Implementation of Reinforcement Learning Architecture to Augment an AI That Can Self-Learn to Play Video Games

by

Aqil Mahmud
18341010
Aswat Karim Khan
18301282
Mohammad Mehdi Hasan Rafi
18101629
Kazi Rayhan Fahim
18301114

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
School of Data and Sciences
Brac University
January 2023

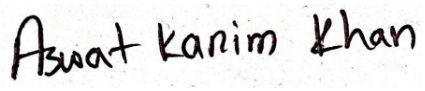
Declaration

It is hereby declared that

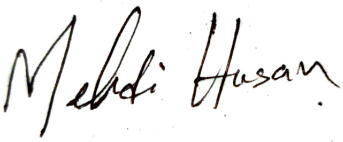
1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Aqil Mahmud
18341010



Aswat Karim Khan
18301282



Mohammad Mehdi Hasan Rafi
18101629

Kazi Rayhan Fahim
18301114

Approval

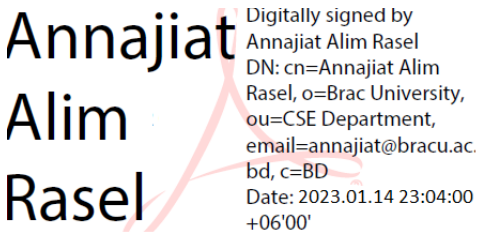
The thesis/project titled "Implementation of Reinforcement Learning Architectures to Augment an AI That Can Self-Learn to Play Video Games" submitted by

1. Aqil Mahmud (18341010)
2. Aswat Karim Khan (18301282)
3. Mohammad Mehdi Hasan Rafi (18101629)
4. Kazi Rayhan Fahim (18301114)

Of Fall, 2022 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on January 23, 2022.

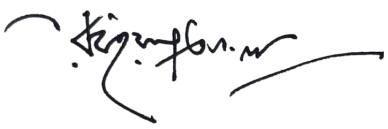
Examining Committee:

Supervisor:
(Member)



Annajiat Alim Rasel
Senior Lecturer
Department of Computer Science and Engineering
BRAC University

Co-Supervisor:
(Member)



Rubayat Ahmed Khan
Senior Lecturer
Department of Computer Science and Engineering
BRAC University

Thesis Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam
Professor
Department of Computer Science and Engineering
BRAC University

Head of Department:
(Chair)

Sadia Hamid Kazi PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
BRAC University

Abstract

This paper is intended to be a practical guide in terms of getting up and running with reinforcement learning. Ideally, it aims to bridge the gap between practical implementation and the theories available for RL. The theory of reinforcement learning involves two main components: an environment, which is the game itself and an agent, which performs an action based on its observation from the environment. Initially, no in-game rules will be given to the agent and it will be rewarded or punished based on the action that it will take. The goal is to increase Proximal Policy Optimization (PPO) to maximize the reward that our agent will get, so over time it will learn what action to take in order to do so. Therefore, we will develop an AI agent that will be able to learn how to play one of the most popular arcade games of all time, Street Fighter. We preprocess our game environment and apply hyperparameter tuning using PyTorch, Stable Baselines, and Optuna to do it. This approach will basically train different types of RL architecture and find a model with the most weighted parameters. Moreover, we are going to Fine Tune that model and run our test cases on it. We are going to see how a reinforcement learning algorithm learns to play.

Keywords: Reinforcement Learning, Neural Networks, Games, AI, Proximal Policy Optimization

Acknowledgement

We would like to express our sincere gratitude to our supervisors, Mr. Annajiat Alim Rasel and Mr. Rubayat Ahmed Khan, for their invaluable guidance, support, and encouragement throughout the duration of this research project. Their expertise in the field and dedication to helping us succeed were truly inspiring.

We would also like to thank the members of our thesis committee, for their valuable input and feedback on our work. Their insights and suggestions greatly contributed to the improvement of our thesis.

We would like to extend our appreciation to the School of Data and Sciences for providing us with the necessary resources and support to complete this project.

We are grateful to our, friends, and family for their unwavering support and encouragement during the challenging times while completing this thesis.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificial Intelligence and Gaming, in spite of mainstream thinking, do not manage everything well together. Is this a disputable assessment? Indeed, it is, however, we will make sense of it. There is a distinction between Artificial Intelligence and Artificial way of behaving. We do not maintain that agents in our games should outfox players. We believe they should be however brilliant as it seems to be important to give tomfoolery and commitment. We would rather not stretch the boundary of our ML bot, as we ordinarily do in various Industries. The adversary should be flawed, mirroring a human-like way of behaving. However, games are not just diversion. Preparing a virtual agent to beat human players can show us how to streamline various cycles in a wide range of fun and engagement. This is how Google DeepMind managed its well-known AlphaGo to beat the most grounded Go player in history and scored an objective that was viewed as incomprehensible at that point. In this paper, we will foster an AI agent that can figure out how to play the well-known game Street Fighter, without any preparation. To do it, we execute Reinforcement Learning algorithms. [2]
So, what is Reinforcement Learning in a nutshell and where does it fit in the big world of machine learning and data science? Reinforcement learning focuses on teaching agents through trial and error. In general, you've got an agent and it learns based on the reward that it gets. It will try to perform a different action if it does not get the reward or it will perform an action multiple times if it gets a bigger reward for that specific action. Moreover, Reinforcement Learning is based on actively engaging with an environment which brings us to how the framework actually fits together [14]. Therefore, there are four key points we will have to consider whenever we are working within reinforcement learning as shown in figure 1. They are the agent, the environment, the action and reward plus observations.

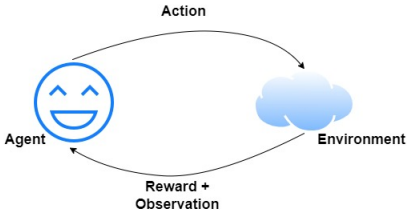


Figure 1.1: An overview of Reinforcement Learning

Agent: Think of your agent as something which is operating within an environment,

so this might be a machine learning model, might also be a person or a player if you're working in a game environment. A policy usually governs it (decides what action to take).
The Environment: This is where an agent is actually operating in. For instance, in our case the player is operating within the game environment, so it is getting a reward based on what it actually does there.
Reward & Observations: The player (agent) will see what's happening within the environment. For example, the player in the game will be able to see what is around them, so in terms of observation, it will see what the game environment actually looks like and then it will also see what reward it accrues based on the actions it takes.
Action: Generally, the player might walk around the environment, it might do something and accumulate a point. It might do something else and does not accumulate a point. It might even lose a life that might be a negative reward.
As it was stated in [1], a really good way to get your head around this is to think of how we might go about training a dog. For example, we want to teach a dog how to sit or how to lay down. Our player (agent) in this case is going to be the dog because we are trying to train our player to be able to take the right action. Now the reward in this case is us giving the dog a treat every time they do the right thing. Initially, you might say sit and the dog might not actually do anything and in this case it has taken an action of doing nothing and in this particular case the environment that it's working with is the environment with yourself in it. The dog will eventually see that it gets no reward because it did not sit down so it might try something else. So, in this case you might say sit again and it might then sit and then it will get a treat. Ideally, it will then start to learn what action to take in response to the environment in order to maximize the reward. Therefore, it is observing the command that you are giving to be able to take the right action, and this is how reinforcement learning works. Your agent tries to take an action in order to maximize its rewards in response to the observations within the environment. It is a little bit different in terms of how you might work with tabular deep learning and machine learning because your agent is actively engaging with a simulated or a real environment and in this case we are going to be dealing with simulated environments. [8]

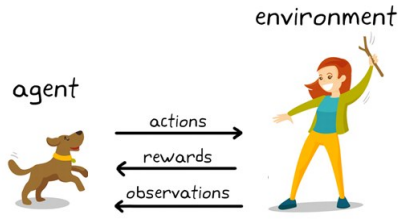


Figure 1.2: Reinforcement Learning in Dog Training

1.1 Research Problems

A reinforcement framework "teaches itself," as it were, by gathering reward signals in light of various activities and states through which the agent runs. In any case, it must be directed by human hands at fundamental levels somewhat. Perhaps the greatest inquiry is additionally one of the most natural: What should be rewarded?

That is simple enough with regards to, say, a straightforward game. The agent figured out how to score a point, give that agent a prize. In any case, what's the significance here with regards to a recommendation system? Leading reinforcement learning researcher John Langford made sense of it via counterexample. One might possibly abuse Personalizer to attempt to anticipate the number of promotions to put on a site. To do as such, you could attach the award to how much income created per occasion. The framework would cheerfully decide to litter your site with advertisements, and users would presumably escape. The methodology ought to be to adjust the reward to the drawn out objective as best as could really be expected, Langford said. That is, at times far from simple or easy. The example intricacy, for example, can shoot up while looking for transient intermediaries — or, in other words, it can require a truly lengthy investment before the model becomes dependable. "There's somewhat of a strain between the most normal statement of the issue, which might include simply upgrading the drawn out remuneration, and the most manageable rendition of the issue to an address, which might include streamlining momentary intermediary," he said. "That is one of the finesses in attempting to approach reinforcement learning issues really." [15]

At an appropriate arrangement of logged encounters with insignificant association with the environment, the offline RL chips away when the agent works on its arrangement with new encounters. This terminates the requirement for the training of AI agents to scale. In any case, it proposes the test where assuming that the model, which is being prepared with a current dataset, makes a move unique in relation to the information assortment agent, one can't decide the award gave to the learning model. Another issue, as recommended by Google AI, is the distributional shift. This happens when the RL algorithms should figure out how to go with choices that contrast from the choices taken in the dataset to work on over the verifiable data. [16]

According to [16], Facebook did a research to imitate DeepMind's AlphaZero. In their mission, it was showed when joined with the inaccessibility of code and models, the outcome is that the methodology is truly challenging, on the off chance that certainly feasible, to replicate, study, refine, and broaden. Neural networks are hazy secret elements whose operations are mysteries to even the makers. They are additionally expanding in size and complexity, upheld by colossal datasets, processing power and long periods of training. These variables make RL models extremely challenging to repeat. Lately, there's been a growing development in AI to neutralize the supposed reproducibility problem, a high-stakes variant of the exemplary it-dealt with the my-machine coding issue. The crisis appears in issues going from AI research that specifically reports algorithm races to admired results because of weighty GPU capability. The Leiden Institute of Advanced Computer Science paper proposes utilizing the 'minimal traces' idea. The thought upholds re-reproduction of activity arrangements in deterministic RL conditions, permitting commentators to check, re-use, and physically review trial results without requiring enormous figure bunches. Different arrangements incorporate following and logging tests, submitting code and making a metadata repository. Second, as opposed to having entrants present their agents, which might possibly be prepared with research-lab levels of GPU wattage, they're expected to submit code prepared utilizing the coordinators' machine. At last, they likewise acquaint randomizing components with ensured results track across various game renditions.

By keeping the actual training on coordinators' machines, with purposely confined computing power, the opposition likewise addresses the test of access. For all around financed research labs, admittance to elevated degrees of processing power isn't quite a bit of an issue. Yet, those assets are not so very much disseminated somewhere else. At the end of the day, to arrive at much more noteworthy levels, reinforcement learning should be both sample efficient and equitable. [15]

1.2 Research Objectives

This research aims to develop a reinforcement learning model which will be able to play an arcade game called Street Fighter. In order to do that we are going to set up the gym retro environment with the Street Fighter ROM or effectively the game cartridge. Moreover, we are going to perform Hyperparameter Tuning using a library called Optuna, which is a critical step to build any machine learning model. This will help us to produce a good model for our reinforcement learning environment. Finally, we will be fine tuning our model using stable baselines. Therefore, the objectives of this research are:

- To understand different types Reinforcement learning algorithms and how they work.
- Implement the RL algorithms using different Python packages.
- To augment a model which learns to play a game without any supervision.
- To test and evaluate the model.

Chapter 2

Literature Review

As game universes develop more tremendous and complex, ensuring they are playable and bug-free is turning out to be progressively hard for engineers. Furthermore, gaming organizations are searching for new instruments, including artificial intelligence, to assist with beating the mounting challenge of testing their products. Another paper by a group of AI specialists at Electronic Arts shows that deep reinforcement learning agents can assist with testing games and ensure they are adjusted and feasible. In the previous 10 years, AI research labs have utilized reinforcement learning to dominate complex games. All the more as of late, gaming organizations have additionally become keen on involving reinforcement learning and other AI strategies in the game improvement life cycle.

2.1 Network of RL Architecture

Figure 2.1, depicts different type reinforcement learning architecture which are commonly used in game industry.

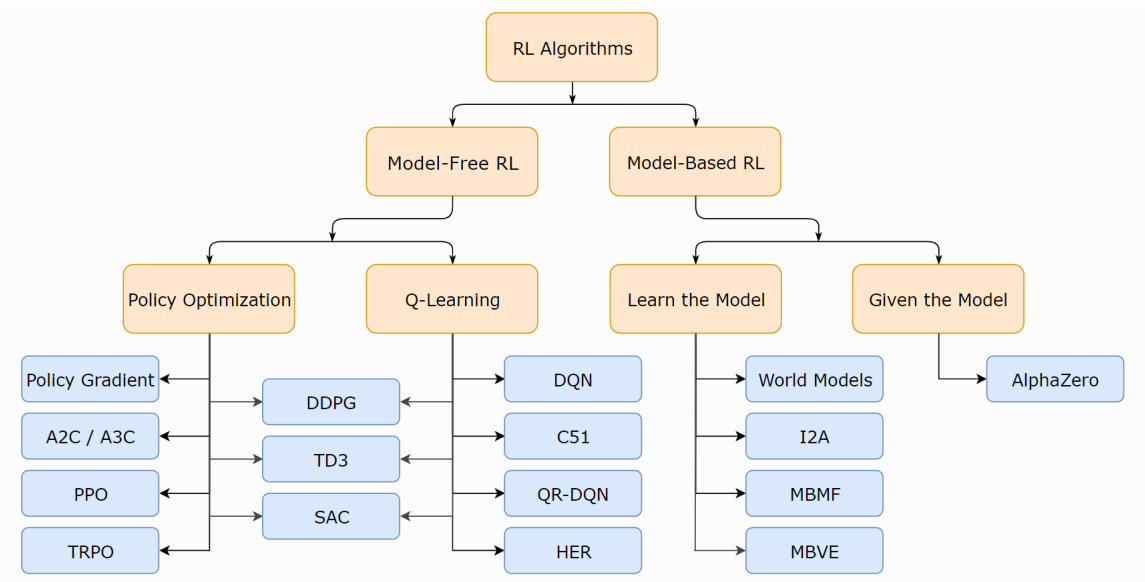


Figure 2.1: Network of RL Architecture [7]

Reinforcement learning (RL) encompasses a diverse set of techniques, which are commonly divided into two broad categories: model-based RL and model-free RL.

The development of model-free RL is more active currently, but this does not mean that model-based RL is not useful. The main idea behind model-free RL is that it uses the current state values to make predictions, while model-based RL uses predictions about the future state of the model to generate the best possible action. [7]
In this paper, we will focus on model-free RL and specifically look at the PPO algorithm. However, it is worth noting that the best algorithm for a particular use case depends on the type of action space that is being used. For example, the A2C algorithm works well with box, discrete, multi-discrete, and multi-binary action spaces, while DQN works only in discrete spaces. Therefore, it is crucial to choose an algorithm that is compatible with the action space of the problem at hand. It is essential to know which algorithm should be used for a specific action space, but the different algorithms should be considered as options to choose from, as some may perform better than others.

2.2 Related Work

2.2.1 StarCraft

Players need to defeat the enemies by performing actions according to real-time game states in StarCraft. Including temporal and spatial reasoning, opponent modelling, multi-agent collaboration and adversarial planning, designing a machine learning model is difficult. With limited flexibility and intelligence, human experiences and replays are the only basis of most models, as of now. The first stage to solve StarCraft AI is to study the micromanagement. To tackle micromanagement scenario, an algorithm for optimizing Greedy Markov Decision Processes in an episodic manner, zero-order method was introduced, which performs better than policy gradient and DQN [4]. Multi-agent deep reinforcement learning technique named BiCNet is used to play StarCraft combat games. The technique utilizes bi-directional neural networks to acquire the ability to collaborate and employs actor-critic reinforcement learning as its structure. BiCNet effectively learns a few helpful techniques, and is versatile to different undertakings, showing preferred exhibitions over GMEZO. In previously mentioned works, researchers predominantly creates concentrated techniques to play micromanagement. To reuse the information between different micromanagement situations, they likewise consolidate curriculum transfer learning figuring out how to this technique. This further develops the example proficiency, and beats GMEZO and BiCNet in enormous scope situations. This technique has preferred execution over different strategies in micromanagement tasks. There are a few testing StarCraft II micromanagement undertakings, and utilize incorporated preparing and decentralized execution to learn helpful ways of behaving. This at last beats cutting edge multi-agent deep reinforcement learning methods. [5]

2.2.2 MOBA and Dota2

RTS games is the origin of Multiplayer Online Battle Field (MOBA), and it contains two teams, moreover, there are five players in each team. Five players in a group should participate to eliminate adversaries, redesign legends, and ultimately obliterate the enemy base, to beat the rival. However, research on MOBA games

is still at an early stage, resulting in fewer studies than in traditional RTS games. Dataset analysis and contextual analysis are the works that are mostly done on MOBA. However, MOBA received more attention from researchers recently, due to a progression of forward leaps that DRL accomplishes in game AI.The most popular mobile MOBA game in China is a simplified version of Dota, known as King of Glory. The architecture of this game contains Tree Search and deep neural networks. In 1v1 MOBA scenario, MCTS-based deep reinforcement learning algorithm is efficient and can be utilized as it was demonstrated in experimental results. OpenAI is responsible for proposing most of the extraordinary works on MOBA. DRL technique with self-play is also successful in 5v5 Dota2 scenario as it was in 1v1 and 2v2, according to the demonstration of their results. The neural network contains LSTM layer as its core component making a simple model architecture. OpenAI Five has mastered the skills of pursuing, navigating through a forest, trickery, group fighting and strategizing for the team's victory, and it defeated human champion OG with a score of 2:0, using the support of proximal policy optimization algorithm and massively distributed cloud computing. Their works on MOBA research with DRL techniques truly opens a new door. [13]

2.2.3 DeepMind lab

Based on Quake3, OpenArena has an extension of a 3D first-person game platform called DeepMind lab. This platform is more composite containing more realistic physics and richer visuals compared to other first-person game platforms. The UNREAL agent has an average performance of 87% compared to expert human players, and it results in an average increase in learning speed of 10 times when compared to A3C on a challenging set of DeepMind lab tasks. Continual learning has made quick progress, as learning agents became more dynamic. In DeepMind lab, there is a consideration of an implicit sequence of tasks to test consistent learning proficiency [9]. Unicorn, a novel agent design, exhibits solid nonstop learning and beats a few standard agents on the proposed space. A method was introduced that uses teacher agents to initiate the training of another learning agent. When performing various tasks and testing on the DMLab-30 suite, the introduced training method significantly improves the sample efficiency of new agents, and it outperforms the previous demonstration by 42%.

2.2.4 Minecraft

Across different game modes, players can assemble innovative manifestations, designs, and work of art across a sandbox construction game called Minecraft. Recently, Project Malmo has become a popular platform for AI research in gaming, particularly for working with 3D data that varies widely. It is an experimental platform that is based on the popular game Minecraft. It upholds an enormous number of situations, including route, critical thinking errands, and endurance to cooperation. A clever Q-learning approach was proposed [12] with state-activity reflection and warm beginning involving human thinking to learn compelling strategies in the Microsoft Malmo cooperative AI challenge. One of the major challenges in Minecraft is the ability to transfer information from one task to another. A DRL (deep reinforcement learning) agent that can transfer information by learning

reusable skills, which can then be integrated into a hierarchical DRL network (H-DRLN).H-DRLN (hierarchical deep reinforcement learning network) demonstrates superior performance and low learning complexity compared to standard DQN in Minecraft, and has the ability to transfer information between related Minecraft tasks without additional training. To settle the fractional or non-Markovian perceptions issues, another DRL algorithm was proposed in view of counterfactual lament minimization that iteratively refreshes an estimate to an aggregate cut advantage work. On challenging Minecraft first-person navigation benchmarks, this algorithm has the potential to significantly outperform existing techniques. [3]

Chapter 3

Methodology

3.1 Environment Preprocessing

Generally, in machine learning or deep learning data pre-processing is the most important step and it is no different for reinforcement learning [10]. Even though, we are not dealing with datasets for this particular project, getting the environment in an appropriate state is very important for its training. Some important attributes of the Street fighter game environment are, “enemy_matches_won”, “score”, “matches_won”, “continuetimer”, “enemy_health” and “health”. Therefore, to preprocess our environment, we did observation preprocess first. In this step, we calculated the change in pixels from our current frame versus our last frame in order to capture movement and then we dropped pixels from the frames to train faster. We did this by gray-scaling and reshaping our frames.

Figure 3.1: Gray scaling and reshaping of a frame to drop pixels

Moreover, we filtered actions which is a parameter available inside the Gym Retro package. Then changed our reward function to “score” initially. The reward function could be any of the attributes from our environment. For example, if we set the reward function to “enemy_health”, then it might get points every time the health of the enemy decreases and it might even lose points every time the agents’ health decreases.
To be more comprehensive, we created a method called ”init”, to inherit from the base environment (the game) and set up the ”observation space” and ”action space.” Here, the ”observation space” is a ”box” shape of 200, 256, 3 with a data type of

uint8. Then we reshaped the "observation space" to 84 by 84 by 1 to make it a gray-scaled frame with a smaller number of pixels. We set the "action space" to "multibinary" with 12 options, and changed it to a "discrete observation space". To set up an instance of the game, an additional parameter was added. This filters out the actions in the environment to only give valid button combinations rather than all multi-binary values.
Moreover, we also created a method called "step". This method allows the agent to take a step in the environment, processing the observation, calculating the frame delta, reshaping the reward function and returning the processed observation, frame delta, and reward. The step method will use the base environment to take a step and return the unprocessed "observation", "reward", "done" and "info" values.

3.2 Custom Environment

For our research, we created a class in Python to be used as a custom environment, including a number of methods that were initialized in order to create the environment. These methods include "init," "step," "render," "reset," and "pre-process." The "init" method gets called when the environment is created, and "step" is the method that defines what actions are taken in the environment. "Render" is the method used for rendering the environment, and "reset" is the method called to restart the environment.

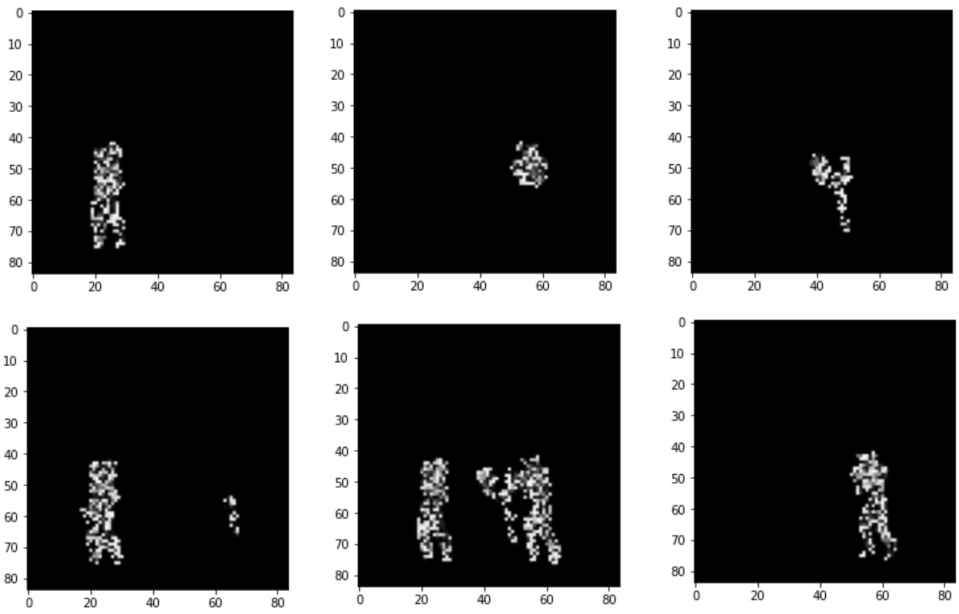


Figure 3.2: The output of our custom environment

Upon examination of the action space, it was determined that it is multi-faced but not zoomed in enough. Further examination of the shape revealed it to be satisfactory. The pre-processed values were visible as they were being rendered separately. Upon initial testing, it was determined that the last frame could be obtained as expected. However, an issue arose in which renders were not being returned. Restarting the kernel resolved this issue, shutting down any existing game frames and allowing the re-import of necessary components. The observation

space and action space were determined to be in good condition upon re-creating the environment. The pre-processed environment was also examined, revealing an ungrayscaled version being used with the base render class. However, it was noted that a higher number of rewards were being obtained as a result of scoring hits. As such, the score will increment accordingly.
It has been determined that the system is only able to see the pixel values that have undergone changes. As we continue to run the program, no changes have occurred upon initiation of the environment and taking steps. The movement of our agent was evidently visible, as well as the movement of the opponent. It can be observed that the system returns the changes in pixel values to the game. This is an advantage of using the frame delta method, as it does not pick up unnecessary information such as changes in the health bar. By noting the movement of the opponent, the agent can be trained on when and how to respond appropriately. It is important to validate that the agent is able to comprehend the information it is receiving. Through testing, the movement of pixels can be monitored. The question of whether or not changes without using frame delta would have yielded different results has been raised, as it has been previously experimented with. However, this is an important aspect of the process. Occasionally, frame changes may occur. In order to capture these changes, we implemented the process of stacking multiple frames. This will also involve processing the last four frames, allowing for the observation of movement in relation to movement, effectively tracking trajectories.

3.3 Proximal Policy Optimization

As we have discussed earlier that the four key elements of reinforcement learning are agent, reward, environment and action. The character in our game is the agent and it can take some actions such as moving forward or backward, jumping and throwing punches. Inside the game environment, it might get a reward depending on the results of the actions that it took. The model controlling our game character learns what actions to take inside the environment in order to maximize that reward. Since, both of our action space and observation is much more sophisticated now, we decided to use the Proximal Policy Optimization (PPO) algorithm to perform reinforcement learning. Table 1 lists different types of action spaces and observation spaces with which the PPO algorithm works.

Table 3.1: Significance of PPO algorithm under different types of action space and observation space

Space	**Action**	**Observation**
Discrete	Yes	Yes
Box	Yes	Yes
MultiDiscrete	Yes	Yes
MultiBinary	Yes	Yes
Dict	No	Yes

OpenAI's Proximal Policy Optimization (PPO) is a reinforcement learning algorithm that strives to find a compromise between ease of implementation, sample

efficiency, and tuning simplicity [6]. PPO employs a policy gradient method of learning, which occurs online and not via stored offline data like DQN. PPO does not have a mechanism to save past experiences like a replay buffer, instead it learns from its interactions with the environment directly. Policy gradient methods are typically not as efficient in terms of the amount of experience required as methods like Q-learning because they use the collected experience only once before discarding it for updating the model.
During the learning process, the agent creates and constantly evolves the observations and rewards used to teach the model in reinforcement learning. Here, a model is trained through the interactions of an agent with an environment, rather than using a fixed dataset in supervised learning. This can make the training process more unstable compared to supervised learning. Additionally, the success of reinforcement learning is more sensitive to the selection of hyperparameters such as the initialization of the model, because it has a high learning rate. This means that the policy network may collect data under a poor policy, leading to the possibility of failure to recover.
The OpenAI team created proximal policy optimization (PPO) to address the difficulties of reinforcement learning. PPO is a policy gradient method that aims to be easy to implement, efficient with sample usage, and simple to adjust. It learns directly from the environment rather than stored data like DQN. Therefore, PPO does not use a replay buffer and discards experience after it is used for a gradient update. Policy gradient methods like PPO are generally less efficient with sample usage compared to queue learning methods, which can reuse stored experience. In PPO, the policy gradient is often first established by taking the average of the log of the policy actions multiplied by a calculated approximation of the advantage function, as shown in equation 3.1.

$$\hat{E}_t[log\pi\theta(a_t|s_t)\hat{A}_t] \tag{3.1}$$

The policy $\pi\theta$ is a neural network that takes the observations of the states from the environment as input and outputs the suggested actions as output. The function $\hat{A}_t$ estimates the value of the selected action in the present state. In order to compute the advantage, we require a baseline estimation and the discounted total of rewards. The calculation of the advantage occurs after the series of episodes from the environment has been obtained, meaning that all rewards are already known and there is no need for estimation using the discount or return. The return $\sum_{k=0}^{\infty} \delta^k r_t + k$, commonly referred to as the discounted sum of rewards, it is the total of all rewards received by the agent for each time step of the current episode. The discount factor "gamma" assigns a weight to rewards, typically between 0.9 and 0.99. This factor takes into account that the agent places a higher value on rewards received at an earlier time compared to those received later.
The second part of the calculation of advantage is an estimate of the expected value, also known as the value function. The function predicts the overall rewards that will be received from the current point in the episode by factoring in the decrease of future rewards in the prediction of the total return in the episode, it is based on the present state. As the agent interacts with the environment, the neural network responsible for representing the value function is frequently refined through supervised learning, using the experience gained during training. We input the states, and the neural network attempts to forecast the total discounted rewards that will be obtained

starting from that particular state. However, the value estimate produced by the neural network will be inaccurate due to variance, as the network does not always predict the exact value of the state. Finding the deviation between the estimated baseline and the discounted rewards determines the advantage estimate $\hat{A}_t$. This advantage estimate measures the extent to which the action taken by the agent was better than the expected outcome given the state it was in. To achieve the final optimization goal of enhancing the policy, we take the product of the log probabilities of the policy's actions and the advantage function.
The goal of the objective function is to increase the likelihood of choosing actions that are predicted to lead to above-average returns in similar situations in the future, and decrease the probability of actions that are predicted to have below-average returns. However, using gradient descent on a single batch of collected experience may result in the network parameters being updated to a point where the noisy estimate of the advantage function becomes completely inaccurate. This can ruin the policy if gradient descent is continued to be used on a single batch of collected experience. To prevent the updated policy from deviating too far from the current policy when it is updated, the "Trust Region Policy Optimization" (TRPO) paper introduced adding a KL divergence constraint to the optimization objective [6], which served as the foundation for PPO. This constraint ensures that the updated policy stays in the region where it is known to work well. But the KL divergence constraint can make the optimization process more complex, and can sometimes cause undesired training outcomes. PPO addresses this issue by incorporating the constraint directly into the optimization objective as shown in eqauation 3.2,

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{3.2}$$

The probability ratio $r_t(\theta)$ is a measure of the likelihood of an action in the updated policy in comparison to the probability of the same action in the prior policy network. The value of $r_t(\theta)$ will be greater than 1 if the action is more likely according to the updated policy, and between 0 and 1 if it is less likely when evaluated on a series of actions and states. The objective of TRPO is obtained by multiplying the advantage function with $r_t(\theta)$. The central objective function used in PPO can be written as shown in equation 3.3,

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{3.3}$$

The PPO algorithm aims to optimize an objective function, which is the expectation operator, that is computed over a set of several sequences of states and actions. The operator is the least of two expressions. The primary expression, $r_t(\theta) \times \hat{A}_t$, is the standard objective for policy gradients and favors the selection of actions that result in a large positive advantage compared to the baseline. The second term is comparable to the first, but it incorporates a truncated form of the $r_t(\theta)$ ratio obtained through a clipping procedure between $1-\epsilon$ and $1+\epsilon$ (where ϵ is usually 0.2). We take the minimum of these two terms to obtain the final result.
The objective function in PPO is designed to handle both positive and negative values of the advantage estimate. The plot in figure 3.3 illustrates how the objective function behaves for different values of the advantage estimate.When the advantage function is positive on the left half of the diagram, the chosen action resulted in a better outcome than expected. On the right half of the diagram, the action had

an estimated detrimental effect on the outcome. Observe how the loss function flattens on the left side when 'r' becomes too large, this happens when the action is much more probable under the current policy as compared to the old policy. In this case, the objective function is truncated to prevent the action update from being overdone. When the action is believed to have a negative impact on the outcome, and the value of 'r' approaches zero, the objective flattens on the right side of the diagram. This indicates actions that are less frequently taken by the current policy compared to the previous one. It serves to prevent updates that would decrease the probability of taking these actions to zero.

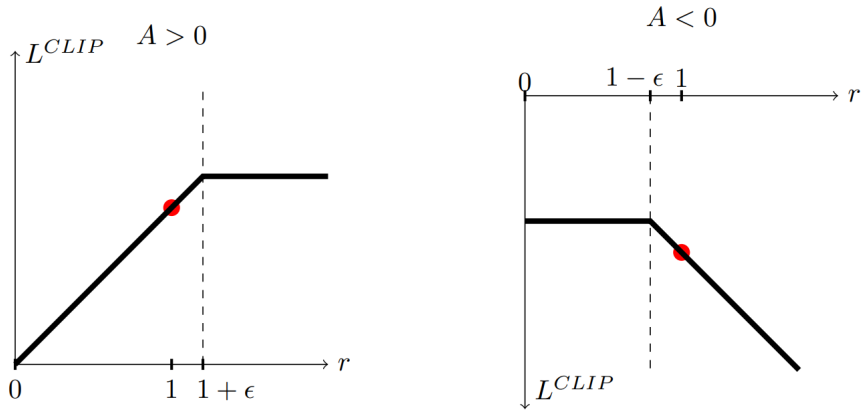


Figure 3.3: L^{CLIP} surrogate function plots and its effects on objectives and policy updates.[6]

The advantage function is prone to noise, so we want to avoid relying on a single estimate when determining our policy. The objective function on the right side of the plot will only be in the shown region if the last gradient step significantly increased the probability of the selected action (thus resulting in a large 'r') and also led to a worse policy due to a negative advantage estimate. In this case, the PPO objective function allows us to reverse the effects of the last gradient step. The function is negative, which means the gradient will indicate that we should decrease the probability of the action by an amount proportional to the negative action. The unmodified version of the objective function only has a smaller value when compared to the clipped version in this case, therefore it will be selected by the minimize operator.

Algorithm 1 PPO, Actor-Critic Style

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

Figure 3.4: Proximal Policy Optimization Algorithm [6]

PPO's objective function promotes conservative policy updates that do not deviate too far from the current policy. Unlike the objective function in TRPO, PPO's is simple and does not require calculating additional constraints or KL divergences. In fact, the simpler PPO objective function often outperforms the more complex TRPO version.
PPO utilizes two alternating processes. In the first process, the current policy is employed to interact with the environment, resulting in the creation of episode sequences. We determine the advantage of these sequences by calculating the fitted baseline estimate of the state values. In the second process, all of the gathered experience is utilized to perform gradient descent on the policy network through the use of the PPO objective, which is stabilized through the application of clipping.

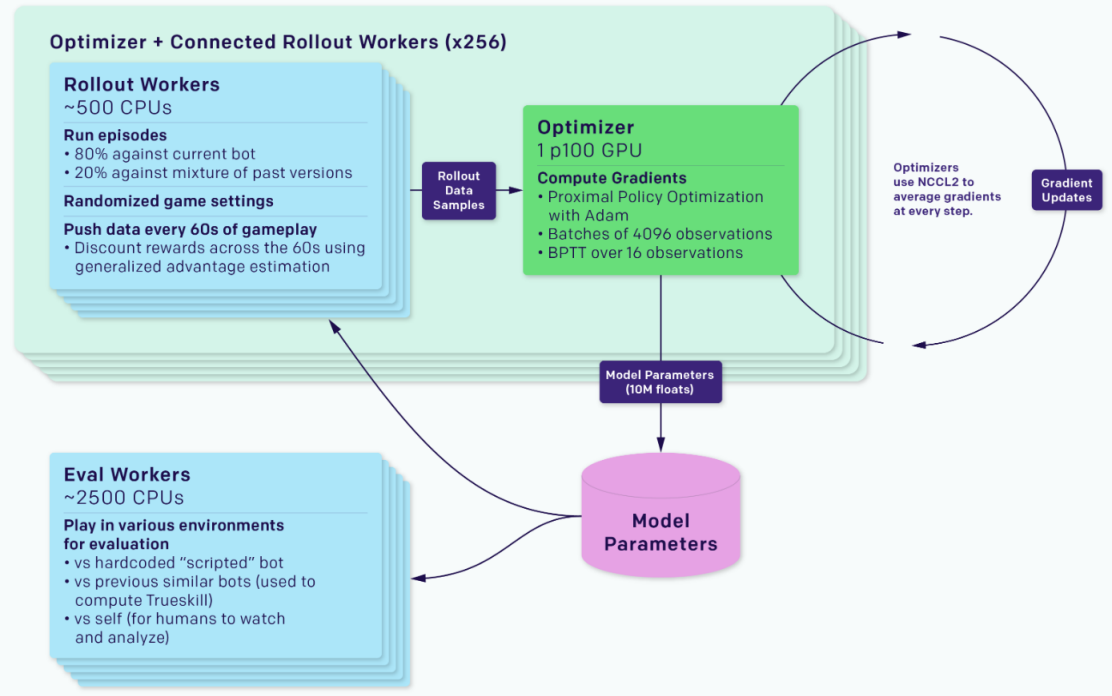


Figure 3.5: Workers gather game experience, optimize, evaluate agents, and monitor through the system. [11]

By using PPO and separating it into two different threads, the agent in the OpenAI 5 system was trained effectively [11]. While thousands of remote workers utilized a recent copy of the policy network and a GPU cluster to interact with the environment, the gradient descent on the network weights was performed using the collected experience from these workers. It was necessary for the workers to regularly update their local versions of the policy network to ensure that they were using the most current version. The final loss function used in the training of the agent was a combination of the clipped PPO objective mentioned earlier and two additional terms, as shown in equation 3.4,

$$L_t^{PPO}(\theta) = \hat{E}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \tag{3.4}$$

The loss function also includes a first additional term that updates the baseline network, which estimates the expected reward that can be obtained from a specific state. The value and policy outputs are separate components of the same network,

yet they are integrated into the same computation process and can be combined in a single loss function. The network for estimating value and the network for selecting the optimal action have a number of shared parameters because both require similar feature extractions from the current state observation. These common parts of the network are included in the same objective function.
The entropy term is included in the objective function to encourage the agent to explore during training. The policy head of PPO produces parameters for a Gaussian distribution for each action type, rather than outputting probabilities for different actions like a discrete action policy. During training, the policy employs these distributions to generate a continuous value for each action. Entropy reflects the unpredictability of the outcome of a random variable and determined by its probability distribution. It represents the average number of bits needed to encode the outcome of the variable. By maximizing entropy, the variable will have a wide range of possible values, leading to the most uncertain outcome. This is why the entropy term causes the policy to behave randomly until the other terms in the objective function become more dominant.
PPO has become a popular algorithm in deep reinforcement learning due to its simplicity in implementation and tuning, as well as its ability to perform at or above the current state-of-the-art on a variety of tasks. The hyperparameters c1 and c2 are used to balance the different parts of the loss function. PPO was not created with an emphasis on sample efficiency, but rather to address the complexity and user-friendliness issues present in many other algorithms. It retains the stability and reliability of TRPO while being easier to implement, requiring only minor changes to basic policy gradient methods. To summarize, PPO is a reliable and easy-to-use policy gradient technique that can be applied to numerous reinforcement learning tasks.

Chapter 4

Implementation and Results

4.1 Hyperparameter Tuning

In this step, we began training our model. Instead of training it directly with high parameters, we performed Hyperparameter Tuning. Hyperparameter tuning is crucial in reinforcement learning as the algorithms can be affected by various hyperparameters [17]. To accomplish this, we needed three libraries: PyTorch, the deep learning framework that Stable Baselines runs on; Stable Baselines, the package/library where most of our reinforcement learning will take place; and Optuna, an optimization framework that can be applied to many algorithms, which makes it convenient to tune hyperparameters. For Proximal Policy Optimization (PPO), the parameters we adjusted are n_steps, gamma, learning_rate, clip_range, and gae_lambda.
We then saved each model and reloaded the best one for fine tuning later on. Moreover, we performed hyperparameter tuning using the PPO algorithm where we set up 10 trials. We trained for 10000 n_steps in each trial.
Table 4.1 shows the results of the best performing parameters, while Table 4.2 shows the results of the worst performing parameters.

Table 4.1: Result of the best performing parameters

ep_len_mean	**5000**
ep_rew_mean	**7200**
entropy_loss	-5.62
approx_kl	0.14903
clip_fraction	0.31400
clip_range	0.31900
learning_rate	0.00005

Table 4.2: Result of the worst performing parameters

ep_len_mean	**2346**
ep_rew_mean	**2000**
entropy_loss	-2.36
approx_kl	0.15423
clip_fraction	0.57642
clip_range	0.88794
learning_rate	0.00231

4.2 Fine Tuning

After completing the hyperparameter tuning process, we reloaded the weights from the Optuna trial that performed the best and continued training. To do this, we first set up a callback. We used a key dependency called Base Callback for this. This step basically logs the entire saved model at a certain check frequency to a specific log path. This allows us to automatically save the reinforcement learning model as we go through training. This also means that if anything goes wrong, we now have different versions of the model saved at different stages of training. Finally, we trained the best model for an additional 100000 n_steps. The results of the model after training are shown in Table 4.3.

Table 4.3: Results after training for 100000 n_steps

ep_len_mean	**11000**
ep_rew_mean	**18500**
entropy_loss	-9.61
approx_kl	0.55383
clip_fraction	0.59872
clip_range	0.83451
learning_rate	0.0000202

4.3 Result and Analysis

As shown in Figure 4.1, we selected a combination of high-performing models from the Hyperparameter Tuning phase, including our best-performing model. The graph illustrates a marked decrease in mean episode length and reward. While this pattern is also evident in our model, the results begin to improve at around 68,000 n_steps. This suggests that our model exhibits a similar trend, but eventually outperforms the other models in terms of performance metrics.

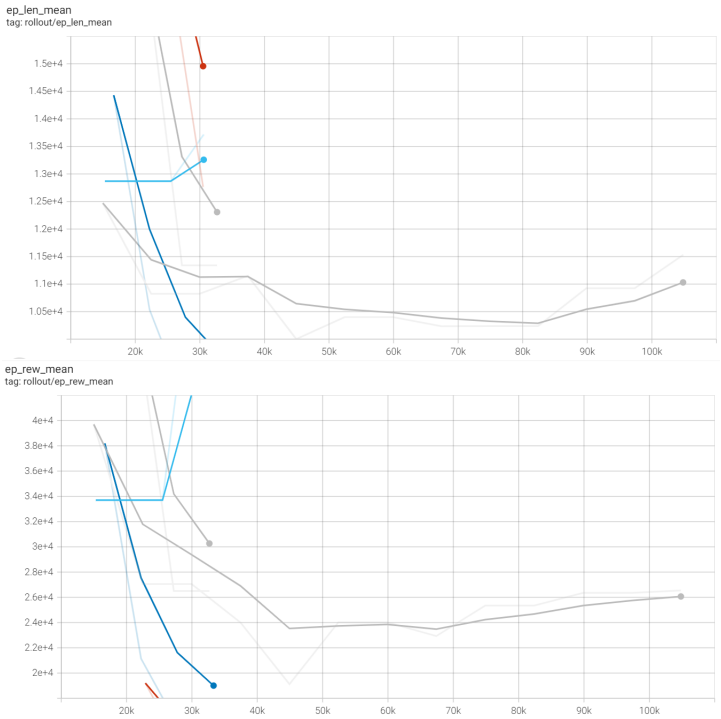


Figure 4.1: Graphical representation of episode length mean and episode reward mean

Figure 4.2 illustrates the performance of the clip fraction parameter during the training phase. The clip fraction of our top performing model was relatively stable throughout training, indicating consistent and stable learning.

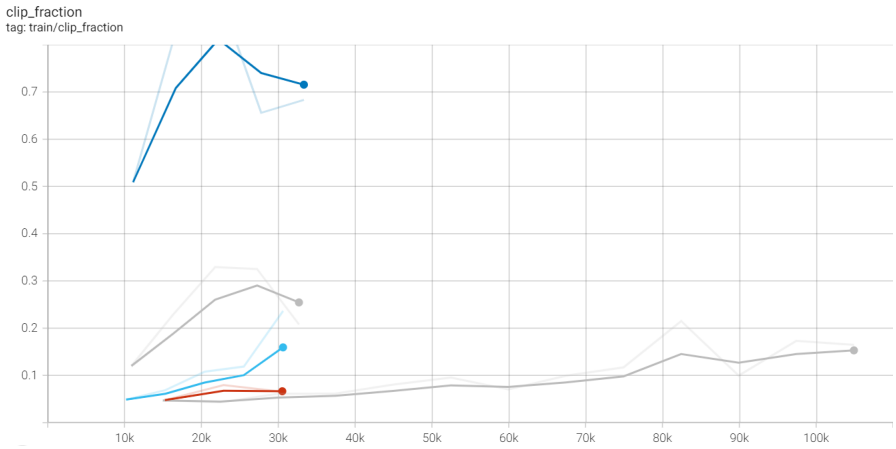


Figure 4.2: Performance of the clip fraction parameter

Additionally, figure 4.3 shows that the entropy loss was minimal during the training period, suggesting an increase in information gain.

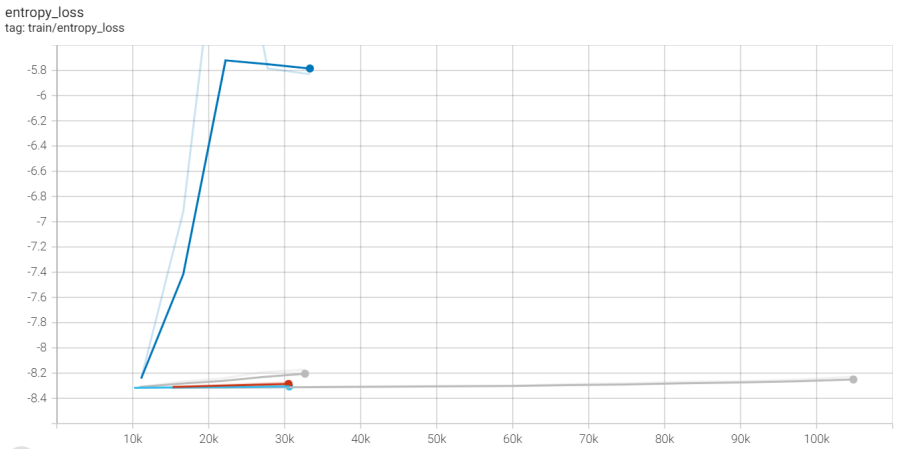


Figure 4.3: Entropy loss during training

Moreover, the value loss for our best-performing model also increased significantly at 75,000 n_steps of training, likely due to the exponential increase in reward. However, once the reward leveled off, the value loss decreased accordingly as shown in figure 4.4.

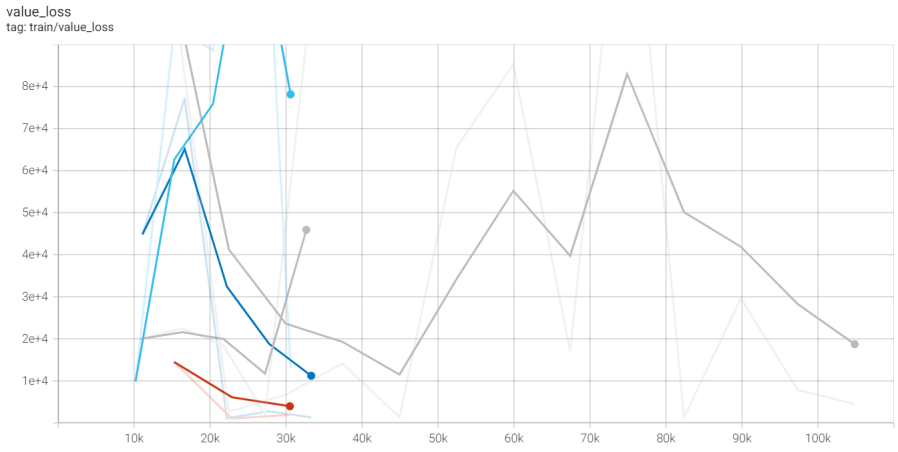


Figure 4.4: The value loss of the model

Also, figure 4.5 shows that the explained variance of our fine tuned model has been more consistent compared to other hyperparameter tuned models.Despite deviating towards the negative axis at around 82,000 steps, it successfully returned to its neutral position. This confirms that the predictions made in the actor-critic network of the proximal policy optimization algorithm were relatively accurate.

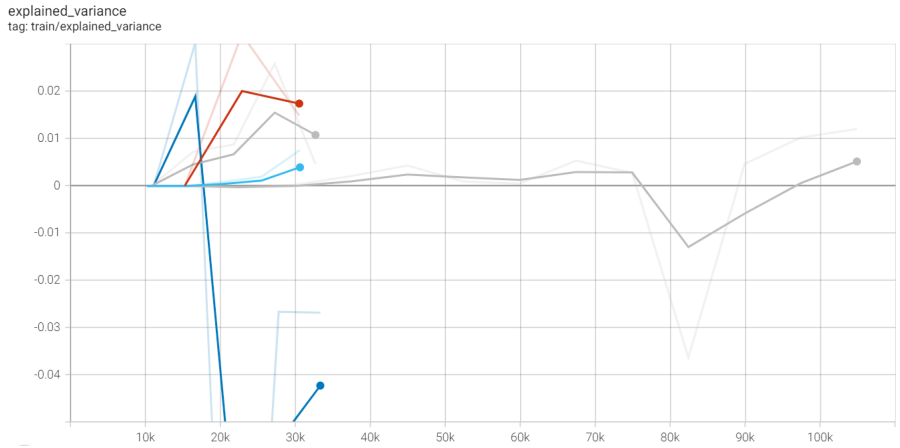


Figure 4.5: Explained variance of the model during training

Furthermore, as illustrated in Figure 4.6, the approx_kl remained close to 0 during the entire training phase. This indicates that the KL divergence constraint effectively ensured that the updated policy remained within the region where it was known to perform optimally. This suggests that the training was able to maintain a balance between exploration and exploitation which is crucial for the policy optimization process.

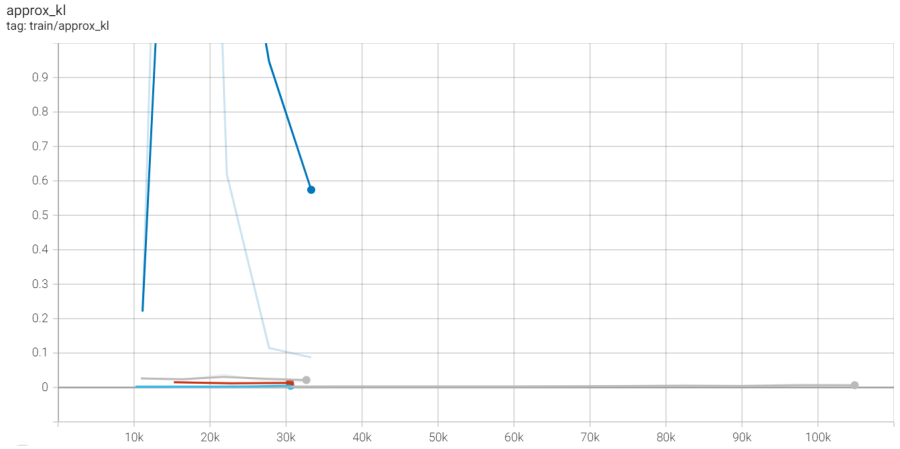


Figure 4.6: Approximate KL divergence of the model during training

In the evaluation phase, we found that the agent we trained was successful in defeating three opponents in individual multiplayer games over the course of three different episodes. However, in the fourth episode, the opponent was an inanimate object and the goal was to destroy it within a set time limit. Our agent was unable to achieve this objective because it was only trained in an environment where there was no time limit and the opponents were actively attacking.

Chapter 5

Conclusion

In conclusion, reinforcement learning for game AI is a complex area. Recent advancements in the field of Artificial Intelligence have led to significant growth and development in the research. The focus of this paper was to demonstrate the ability of a character in an arcade game, and we believe similar techniques can be applied to other scenarios and tasks, leading to more successful outcomes. As we have seen in the Literature Review chapter, various DRL algorithms have been used in a number of successful video games, from single-agent, 2D perfect data to multi-agent, 3D imperfect data, and have achieved human-level performance.Despite the progress, there are still some major obstacles in utilizing DRL techniques in this area, particularly when it comes to multi-agent video games that involve 3D imperfect data. Creating a high-level game AI requires the development of new and improved DRL methods that can be applied to challenging and complex environments. Despite the obstacles, there are many opportunities for further research and development in this field. Applications such as robotics and autonomous driving use raw sensory inputs in their domains, and these data can be used to enhance the learning of behaviors using reinforcement learning. There are still a number of challenges that have not been thoroughly investigated in this area, and it could be a promising direction to pursue further research in the future.

The findings of our study revealed that our agent was unable to defeat an opponent who possessed the same character and abilities. During the training phase, the underlying reinforcement learning model may have difficulty distinguishing between characters because the game was presented in a gray-scaled format, causing this outcome. Despite this setback, the results obtained were considered satisfactory. However, it is worth mentioning that during the training process, some limitations were identified in the gym retro environment which was built some time ago. One of these limitations is the absence of the game integration feature, which allows for continuing from a specific game state or level. In light of these findings, we plan to make significant improvements to the model by fine-tuning it on each individual level, utilizing a technique similar to curriculum learning in our future work.

Bibliography

[1] E. Tzorakoleftherakis, "Three things to know about reinforcement learning," 2011. [Online]. Available: https://www.kdnuggets.com/2019/10/mathworks-reinforcement-learning.html.

[2] R. S. Sutton and A. G. Barto, "Reinforcement learning - an introduction," *The MIT Press*, 2015. [Online]. Available: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

[3] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor, "A deep hierarchical approach to lifelong learning in minecraft," *CoRR*, vol. abs/1604.07255, 2016. arXiv: 1604.07255. [Online]. Available: http://arxiv.org/abs/1604.07255.

[4] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala, "Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks," *CoRR*, vol. abs/1609.02993, 2016. arXiv: 1609.02993. [Online]. Available: http://arxiv.org/abs/1609.02993.

[5] P. Peng, Q. Yuan, Y. Wen, *et al.*, "Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games," *CoRR*, vol. abs/1703.10069, 2017. arXiv: 1703.10069. [Online]. Available: http://arxiv.org/abs/1703.10069.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. arXiv: 1707.06347. [Online]. Available: http://arxiv.org/abs/1707.06347.

[7] J. Achiam, "Openai spinning up," 2018. [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/ppo.html.

[8] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *CoRR*, vol. abs/1811.12560, 2018. arXiv: 1811.12560. [Online]. Available: http://arxiv.org/abs/1811.12560.

[9] D. J. Mankowitz, A. Zıdek, A. Barreto, *et al.*, "Unicorn: Continual learning with a universal, off-policy agent," *CoRR*, vol. abs/1802.08294, 2018. arXiv: 1802.08294. [Online]. Available: http://arxiv.org/abs/1802.08294.

[10] T. N. Minh, M. Sinn, H. T. Lam, and M. Wistuba, "Automated image data preprocessing with deep reinforcement learning," *CoRR*, vol. abs/1806.05886, 2018. arXiv: 1806.05886. [Online]. Available: http://arxiv.org/abs/1806.05886.

[11] M. Petrov, D. Farhi, J. Raiman, *et al.*, "Openai five," 2018. [Online]. Available: https://openai.com/blog/openai-five/.

[12] Y. Xiong, H. Chen, M. Zhao, and B. An, "Hogrider: Champion agent of microsoft malmo collaborative ai challenge," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. DOI: 10.1609/aaai.v32i1.11581. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11581.

[13] C. Berner, G. Brockman, B. Chan, *et al.*, "Dota 2 with large scale deep reinforcement learning," *CoRR*, vol. abs/1912.06680, 2019. arXiv: 1912.06680. [Online]. Available: http://arxiv.org/abs/1912.06680.

[14] M. Naeem, S. Rizvi, and A. Coronato, "A gentle introduction to reinforcement learning and its application in different fields," *IEEE Access*, vol. 8, pp. 209 320–209 344, Jan. 2020. DOI: 10.1109/ACCESS.2020.3038605.

[15] S. Gossett, "7 challenges in reinforcement learning — and how researchers are responding," 2021. [Online]. Available: https://builtin.com/machine-learning/reinforcement-learning.

[16] A. Gopani, "What are the main challenges of reinforcement learning, and how to overcome them?," 2022. [Online]. Available: https://analyticsindiamag.com/what-are-the-main-challenges-of-reinforcement-learning-and-how-to-overcome-them/.

[17] S. Shekhar, A. Bansode, and A. Salim, "A comparative study of hyper-parameter optimization tools," *CoRR*, vol. abs/2201.06433, 2022. arXiv: 2201.06433. [Online]. Available: https://arxiv.org/abs/2201.06433.