Part B

REINFORCEMENT LEARNING

DELE CA2

Chew Yee Jing - P2415860
Vijeyakumar Dakshaa - P2415899

Overview

Introduction 01

Background Research 02

Modelling 03

Model Improvemnet 1 04

Model Improvement 2 05

Final Model 06

Conclusion 07

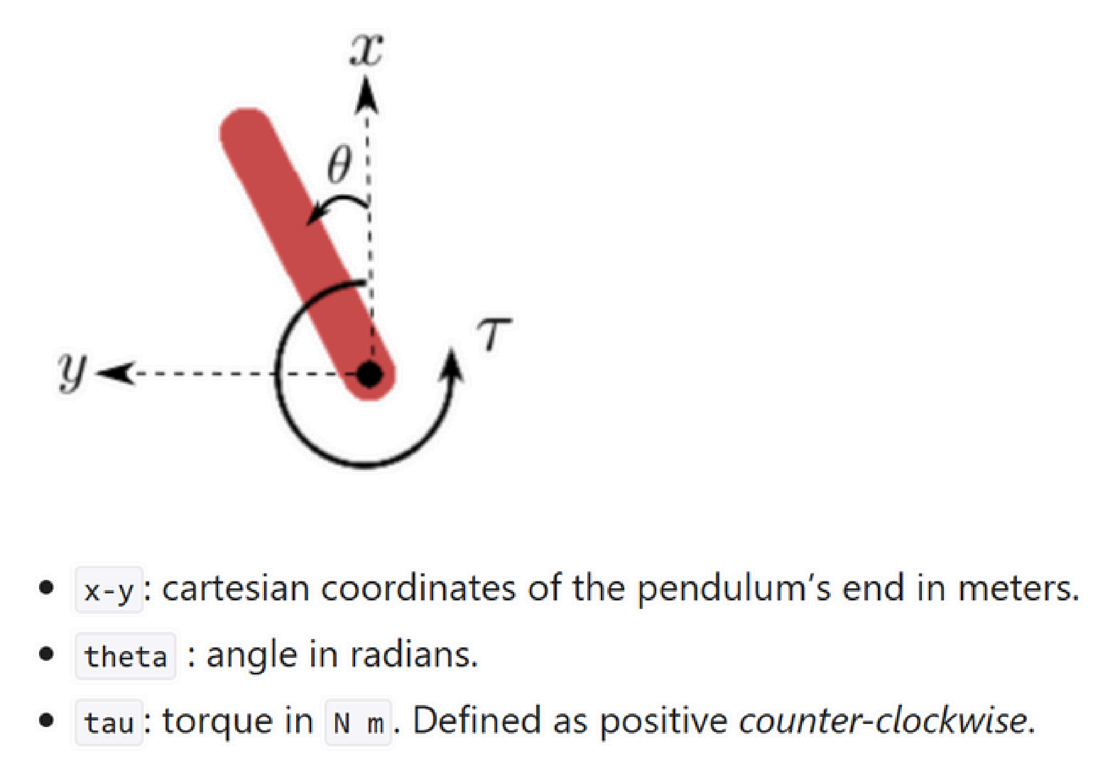
Thank You 08

Introduction

Apply Reinforcement Learning (RL) to solve the classic control problem of balancing a pendulum.

- Use the Pendulum-v0 environment from **OpenAI Gym**.
- Implement a **Deep Q-Network (DQN)** solution.
- Handle the **continuous action space** by applying **discretization**.

Aim: The objective is to train an agent that learns to keep the pendulum upright by applying appropriate torques at each timestep.



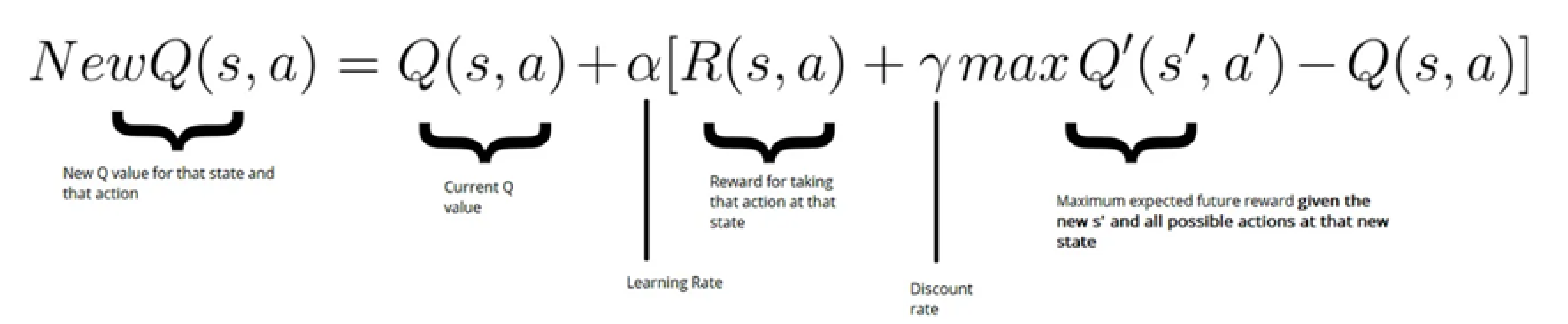
- `x-y`: cartesian coordinates of the pendulum's end in meters.
- `theta` : angle in radians.
- `tau`: torque in `N m`. Defined as positive *counter-clockwise*.

Background Research

Pendulum-v0

What is DQN?

DQN is a model-free, value-based reinforcement learning algorithm. It approximates the **Q-value function** using a neural network to evaluate the **expected future reward** of taking an action in a given state.



$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma \, max \, Q'(s',a') - Q(s,a)]$$

New Q value for that state and that action
Current Q value
Learning Rate
Reward for taking that action at that state
Discount rate
Maximum expected future reward given the new s' and all possible actions at that new state

Pendulum-v0 environment

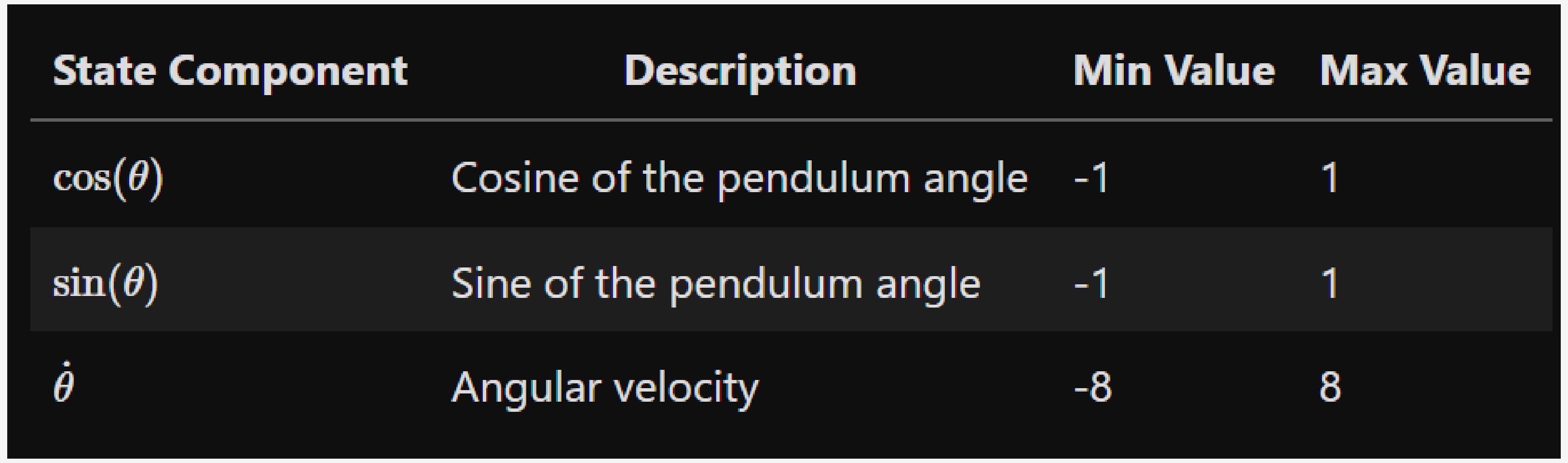
The Pendulum-v0 environment simulates a simple pendulum with the goal of keeping it upright (angle = 0) using minimal energy. It is designed as a continuous control task.

State Space: Observation Space

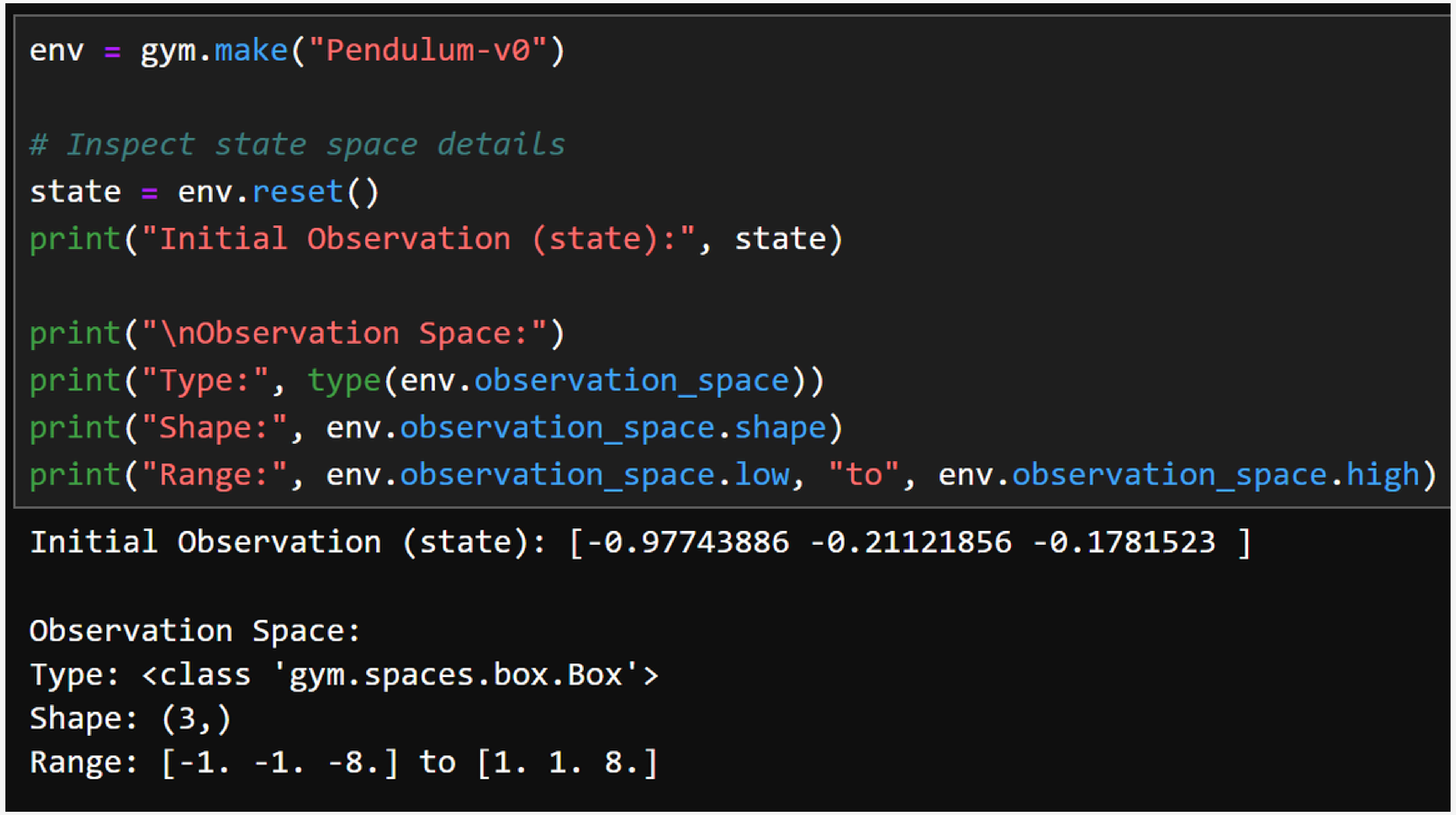
The observation space in Pendulum-v0 consists of 3 continuous values:

- cosine of the pendulum angle
- sine of the pendulum angle
- angular velocity (i.e. the rate at which the pendulum is swinging)

This 3-dimensional state vector is used by the agent to determine its next action.



State Component	Description	Min Value	Max Value
$\cos(\theta)$	Cosine of the pendulum angle	-1	1
$\sin(\theta)$	Sine of the pendulum angle	-1	1
$\dot{\theta}$	Angular velocity	-8	8



```python
env = gym.make("Pendulum-v0")

# Inspect state space details
state = env.reset()
print("Initial Observation (state):", state)

print("\nObservation Space:")
print("Type:", type(env.observation_space))
print("Shape:", env.observation_space.shape)
print("Range:", env.observation_space.low, "to", env.observation_space.high)
```

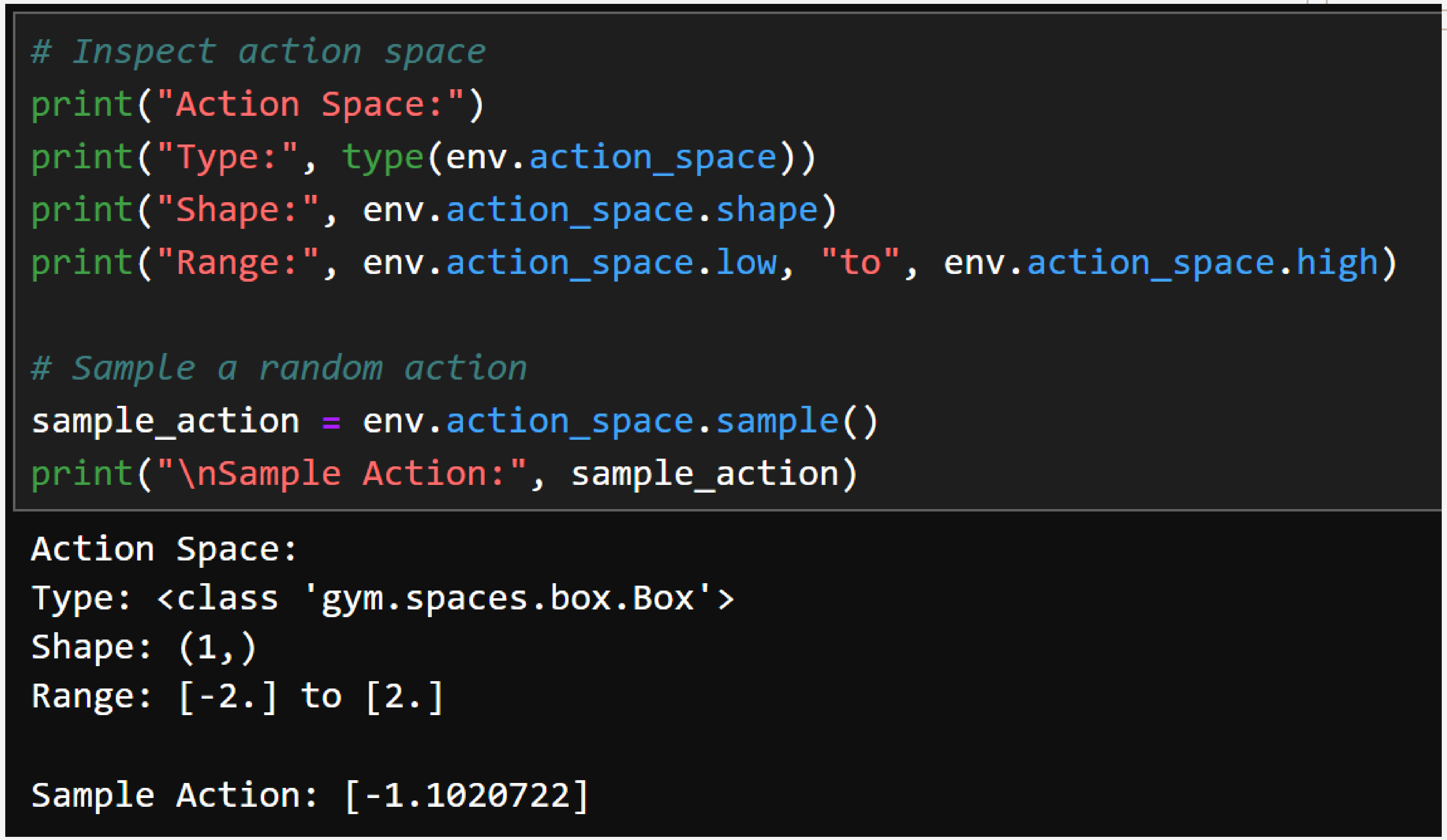
```
Initial Observation (state): [-0.97743886 -0.21121856 -0.1781523 ]

Observation Space:
Type: <class 'gym.spaces.box.Box'>
Shape: (3,)
Range: [-1. -1. -8.] to [1. 1. 8.]
```

Action Space: Torque Control

The Pendulum-v0 environment uses a continuous action space where the agent must decide how much torque (force) to apply to the base of the pendulum at each step.

- The action is a single real-valued number.
- This torque directly affects the pendulum's angular velocity and position.
- The valid torque range is from -2.0 to 2.0.

Since DQN only supports discrete actions, discretization is performed.



```python
# Inspect action space
print("Action Space:")
print("Type:", type(env.action_space))
print("Shape:", env.action_space.shape)
print("Range:", env.action_space.low, "to", env.action_space.high)

# Sample a random action
sample_action = env.action_space.sample()
print("\nSample Action:", sample_action)
```

```
Action Space:
Type: <class 'gym.spaces.box.Box'>
Shape: (1,)
Range: [-2.] to [2.]

Sample Action: [-1.1020722]
```

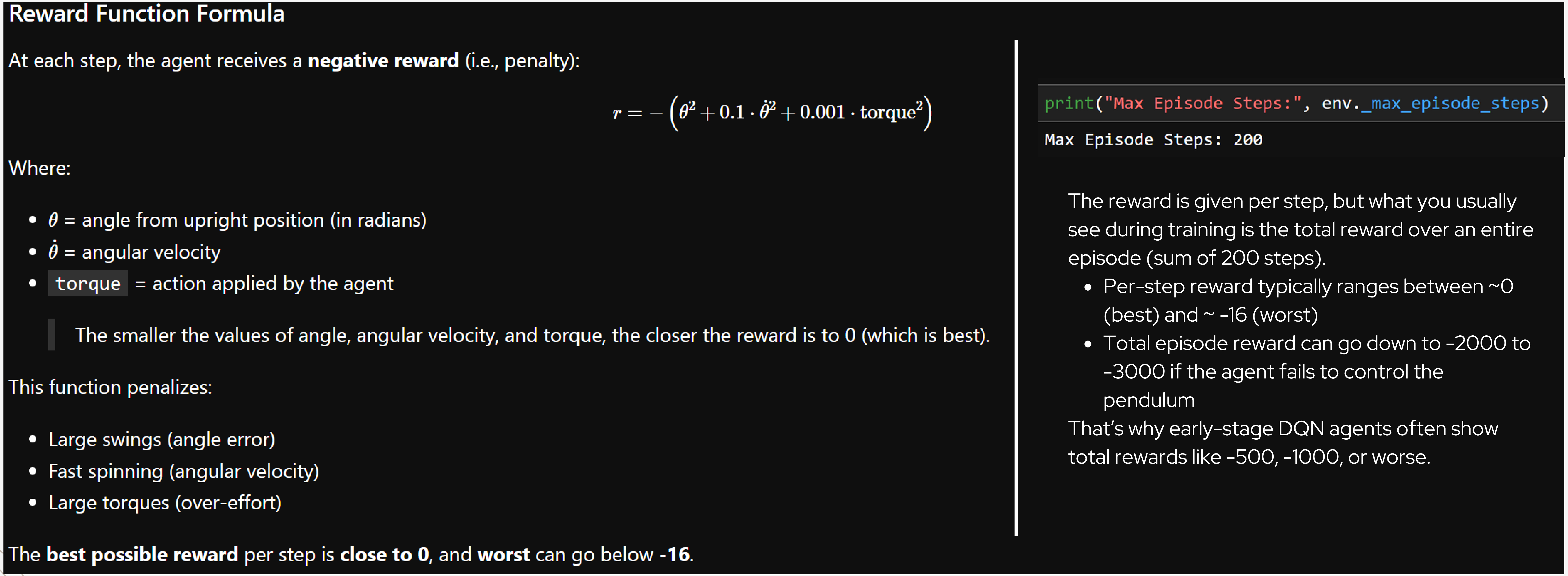
- The action is a 1D continuous Box, meaning a single float value.
- Sampled action -1.10 → The agent is applying a moderate clockwise torque.
- Values near -2.0 apply strong clockwise force, and values near 2.0 apply strong counterclockwise force.

This torque becomes the output of the agent's policy, and we'll discretize this space into fixed steps so DQN can work with it.

Reward Function

In reinforcement learning, the reward function defines the goal. The agent learns by maximizing rewards over time.

In the Pendulum-v0 environment, the goal is to keep the pendulum upright (angle = 0), steady (low angular velocity), and with minimal effort (low torque).



Reward Function Formula

At each step, the agent receives a **negative reward** (i.e., penalty):

$$r = -\left(\theta^2 + 0.1 \cdot \dot{\theta}^2 + 0.001 \cdot \text{torque}^2\right)$$

Where:

- θ = angle from upright position (in radians)
- $\dot{\theta}$ = angular velocity
- `torque` = action applied by the agent

> The smaller the values of angle, angular velocity, and torque, the closer the reward is to 0 (which is best).

This function penalizes:

- Large swings (angle error)
- Fast spinning (angular velocity)
- Large torques (over-effort)

The **best possible reward** per step is **close to 0**, and **worst** can go below **-16**.

```
print("Max Episode Steps:", env._max_episode_steps)
```

```
Max Episode Steps: 200
```

The reward is given per step, but what you usually see during training is the total reward over an entire episode (sum of 200 steps).

- Per-step reward typically ranges between ~0 (best) and ~ -16 (worst)
- Total episode reward can go down to -2000 to -3000 if the agent fails to control the pendulum

That’s why early-stage DQN agents often show total rewards like -500, -1000, or worse.

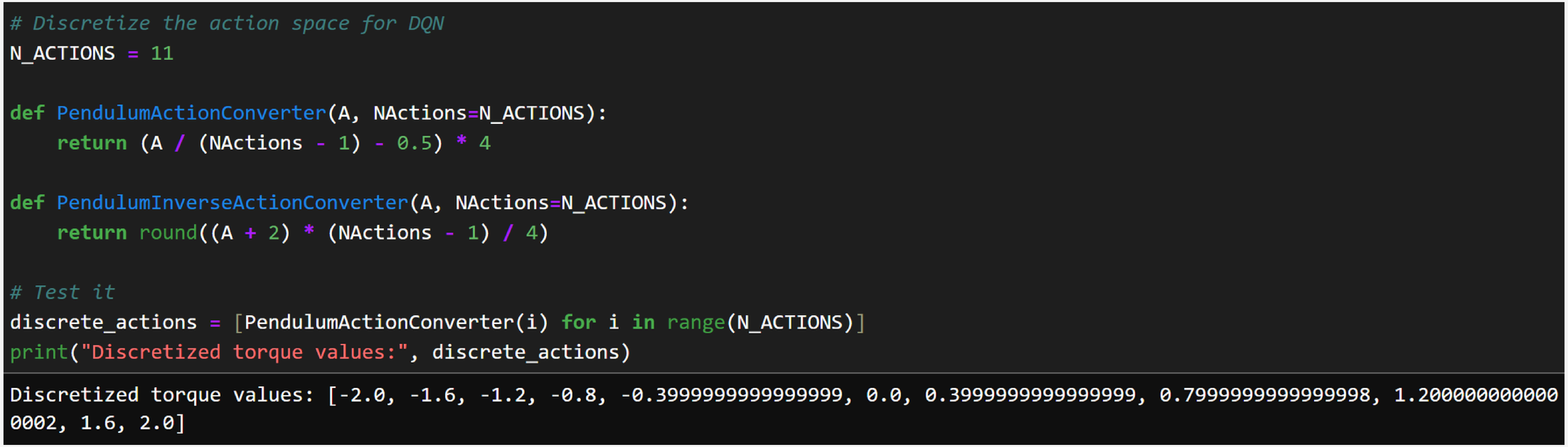
Action Discretization

The Pendulum-v0 environment has a **continuous action space**, where the agent must choose a real-valued torque between **-2.0 and 2.0**.
However, the **Deep Q-Network (DQN)** algorithm only works with **discrete** action spaces, it needs a fixed number of actions to choose from.

Solution: Discretization

We solve this by converting the continuous torque range into a fixed set of **evenly spaced discrete values** (e.g. 11 values):

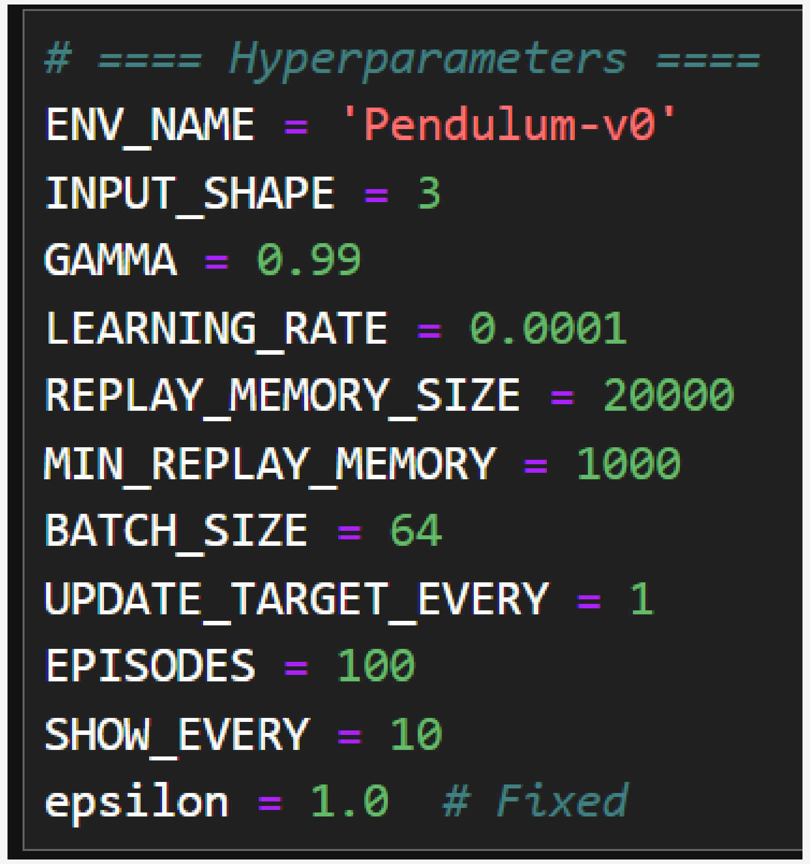
This allows DQN to output a discrete action index (e.g. 0 to 10), which we then **map back to a real torque value**.
Discretization bridges the gap between continuous control tasks and discrete-action RL algorithms like DQN.



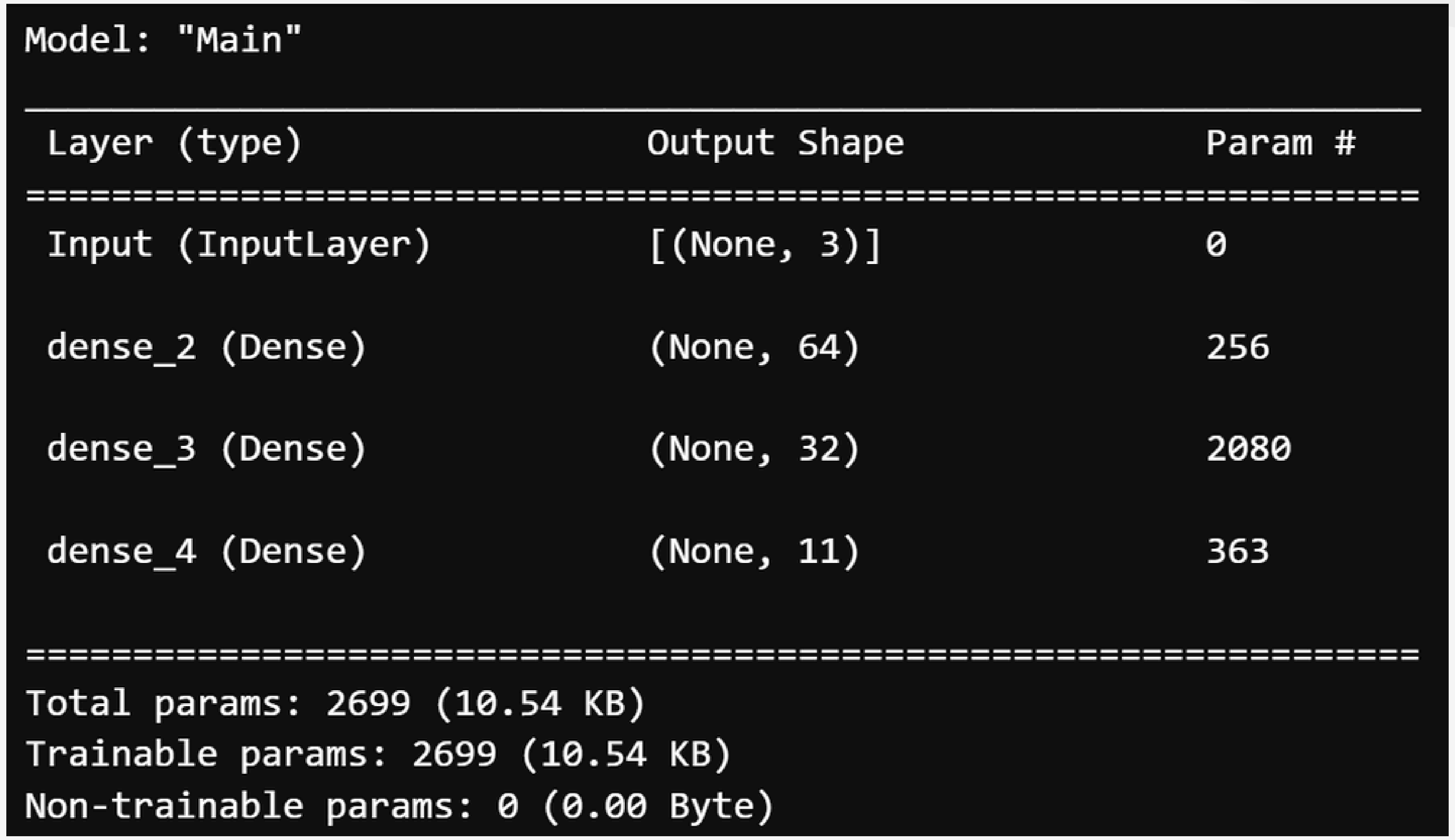
```
# Discretize the action space for DQN
N_ACTIONS = 11

def PendulumActionConverter(A, NActions=N_ACTIONS):
    return (A / (NActions - 1) - 0.5) * 4

def PendulumInverseActionConverter(A, NActions=N_ACTIONS):
    return round((A + 2) * (NActions - 1) / 4)

# Test it
discrete_actions = [PendulumActionConverter(i) for i in range(N_ACTIONS)]
print("Discretized torque values:", discrete_actions)
```

```
Discretized torque values: [-2.0, -1.6, -1.2, -0.8, -0.3999999999999999, 0.0, 0.3999999999999999, 0.7999999999999998, 1.200000000000
0002, 1.6, 2.0]
```

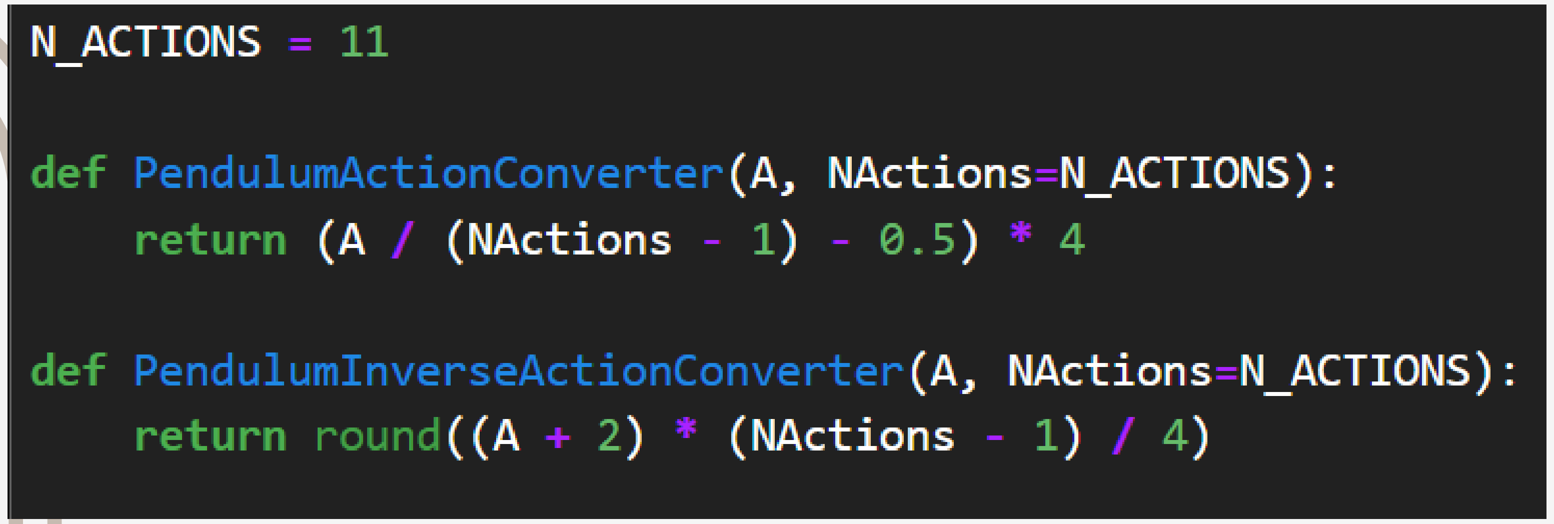
Modelling



```python
# ==== Hyperparameters ====
ENV_NAME = 'Pendulum-v0'
INPUT_SHAPE = 3
GAMMA = 0.99
LEARNING_RATE = 0.0001
REPLAY_MEMORY_SIZE = 20000
MIN_REPLAY_MEMORY = 1000
BATCH_SIZE = 64
UPDATE_TARGET_EVERY = 1
EPISODES = 100
SHOW_EVERY = 10
epsilon = 1.0  # Fixed
```



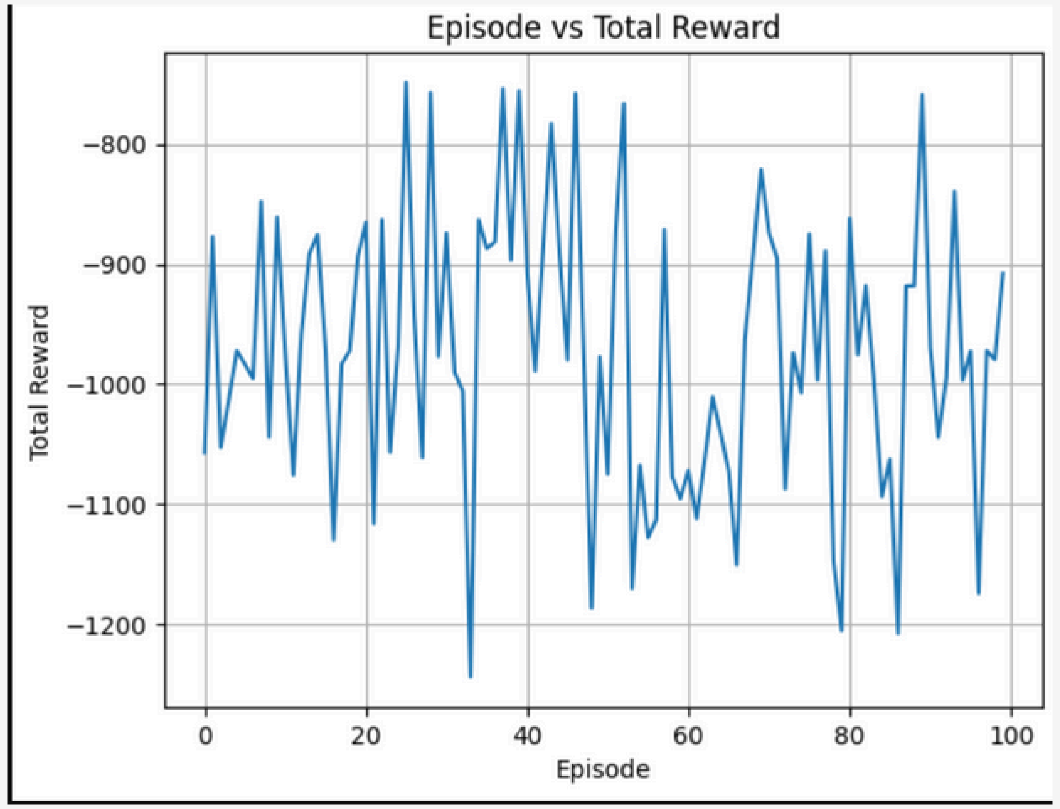
```
Model: "Main"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (InputLayer)          [(None, 3)]               0

 dense_2 (Dense)             (None, 64)                256

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 11)                363

=================================================================
Total params: 2699 (10.54 KB)
Trainable params: 2699 (10.54 KB)
Non-trainable params: 0 (0.00 Byte)
```

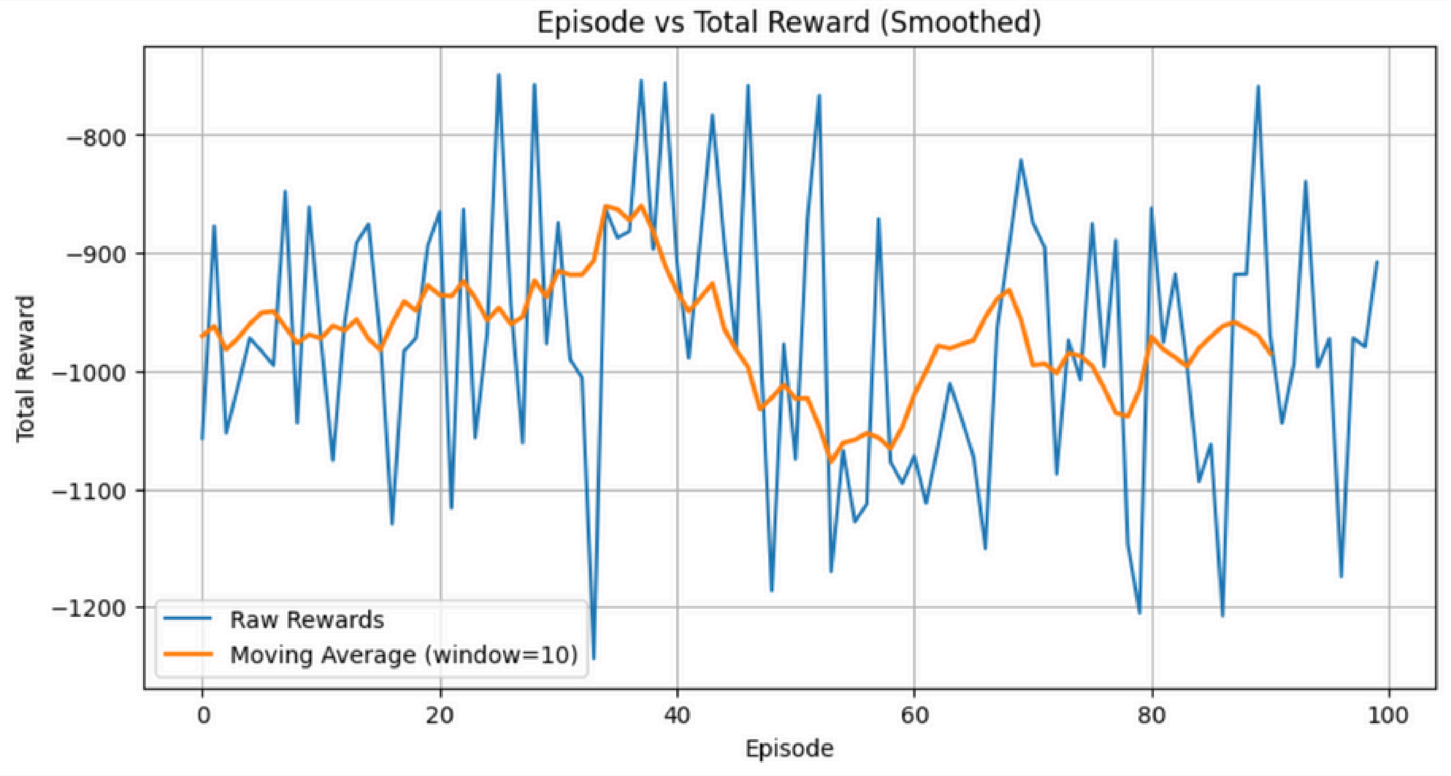


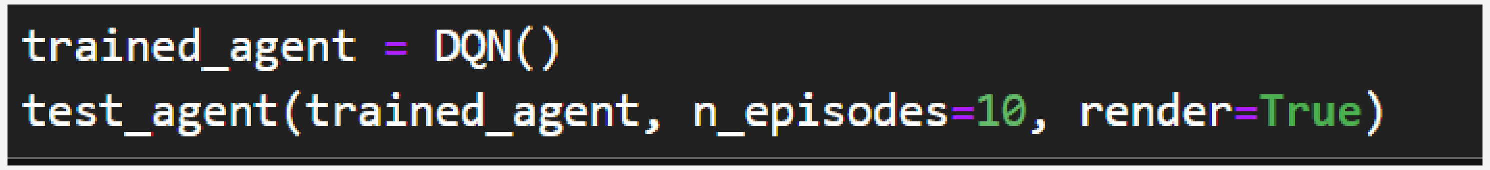
```python
N_ACTIONS = 11

def PendulumActionConverter(A, NActions=N_ACTIONS):
    return (A / (NActions - 1) - 0.5) * 4

def PendulumInverseActionConverter(A, NActions=N_ACTIONS):
    return round((A + 2) * (NActions - 1) / 4)
```

- LR: 0.0001
- Replay Memory Size: 20000
- N Actions: 11
- Model Architecture → 2 layers: 64, 32

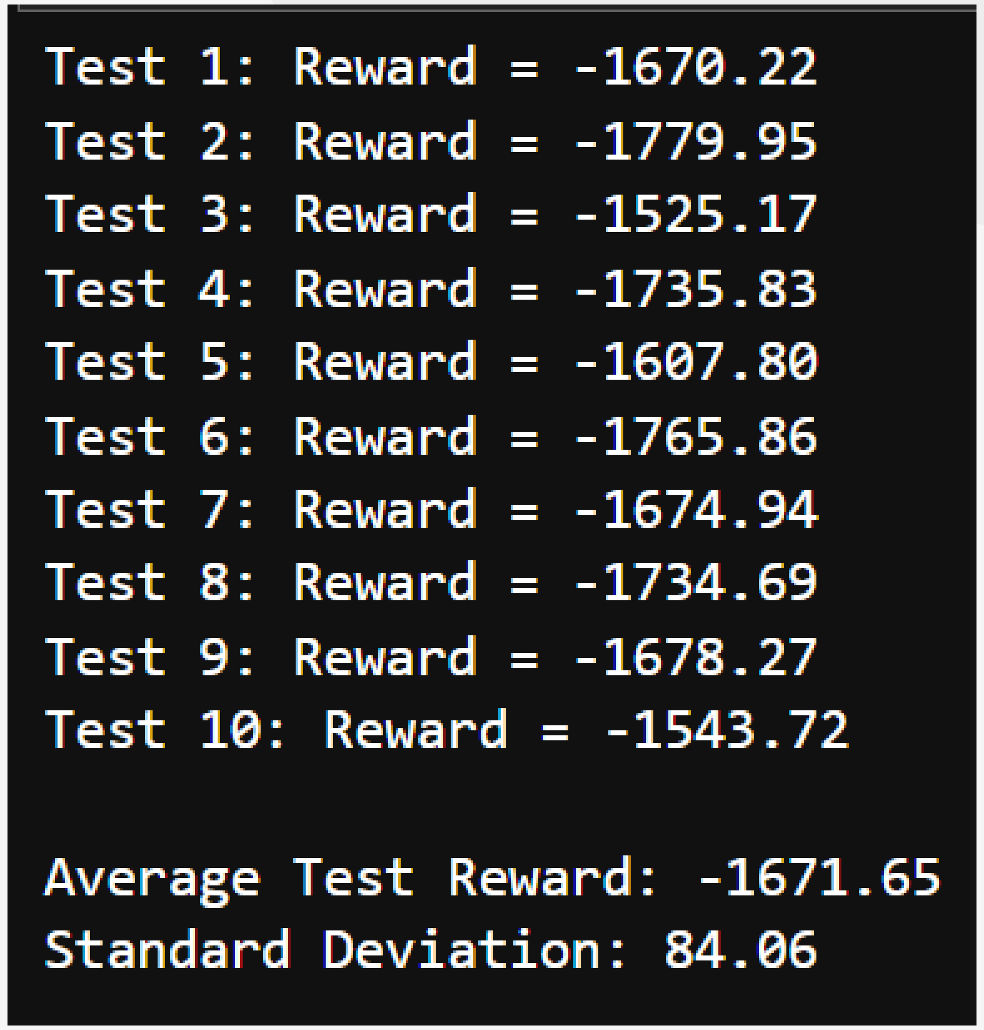
Modelling



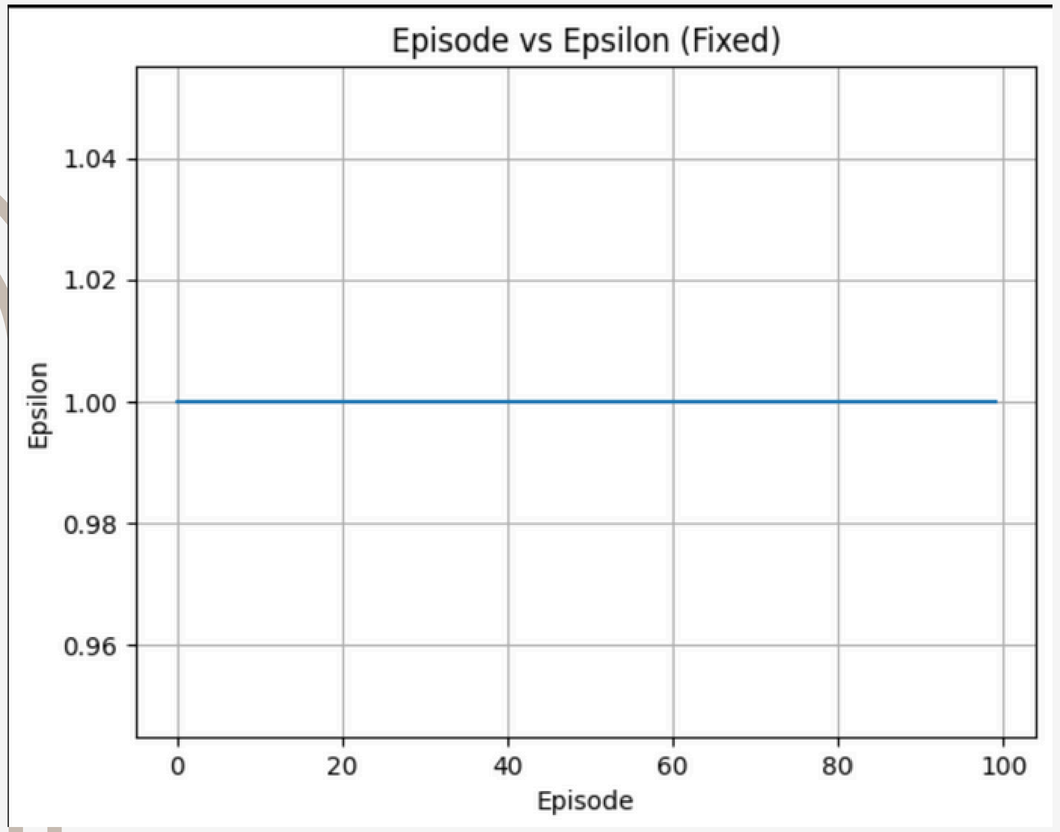




```
trained_agent = DQN()
test_agent(trained_agent, n_episodes=10, render=True)
```

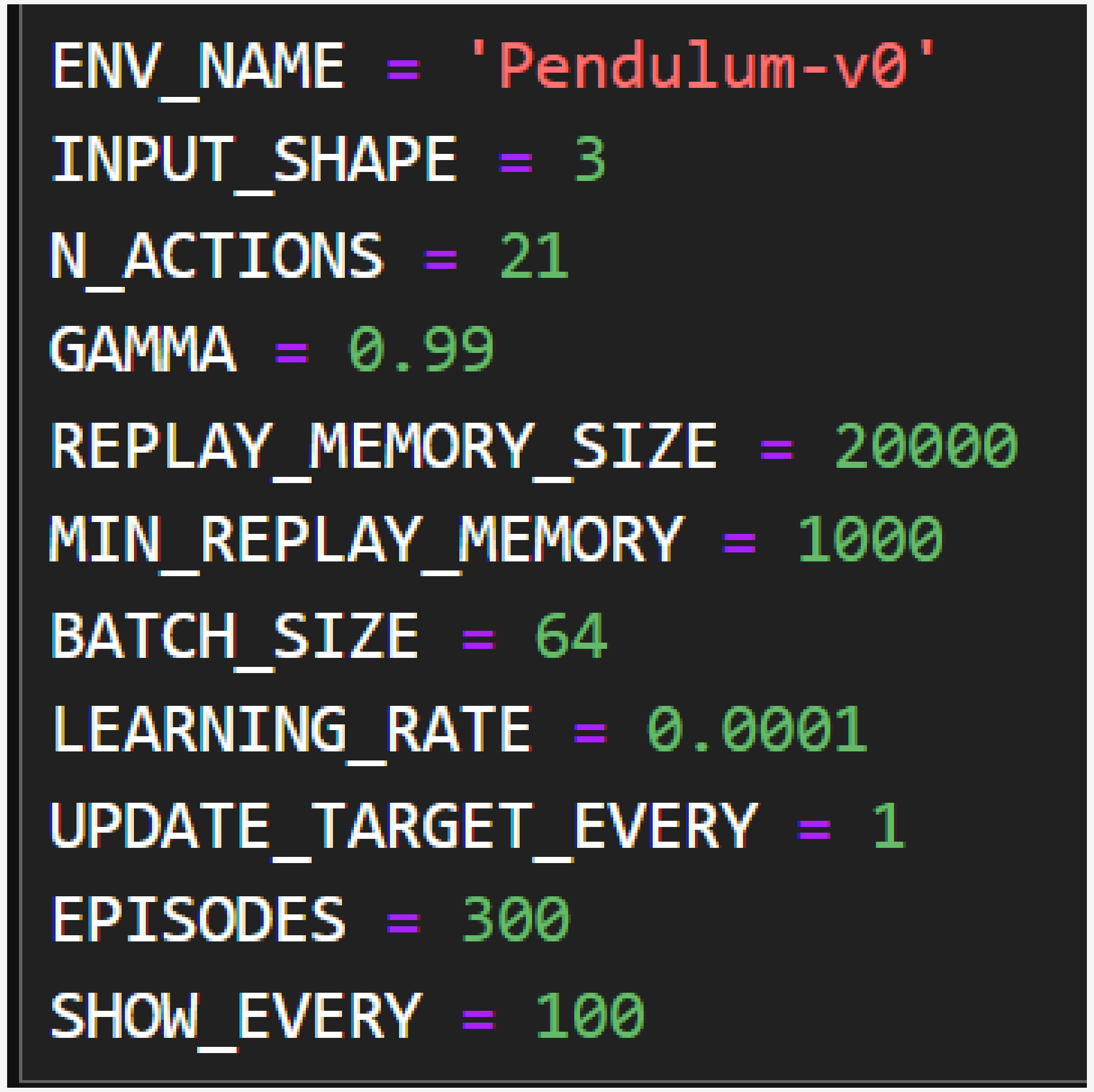


```
Test 1: Reward = -1670.22
Test 2: Reward = -1779.95
Test 3: Reward = -1525.17
Test 4: Reward = -1735.83
Test 5: Reward = -1607.80
Test 6: Reward = -1765.86
Test 7: Reward = -1674.94
Test 8: Reward = -1734.69
Test 9: Reward = -1678.27
Test 10: Reward = -1543.72

Average Test Reward: -1671.65
Standard Deviation: 84.06
```

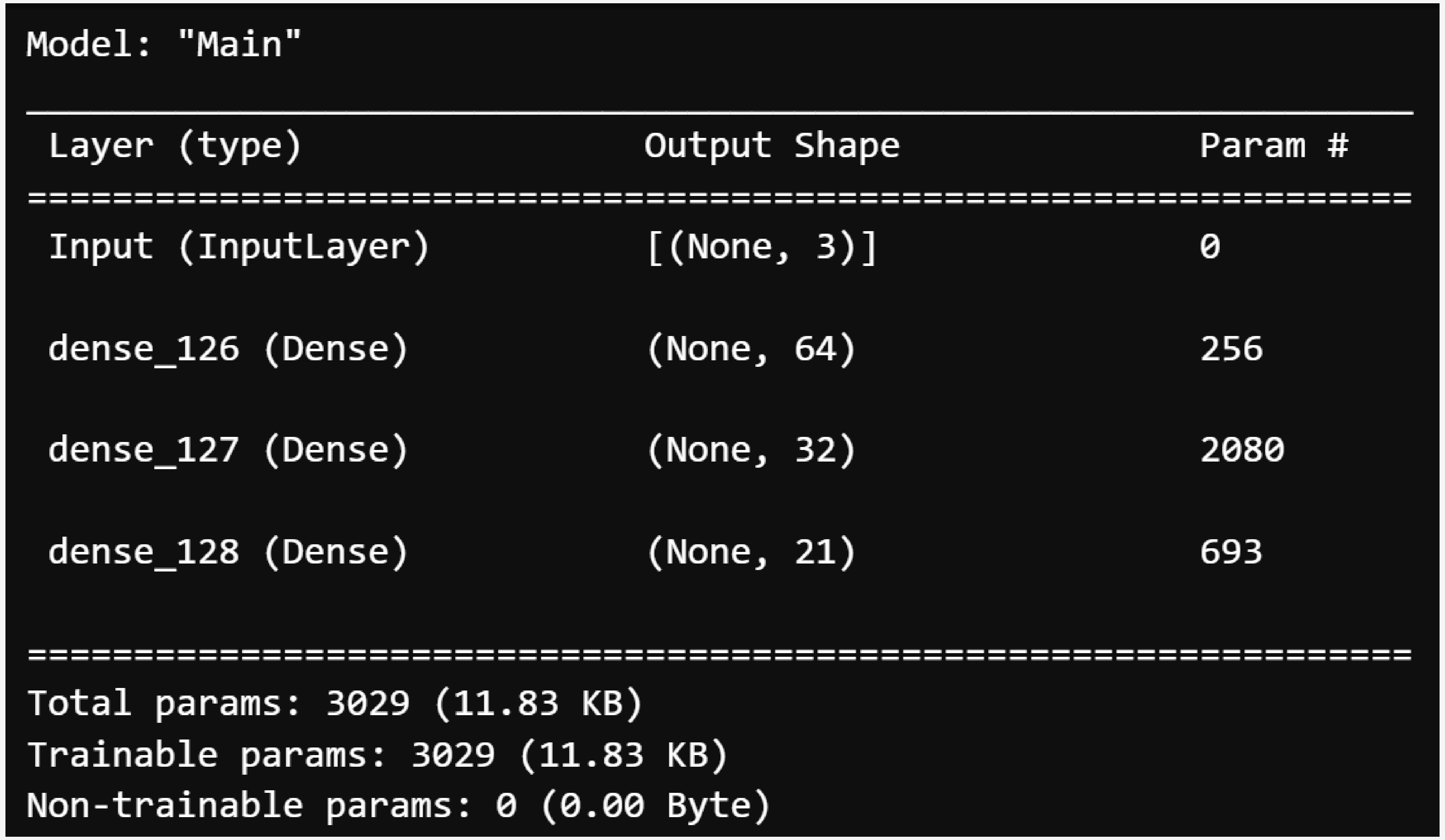


- Episode vs Total Reward: Highly volatile, fluctuating between approximately -800 and -1200 across episodes → agent is not improving its policy over time.
- Moving Average: Unstable
- Fixed Epsilon
- High negative Reward (-1671). Far away from ideal value of 0.

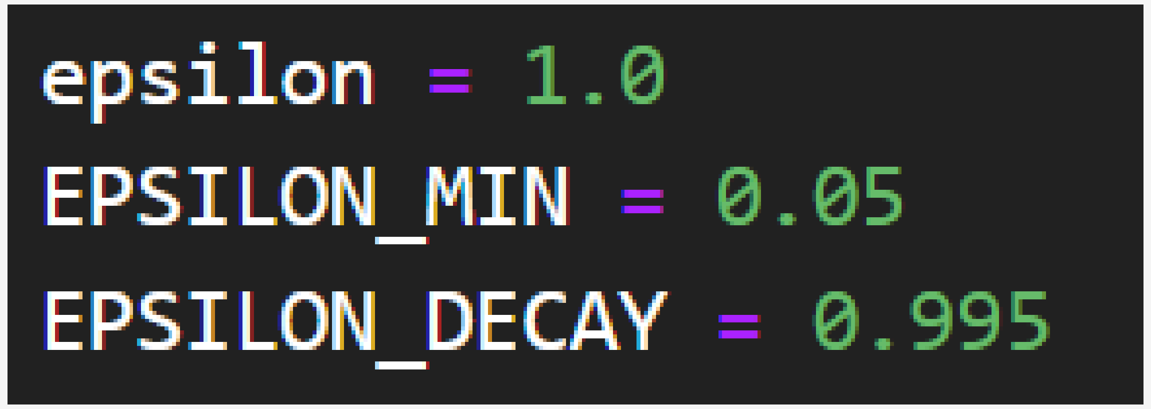
Model Improvement 1 – Epsilon Decay



```
ENV_NAME = 'Pendulum-v0'
INPUT_SHAPE = 3
N_ACTIONS = 21
GAMMA = 0.99
REPLAY_MEMORY_SIZE = 20000
MIN_REPLAY_MEMORY = 1000
BATCH_SIZE = 64
LEARNING_RATE = 0.0001
UPDATE_TARGET_EVERY = 1
EPISODES = 300
SHOW_EVERY = 100
```



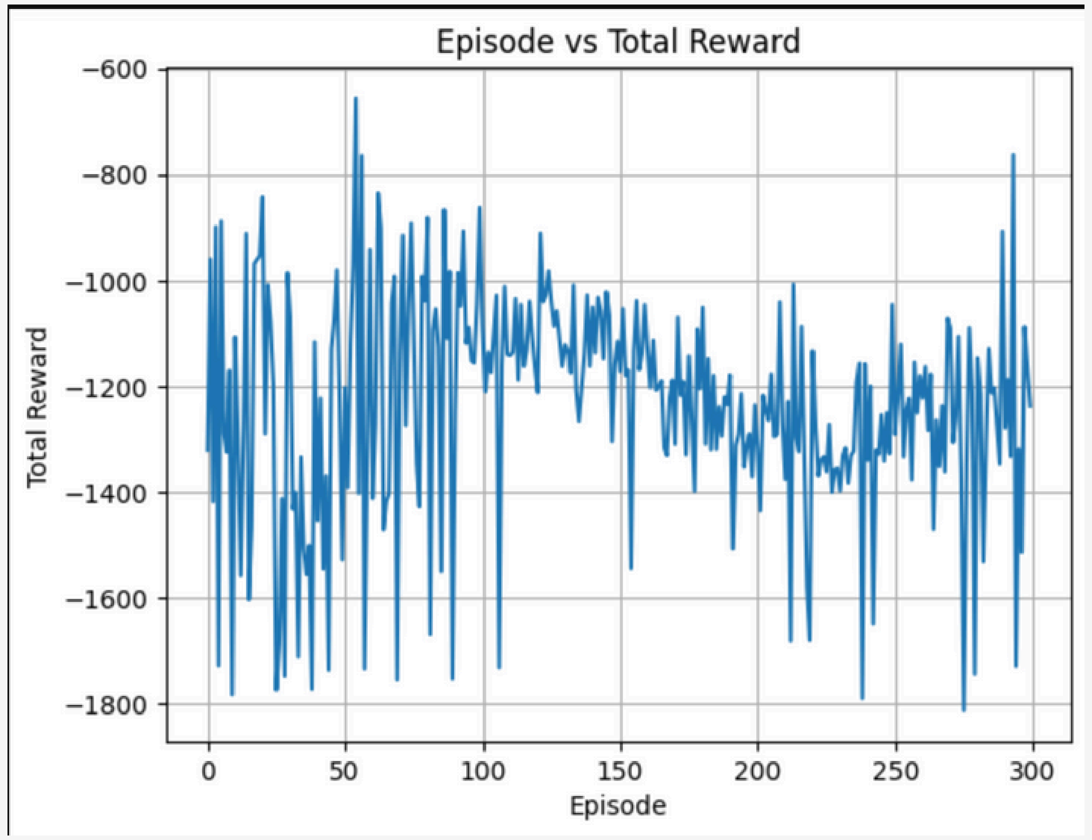
```
Model: "Main"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (InputLayer)          [(None, 3)]               0

 dense_126 (Dense)           (None, 64)                256

 dense_127 (Dense)           (None, 32)                2080

 dense_128 (Dense)           (None, 21)                693

=================================================================
Total params: 3029 (11.83 KB)
Trainable params: 3029 (11.83 KB)
Non-trainable params: 0 (0.00 Byte)
```

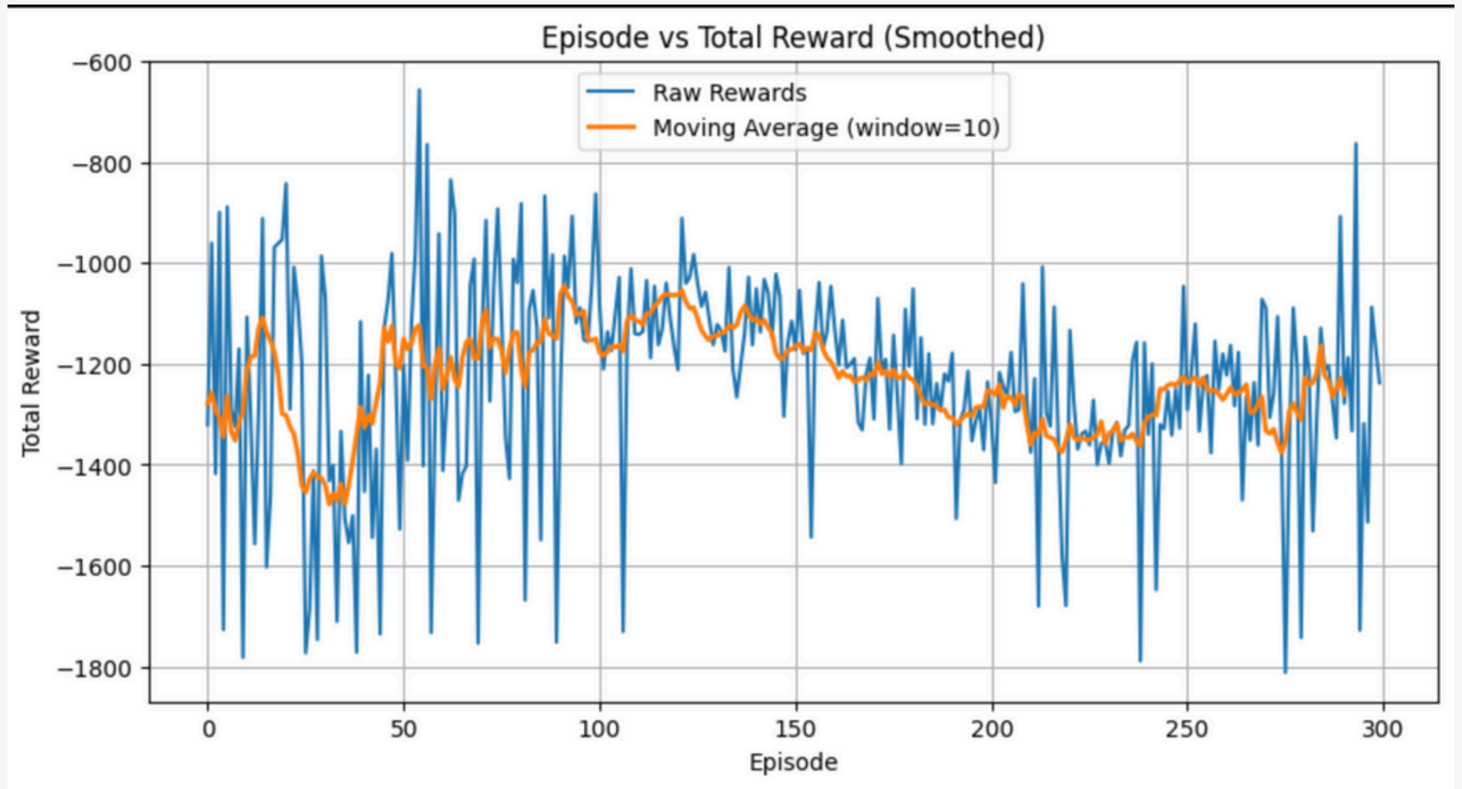


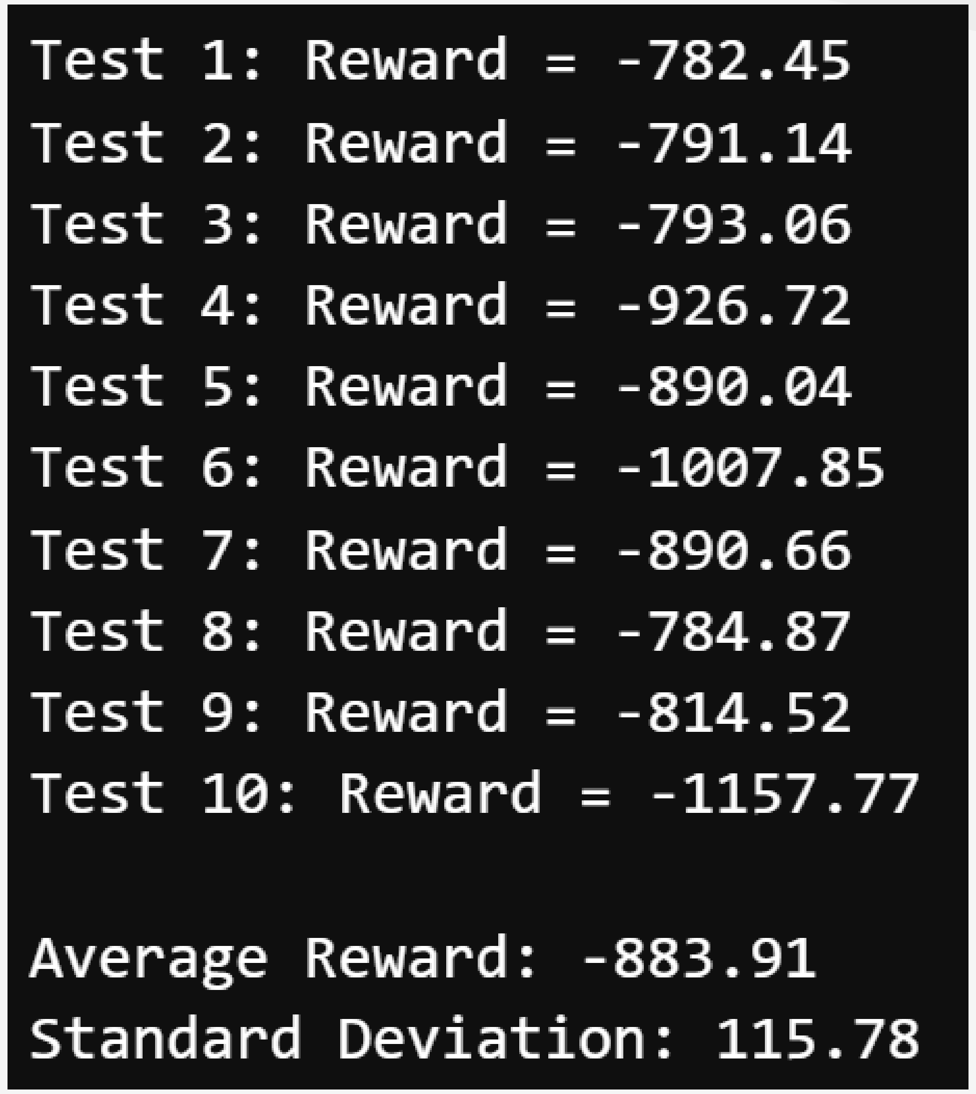
```
epsilon = 1.0
EPSILON_MIN = 0.05
EPSILON_DECAY = 0.995
```

- Introduce epsilon decay
- Increase episodes to 300 in hopes to see convergence
- N Actions: 21 (standard for Pendulum) to reduce reward volatility and standard deviation

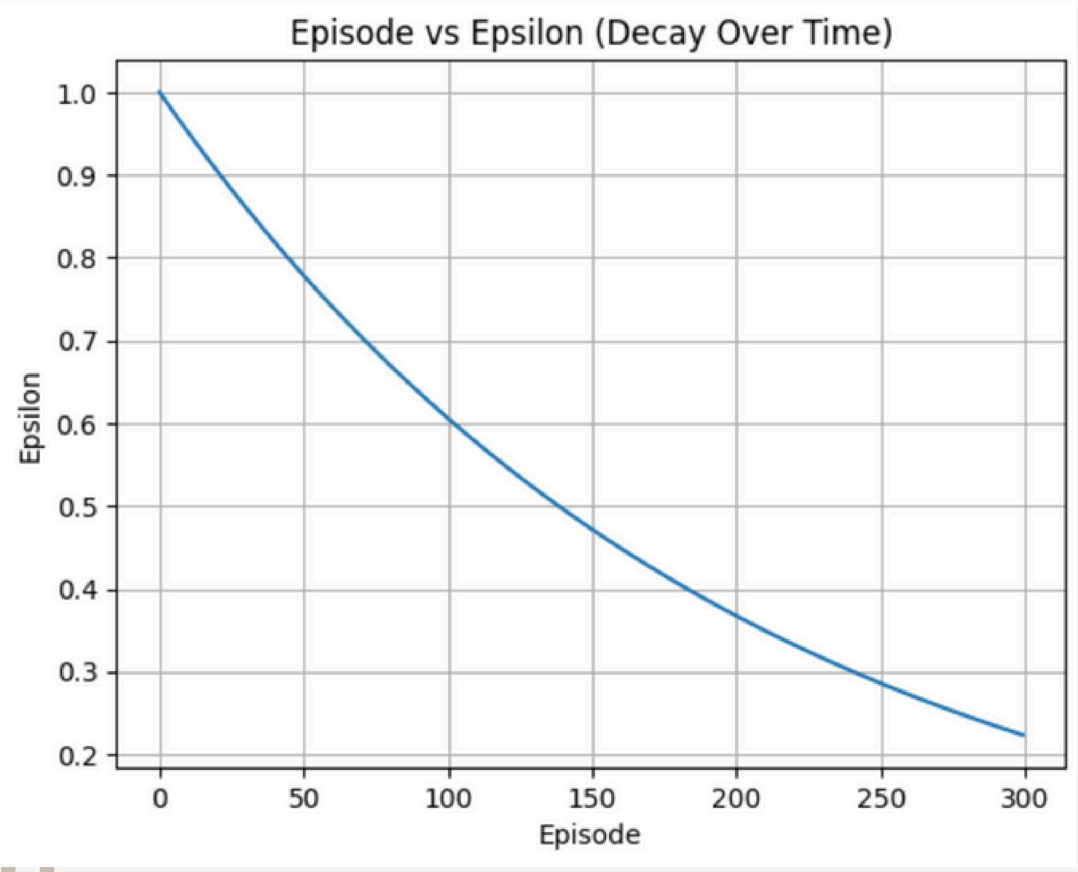
Model Improvement 1 – Epsilon Decay

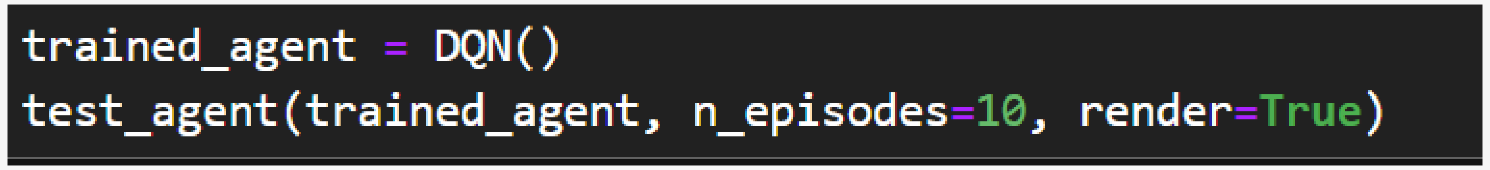






```
Test 1: Reward = -782.45
Test 2: Reward = -791.14
Test 3: Reward = -793.06
Test 4: Reward = -926.72
Test 5: Reward = -890.04
Test 6: Reward = -1007.85
Test 7: Reward = -890.66
Test 8: Reward = -784.87
Test 9: Reward = -814.52
Test 10: Reward = -1157.77

Average Reward: -883.91
Standard Deviation: 115.78
```

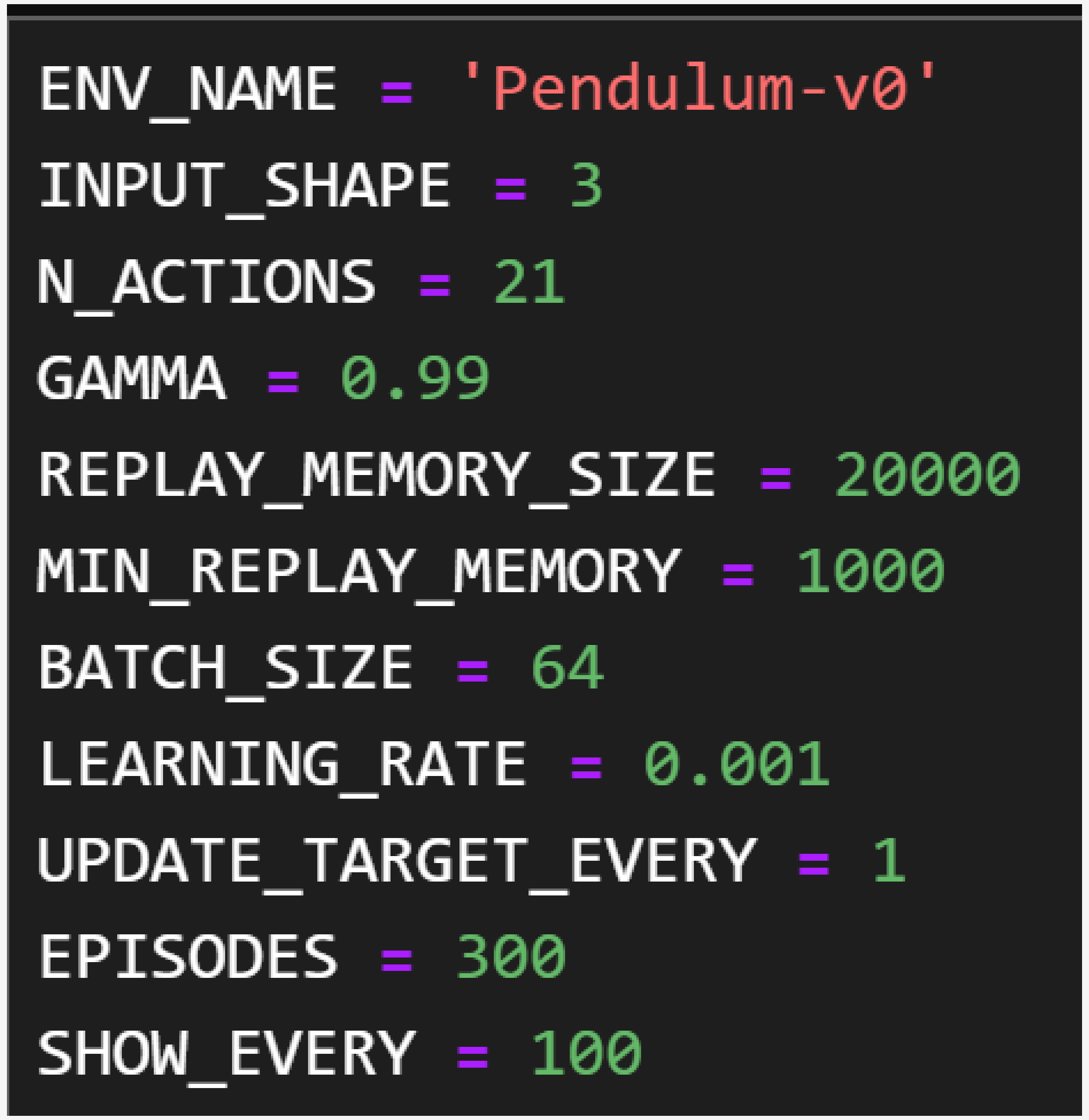




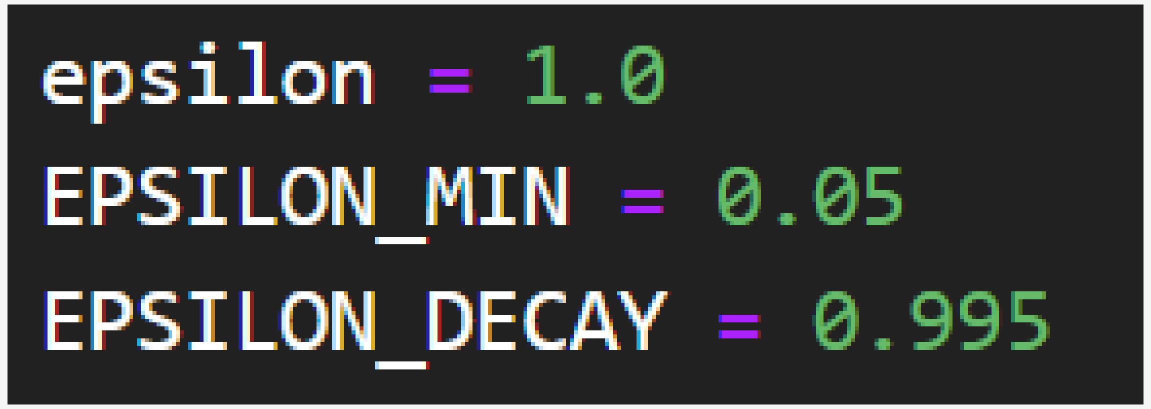
```
trained_agent = DQN()
test_agent(trained_agent, n_episodes=10, render=True)
```

- Episode vs Total Reward: Lesser volatility but plateau not seen. Noisy.
- Moving Average: Partial Improvement but no upward climb.
- Epsilon Deacy is working is expected.
- Lower Negative Reward than baseline → agent can now maintain the pendulum in a more stable position. SD is still high so refinement can be made.

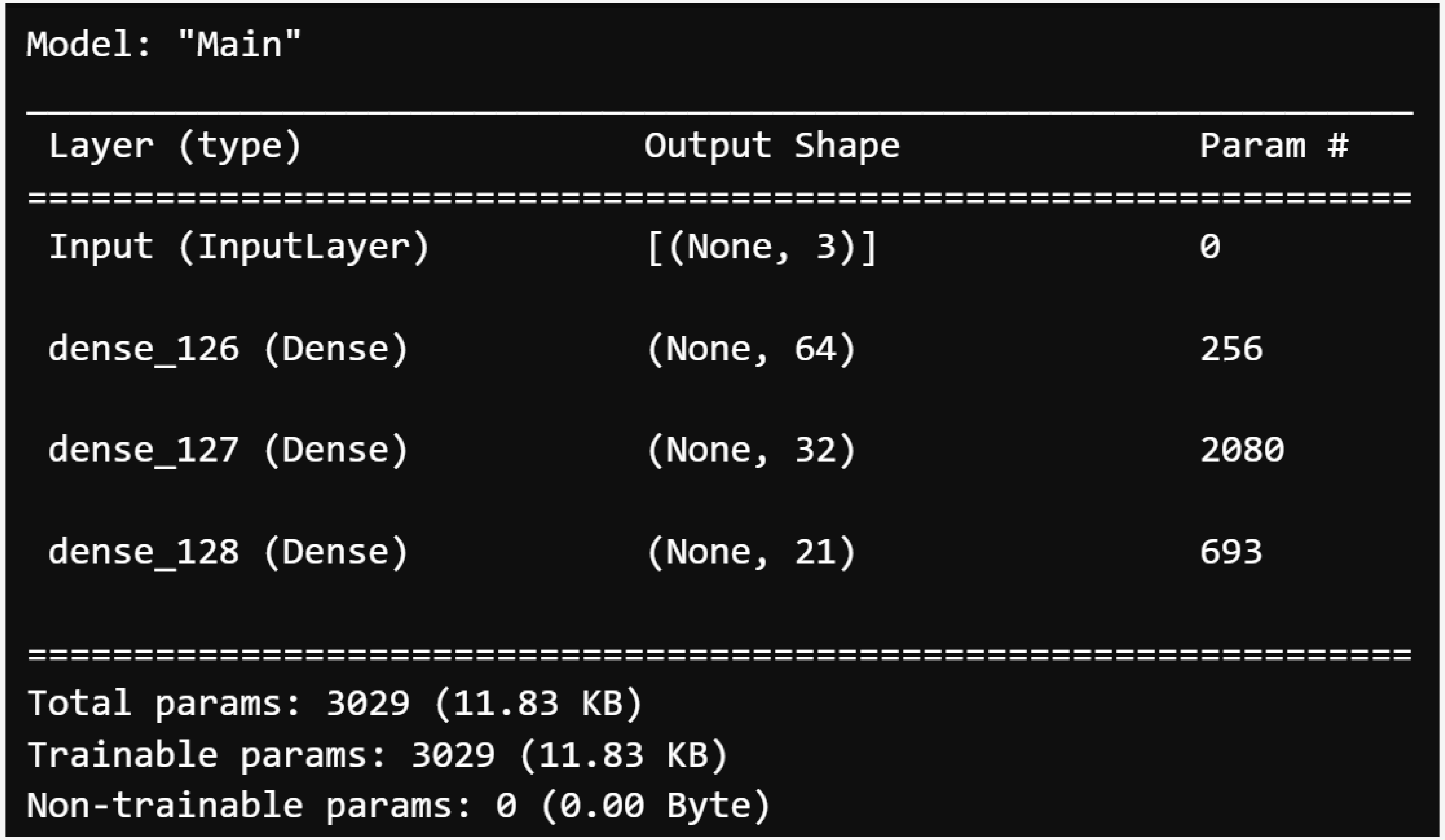
Model Improvement 2 - Speed up LR



```
ENV_NAME = 'Pendulum-v0'
INPUT_SHAPE = 3
N_ACTIONS = 21
GAMMA = 0.99
REPLAY_MEMORY_SIZE = 20000
MIN_REPLAY_MEMORY = 1000
BATCH_SIZE = 64
LEARNING_RATE = 0.001
UPDATE_TARGET_EVERY = 1
EPISODES = 300
SHOW_EVERY = 100
```



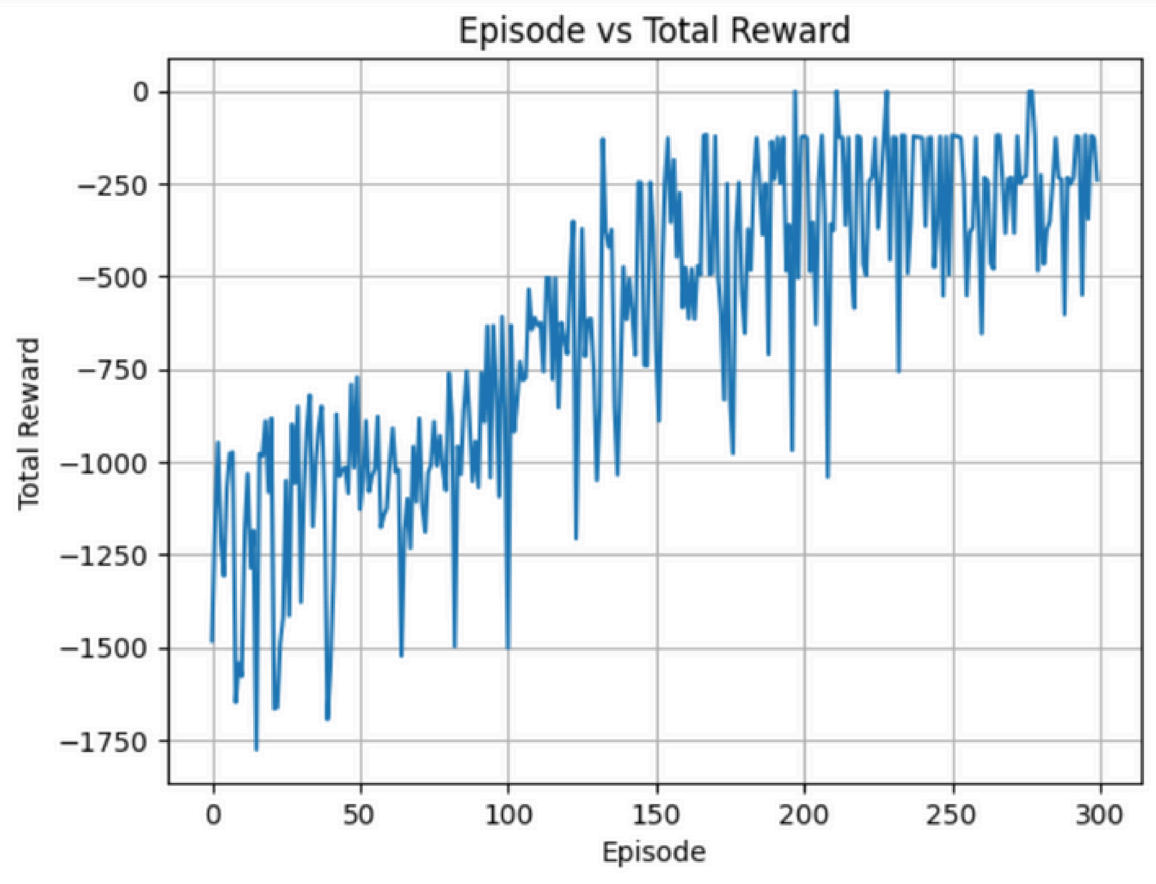
```
epsilon = 1.0
EPSILON_MIN = 0.05
EPSILON_DECAY = 0.995
```

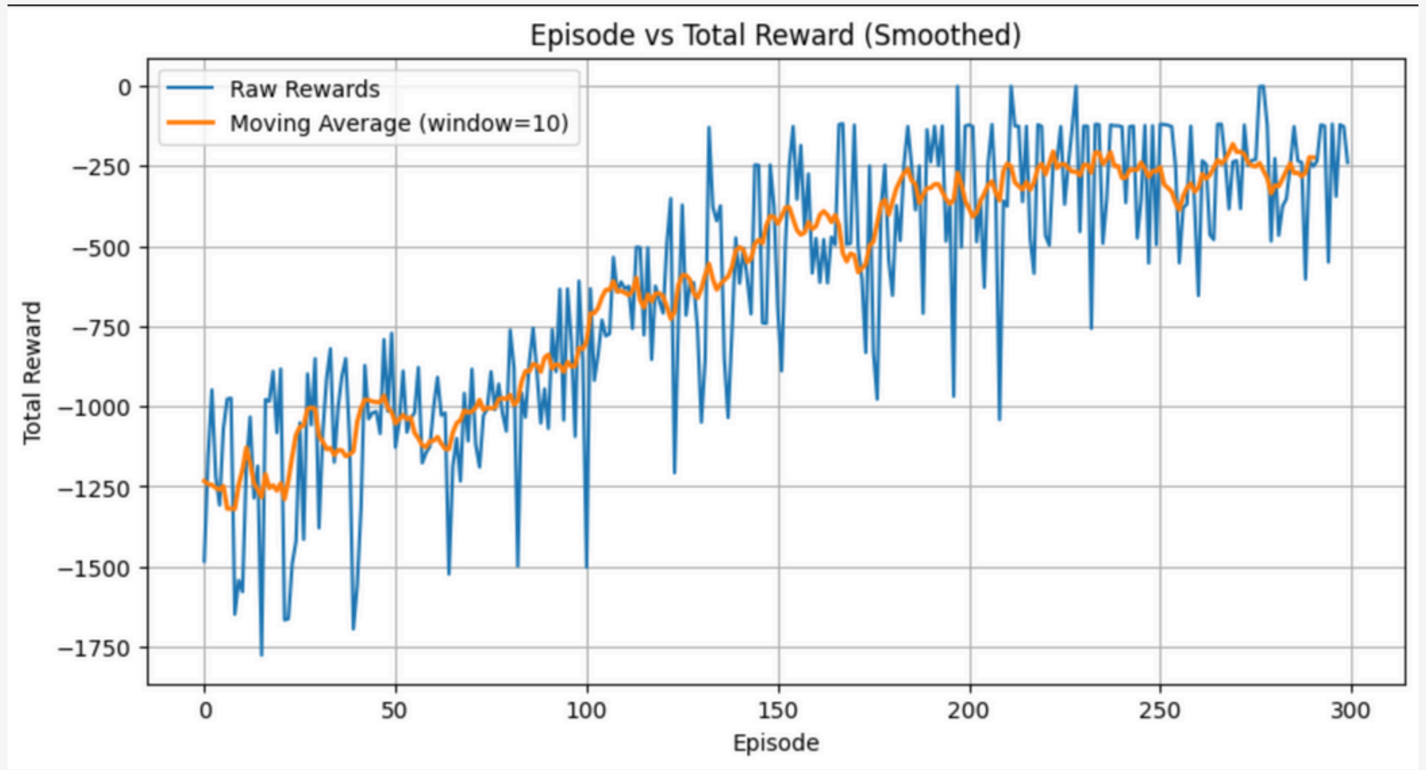


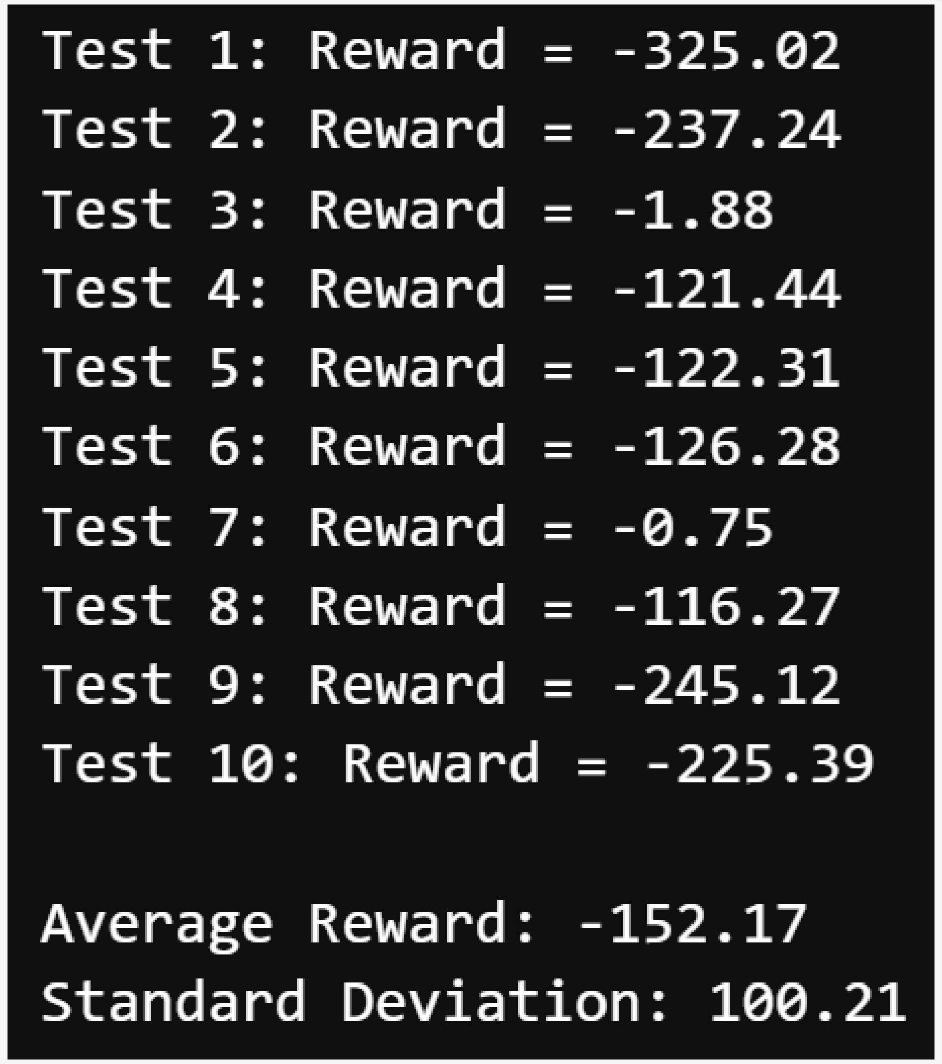
```
Model: "Main"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (InputLayer)          [(None, 3)]               0

 dense_126 (Dense)           (None, 64)                256

 dense_127 (Dense)           (None, 32)                2080

 dense_128 (Dense)           (None, 21)                693

=================================================================
Total params: 3029 (11.83 KB)
Trainable params: 3029 (11.83 KB)
Non-trainable params: 0 (0.00 Byte)
```

- Increase the speed and strength of Q-value updates to help the agent adapt more quickly to valuable state–action pairs, while keeping the network architecture and replay memory unchanged to isolate the effect of learning rate.
    - LR: 0.0001 → 0.001

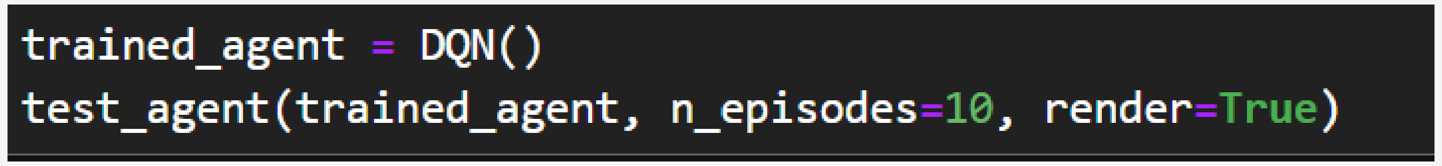
Model Improvement 2 - Speed up LR



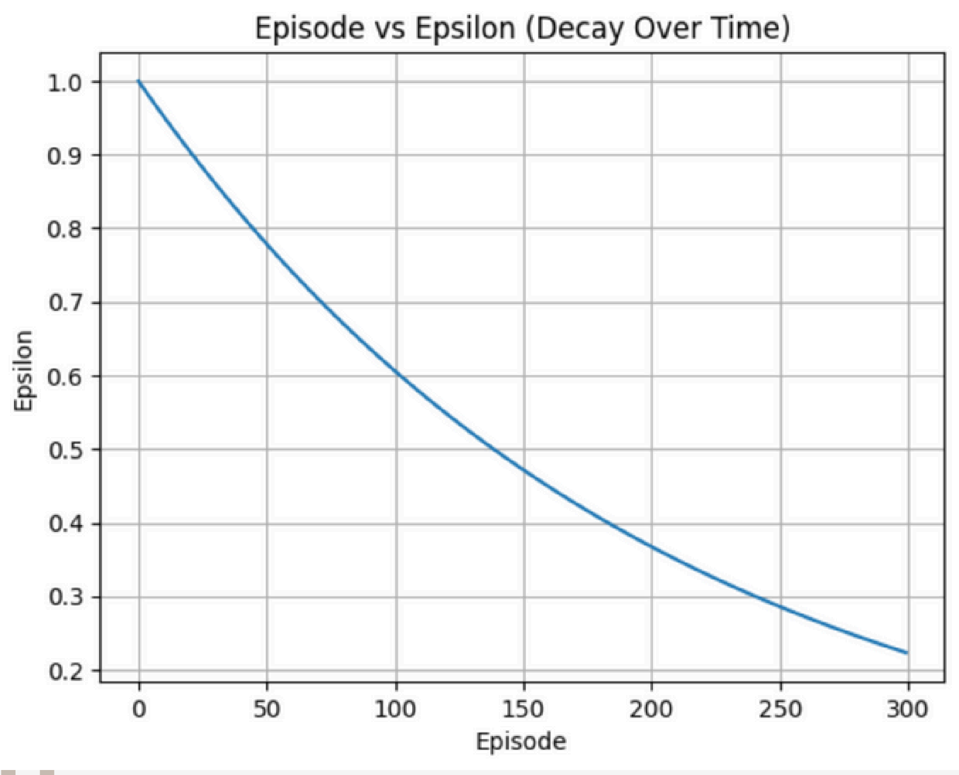




```
Test 1: Reward = -325.02
Test 2: Reward = -237.24
Test 3: Reward = -1.88
Test 4: Reward = -121.44
Test 5: Reward = -122.31
Test 6: Reward = -126.28
Test 7: Reward = -0.75
Test 8: Reward = -116.27
Test 9: Reward = -245.12
Test 10: Reward = -225.39

Average Reward: -152.17
Standard Deviation: 100.21
```

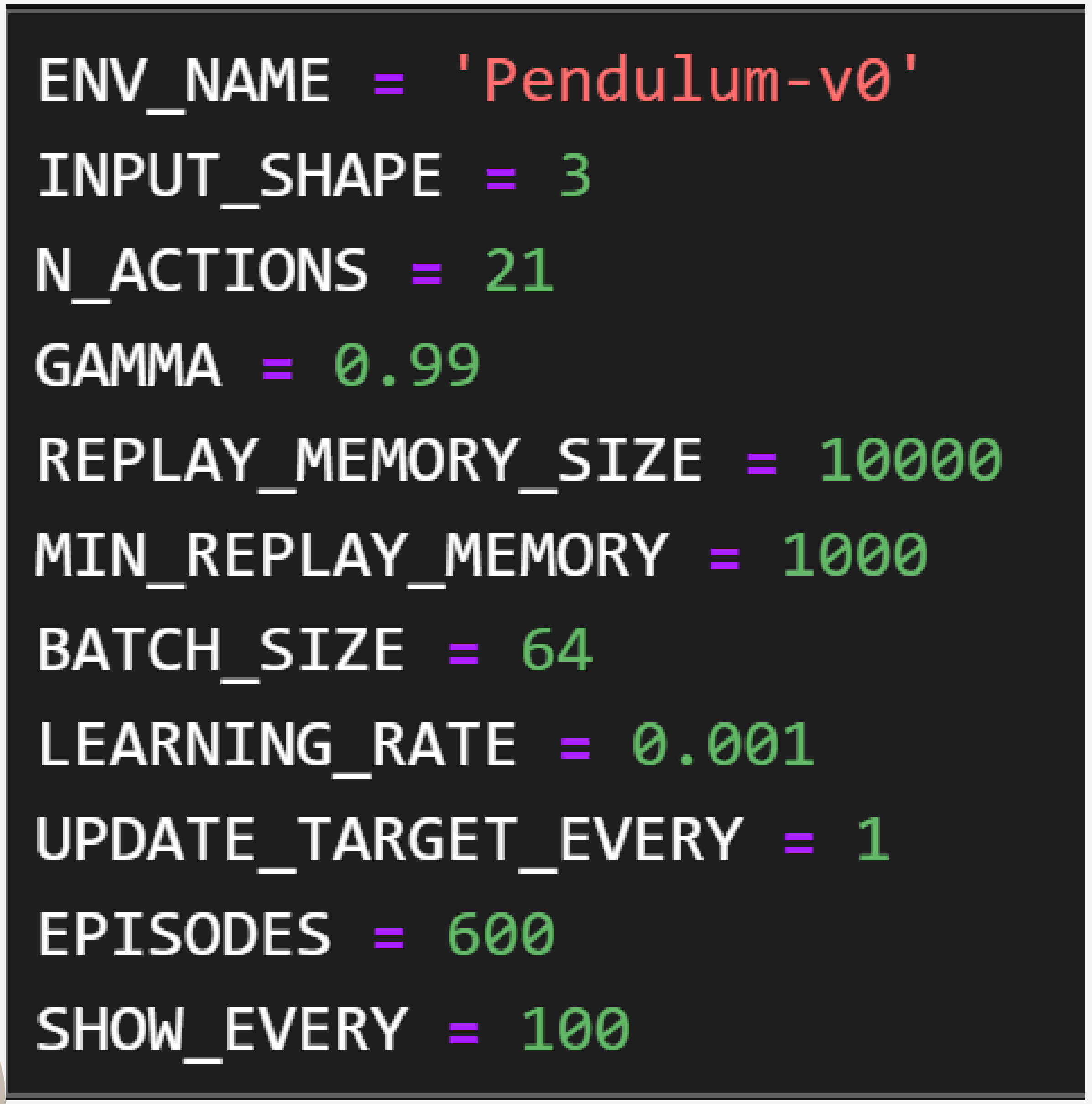


```
trained_agent = DQN()
test_agent(trained_agent, n_episodes=10, render=True)
```

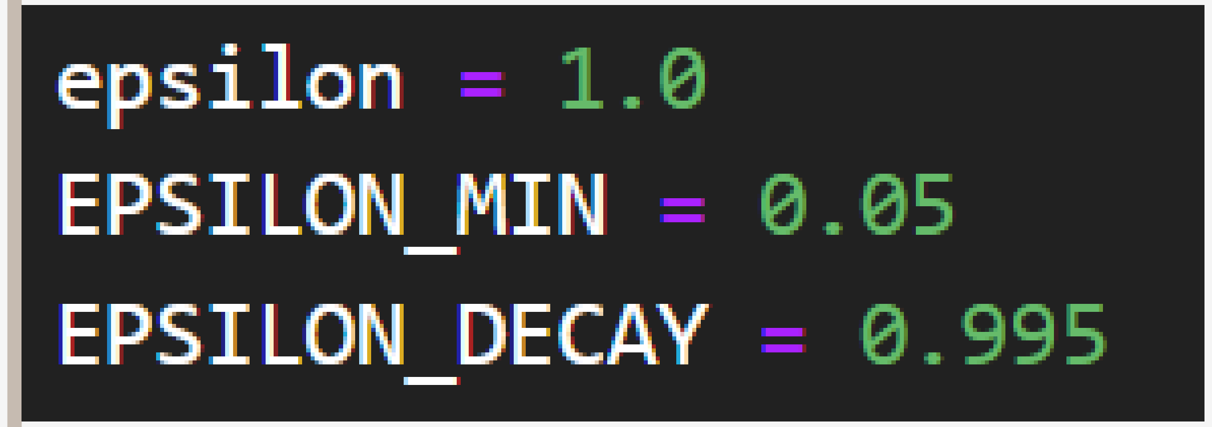


- Episode vs Total Reward: Volatility is reduced. (increased learning rate allowed the agent to update Q-values more effectively and exploit learned strategies sooner)
- Moving Average: Rises steadily, reaching close to -200 by the final episodes.
- Epsilon Deacy is working is expected.
- Dramatic improvement in average reward compared to Improvement 1 (-884 → -152), showing that the agent can maintain the pendulum near upright for most of the episode.
    - Better aniticipation and correction of angular deviation

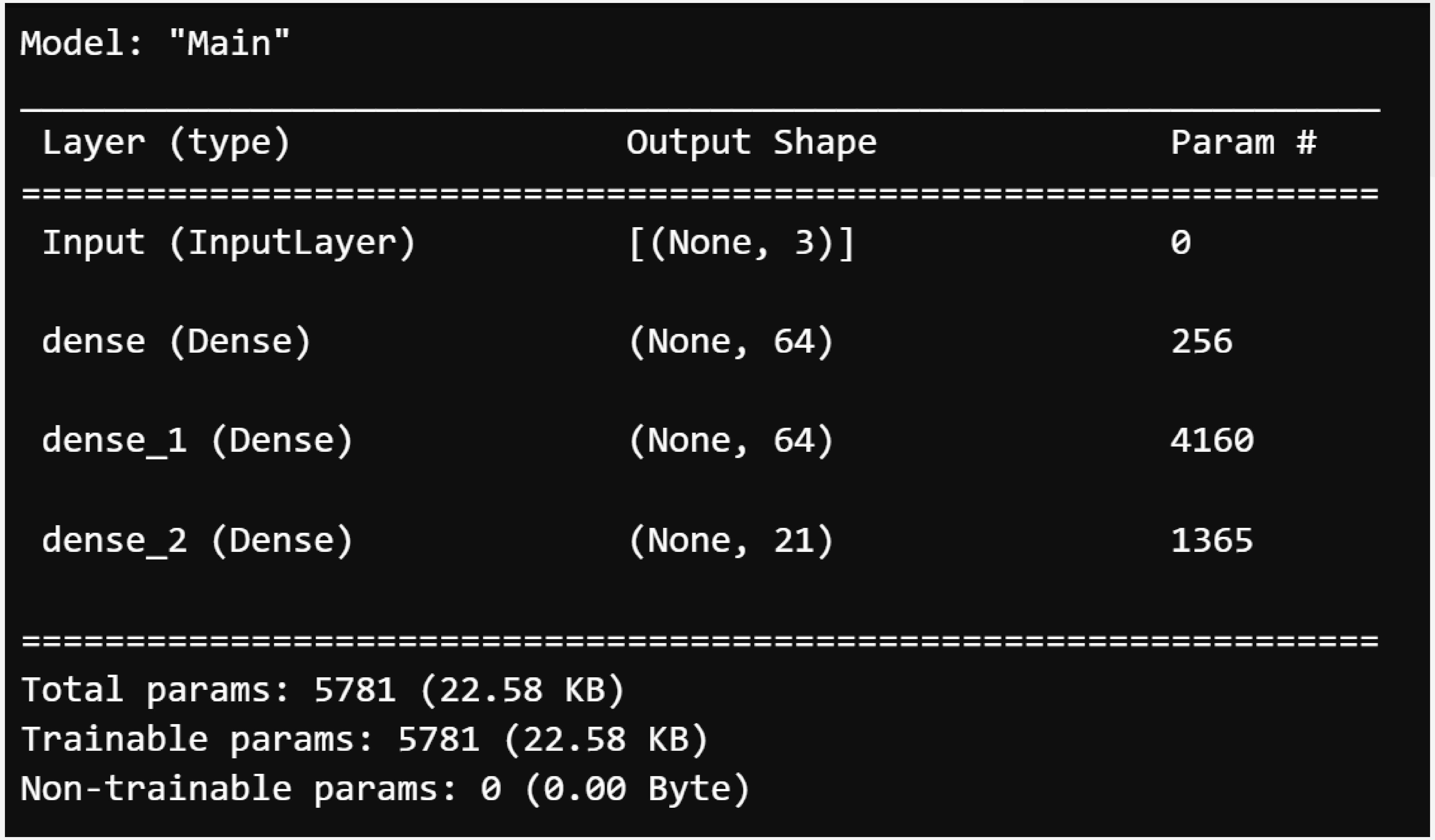
Final Model



```
ENV_NAME = 'Pendulum-v0'
INPUT_SHAPE = 3
N_ACTIONS = 21
GAMMA = 0.99
REPLAY_MEMORY_SIZE = 10000
MIN_REPLAY_MEMORY = 1000
BATCH_SIZE = 64
LEARNING_RATE = 0.001
UPDATE_TARGET_EVERY = 1
EPISODES = 600
SHOW_EVERY = 100
```



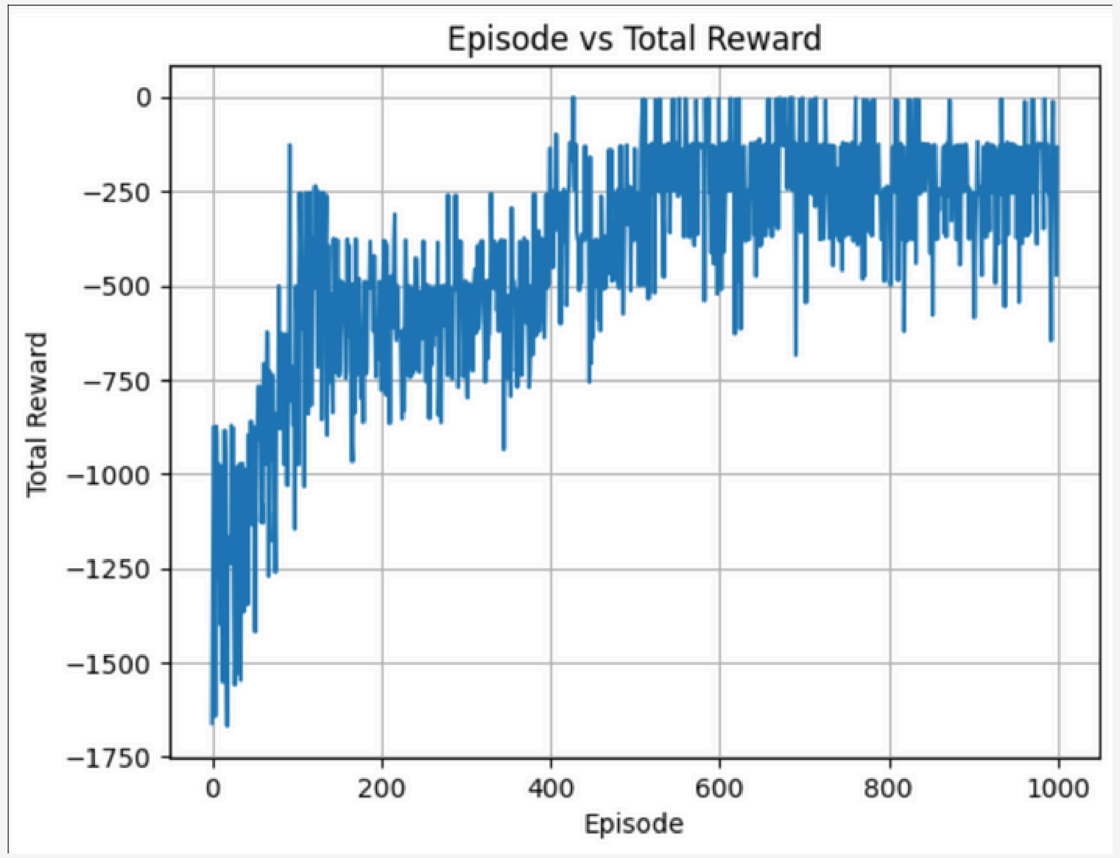
```
epsilon = 1.0
EPSILON_MIN = 0.05
EPSILON_DECAY = 0.995
```

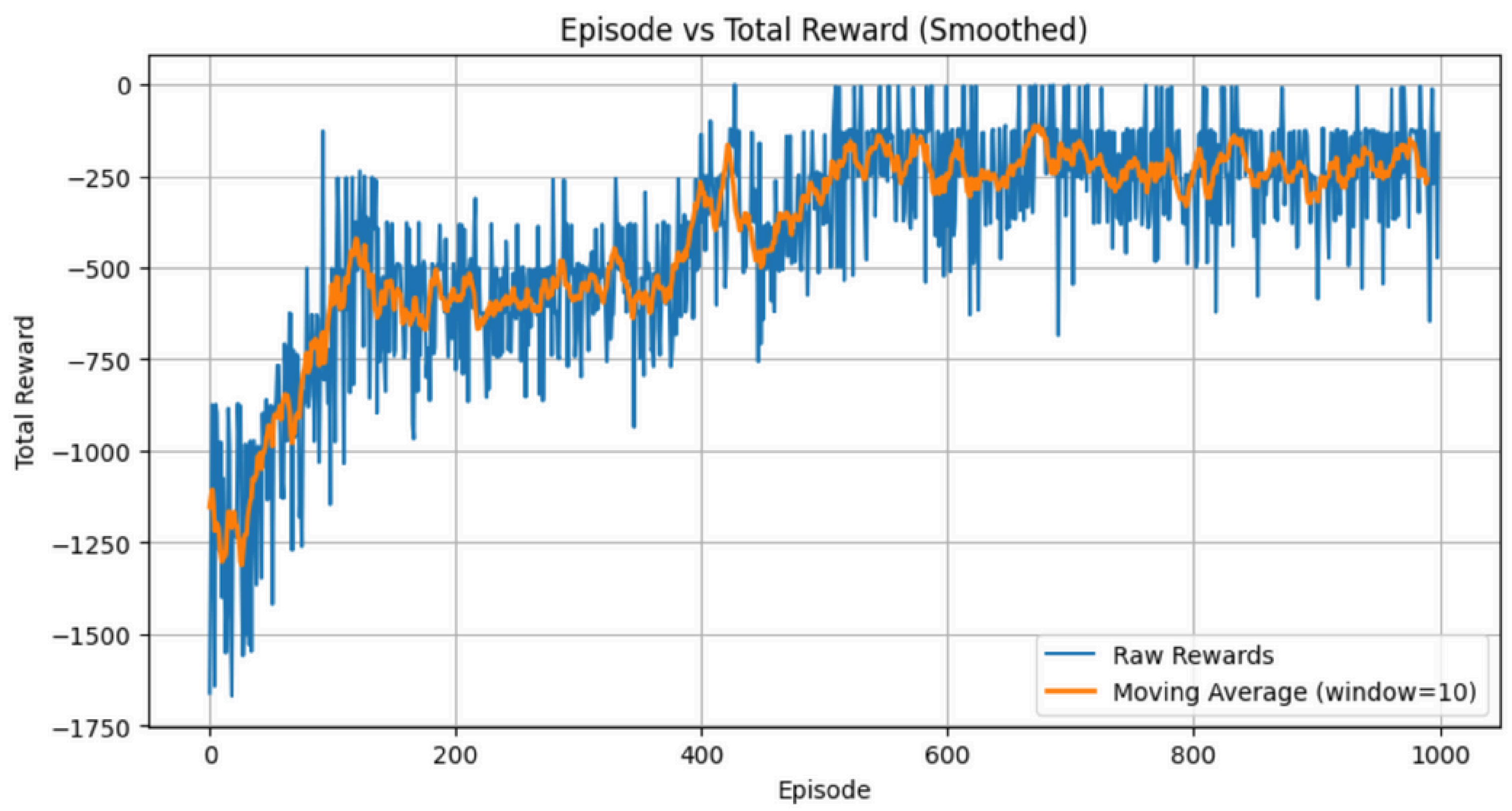


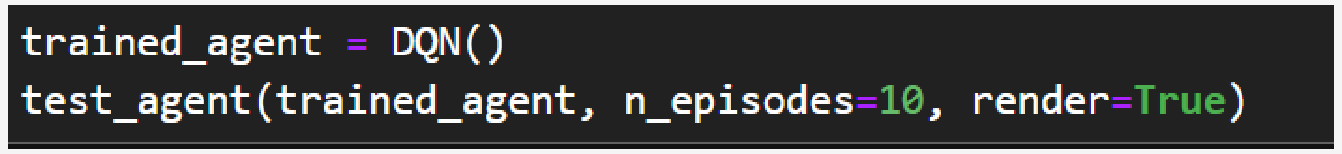
```
Model: "Main"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (InputLayer)          [(None, 3)]               0

 dense (Dense)               (None, 64)                256

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 21)                1365

=================================================================
Total params: 5781 (22.58 KB)
Trainable params: 5781 (22.58 KB)
Non-trainable params: 0 (0.00 Byte)
```

- Change Model Architecture: Increase second layer's neurons from 32 → 64
- Reduce replay memory from 20000 to 10000 so model refreshes experiences faster, so the network trains more on recently learned strategies.

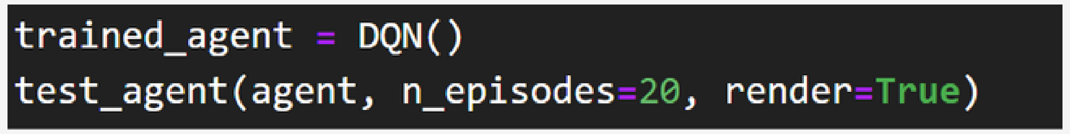
Final Model


Episode vs Total Reward

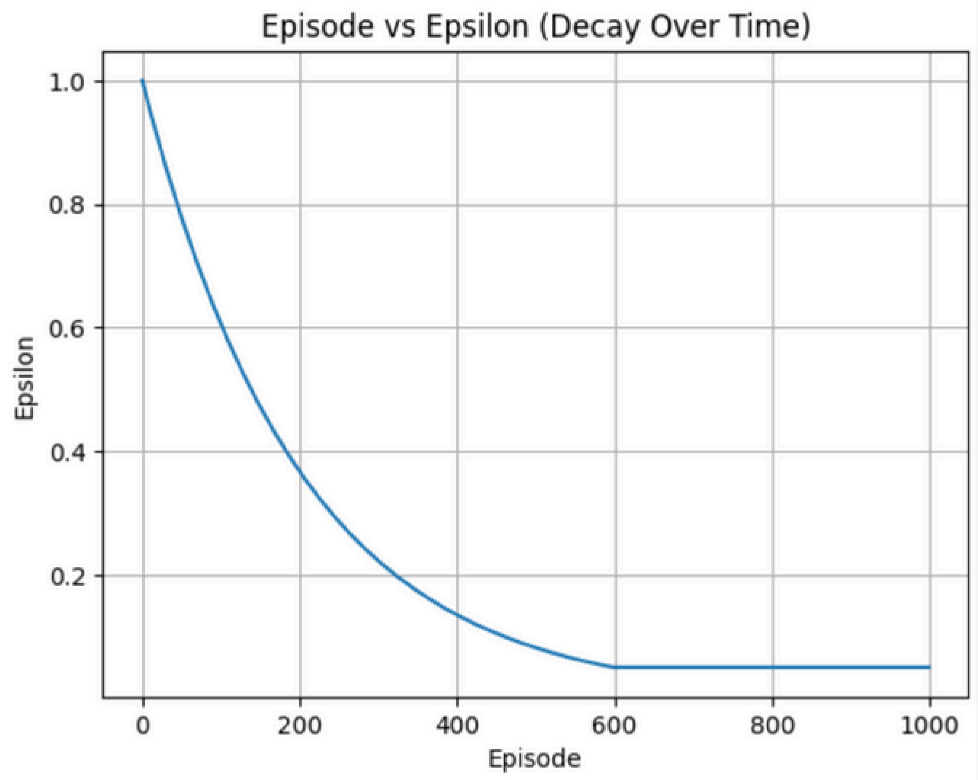

Episode vs Total Reward (Smoothed)



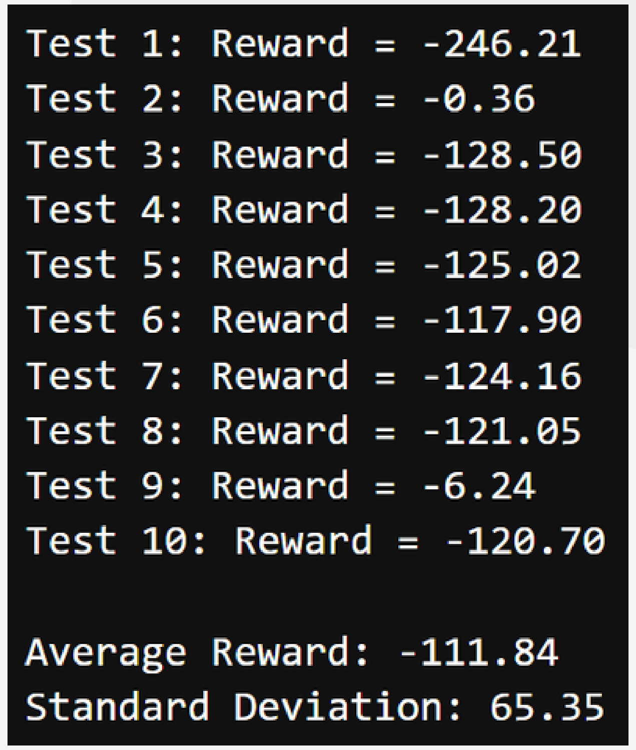
```
trained_agent = DQN()
test_agent(trained_agent, n_episodes=10, render=True)
```



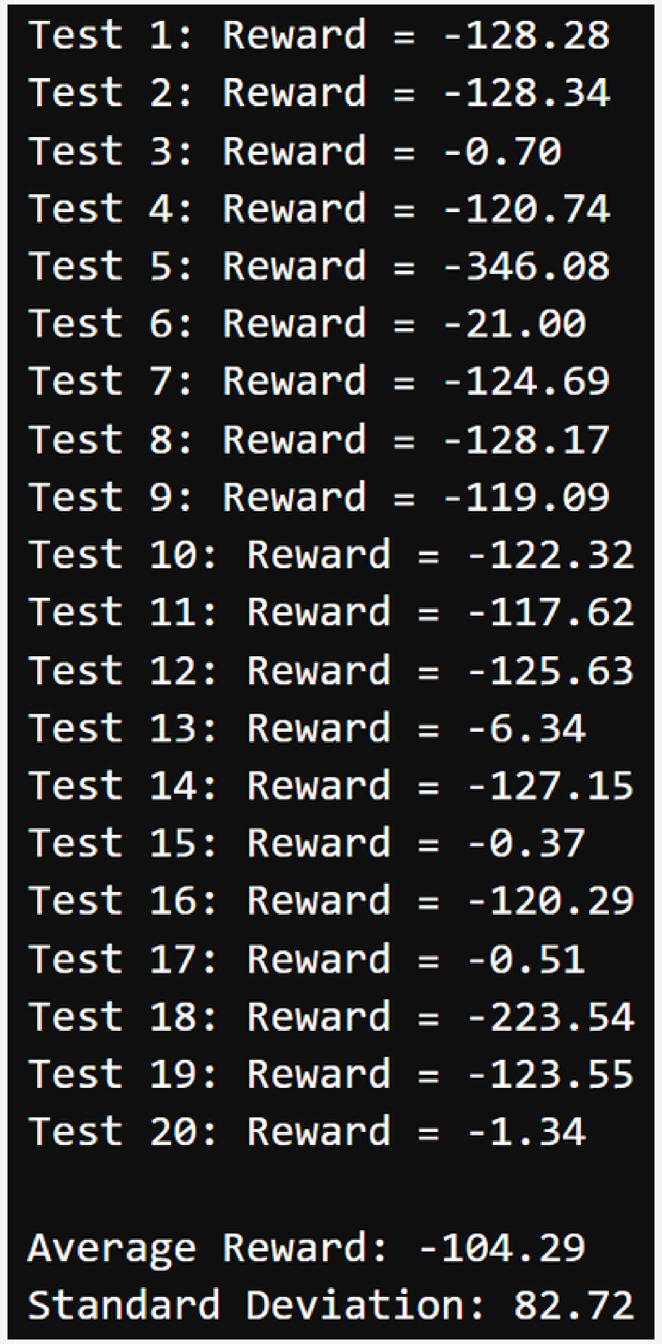
```
trained_agent = DQN()
test_agent(agent, n_episodes=20, render=True)
```


Episode vs Epsilon (Decay Over Time)

- Episode vs Total Reward: Clear upward trend over time. Rewards steadily improve toward 0 as training progresses. Indicates that the agent is learning an effective control policy. Graph plateaus.
- Moving Average: Confirms consistent improvement, gradual convergence with less volatility over time.
- Epsilon Deacy is working is expected.
- Substantial improvement. Multiple episodes achieved near-optimal performance (e.g. -0.37, -0.51, -0.70), showing the agent's ability to sustain the pendulum upright for nearly the entire episode.



```
Test 1: Reward = -246.21
Test 2: Reward = -0.36
Test 3: Reward = -128.50
Test 4: Reward = -128.20
Test 5: Reward = -125.02
Test 6: Reward = -117.90
Test 7: Reward = -124.16
Test 8: Reward = -121.05
Test 9: Reward = -6.24
Test 10: Reward = -120.70

Average Reward: -111.84
Standard Deviation: 65.35
```

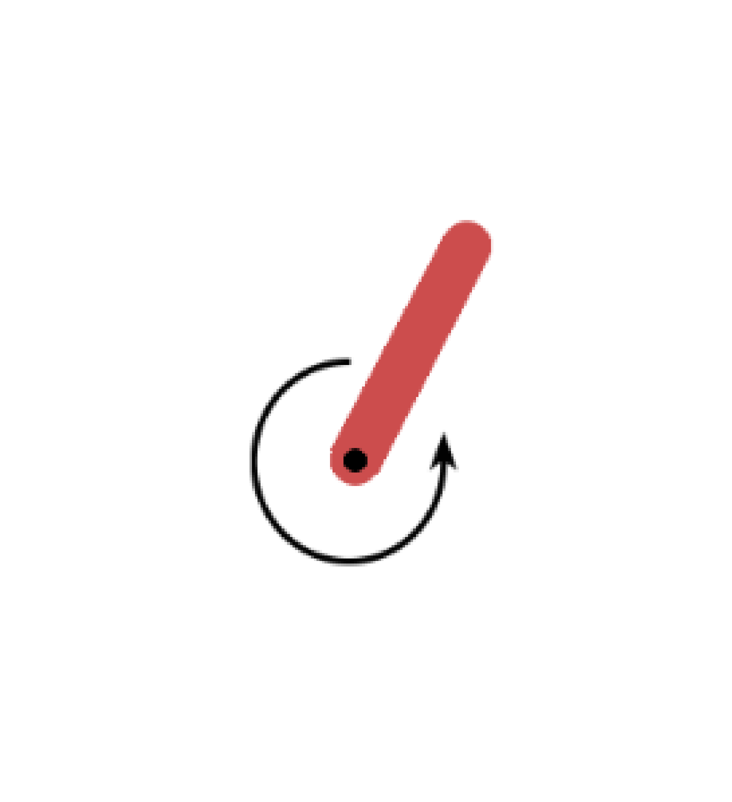


```
Test 1: Reward = -128.28
Test 2: Reward = -128.34
Test 3: Reward = -0.70
Test 4: Reward = -120.74
Test 5: Reward = -346.08
Test 6: Reward = -21.00
Test 7: Reward = -124.69
Test 8: Reward = -128.17
Test 9: Reward = -119.09
Test 10: Reward = -122.32
Test 11: Reward = -117.62
Test 12: Reward = -125.63
Test 13: Reward = -6.34
Test 14: Reward = -127.15
Test 15: Reward = -0.37
Test 16: Reward = -120.29
Test 17: Reward = -0.51
Test 18: Reward = -223.54
Test 19: Reward = -123.55
Test 20: Reward = -1.34

Average Reward: -104.29
Standard Deviation: 82.72
```

Conclusion

Best Setup Identification

- All models tested under identical conditions (same env, metrics, evaluation).
- Final model verified over 10 & 20 episodes for stability.
- Chosen for:
  - **Highest avg reward** (closest to 0)
  - **Lowest variance** across runs. (Standard Deviation)



Success Factors

- **Epsilon decay**: smooth exploration → exploitation shift.
- **Higher learning rate**: faster Q-value updates.
- **Bigger network (64-64)**: better value approximation.
- **Smaller replay buffer (10k)**: fresher, relevant samples.
- **Longer training** (1000 eps): allows larger network to converge.

Outcome

Stable, high-reward control with minimal dips, confirmed by metrics and training graphs.

THANK YOU