AIIR Project - AI Mario

By Blake Muchmore, Dominic Manno, Matthew Georgievski, Ryan Kim

# 1. Mario Environment

We use an OpenAI Gym environment that utilises different levels from the game Super Mario Bros. & Super Mario Bros. 2 (Lost Levels) (https://pypi.org/project/gym-super-mario-bros/).

The goal of this project is to complete Mario levels as fast as possible with a few custom additional reward conditions such as coin collection and score increase. Episodes end when Mario reaches the end of the level, if Mario dies, or if a certain time has elapsed.

# 

## 1.1 Action Space

As the Gym is built upon the nes-py emulator, the default NES action space has a full 256 possible discrete actions. This is constrained down into three action lists (RIGHT\_ONLY, SIMPLE\_MOVEMENT, and COMPLEX\_MOVEMENT) that dictates the complexity of the actions used in the training environment. We have decided to utilise COMPLEX\_MOVEMENT for our Mario Environment to realistically replicate valid inputs done by a NES controller. Having more complex sequences of actions also allows us to train Mario to interact with the environment more dynamically.

### 

### Possible Actions:

* 0: No Movement
* 1: Move Right
* 2: Move Right + Jump
* 3: Move Right + Sprint
* 4: Move Right + Jump + Sprint
* 5: Jump
* 6: Move Left
* 7: Move Left + Jump
* 8: Move Left + Sprint
* 9: Move Left + Jump + Sprint
* 10: Down
* 11: Up

## 1.2 Observation Space

The info dictionary returned by the step method contains the following keys:

|  |  |  |
| --- | --- | --- |
| Key | Unit | Description |
| coins | int | Number of collected coins |
| flag\_get | bool | True if Mario reached a flag |
| life | int | Number of lives left |
| score | int | Cumulative in-game score |
| stage | int | Current stage |
| status | str | Mario's status/power |
| time | int | Time left on the clock |
| world | int | Current world |
| x\_pos | int | Mario's x position in the stage |
| y\_pos | int | Mario's y position in the stage |

## 1.3 Rewards Space

### Base Reward Function

The innate gym reward function is designed to encourage the agent to move as far right as possible (increasing displacement x of the agent's starting position), speedrun to the end as fast as possible and all while avoiding death.

The base reward function is composed of three variables:

v (Velocity): measures the change in the agent's x-position between states

* Moving right: v > 0
* Moving left: v < 0
* Not moving: v = 0

c (Clock Penalty): discourages the agent from standing still by penalising clock ticks

* No clock tick: c = 0
* Clock tick: c < 0

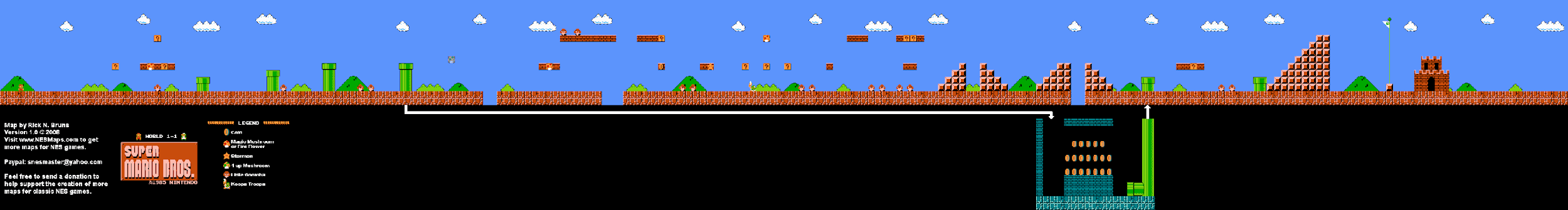
d (Death Penalty): penalises the agent for dying in a state

* Alive: d = 0
* Dead: d = -15

The total reward r is the sum of these variables: r = v + c + d

# 2. Training Assessment Results:

We have completed a set of training assessments to pinpoint and establish appropriate hyperparameters and epsilon-greedy weights policy. This section depicts this process.



## 2.1 Base Reward Policy

Initially, the AI was trained on the innate reward space, prioritising quick rightward movements, aiming to decrease completion time and enhance agent efficiency. Additionally, the agent was discouraged from death. This assessment was set with the following parameters.

#### Epsilon-greedy policy:

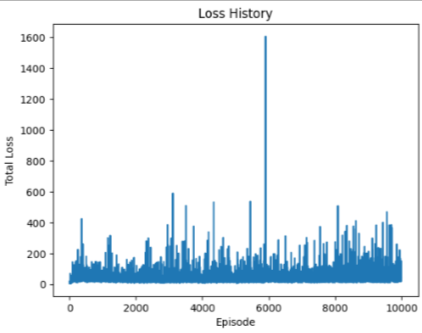
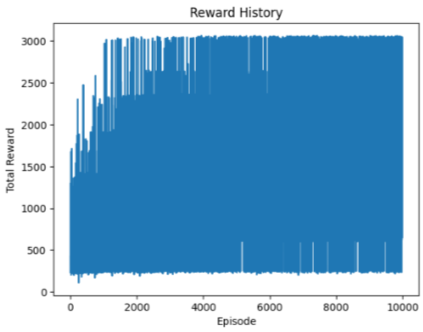
if random.random() < eps\_threshold:

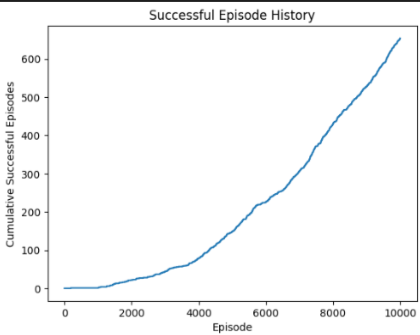
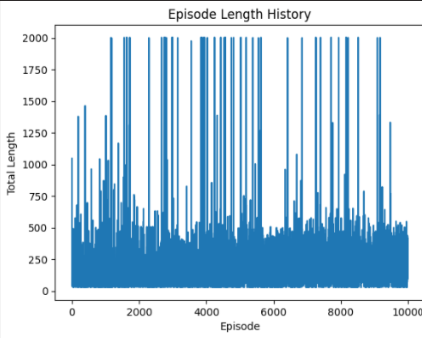
return np.random.choice(np.array(range(12)), p=[0.05, 0.1, 0.1, 0.1, 0.1, 0.05, 0.1, 0.1, 0.1, 0.1, 0.05, 0.05])

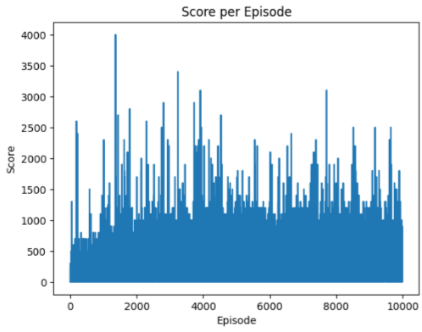
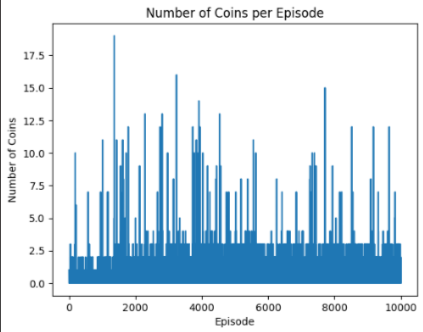
#### Hyperparameters:

|  |  |
| --- | --- |
| **Episodes** | 10,000 |
| **Memory Size** | 50,000 |
| **Replay Start Size** | 5,000 |
| **Network Update Iteration** | 5,000 |
| **Memory Retain** | 0.1 (10%) |
| **Batch Size** | 32 |
| **Learning Rate** | 0.00025 |
| **Gamma** | 0.9 |
| **Epsilon Start** | 1.0 |
| **Epsilon End** | 0.1 |
| **Epsilon Decay** | 0.99999975 |
| **DQN\_DIM1** | 512 |
| **DQN\_DIM2** | 512 |

#### Performance Metrics:

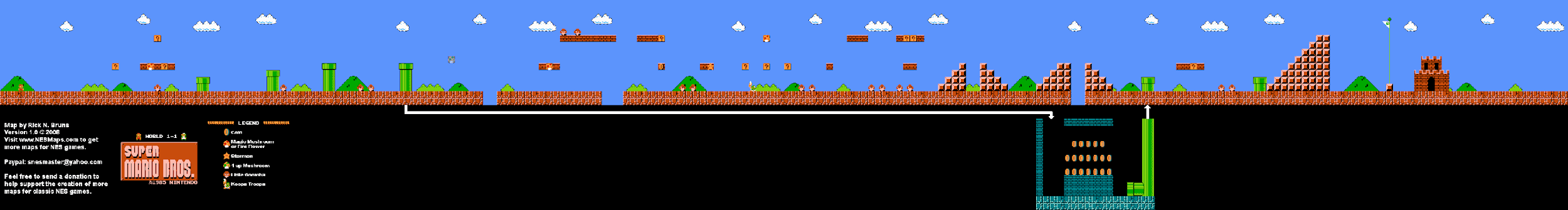






[Recording 2024-05-10 085928.mp4](https://studentutsedu-my.sharepoint.com/:v:/g/personal/blake_a_muchmore_student_uts_edu_au/EQDNmAARfjBJgiUPn05ZCSABB9CQp8Mn3AXp8qh7F6u5NQ?e=GoRY7z)

AI completes approximately 40% of the stage during testing



## 2.2 Custom Reward Policy

Simultaneously, we also assessed the training when our custom reward space was appended to the innate reward space of the gym environment.

### Custom Reward Function

The custom reward shaping function was designed to incentivize the agent to also actively pursue coin collection, elevate its score, and ultimately reach the level’s end flag. This supplementary reward function was integrated into the existing reward system provided by the gym environment. The aim of this was to imitate real-player activity by continuing to prioritise level completion but also incentivize coin and score increases.

The custom reward function is composed of three variables:

COIN\_REWARD = 0.1

SCORE\_REWARD = 1

FLAG\_REWARD = 50

coin\_reward = COIN\_REWARD if info['coins'] > prev\_info['coins'] else 0

score\_reward = SCORE\_REWARD \* (info['score'] - prev\_info['score'])

flag\_reward = FLAG\_REWARD if info['flag\_get'] else 0

custom\_reward = coin\_reward + score\_reward + flag\_reward

The total custom reward is the sum of these variables: custom\_reward = coin\_reward + score\_reward + flag\_reward. This custom reward is added to environment base reward.

### Version 1

To ensure accurate, reliable, and valid comparison between the custom reward and innate reward assessments, we retained the epsilon-greedy policy and hyperparameters between them.

#### Epsilon-greedy policy

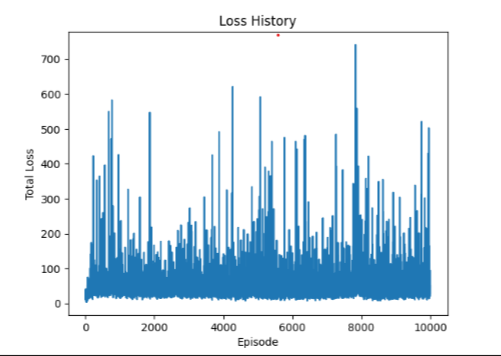
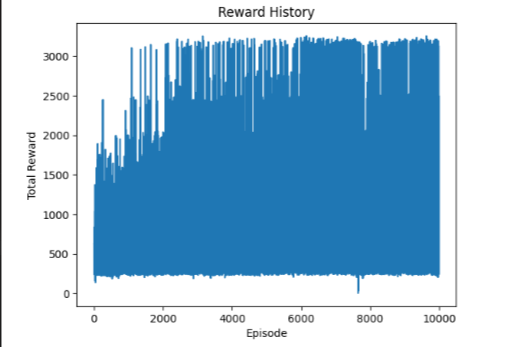
if random.random() < eps\_threshold:

return np.random.choice(np.array(range(12)), p=[0.05, 0.1, 0.1, 0.1, 0.1, 0.05, 0.1, 0.1, 0.1, 0.1, 0.05, 0.05])

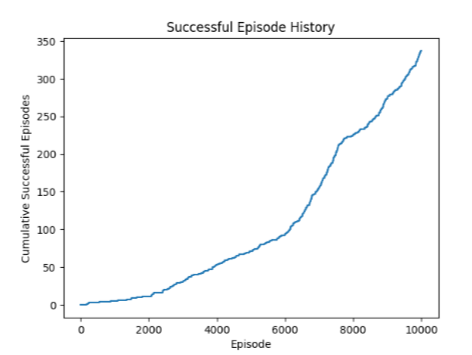
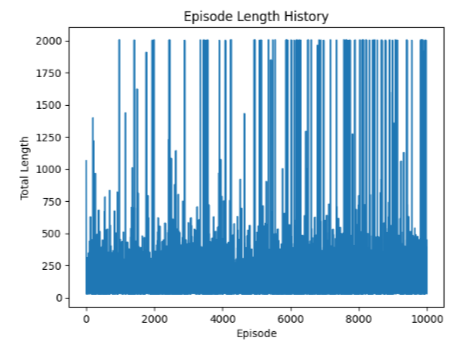
#### Hyperparameters

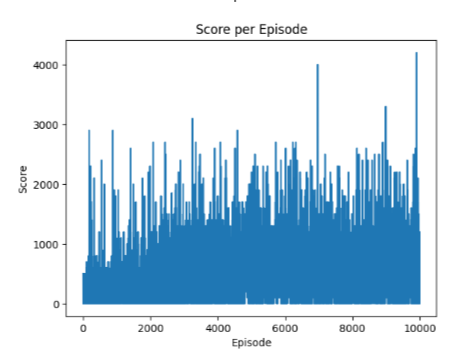
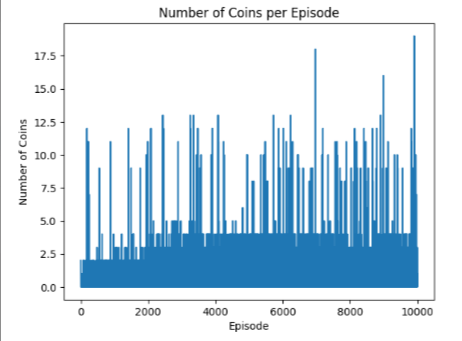
|  |  |
| --- | --- |
| **Episodes** | 10,000 |
| **Memory Size** | 50,000 |
| **Replay Start Size** | 5,000 |
| **Network Update Iteration** | 5,000 |
| **Memory Retain** | 0.1 (10%) |
| **Batch Size** | 32 |
| **Learning Rate** | 0.00025 |
| **Gamma** | 0.9 |
| **Epsilon Start** | 1.0 |
| **Epsilon End** | 0.1 |
| **Epsilon Decay** | 0.99999975 |
| **DQN\_DIM1** | 512 |
| **DQN\_DIM2** | 512 |

#### Performance Metrics:

A screenshot of a computer screen

Description automatically generated





These performance metrics agree with our proposed hypothesis that this training assessment will produce an increase in average score and coin collection. Although through the comparison with this assessment and the innate reward function, the following can be determined:

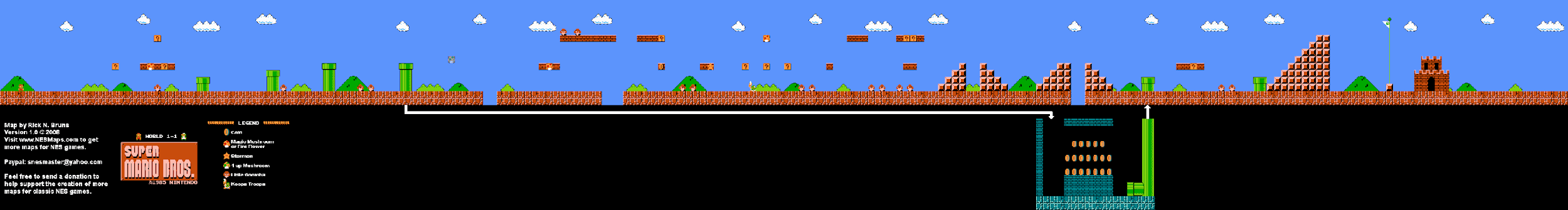
* The model struggles to learn appropriately throughout its episodes
  + The change in reward between episodes is extremely volatile and doesn’t produce a positive gradient trend
  + The success rate of training episodes is significantly low (~3-6%)
* Episode length didn’t always correspond with a higher reward, indicating that the agent consistently gets stuck within the level

This indicates that the method of training is insufficient which could be caused for a variety of reasons, such as:

* Episode count too low to train the agent sufficiently
* Learning rate was too low
* Epsilon was either decaying too quickly or epsilon end too low of a value to promote adequate exploration

[Desktop 2024.05.09 - 23.34.40.02.mp4 (sharepoint.com)](https://studentutsedu-my.sharepoint.com/personal/ryan_j_kim_student_uts_edu_au/_layouts/15/stream.aspx?id=%2Fpersonal%2Fryan%5Fj%5Fkim%5Fstudent%5Futs%5Fedu%5Fau%2FDocuments%2FMicrosoft%20Teams%20Chat%20Files%2FDesktop%202024%2E05%2E09%20%2D%2023%2E34%2E40%2E02%2Emp4&ga=1&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview%2Ed683affb%2D3357%2D4114%2Db625%2D3ab059976571)

AI completes approximately 60% of the stage during testing.



## Version 1.5

To test our hypothesis that training didn’t undergo enough episodes to successfully train the agent, the above test was extended to complete 30,000 episodes. Relevant changes to the code were made to imitate that the test was never stopped.

A graph and a diagram

Description automatically generated with medium confidenceA barcode and graph

Description automatically generatedA graph of a graph of a graph

Description automatically generated with medium confidenceA screenshot of a computer screen

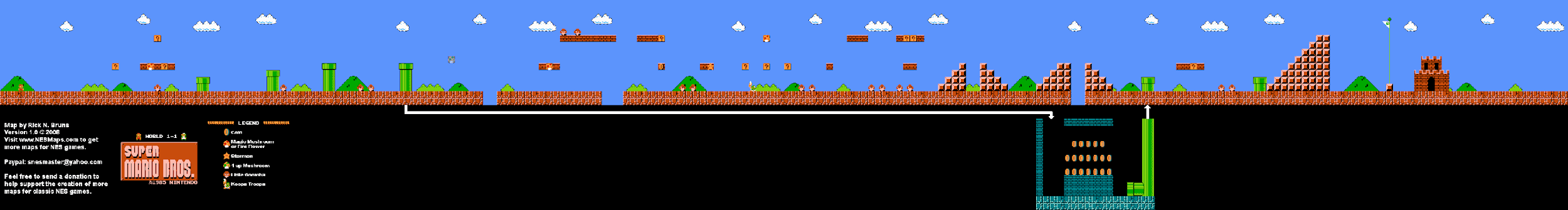
Description automatically generated

From the above performance metric, it was determined that increasing the number of episodes affected the results in the following ways:

* The agent underwent more successful attempts (although this was assumed to be due to more episodes rather than successful learning)
* The agent learned and experienced higher average score and coins collected per episode (although this only showed a minor improvement)
* The average reward per episode didn’t improve and showed the same volatility as the previous assessment
* The episode length average significantly increased indicating that the death reward was somewhat working
* Overall testing performance remained unchanged indicating overall learning to complete the level wasn’t undergone

[Desktop 2024.05.10 - 13.16.32.03.mp4](https://studentutsedu-my.sharepoint.com/:v:/g/personal/ryan_j_kim_student_uts_edu_au/ESy_yOb35TVAhdpyzr0BGVcBRfxAJuaCaz8H5JPI8IY59w?e=tj2apn)

AI completes approximately 60% of the stage during testing



## Version 2

Following the previous training assessments, some further changes to the epsilon-greedy policy and hyperparameters were made. We changed the epsilon-greedy policy to bias rightward movements to hopefully promote further movement through the level. This also removed bias from unwanted leftward movement and redundant actions such as staying still or jumping without movement.

Changes to the hyperparameters were made with the aim of improving agent learning capabilities by changing the following:

* Increased memory size to increase experiences to learn from and reduce removal of important episode memory
* Decrease in replay start size and network update iteration to initiate learning sooner during training and to update the neural network weightings more often
* Increasing epsilon end to promote exploration rates towards the end of the training to increase learning potential by exploring new routes, paths, action sets, etc

Epsilon-greedy policy

We also changed the epsilon-greedy policy to heavily bias rightward motion.

if random.random() < eps\_threshold:

            return np.random.choice(np.array(range(12)), p=[0.005, 0.1675, 0.1675, 0.1675, 0.2175, 0.025, 0.05, 0.05, 0.05, 0.05, 0.05, 0])  # Random action with set priors

#### HYPERPARAMETERS

|  |  |
| --- | --- |
| **Episodes** | 20,000 |
| **Memory Size** | 100,000 |
| **Replay Start Size** | 1,000 |
| **Network Update Iteration** | 1,000 |
| **Memory Retain** | 0.1 (10%) |
| **Batch Size** | 32 |
| **Learning Rate** | 0.0025 |
| **Gamma** | 0.9 |
| **Epsilon Start** | 1.0 |
| **Epsilon End** | 0.33 |
| **Epsilon Decay** | 0.99999975 |
| **DQN\_DIM1** | 512 |
| **DQN\_DIM2** | 512 |

#### Custom Reward Function

The custom reward function was also updated with the aim to favour coin collection and to also further incentivize the agent to complete the level via a greater completion bonus. We also reduced the score reward because we wanted to determine if it was impacting the learning of the agent to prioritise the end of the level.

COIN\_REWARD = 0.5

SCORE\_REWARD = 0.1

FLAG\_REWARD = 1000

coin\_reward = COIN\_REWARD if info['coins'] > prev\_info['coins'] else 0

score\_reward = SCORE\_REWARD \* (info['score'] - prev\_info['score'])

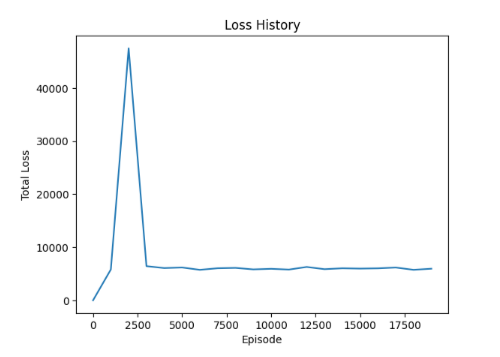
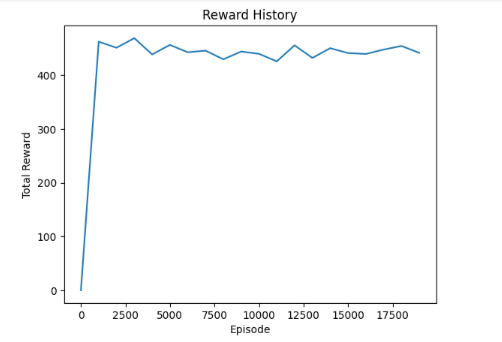
flag\_reward = FLAG\_REWARD if info['flag\_get'] else 0

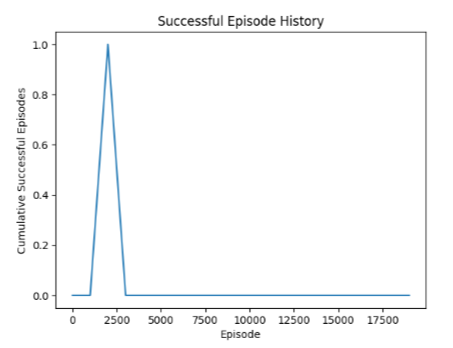
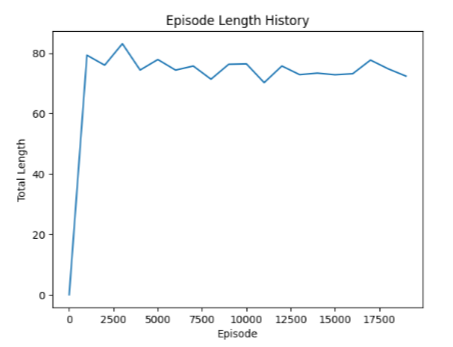
custom\_reward = coin\_reward + score\_reward + flag\_reward

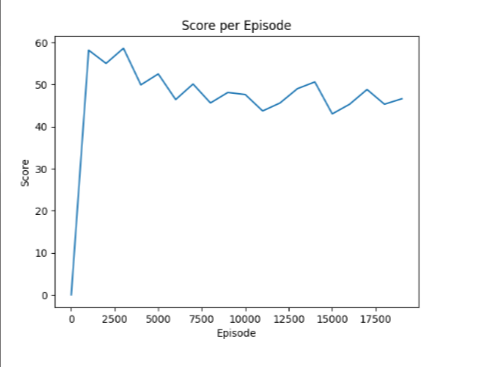
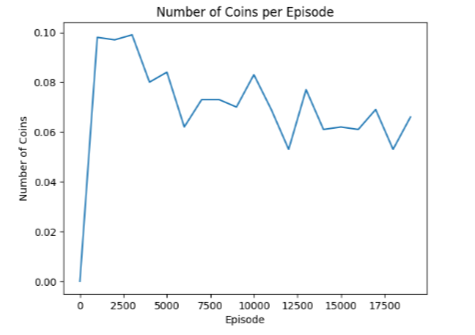
The total custom reward is the sum of these variables: custom\_reward = coin\_reward + score\_reward + flag\_reward. This custom reward is added to environment base reward.

#### Performance Metrics:

Overall, our results still show a struggle with learning and worsened regarding successful attempts. Our model did reach 60% during testing faster though representing our biassed movements in our epsilon-greedy policy.

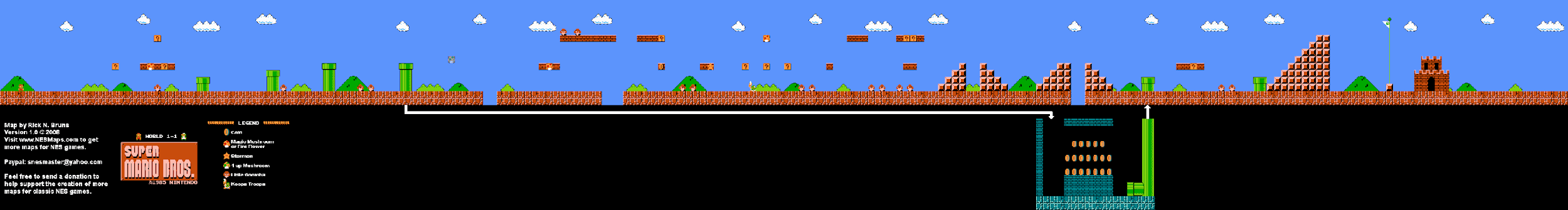






[Desktop 2024.05.10 - 23.53.29.04.mp4 (sharepoint.com)](https://studentutsedu-my.sharepoint.com/personal/ryan_j_kim_student_uts_edu_au/_layouts/15/stream.aspx?id=%2Fpersonal%2Fryan%5Fj%5Fkim%5Fstudent%5Futs%5Fedu%5Fau%2FDocuments%2FMicrosoft%20Teams%20Chat%20Files%2FDesktop%202024%2E05%2E10%20%2D%2023%2E53%2E29%2E04%2Emp4&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview%2Ecd5f6152%2D49b8%2D4fce%2Dad8a%2D2ae2c5dfc594&ga=1)

AI completes 60% of the stage during testing



## 2.3 Experimental Custom Reward Policy Attempts

Due to the minimal changes seen between different training attempts, another team member decided to overhaul major component parameters that influence our reinforcement learning.

#### Hyperparameters:

|  |  |
| --- | --- |
| **Episodes** | 20,000 |
| **Memory Size** | 100,000 |
| **Replay Start Size** | 10,000 |
| **Network Update Iteration** | 2,000 |
| **Memory Retain** | 0.5 (50%) |
| **Batch Size** | 32 |
| **Learning Rate** | 0.000001 |
| **Gamma** | 0.9 |

The purpose of these changes were to increase Mario's memory, drastically increase the retention memory to 50% of the replay buffer, and increase initial starting replay size for more samples in Mario’s early training. Additionally we slowed down the learning rates and network update iterations to simulate slower but smoother training performance.

#### Epsilon-greedy policy:

|  |  |
| --- | --- |
| **Epsilon Start** | 20,000 |
| **Epsilon End** | 100,000 |
| **Epsilon Decay** | Episodes \* 0.8 |

##### Code:

if self.memory.memory\_count > self.replay\_start\_size:

self.exploration\_rate = max(self.epsilon\_end, self.epsilon\_start \* math.exp(-self.episode / self.epsilon\_decay))

Included in the overhaul, the decaying rate of the greedy epsilon value was changed to exponentially decrease until 80% of the total episode which was to incentivize Mario to interchangeably explore more greedily within the first half of the training and then exploiting more on the other half.

#### Random Action Probability Distribution:

if random.random() < self.exploration\_rate:

return np.random.choice(np.array(range(12)), p=[0.005, 0.21, 0.15, 0.17, 0.19, 0.025, 0.1, 0.05, 0.05, 0.05, 0.0, 0]) # Random action with set priors

#### Custom Reward Function:

def shapeRewards(info, prev\_info, done):

# Creating constants for reward shaping

COIN\_REWARD = 0.1

SCORE\_REWARD = 0.1

FLAG\_REWARD = 1000

RIGHT\_REWARD = 0.1

DEATH\_REWARD = -50

TIME\_REWARD = -0.2

# Checking if death reward should be added

death\_reward = 0

if done and info['flag\_get'] == False:

death\_reward = DEATH\_REWARD

# Checking if timeout reward should be added

timeout\_reward = 0

if info['time'] < 1: #on the cut list

print("TIME OUT")

timeout\_reward = -10 # Subtract 10 if time ran out

# Checking if mario got a powerup

powerup\_reward = 0

if info['status'] == 'tall' and prev\_info['status'] == 'small':

print('BIGG MODE')

powerup\_reward = 50 # reward for being tall

elif info['status'] == 'fireball' and prev\_info['status'] in ['tall', 'small']:

print('FIRE MODE')

powerup\_reward = 50 # reward for changing to fireball

# killstreak

kill\_reward = 0

if (info['score'] - prev\_info['score']) == 100 and info['coins'] == prev\_info['coins']: #checks if mario killed a goomba or koopa

kill\_reward = 2 # or any other reward value you choose #TWEAK THIS

# Checking if mario is stuck

stuck\_reward = 0

if abs(info['x\_pos'] - prev\_info['x\_pos']) <= 1:

stuck\_reward = -1

# Checking if any custom rewards should be added

coin\_reward = COIN\_REWARD \* (info['coins'] - prev\_info['coins'])

score\_reward = SCORE\_REWARD \* (info['score'] - prev\_info['score'])

flag\_reward = FLAG\_REWARD if info['flag\_get'] else 0

if (info['x\_pos'] > prev\_info['x\_pos']):

right\_reward = RIGHT\_REWARD \* (info['x\_pos'] - prev\_info['x\_pos'])

else:

right\_reward = 0

time\_reward = TIME\_REWARD \* (prev\_info['time'] - info['time'])

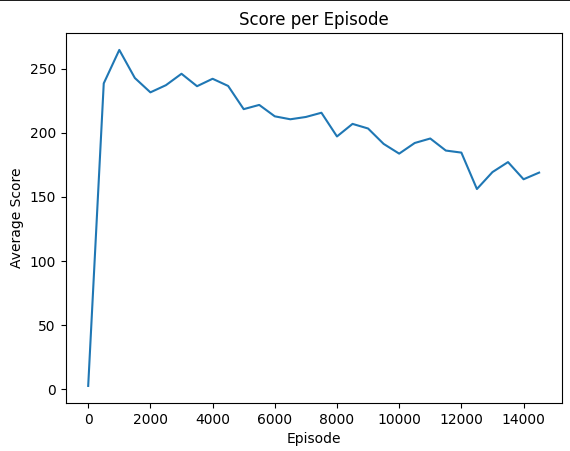
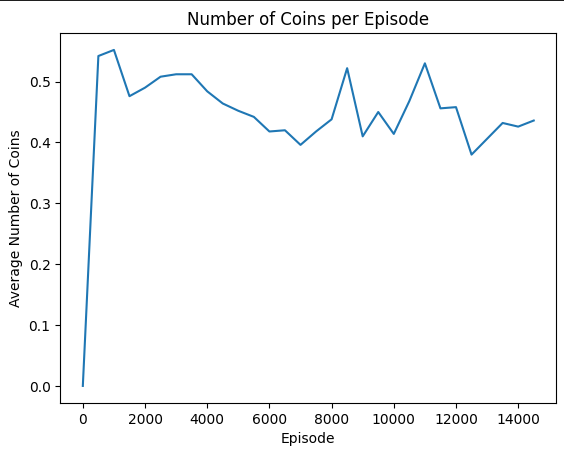
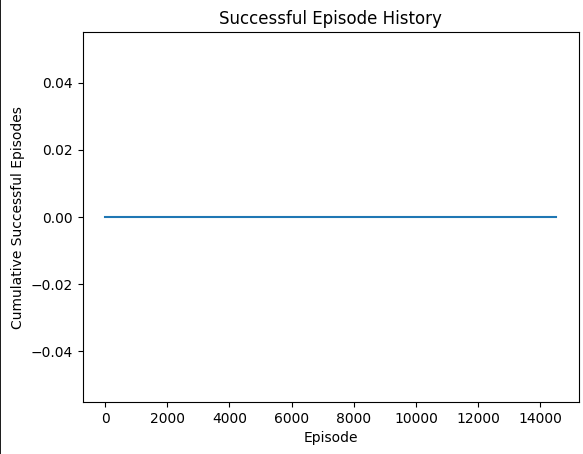
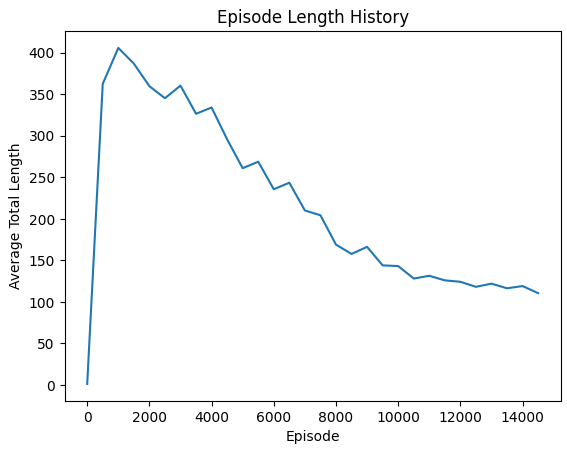
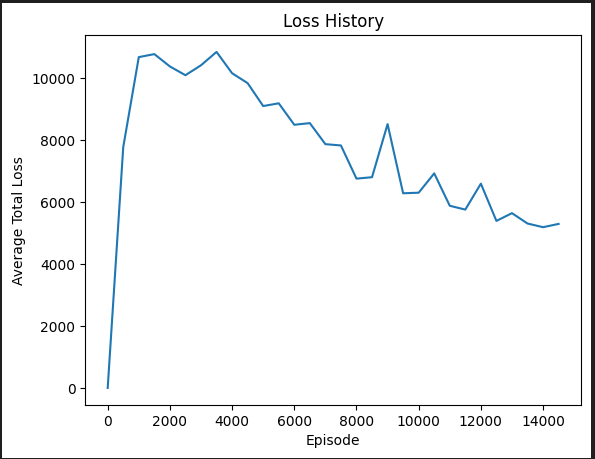
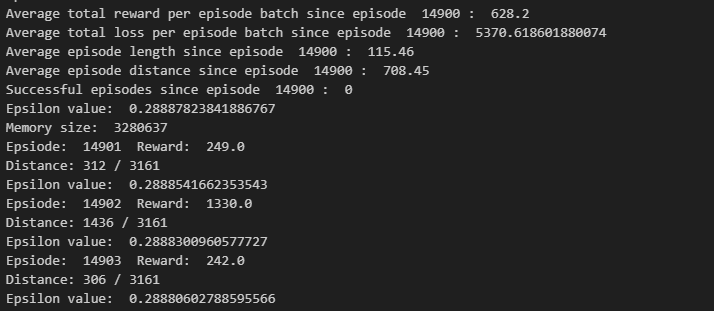
# Calculating the custom reward

custom\_reward = coin\_reward + score\_reward + flag\_reward + right\_reward + death\_reward + time\_reward + powerup\_reward + stuck\_reward + timeout\_reward + kill\_reward

return custom\_reward # Returning the custom reward

Extensive work to implement a more interactive reward system was used to test its influence on Mario’s actions within the environment, power ups giving Mario a second chance within the run were given a fair amount of feedback (although a much higher or total reward percentile increase was proposed but reserved for later testing). An additional stuck detection condition punishes Mario for being in place for long durations (observing the training runs, Mario will often be stuck at the beginning pipes, the 4 pipe problem). A kill reward condition, which detects when Mario successfully stomps on a Goomba (also important during Mario’s climb in between pipes being caged in with Goomba’s). A function to detect when Mario falls down a bottomless hole was going to be implemented, however due to a hidden underground section of the level, determining y\_pos of Mario was too finicky and potentially exploitable.

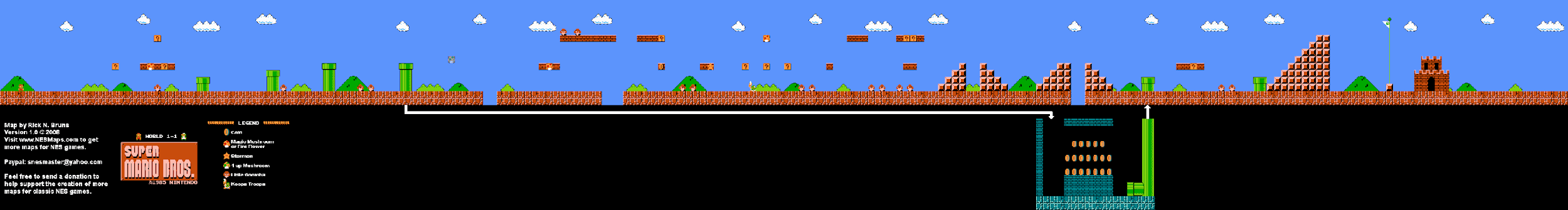
#### Performance Metrics:



An unexpected catastrophic performance with Mario occurred over 20,000 episodes that worsened to a localised lower minima near the end of training. We suspected an error/bug was prevalent within the code base or libraries as other team members experienced the sudden degrading performance of Mario within their local setups.

<https://studentutsedu-my.sharepoint.com/:v:/g/personal/blake_a_muchmore_student_uts_edu_au/EXE2rSFDvbJFm-iZqukAW1ABLza1I6xown1LOAugbW7xTg?e=DemZJV>

AI completes 5% of the stage during testing



## 2.4 Final Custom Reward Policy Attempt

From all the above tests, we collated knowledge from various iterative experiments to determine the most appropriate learning model for our AI to speedrun Super Mario Bros levels, while also collecting coins and increasing score.

Epsilon-greedy policy

We used the same epsilon greedy bias terms as previous sections. This heavily biases rightward movements to get further into the level faster and more efficiently.

if random.random() < eps\_threshold:

            return np.random.choice(np.array(range(12)), p=[0.005, 0.1675, 0.1675, 0.1675, 0.2175, 0.025, 0.05, 0.05, 0.05, 0.05, 0.05, 0])  # Random action with set priors

#### HYPERPARAMETERS

|  |  |
| --- | --- |
| **Episodes** | 10,000 |
| **Memory Size** | 50,000 |
| **Replay Start Size** | 5,000 |
| **Network Update Iteration** | 3,000 |
| **Memory Retain** | 0.1 |
| **Batch Size** | 32 |
| **Learning Rate** | 0.00025 |
| **Gamma** | 0.99 |
| **Epsilon Start** | 1.0 |
| **Epsilon End** | 0.05 |
| **Epsilon Decay** | 0.7 \* Episodes \* 250 |
| **DQN\_DIM1** | 512 |
| **DQN\_DIM2** | 512 |

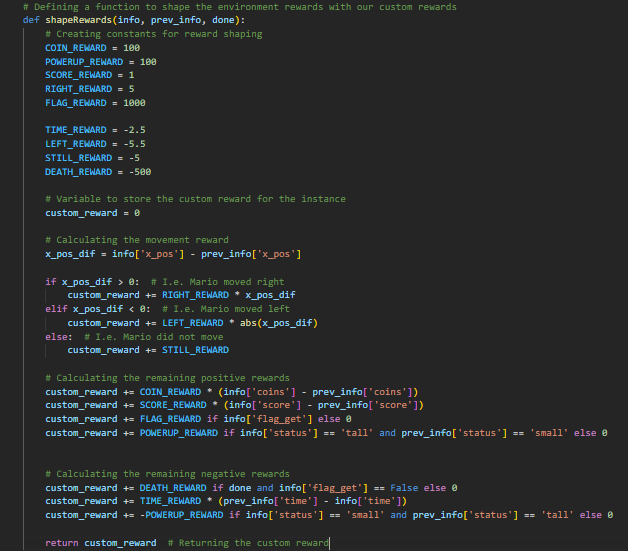
The following changes to the hyperparameters were as follows:

* Replay start size was slightly reduced to implement training sooner
* Network update iteration was increased to reduce training rate once training started
* Memory retain reflected original 10%
* Gamma increased to 0.99 to further incentivise future rewards over immediate rewards
* Epsilon decay was changed to finish decaying after roughly 70% of episodes (episodes from previous training showed length to be roughly 250 per episode)

#### Custom Reward Function

The finalised custom reward function incentivises Mario to collect coins and powerups, increase score, move rightwards, and increase the goal. The Mario agent is discouraged to waste time, move leftwards, stay still, and die before completing the level.

The rewards are scaled to bias rightwards movement more than other goals as the main project aim was to incentivize Mario to speedrun levels. Collecting coins and powerups, and increasing score was defined as a secondary objective. While the right reward has a lower value than the other rewards, there are significantly more opportunities to get right movement rewards than others such as coins and powerups.



#### Performance Metrics:

A graph of a blue line

Description automatically generatedA graph of loss of a loss

Description automatically generated

A graph of a bar graph

Description automatically generated with medium confidence A graph of a successful episode

Description automatically generated A graph of a number of coins

Description automatically generated A graph with blue lines

Description automatically generated

Overall, the training was fairly successful even though we couldn’t fully complete the level. The training showed an increase in reward gained overtime, an average low loss, and roughly ~650-700 successful episodes. The results also showed a gradual increase in coin collection reaching ~4.5 coins per episode, and an average score of 2000. The score especially is high considering the agent isn’t incentivised to defeat enemies, meaning the agent is successfully avoiding them, defeating them only when needed, and collecting coins.

[mario vid.mp4](https://studentutsedu-my.sharepoint.com/:v:/g/personal/dominic_manno_student_uts_edu_au/EWv0bljmlAhBnHq5eXBwWZUB8BlPa_fI6zHIEY3SQdmRRg?e=rAPFEK)

AI completes approximately 75% of the stage during testing

