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.

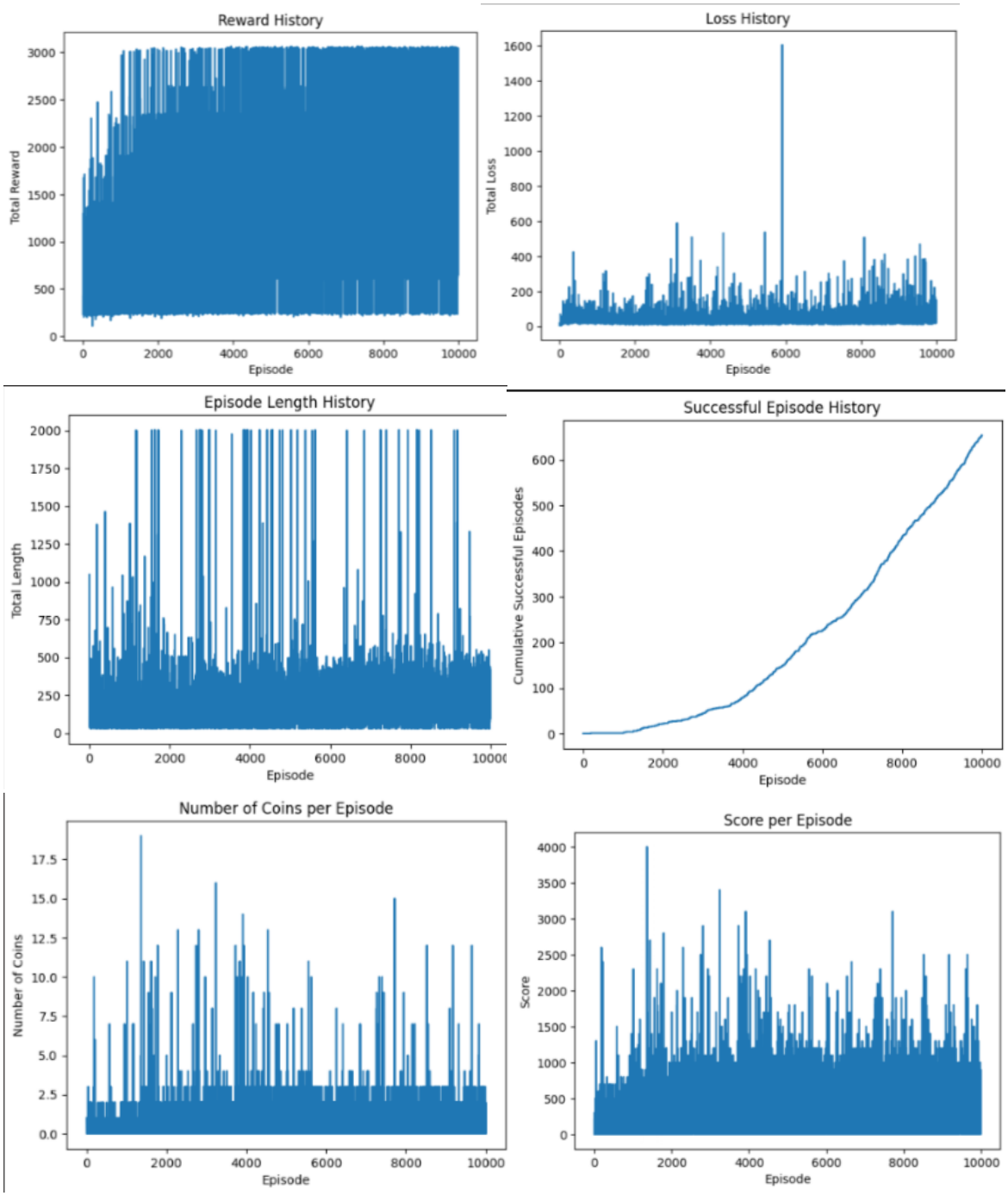
Epsilon-greedy policy:

```
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]

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

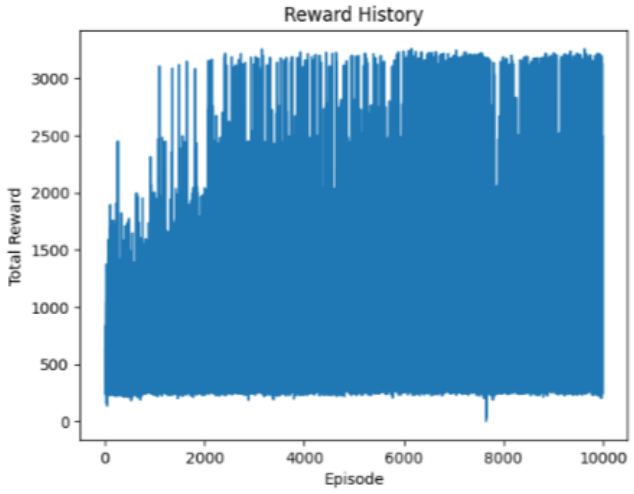
```
if random.random() < eps_threshold:
    return np.random.choice(np.array(range(12)), p=[0.05, 0.1, 0.1, 0.1, 0.1, 0.05, 0.1, 0.1, 0.1, 0.1, 0.05, 0.05])
```

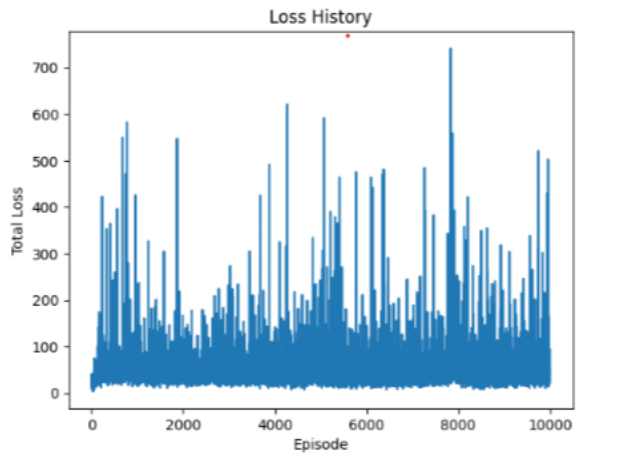
Hyperparameters

Episodes	10,000
Memory Size	50,000
Replay Start Size	5,000
Network Update Iteration	5,000

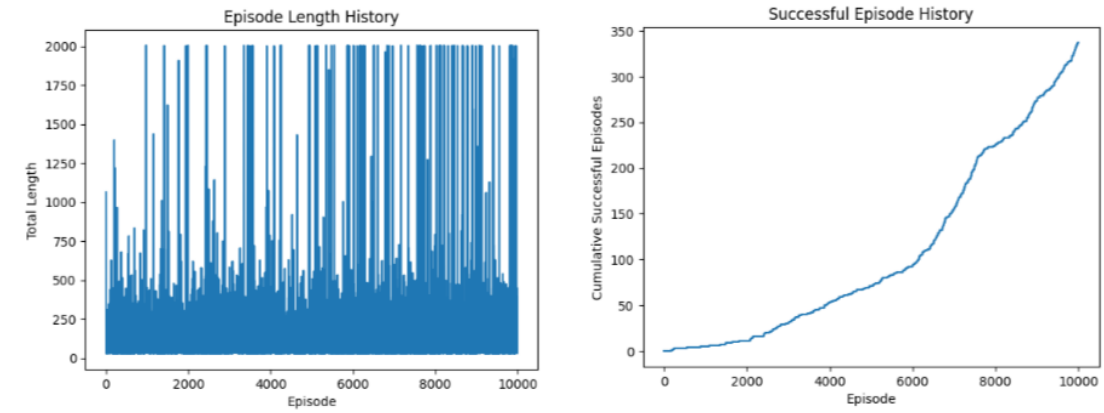
Memory Retain	0.1 (10%)
Batch Size	32
Learning Rate	0.00025
Gamma	0.9
Epsilon Start	1.0
Epsilon End	0.1
Epsilon Decay	0.99999975
DQN_DIM1	512
DQN_DIM2	512

Performance Metrics:






```
Epsiode:  9971  Reward:  771.0
Updating target network
Epsiode:  9972  Reward:  3132.0
Epsiode:  9973  Reward:  628.0
Epsiode:  9974  Reward:  2016.0
Epsiode:  9975  Reward:  625.0
Updating target network
Epsiode:  9976  Reward:  3193.0
Epsiode:  9977  Reward:  791.0
Epsiode:  9978  Reward:  2446.0
Epsiode:  9979  Reward:  1191.0
Epsiode:  9980  Reward:  1126.0
Updating target network
Epsiode:  9981  Reward:  1392.0
Epsiode:  9982  Reward:  2434.0
Epsiode:  9983  Reward:  1075.0
Epsiode:  9984  Reward:  3132.0
Updating target network
Epsiode:  9985  Reward:  1387.0
Epsiode:  9986  Reward:  607.0
Epsiode:  9987  Reward:  623.0
Epsiode:  9988  Reward:  1393.0
Epsiode:  9989  Reward:  2426.0
Updating target network
Epsiode:  9990  Reward:  2497.0
Epsiode:  9991  Reward:  241.0
Epsiode:  9992  Reward:  1373.0
Epsiode:  9993  Reward:  246.0
Epsiode:  9994  Reward:  2046.0
Epsiode:  9995  Reward:  1387.0
Epsiode:  9996  Reward:  1457.0
Epsiode:  9997  Reward:  1343.0
Updating target network
Epsiode:  9998  Reward:  1075.0
Epsiode:  9999  Reward:  633.0
Number of successful episodes:  337
```



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:

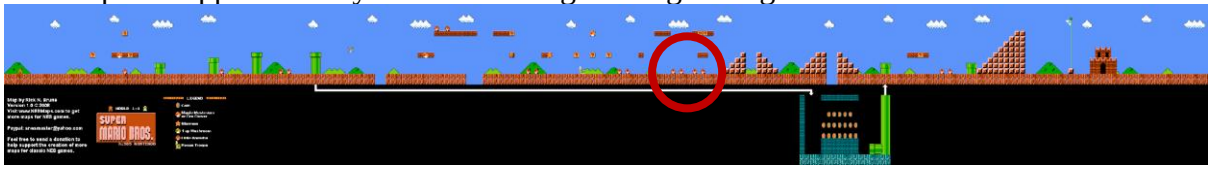
- The model struggles to learn appropriately throughout its episodes
    - The change in reward between episodes is extremely volatile and doesn't produce a positive gradient trend
    - The success rate of training episodes is significantly low (~3-6%)
- Episode length didn't always correspond with 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

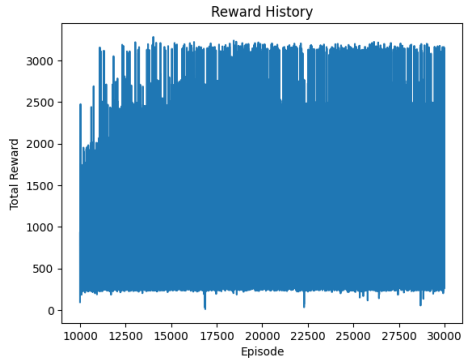
[Desktop 2024.05.09 - 23.34.40.02.mp4 (sharepoint.com)]

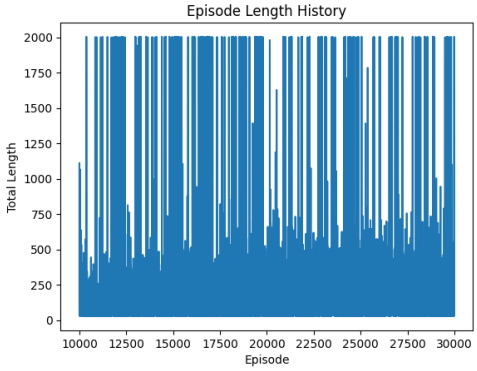
AI completes approximately 60% of the stage during testing.

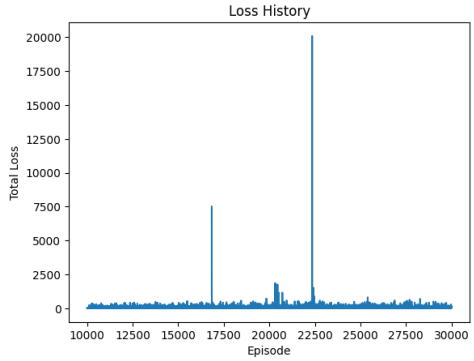


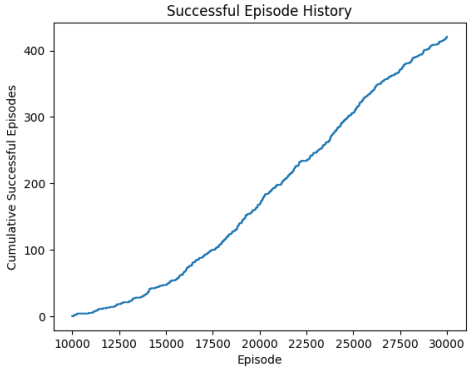
Version 1.5

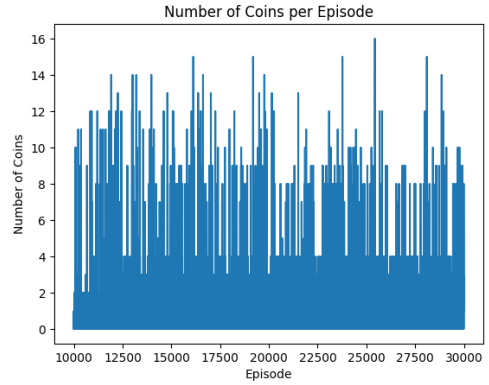
To test our hypothesis that training didn't undergo enough episodes to successfully train the agent, the above test was extended to complete 30,000 episodes. Relevant changes to the code were made to imitate that the test was never stopped.

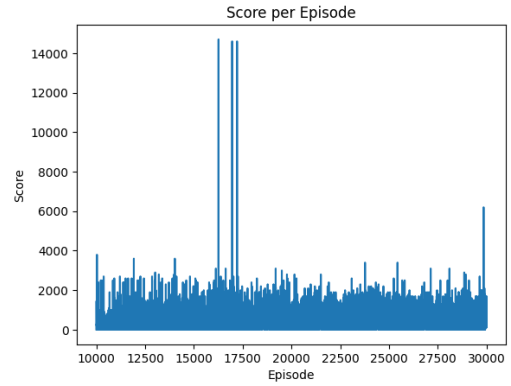













```
Epsiode:  29996  Reward:  1453.0
Epsiode:  29997  Reward:  2445.0
Epsiode:  29998  Reward:  1889.0
Epsiode:  29999  Reward:  1750.0
Number of successful episodes:  421
```

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
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

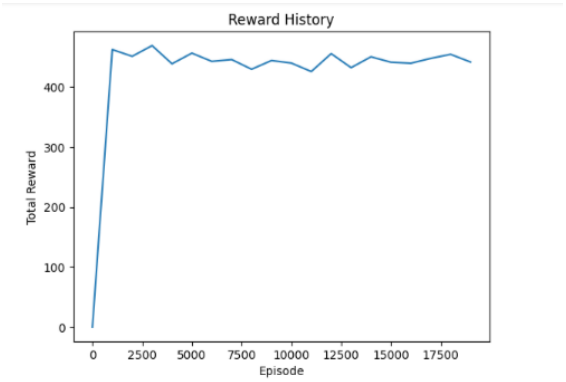
The custom reward function was also updated with the aim to favour coin collection and to also further incentivize the agent to complete the level via a greater completion bonus. We also reduced the score reward because we wanted to determine if it was impacting the learning of the agent to prioritise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

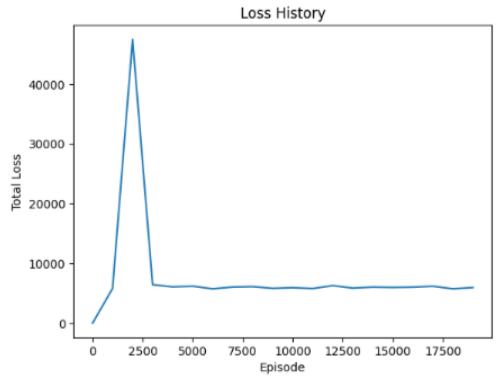
```
COIN_REWARD = 0.5
SCORE_REWARD = 0.1
FLAG_REWARD = 1000

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
```

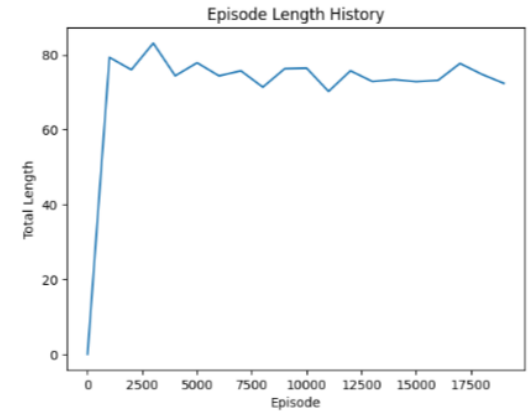
The total custom reward is the sum of these variables: custom_reward = coin_reward + score_reward + flag_reward. This custom reward is added to environment base reward.

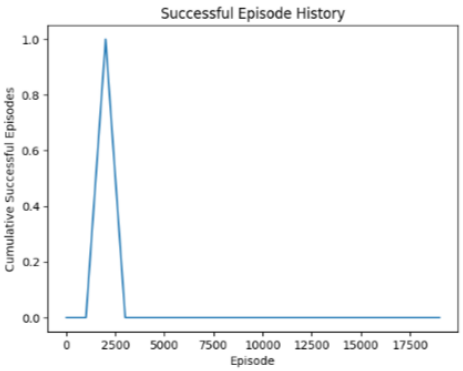
Performance Metrics:

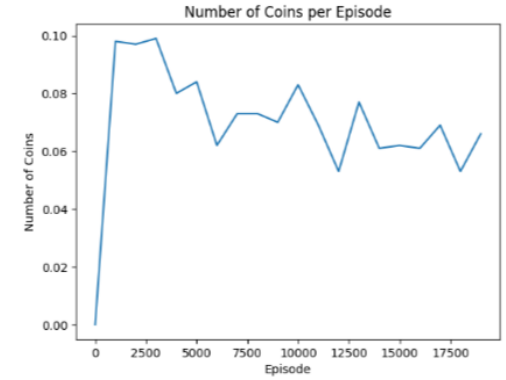
Overall, our results still show a struggle with learning and worsened regarding successful attempts. Our model did reach 60% during testing faster though representing our biassed movements in our epsilon-greedy policy.





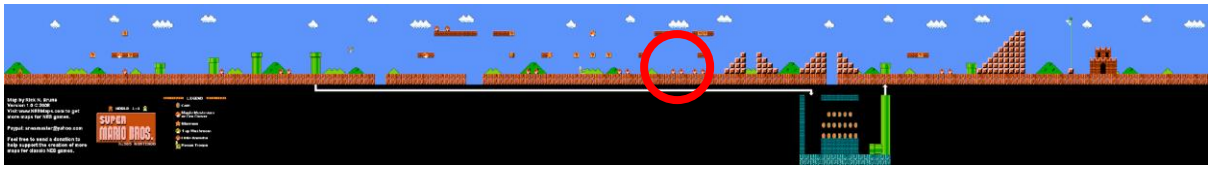






[Desktop 2024.05.10 - 23.53.29.04.mp4 (sharepoint.com)]

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:

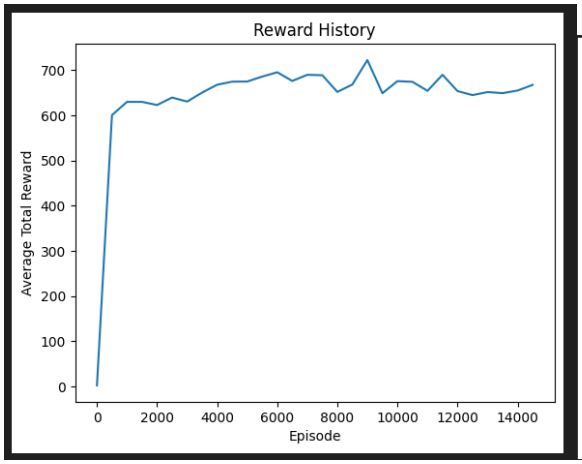
```
def shapeRewards(info, prev_info, done):
    # Creating constants for reward shaping
    COIN_REWARD = 0.1
    SCORE_REWARD = 0.1
    FLAG_REWARD = 1000
    RIGHT_REWARD = 0.1
    DEATH_REWARD = -50
    TIME_REWARD = -0.2

    # 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
```

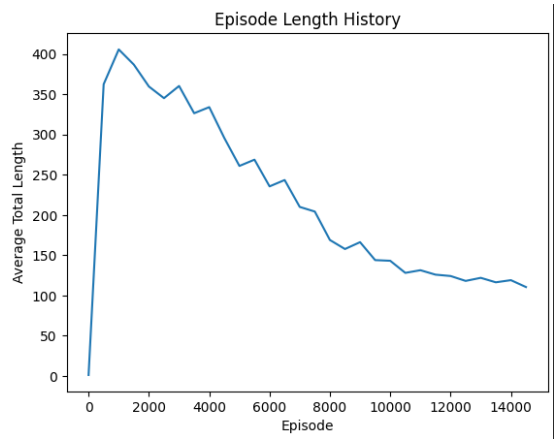
```
    # Checking if mario is stuck
    stuck_reward = 0
    if abs(info['x_pos'] - prev_info['x_pos']) <= 1:
        stuck_reward = -1

    # 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
```

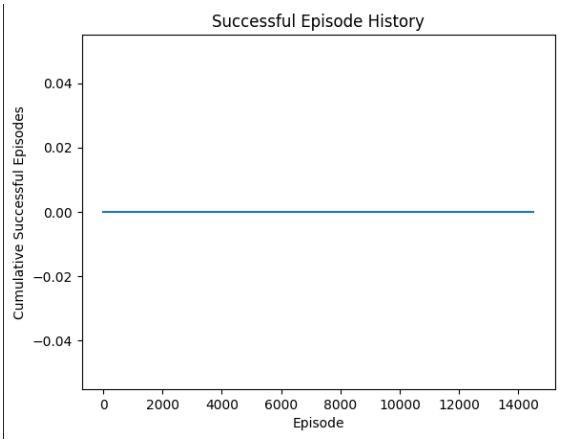
Extensive work to implement a more interactive reward system was used to test its influence on Mario's actions within the environment, power ups giving Mario a second chance within the run were given a fair amount of feedback (although a much higher or total reward percentile increase was proposed but 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.

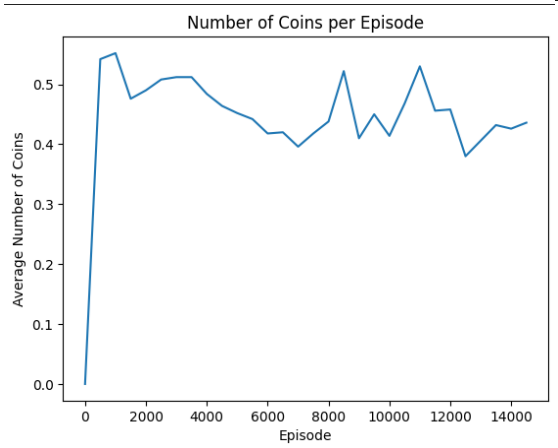
Performance Metrics:

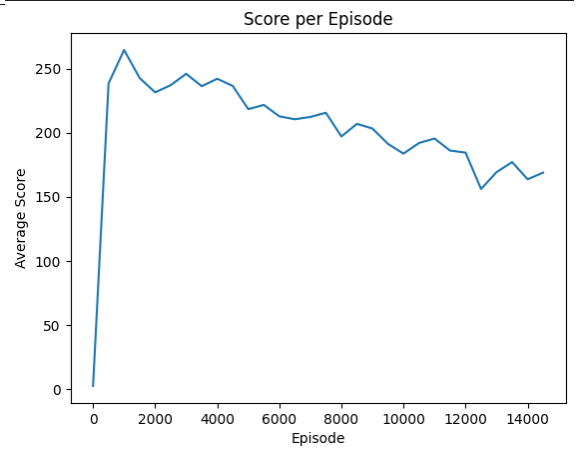
```
Average total reward per episode batch since episode  14900 :  628.2
Average total loss per episode batch since episode  14900 :  5370.618601880074
Average episode length since episode  14900 :  115.46
Average episode distance since episode  14900 :  708.45
Successful episodes since episode  14900 :  0
Epsilon value:  0.28887823841886767
Memory size:  3280637
```











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)

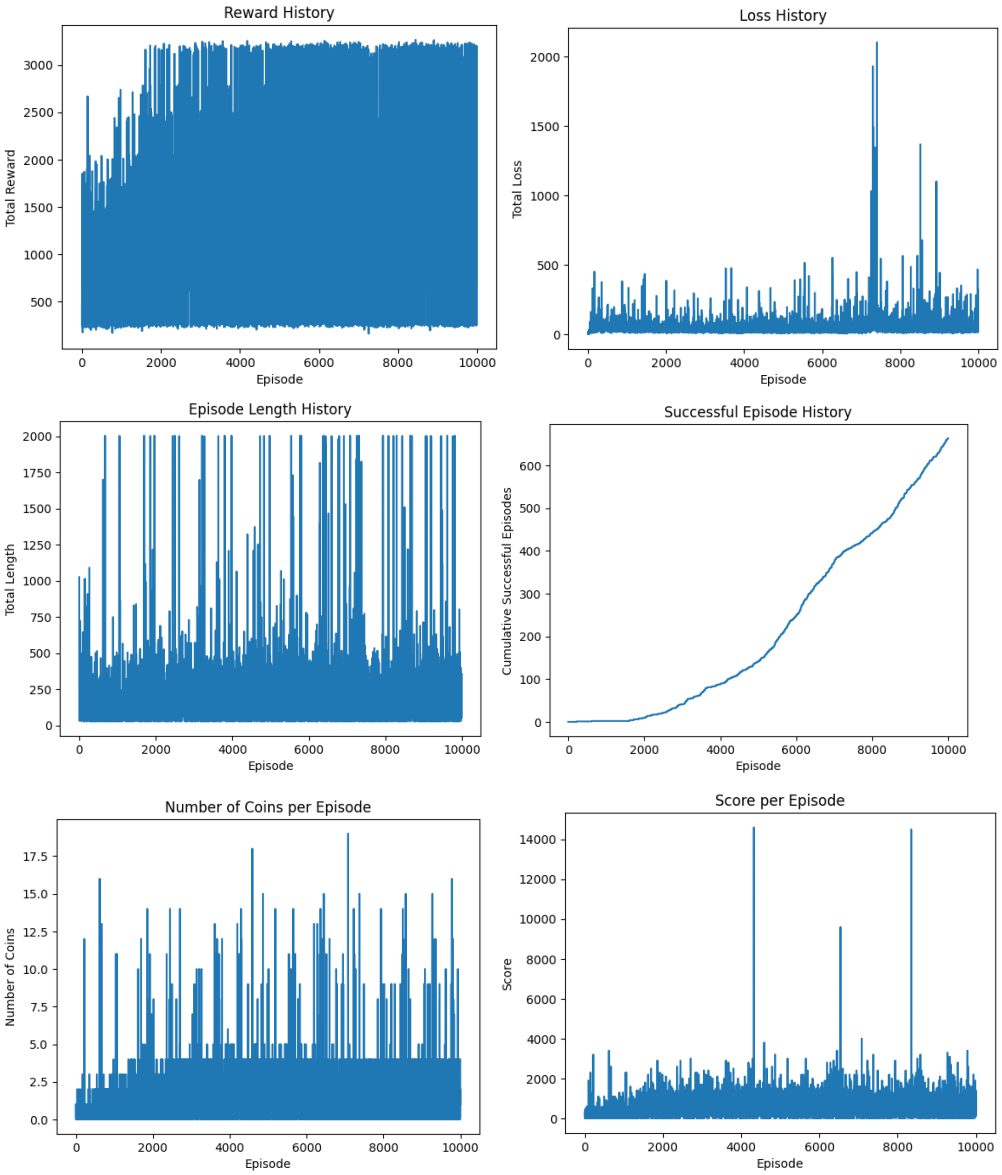
Custom Reward Function

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.

```python
# Defining a function to shape the environment rewards with our custom rewards
def shapeRewards(info, prev_info, done):
    # Creating constants for reward shaping
    COIN_REWARD = 100
    POWERUP_REWARD = 100
    SCORE_REWARD = 1
    RIGHT_REWARD = 5
    FLAG_REWARD = 1000

    TIME_REWARD = -2.5
    LEFT_REWARD = -5.5
    STILL_REWARD = -5
    DEATH_REWARD = -500

    # Variable to store the custom reward for the instance
    custom_reward = 0

    # Calculating the movement reward
    x_pos_dif = info['x_pos'] - prev_info['x_pos']

    if x_pos_dif > 0:  # I.e. Mario moved right
        custom_reward += RIGHT_REWARD * x_pos_dif
    elif x_pos_dif < 0:  # I.e. Mario moved left
        custom_reward += LEFT_REWARD * abs(x_pos_dif)
    else:  # I.e. Mario did not move
        custom_reward += STILL_REWARD

    # Calculating the remaining positive rewards
    custom_reward += COIN_REWARD * (info['coins'] - prev_info['coins'])
    custom_reward += SCORE_REWARD * (info['score'] - prev_info['score'])
    custom_reward += FLAG_REWARD if info['flag_get'] else 0
    custom_reward += POWERUP_REWARD if info['status'] == 'tall' and prev_info['status'] == 'small' else 0


    # Calculating the remaining negative rewards
    custom_reward += DEATH_REWARD if done and info['flag_get'] == False else 0
    custom_reward += TIME_REWARD * (prev_info['time'] - info['time'])
    custom_reward += -POWERUP_REWARD if info['status'] == 'small' and prev_info['status'] == 'tall' else 0

    return custom_reward  # Returning the custom reward
```

Performance Metrics:



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

AI completes approximately 75% of the stage during testing