An Exploration of Feature Engineering For Super Mario Agent

Lingyun Xie JiaRong Wu

ly-xie22@tsinghua.mails.edu.cn jr-wu22@tsinghua.mails.edu.cn

***Abstract*— Deep reinforcement learning has been successfully applied to a wide range of tasks, including playing Atari games, chess, and go. In this paper, we extend the use of deep reinforcement learning to a classic platformer game: Super Mario Bros. Our approach uses a convolutional neural network to process game screen pixels and predict the optimal action at each time step. Our results demonstrate the potential of deep reinforcement learning to learn complex tasks from raw sensory input and provide a new approach for playing and potentially even designing platformer games.**

***Index Terms*— Reinforcement learning, Super Mario games, Game AI, Value function estimation, Policy iteration, Experience replay, Sample efficiency, Exploration strategies, 2D Game, Platformer, Game Physics, Mario AI, Mario Reinforcement Learning**

I. Introduction

Super Mario Bros. is a beloved video game that has been enjoyed by players of all ages. In this paper, we present a deep reinforcement learning approach for training an intelligent agent to play Super Mario Bros. Reinforcement learning is a type of machine learning that focuses on teaching agents to take actions in an environment that maximize their reward.

Reinforcement learning (RL) is a type of machine learning in which an agent learns to interact with an environment in order to maximize a reward. In RL, the agent receives a reward for performing certain actions in the environment, and the goal is to learn a policy that will maximize the cumulative reward over time.

As the general framework of the program has been given, our current one is mainly designed for observation space, action space, and reward space, so that the intelligence can pass).

II. Reinforcement Learning Basics

Part 1:

Reinforcement learning is a method of machine learning that aims to teach intelligent agents to perform certain actions in interactive environments in order to maximize rewards. The basic idea of reinforcement learning is to allow the intelligent agent to learn how to maximize future rewards while exploring and developing new strategies.

Deep learning is a branch of machine learning that uses multi-layered neural networks to model data. Deep learning can be used for many different machines learning tasks, including classification, regression, natural language processing, and image processing. Deep learning is an important tool for reinforcement learning because it can help agents make decisions in the environment.

In summary, reinforcement learning and machine learning are different but related. Machine learning is a general method for solving a variety of different problems. On the other hand, reinforcement learning is a branch of machine learning specifically designed to solve interactive problems, such as teaching an intelligent agent how to win a game. Deep learning is a tool for machine learning that can help us solve various machine learning tasks, including reinforcement learning.

Part 2:

1. Agent: In reinforcement learning, the agent is the subject that is being learned. It can be a robot, a software program, a human, or any other form of entity.

2. Reward: A reward is the value obtained by an intelligent agent after performing a certain action. In reinforcement learning, the goal of the agent is to maximize rewards by performing valuable actions.

3. Return: Return is the total accumulated reward obtained by an intelligent agent in the future. In reinforcement learning, the agent's goal is to maximize the return.

4. State transition function: The state transition function describes the change in state that occurs when an action is taken in a given state. In reinforcement learning, the state transition function is used to predict the future state based on the current state and the chosen action.

5. Markov decision process (MDP): An MDP is a mathematical framework for modeling decision-making in situations where the future is uncertain. In reinforcement learning, MDPs are used to represent

the environment in which the agent is operating.

6. Value function: The value function is a measure of the expected future return of a given state or action. In reinforcement learning, the value function is used to determine the optimal action to take in a given state.

7. Policy: A policy is a rule or set of rules that an agent follows to make decisions in a given environment. In reinforcement learning, the policy is used to determine the action that the agent should take in a given state.

8. Temporal difference (TD) algorithm: The TD algorithm is a method for learning value functions in reinforcement learning. It is based on the idea of using the difference between the expected return and the actual return to update the value function.

9. Observation space: The observation space is the set of all possible states that an agent can observe in the environment.

10. Action space: The action space is the set of all possible actions that an agent can take in the environment.

11. Reward space: The reward space is the set of all possible rewards that an agent can receive in the environment.

III. Literature Review

Part A:

1.MinAtar: An Atari-Inspired Testbed for Thorough and Reproducible Reinforcement Learning Experiments:

This paper provides a scalable, reproducible, and comprehensive testing environment for reinforcement learning research, which is important for more accurately evaluating the performance of reinforcement learning algorithms.

Key idea: Use the MinAtar testing platform to evaluate the performance of reinforcement learning algorithms.

Results: The experiment results show that using the MinAtar test platform allows for a comprehensive and reproducible evaluation of the performance of reinforcement learning algorithms.

2.Playing FPS Games with Deep Reinforcement Learning:

This article proposes a method of playing shooting games using deep reinforcement learning, and utilizes the DQN algorithm, memory and target network to improve learning efficiency, providing a new approach for the research of game reinforcement learning.

Key idea: Use deep reinforcement learning methods to play shooting games and improve learning efficiency using the DQN algorithm, memory, and target networks.

Result: The experiment results show that the proposed method using deep reinforcement learning is able to achieve high scores in the game and reach human-level performance.

3.Action Space Shaping in Deep Reinforcement Learning:

This article proposes a method of using adaptive policy gradient algorithms to adjust the action space, providing a new solution for the research of reinforcement learning that can effectively improve the learning speed and final performance.

Key idea: Use adaptive policy gradient algorithms to adjust the action space to improve the efficiency of reinforcement learning.

Result: The experiment results show that the proposed method of using an adaptive policy gradient algorithm to adjust the action space is effective in improving learning speed and ultimately performance.

4.Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning.:

This paper presents a method of using deep reinforcement learning to achieve human-level control, providing a new approach for the research of reinforcement learning that can achieve high performance in complex environments.

Key idea: Use deep reinforcement learning methods to achieve human-level control and use the DQN algorithm to learn the game.

Result: The experiment results show that the proposed method using deep reinforcement learning is able to achieve human-level performance in Atari 2600 games.

Algorithm Summary:
DQN algorithm:

The DQN algorithm is a method of reinforcement learning using a deep neural network. It stabilizes the convergence of Q values by using a memory buffer and a target network, and can learn in high-dimensional state spaces. The advantage of the DQN algorithm is its ability to achieve high performance in complex environments, but its disadvantage is the need for a large amount of training data and computational resources.

Adaptive Policy Gradient algorithm:

The Adaptive Policy Gradient algorithm is a method of updating a policy using gradient information. It is able to automatically adjust the action space, thus improving the efficiency of reinforcement learning. The advantage of the Adaptive Policy Gradient algorithm is its ability to effectively improve the learning speed and final performance, but it may suffer from overfitting as a disadvantage.

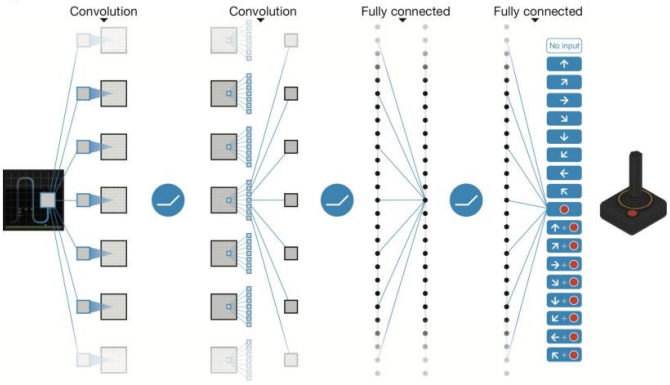
Part B:

1. Pseudo code for DQN

1. Collect training sample group (s,a,r,s')
2. Initialize the Q function
3. Calculate the TD target
4. Calculate the loss function
5. Calculate the maximum value action
6. Update the Q parameters
7. Target Q network is the same as the regular Q network

2. DQN network architecture

Below is the network architecture of a DQN network in figure 1.



(Figure 1)

3.Benefits of using a neural network to replace the Q-table in DQN compared to Q-Learning:

A neural network can learn complex features, while a Q-table can only store discrete state-action pairs. A neural network can handle high-dimensional input, while a Q-table can only handle low-dimensional input. A neural network can adapt to new data, while a Q-table needs to recalculate all state-action pairs.

4.Benefits of using a Target Network:

It can reduce variance in Q-values, improving learning efficiency. It can stabilize convergence of Q-values, improving stability of learning.

5.Benefits of using a Replay Buffer:

It can cache past experiences, improving learning efficiency. It can randomly sample from experiences, reducing the impact of high frequency sampling on training. Operations for adding, deleting, modifying, and accessing elements in the Replay Buffer can be implemented using the “collections.deque” class in Python. The Buffer Size of the Replay Buffer should be determined based on the specific situation. Generally, a larger Buffer Size will improve learning efficiency but also increase storage and computational overhead. Increasing the Buffer Size may lead to excessive memory usage, while decreasing the Buffer Size may result in decreased learning efficiency.

6.Operations for processing observation, reward, and action spaces in the given papers:

The MinAtar test platform uses operations such as image normalization, cropping, and flipping to process the observation space. The "Playing FPS Games with Deep Reinforcement Learning" paper uses action normalization and reward normalization to process the action and reward spaces. The "Action Space Shaping in Deep Reinforcement Learning" paper uses action space shaping to improve the performance of the agent. The "MinAtar: An Atari-Inspired Testbed for Thorough and Reproducible Reinforcement Learning Experiments" paper uses the Arcade Learning Environment to provide a standardized environment for evaluating different reinforcement learning algorithms.

IV. Experiment

A. *3.1 Background*

We tried to pass stage 1-1 using a random action smart body, but it was hard to get a satisfactory result. Subsequently, We manually use keyboard to pass the stage 1-1. During the game, we mainly focus on avoiding monsters and avoiding holes when making action decisions, and then we can pass the stage 1-1.

When we training the agent, we mainly focus on the three main components:

- Observation Space: The collection of each frame of the game, or multiple frames overlapping.
- Action Space: The collection of actions that an agent can take. Mario can take actions including: up and down, left and right, jumping, firing bullets, and combinations of these actions.
- Reward Space: The reward space mainly includes rewards for killing monsters, rewards for getting gold, and rewards for passing the stage

B. *3.2 Baseline*

3.2.1 Training

By reviewing the official PyTorch documentation, as well as the project's Github documentation, we successfully run the baseline model.

The parameters of the model are:

- -s 1 -v 0 -a 7 -o 1

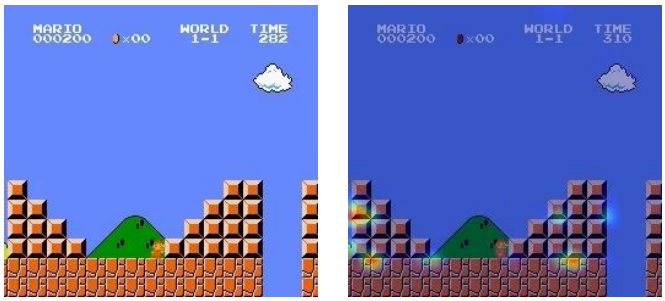
S is the random seed

V is the version of game

A is the number of actions that the agent will take

O is the number of frames that will be overlapped during training

But unfortunately, the baseline model does not pass the stage 1-1. The agent stuck in front of a step and no longer moved as shown in figure 2.



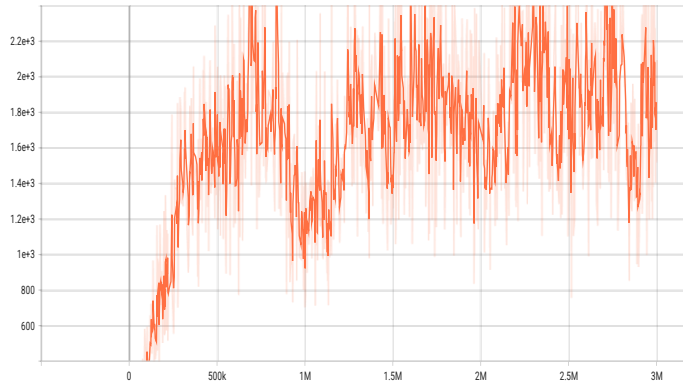
(Figure 2)

As we can see from figure 1, the agent stopped in front of the step and, according to the CAM image, there is very little reward in this frame, and much of it comes from meaningless places.

3.2.1 Analysis of training and test results

The following figure shows the loss function of the training process after smoothing. We learn that the loss is in the interval of 1200~2200, But it’s very unstable and does not converge

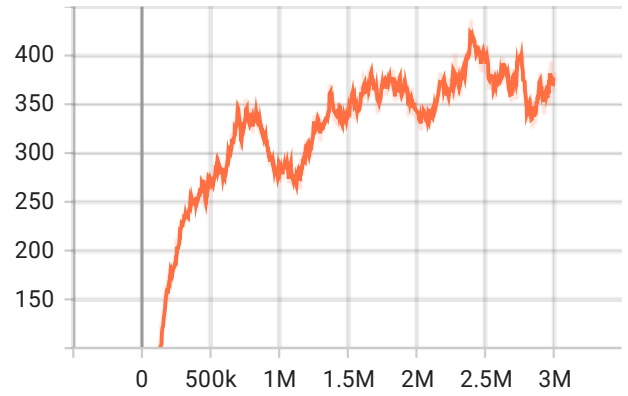
Loss Curve



(Figure 3)

By observing the Q value during the training process in figure 4, we can learn that although it is still rising, the speed is slowing down and the value seems to be coming to a peak of about 400.

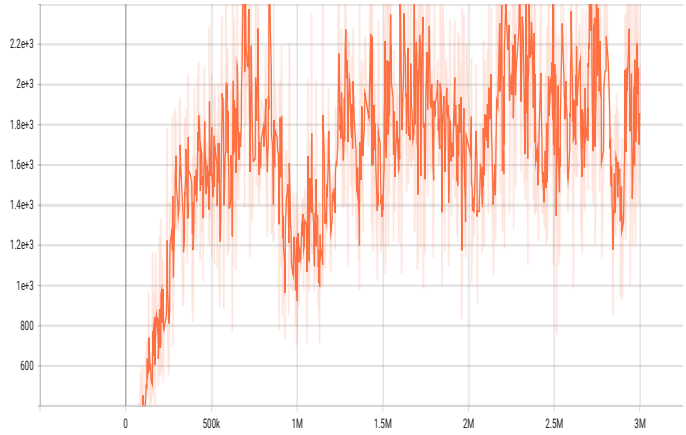
Q Value



(Figure 4)

The following figure 5 is the average reward during the test, we can see that even the maximum reward is only about 1800, and we need the reward to reach about 3000 to pass the stage 1-1. It can be said to be a far cry from our goal.

Reward Mean



(Figure 5)

C. Feature Engineering

3.3.1 Improved Version One

Problems of baseline model:

1. The baseline model uses the original version of the Mario game. But in this version of the game, there are many aesthetic elements. These elements are just to make the game look more beautiful, and do not provide any positive feedback on the movement decision
2. The baseline model includes seven movements in the action space, which seems meaningless in the stage 1-1. We don't need that many movements to pass stage 1-1, and there are even some moves that are ineffective in stage 1-1. And these redundant movements will only slow down the training processing.
3. The observation space contains only the current game frame, but sometimes the current frame will not contain enough information to help the intelligence make decisions, which can cause the agent to get stuck somewhere and stop moving

In summary, we have modified the model with the following parameters:

We run the model with parameters:

- -s 1 -v 2 -a 4 -o 4

A 4 includes the actions of 'Right', 'Left', 'Right Jump', 'Left Jump'

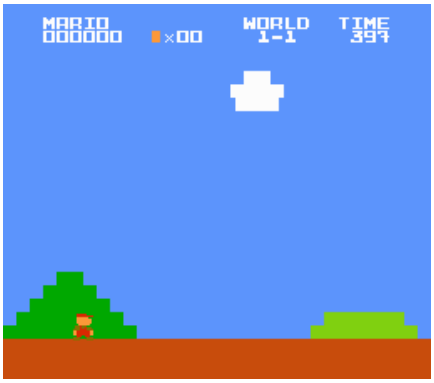
O 4 means overlapping four frames

V 2 is the pixel version of Super Mario

In the action space, we only included four movements, which are sufficient to pass the stage 1-1 and can accelerate the training speed.

In observation space, we overlapped four frames to allow the agent to learn more information.

We also use the pixel version of Mario shown in figure 6, which omits a lot of artwork details. It allows us to speed up the training process.



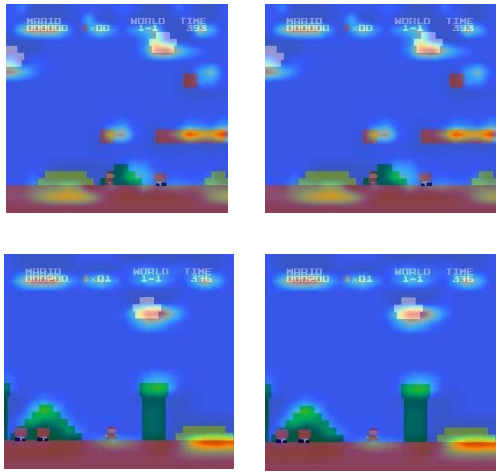
(Figure 6)

3.3.2 Analysis of Version One Result

After 3M iteration, out agent still didn't make it through and its finally jumped into the hole.

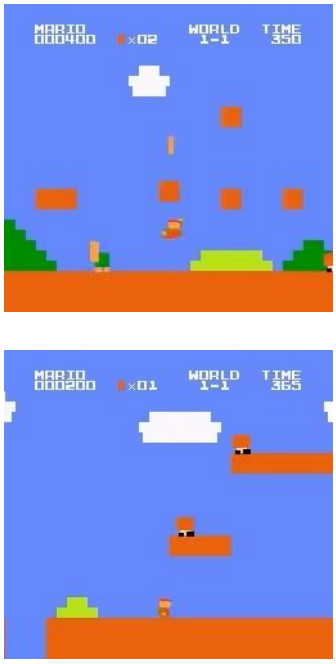
This time we see:

- In figure 7, many clouds and irrelevant things give a strong incentive for the agent. However, these things are not noticed during the process of playing the game by human being.



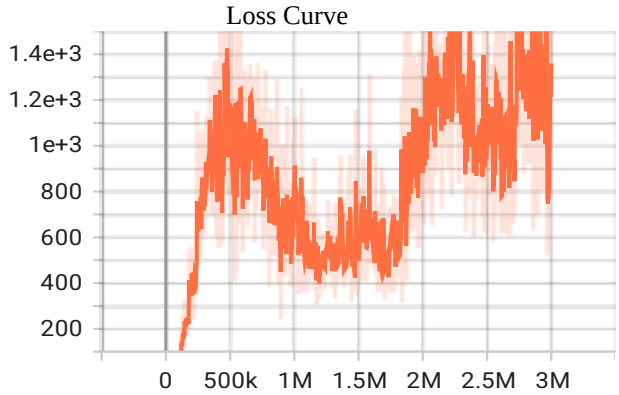
(Figure 7)

- The effective action of the agent is only moving to the right and jumping, thus there is no need for left movements as shown in figure 8. So it seems that two steps are enough to make the agent pass the stage 1-1



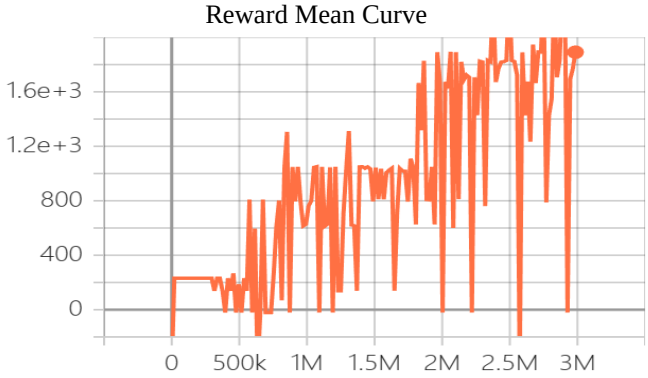
(Figure 8)

- From the loss curve in figure 9, we can see that the loss is smaller and more stable than the baseline model



(Figure 9)

- At the same time, we can see that the reward value shown in figure 10 is also higher than the baseline model, reaching about 1800, which is still some distance away from our goal, but at least it can prove that there is no problem with our improvement direction



(Figure 10)

3.3.3 Improved Version Two

From the above analysis we can see that our improvement is in the right direction.

However, the pixel version of Mario still has some, such as clouds, grass and other irrelevant elements for decision making, so we will choose a more concise version. Also, since in the last version there were actually no leftward and left-upward actions, we will continue to simplify the action space.

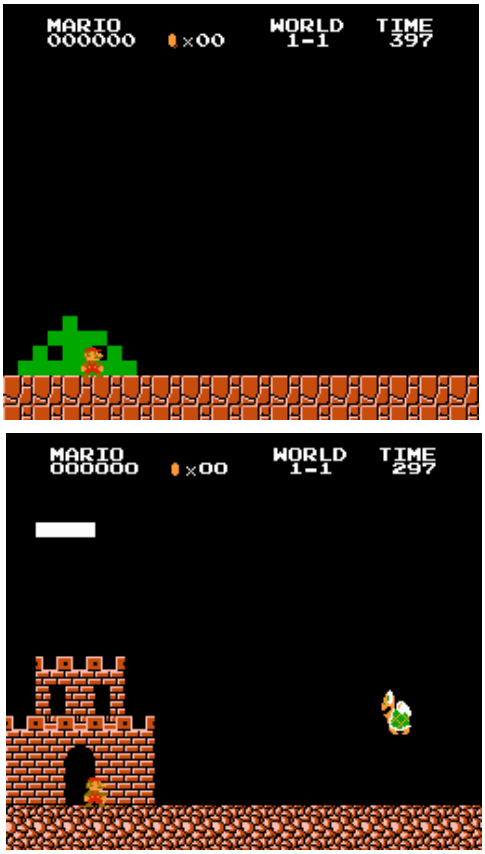
We run the model with parameters:

- -s 1 -v 1 -a 2 -o 8

A 2 includes the actions of ‘Right’ and ‘Right Jump’

O 8 means overlapping four frames

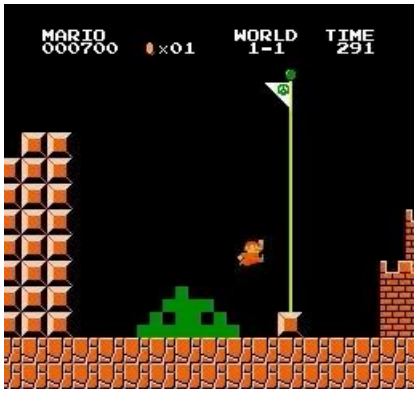
V 1 is the down sample version of Super Mario



(Figure 11)

For figure 11, this version of the game does not have irrelevant things such as clouds, which allows the intelligences to focus their choices more on what is useful and speeds up the convergence process

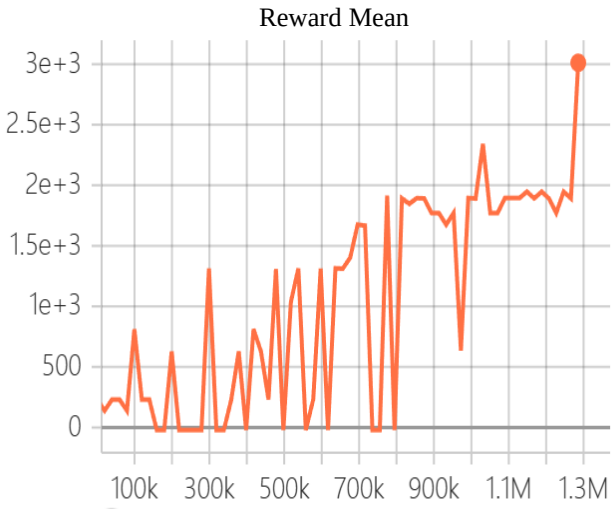
After 1.3M steps, we finally got a model that could pass the game shown in figure 12



(Figure 12)

3.3.4 Analysis of Version Two Result

- In the figure 13, the model converged relatively quickly, taking less than one-half of the 3M step to complete the training and achieving a reward of over 3000.



(Figure 13)

- The movement of the agent is very smooth, it takes 100 seconds to pass the stage 1-1.

V. Conclusion

As the experiment comes to an end, we were able to we learned the basic concepts of reinforcement learning and actually trained an agent that could pass the test. There are still many variables we should tackle so we can produce a more valuable results and create few more variety of approaches to the reinforcement training. For example, we can add coin rewards to make the agent eat as many coins as possible. We could change the seed value to perform the experiment. We can also try to train the agent to pass other stages with jump or shot actions so it won't only be 1-1. Overall, it is very interesting to be able to train Mario to pass the level on its own, once we were able to enhance our equipment to decrease the amount of time to train Mario, we should be able to learn more about reinforcement training by training Mario or other games to compete a more complex objective.

IV. Participation

	谢灵运	吴家荣
Working Code	1/2	1/2
Report	1/2	1/2
Poster	1/2	1/2

(Figure 14)

All of our assignments are split evenly so we are able to have a fair working environment as shown in figure 14.

REFERENCES

1. Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236
2. Kanervisto, A., Scheller, C., & Hautamäki, V. (2020, April 2). Action space shaping in deep reinforcement learning. arXiv.Org. https://arxiv.org/abs/2004.00980
3. Lample, G., & Chaplot, D. S. (2016, September 18). Playing FPS games with deep reinforcement learning. arXiv.Org. https://arxiv.org/abs/1609.05521
4. Young, K., & Tian, T. (2019, March 7). MinAtar: An Atari-inspired testbed for thorough and reproducible reinforcement learning experiments https://arxiv.org/abs/1903.03176