reset is also implemented to record the progress.

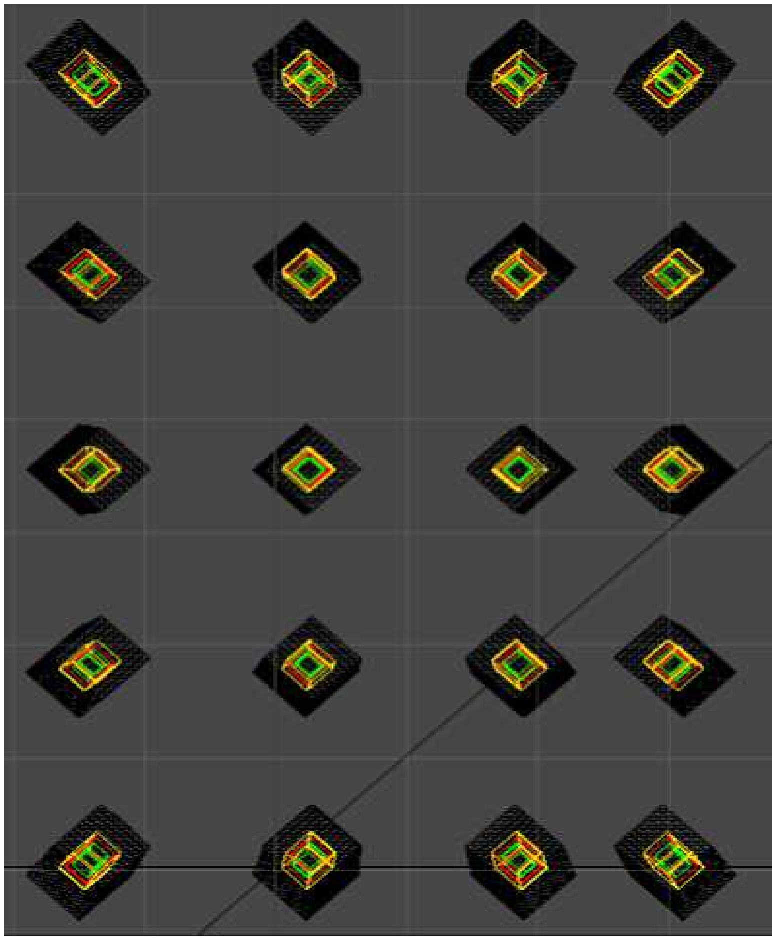


Figure 5.1: Final implementation of the game

The ML-agents training program saves the following statistics:

5.1 Cumulative Reward

We have the plot of the mean cumulative reward per episode over all the training agents. The cumulative reward increases during a successful training session.

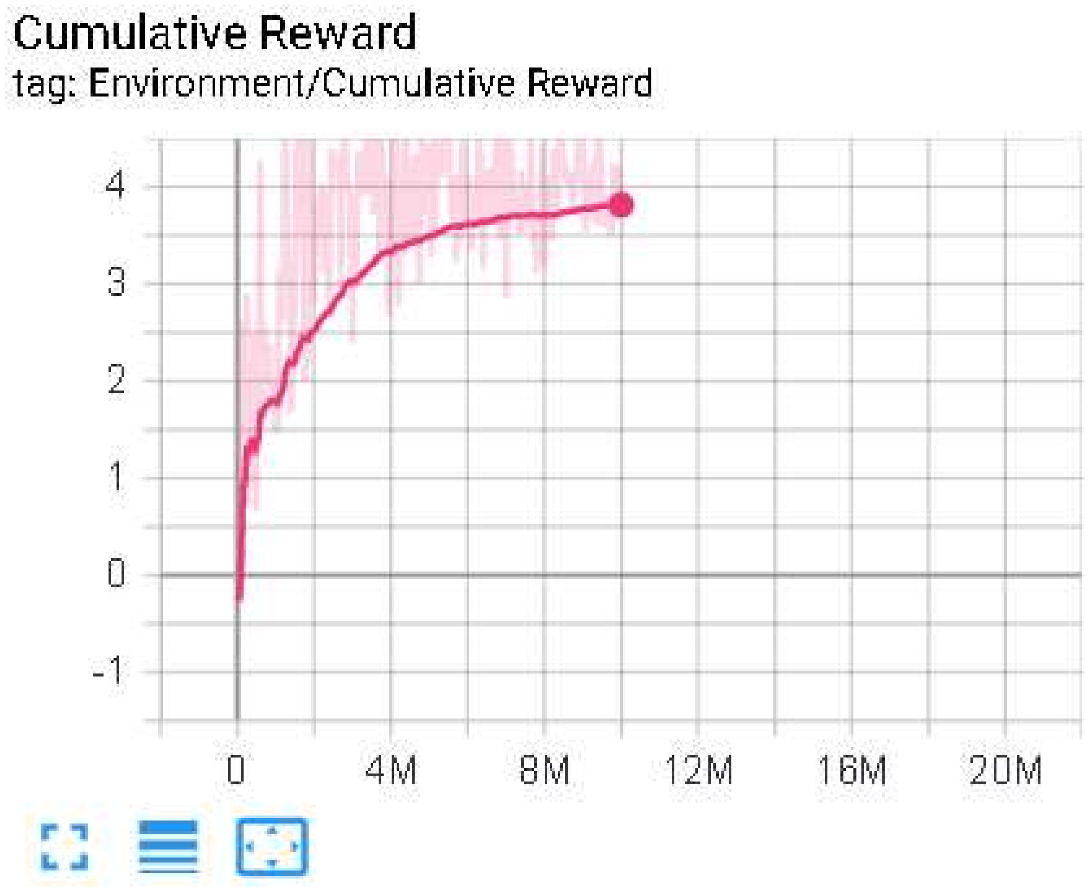


Figure 5.2: Cumulative reward over each step in the environment.

5.2 Episode Length

The mean length of each episode in the environment of all agents [10].

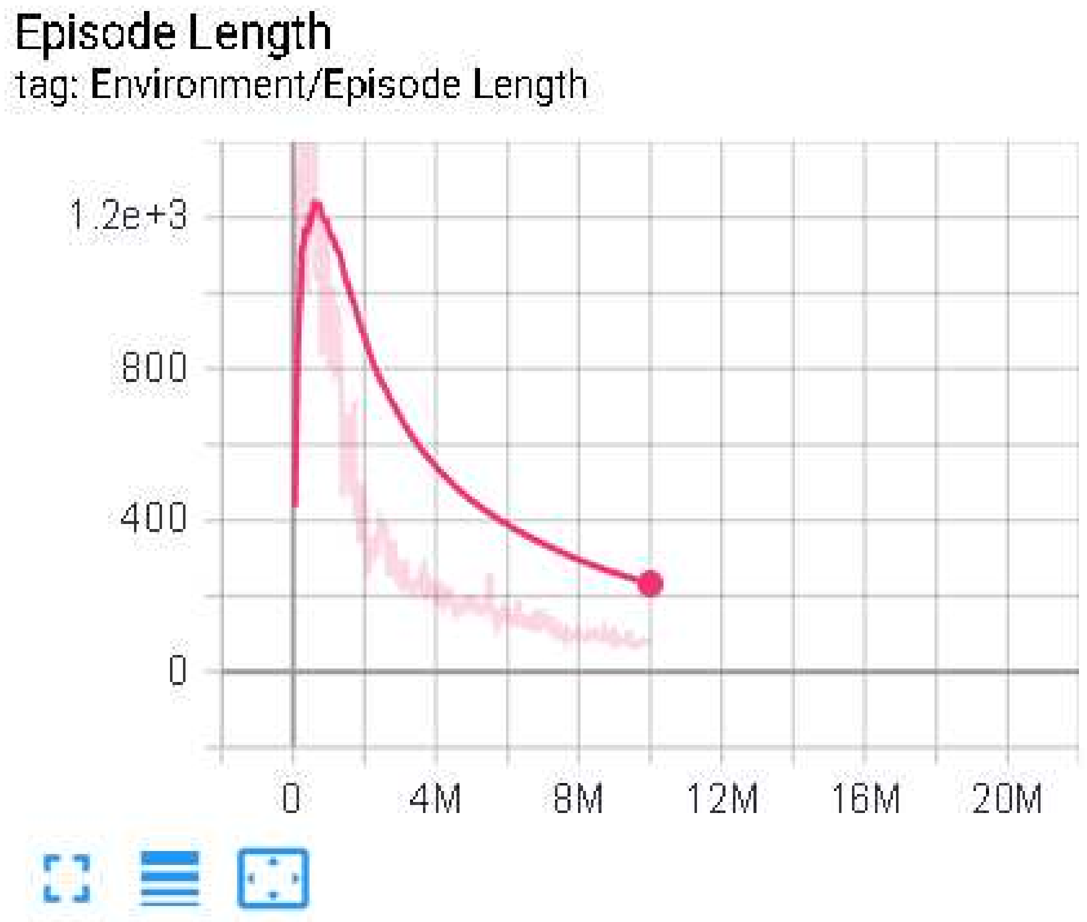


Figure 5.3: Episode length over each step in the environment.

5.3 Entropy

How random the decisions of the model are? Should slowly decrease during a successful training process. If it decreases too quickly. The beta hyperparameter should be increased [10].

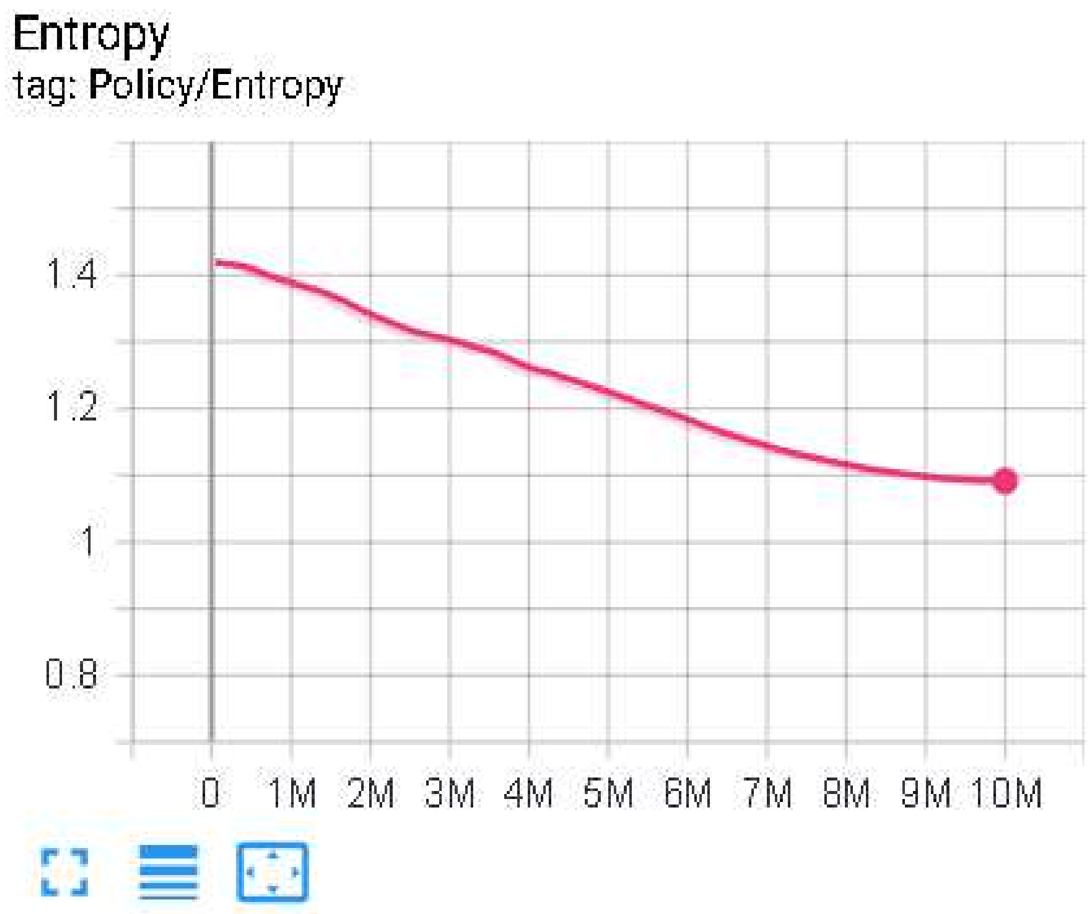


Figure 5.4: Entropy over each step in the environment.

5.4 Learning Rate

How large a step the training algorithm takes as it searches for the optimal policy [10]. It should decrease over time.

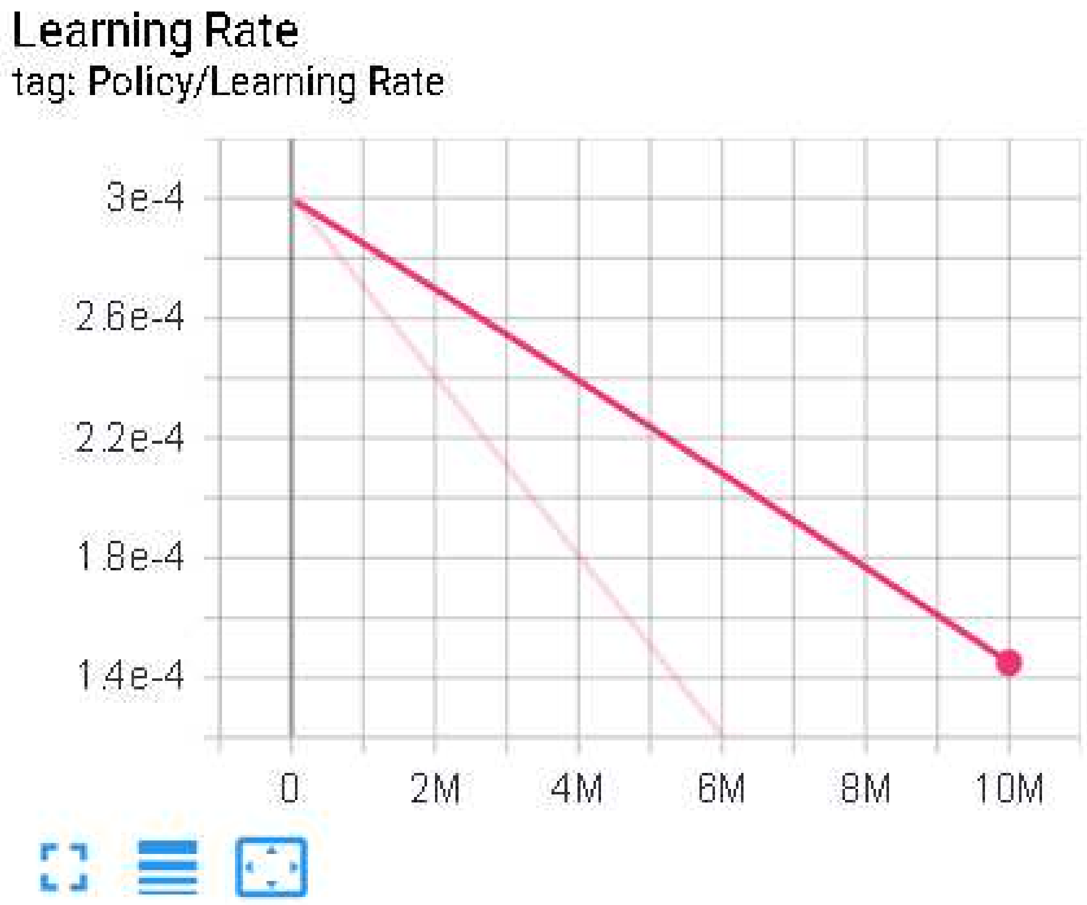


Figure 5.5: Learning rate over each step in the environment.

5.5 Policy Loss

The mean loss of the policy function update. Correlates to how much the policy (process for deciding actions) is changing. The magnitude of this should decrease during a successful training session [10].

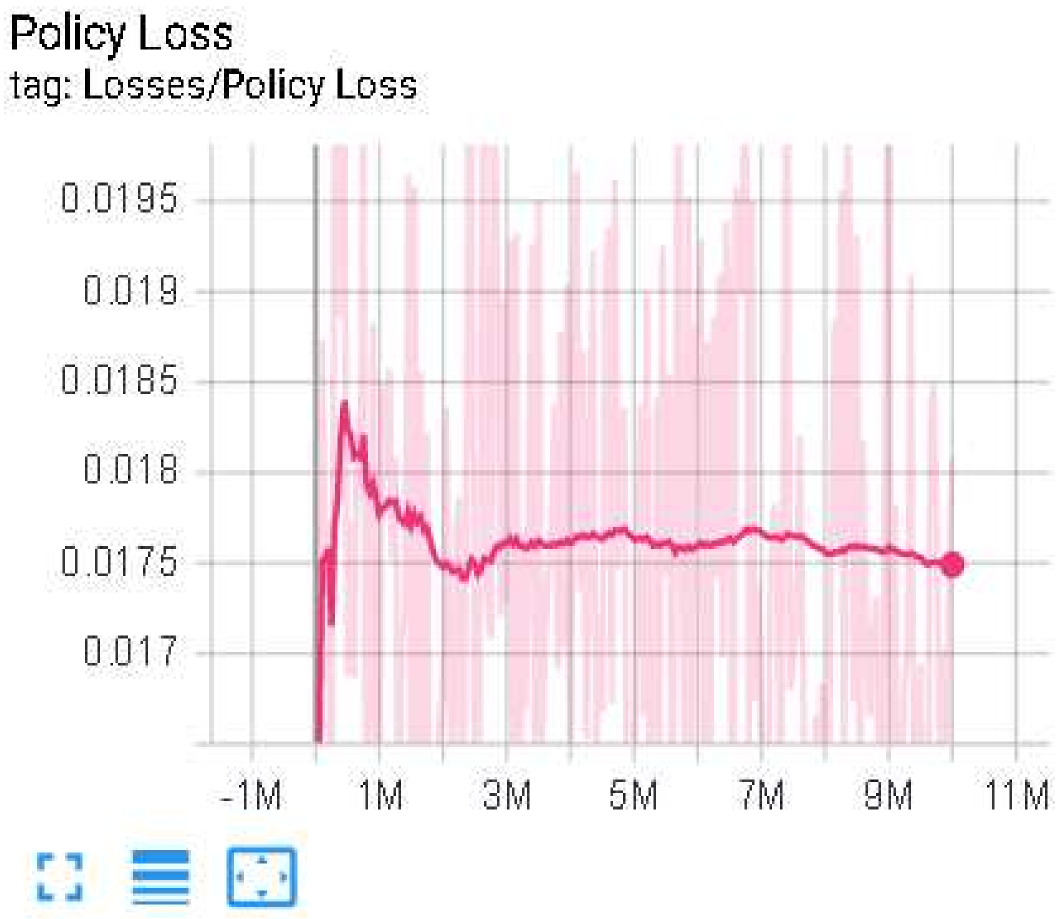


Figure 5.6: Policy loss over each step in the environment.

5.6 Value Estimate

The mean value estimate for all states visited by the agent [10]. It should increase during a successful training session.

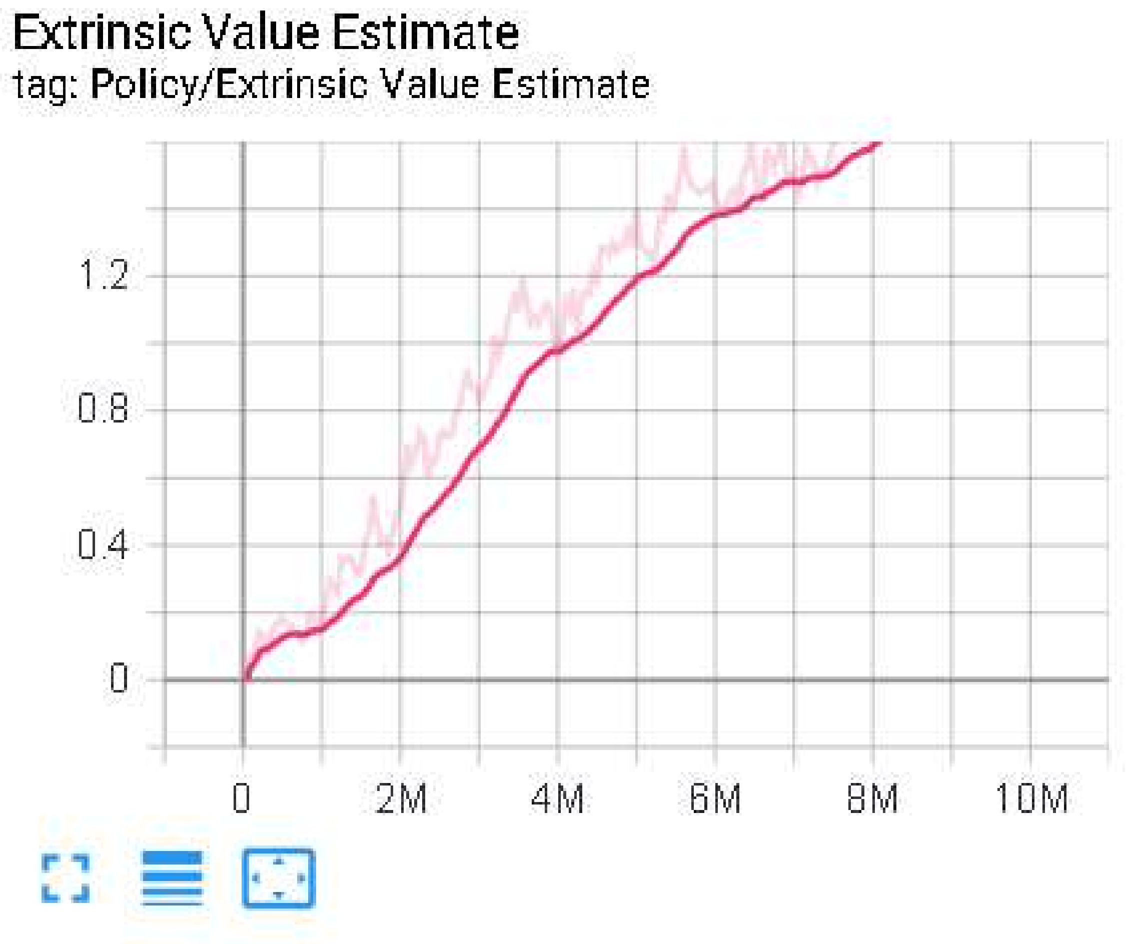


Figure 5.7: Value estimate over each step in the environment.

5.7 Value Loss

The mean loss of the value function update. Correlates to how well the model can predict the value of each state. This should decrease during a successful training session [10].

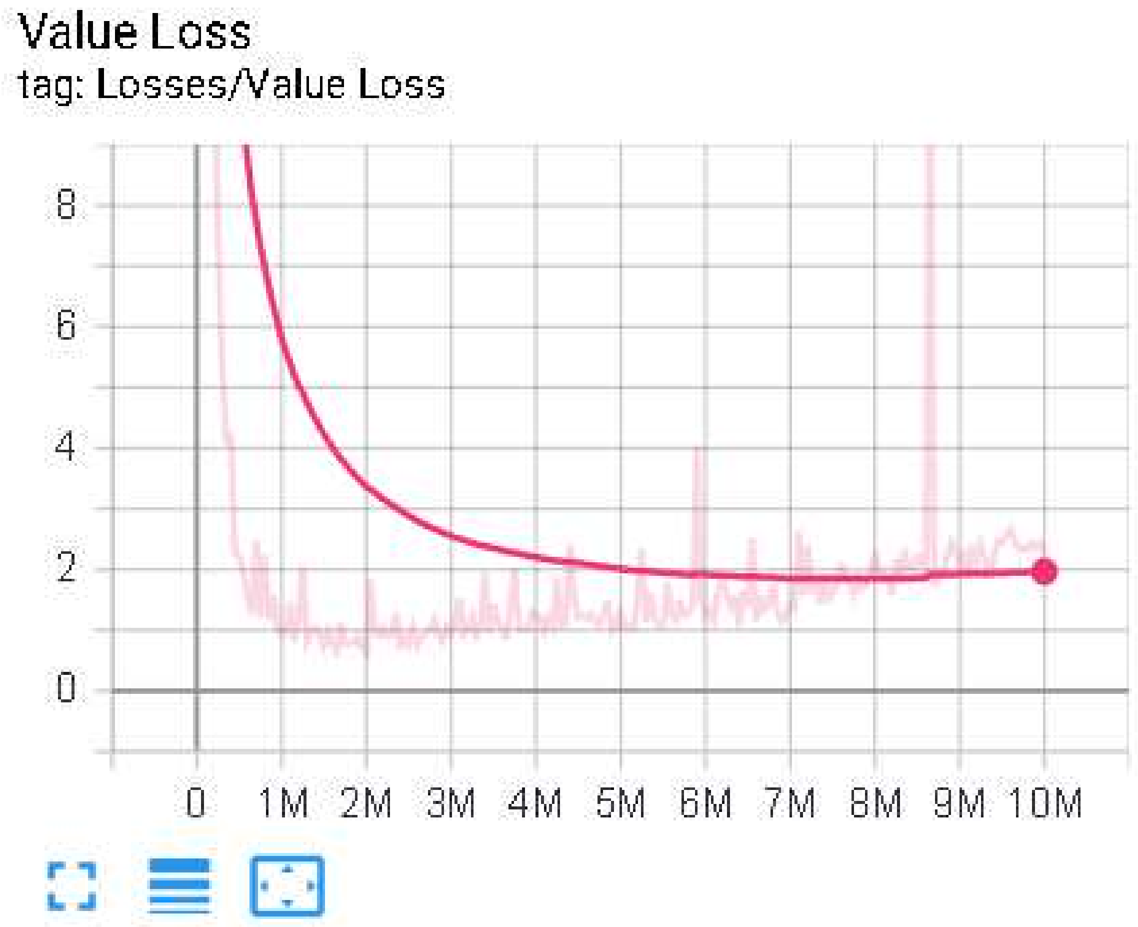


Figure 5.8: Value loss over each step in the environment.

6. CONCLUSION

We have presented to our knowledge the first approach for a 3D game where the player had to search the goal(star) inside a polytope and shoot the ball so it could reach the goal avoiding obstacles in between. This thesis also investigated how artificial intelligence techniques can be used to predict the trajectory of the ball.

To make an integrated game, two scenes were designed. The game scene for a human player to play was achieved with a third-person camera rotating in either direction with respect to the ground. The player was provided with an extra feature with a visual Prediction Line that predicted the trajectory which the ball would follow if the player tended to shoot it that instant. It was all achieved by Simulating an extra scene on top of the current scene.

The thesis presents a trained artificial agent which can predict a conditioned trajectory to model human player motion using imitation learning. We also implemented a reward system in the environment so our agent can navigate the ball correctly on the course. We implemented several topics including Reinforcement Learning techniques, Convolutional Neural Networks, and Imitation Learning.

6.1 FUTURE WORK

Currently, the game is achieved completely from a third-person perspective. The player can enjoy shooting the ball while rotating the camera around the plane. The player is accompanied by a Prediction line which helps the player to follow the trajectory of the ball before it shoots from the fire position.

Creating solutions for game development is time-consuming, thus future work with Deep-RL would improve the implementation of the training model. GAIL and Nature-CNN fulfilled our initial requirements and demonstrated the advantages and possibilities; however, it would be great to investigate the other aspects of AI.

The current version of the game has no more drawbacks but implementing more features to it could make it much better.

- Resource Utilization: We noticed that it is taking a long time to train agents in our Unity scene. Instantiation in unity is very inefficient and our scene instantiated a lot of game objects and destroy it later which consume a lot of resources of our local system. This could be made faster by running on servers using cloud services.
- Agent Rotation: Currently, we can rotate the agent with the combination of two axes. It would be interesting to test the movement in the third dimension.
- Cumulative Reward: The mean cumulative reward of our environments was making small, but not huge progress.

REFERENCES

[1] Xia, Peng. “3D Game Development with Unity A Case Study: A First-Person Shooter (FPS) Game.” (2014)

[2] Juliani, A. 2017, Introducing: Unity Machine Learning Agents Toolkit, Unity Blog, accessed 20 April 2021, <https://blogs.unity3d.com/2017/09/19/introducing-unity-machine-learning-agents/>.

[3] Nakayama, Y., Wang, H., Zhuang, Y. 2020, Training a reinforcement learning Agent with Unity and Amazon SageMaker RL, AWS Machine Learning Blog, accessed 20 April 2021, <https://aws.amazon.com/blogs/machine-learning/training-a-reinforcement-learning-agent-with-unity-and-amazon-sagemaker-rl/>.

[4] Cohen, A. 2020, Training intelligent adversaries using self-play with ML-Agents, Unity Blog, accessed 21 April 2021, < https://blogs.unity3d.com/2020/02/28/training-intelligent-adversaries-using-self-play-with-ml-agents/>.

[5] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)

[6] Training Configuration File 2021, Unity technologies, accessed March 21 2021, < https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md>.

[7] ML-Agents Toolkit Overview 2021, Unity technologies, accessed March 21 2021, < https://github.com/Unity-Technologies/ml-agents/blob/main/docs/ML-Agents-Overview.md#gail-generative-adversarial-imitation-learning>.

[8] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529-533 (2015).

[9] Unity ML-Agents Python Low Level API 2020, Unity technologies, accessed April 20 2021, < https://github.com/Unity-Technologies/ml-agents/blob/8ad72c08d7d9408c4dda4601beec7f8e82f911ed/docs/Python-API.md#engineconfigurationchannel >.

[10] Using TensorBoard to Observe Training 2018, accessed April 2021, < https://github.com/miyamotok0105/unity-ml-agents/blob/master/docs/Using-Tensorboard.md>.

[11] Ho, J., Ermon, S.: Generative adversarial imitation learning. In: Advances in Neural Information Processing Systems. Pp. 4565-4573 (2016)

[12] Yuan, Ye and Kris M. Kitani. "3D Ego-Pose Estimation via Imitation Learning." ECCV (2018)

[13] Espeholt, L., Soyer H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. arXiv preprint arXiv:1802.01561,2018

[14] Brendmoe, M., "Generating gameplay scenarios through faction conflict modeling." (2012)

[15] Knudsen, K., “Freia: Exploring Biological Pathways Using Unity3D.” (2015)

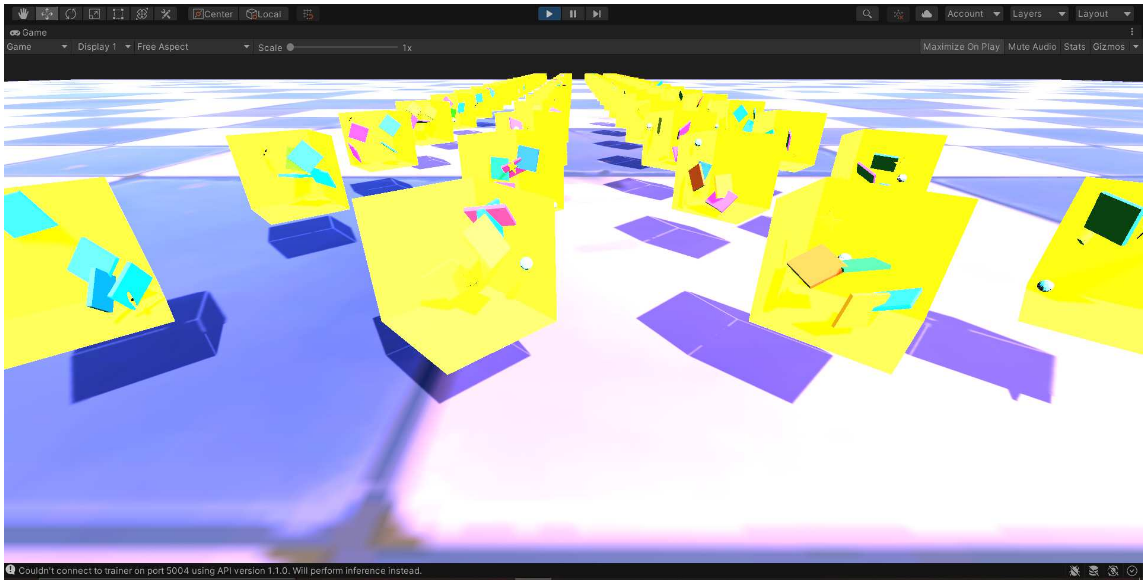
[16] Izgi, E., “Framework for Roguelike Video Games Development.” (2018)

[17] Penzentcev, A., “Architecture and implementation of the system for serious games in Unity 3D.” (2015)

[18] Brown, H., “Applying Imitation and Reinforcement Learning to Sparse Reward Environments.” (2020)

[19] T. Singh, Y. Jain and V. Kumar, “Predicting parole hearing result using machine learning,” 2017 International Conference on Emerging Trends in Computing and Communication Technologies (ICETCCT), 2017, pp. 1-3, doi: 10.1109/ICETCCT.2017.8280342

APPENDIX



Couldn't connect to trainer on port 5004 using API version 1.1.0. Will perform inference instead.

Figure 8.1: Demonstration of the game played by the trained agent.

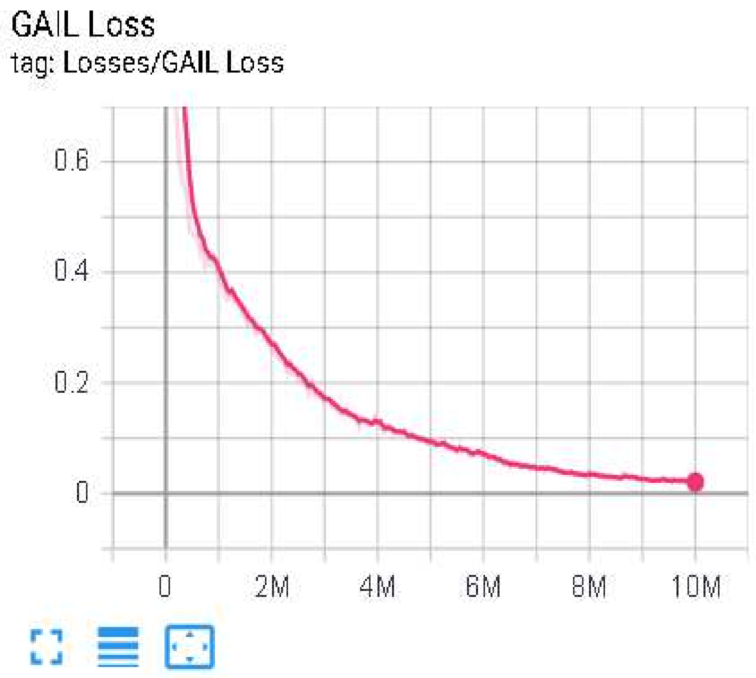


Figure 8.2: GAIL Value loss over each step in the environment.

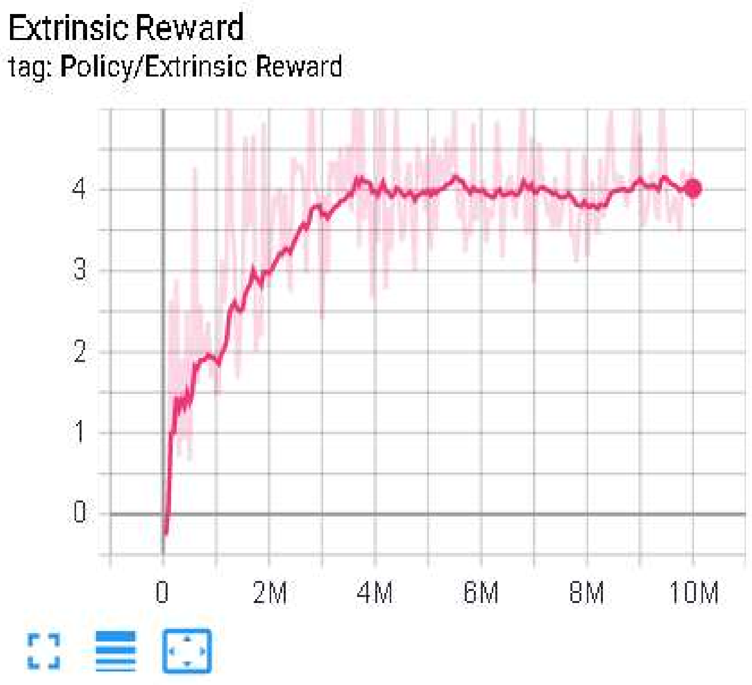


Figure 8.3: Extrinsic Reward over each step in the environment.

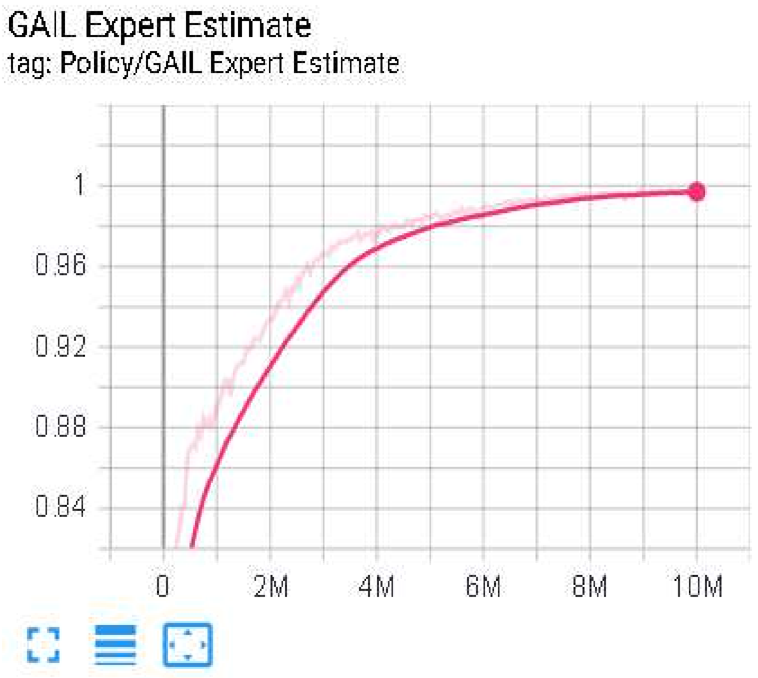


Figure 8.4: Gail Policy Estimate over each step in the environment.

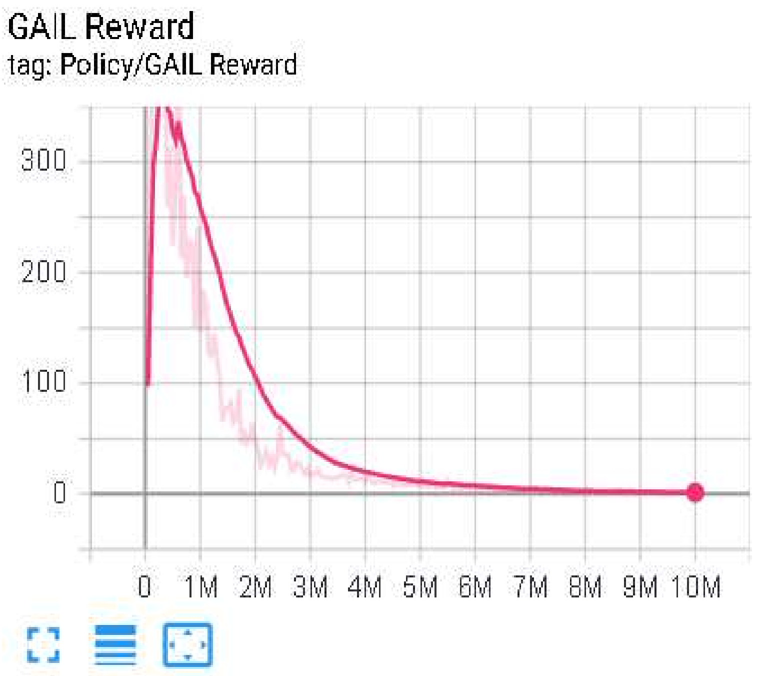


Figure 8.5: Gail Reward over each step in the environment.