ASSIGNMENT 3

Full Name	UBIT Name	UB Number
Dharma Acha	Dharmaac	50511275
Gokul	gokulpul	50510470

Part 1

Grid Environment:

The environment we chose has 12 states and 5 rewards of which two of them are positive rewards, two of them are negative rewards, and one final positive reward when the agent reaches the goal state. The final aim of the agent is to reach the goal point by taking suitable actions.

Agent reaches the goal by taking suitable actions to maximize the reward. When the agent reaches the positive reward it gets a +5 reward and when it reaches the negative reward it gets a -5 reward. In our grid environment, the gold treasure is the positive reward, and the danger zone is the negative reward. For the final goal, the state has a reward of a +7.

States: {S1 = (0,0), S2 = (0,1), S3 = (0,2), S4 = (1,0), S5 = (1,1), S6 = (1,2), S7 = (2,0), S8 = (2,1), S9 = (2,2), , S10 = (3,0), S11 = (3,1), S12 = (3,2) }

Actions: {Up, Down, Right, Left}

Rewards: {-5, +5, -5, +5}

Goal Reward: +7

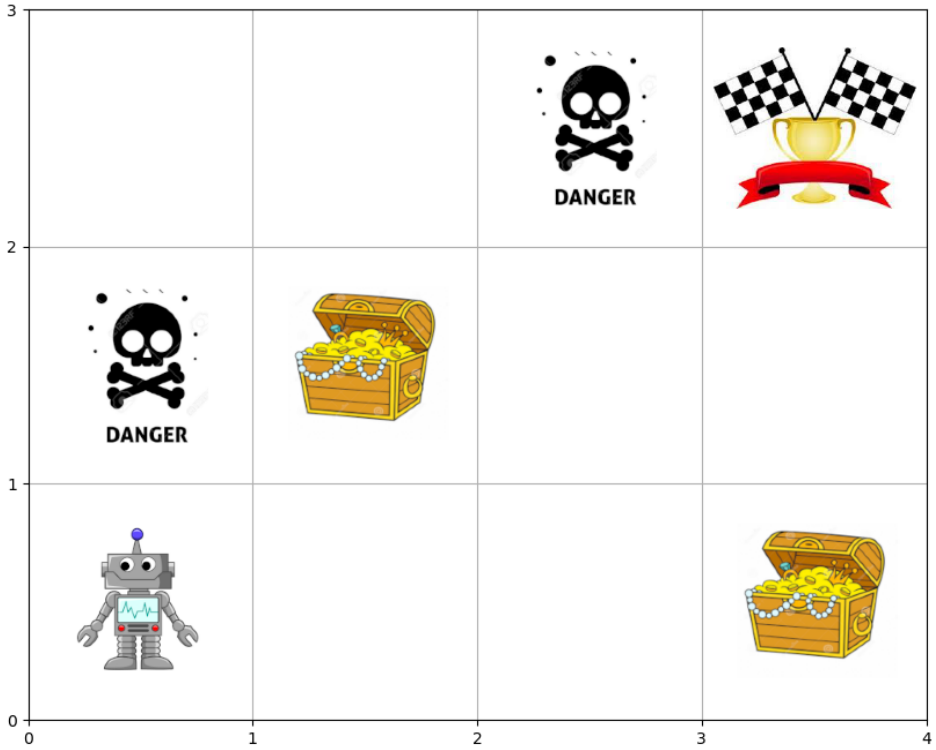
Positions:

Agent position = [0, 0]
Positive reward = [[3, 0],[1,1]]
Negative reward = [[0,1],[2,2]]
Goal = [[3,2]]

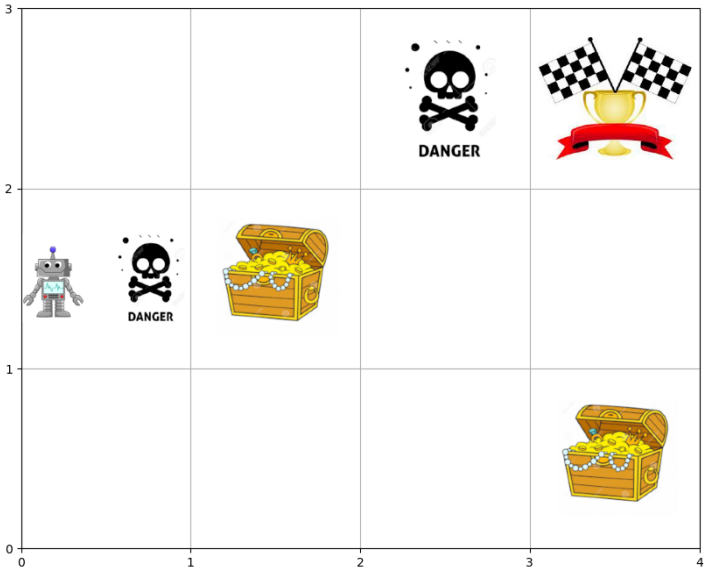
Main objectives:

Reach the goal state with maximum reward.

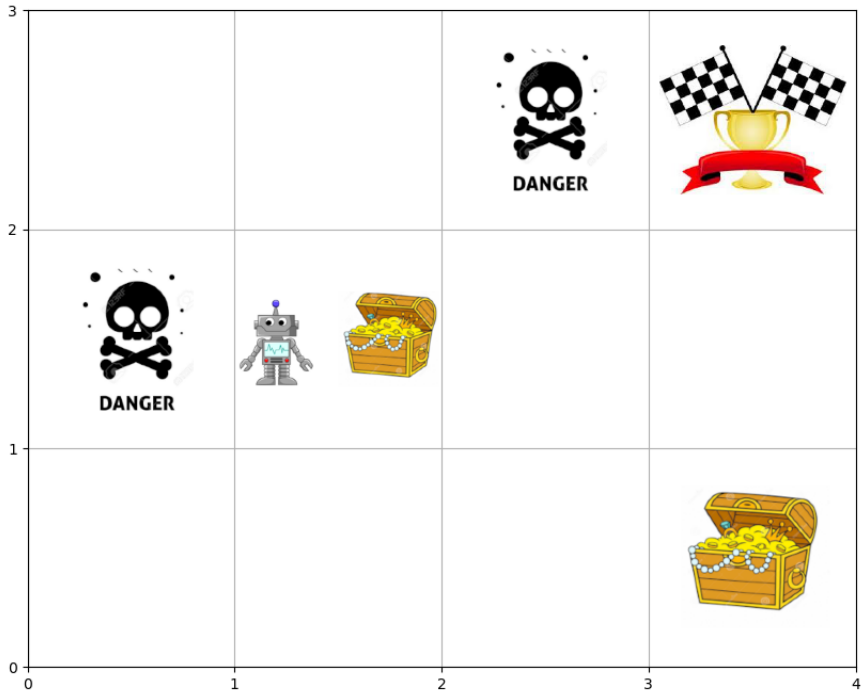
Visualization



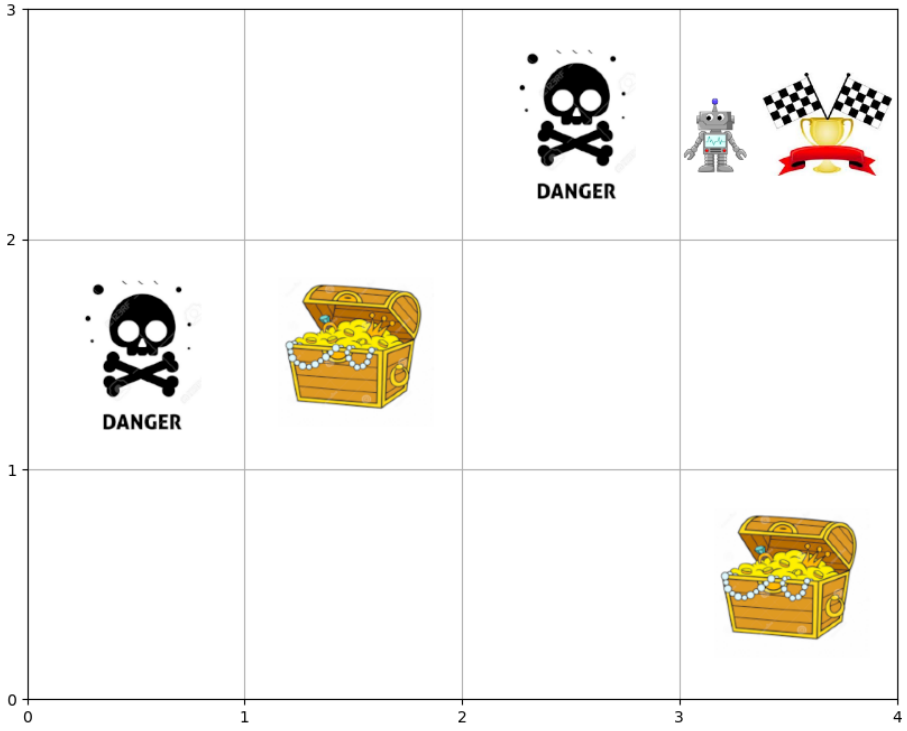
Negative Action:



Positive Action:



Goal:



Safety in AI

1. We implemented required action constraints to make the agent take the necessary actions
2. We introduced termination as soon as the agent reached the goal.
3. To avoid the agent moving out of the environment we used the clip method
4. We used gymnasium spaces which define observation and action spaces for agents interacting with an environment. This is important for ensuring the safety of the learning process.
5. To make the agent operate within the specified limits we use the gym.spaces to define and restrict the action spaces by preventing undesired or unsafe behaviors.

Tabular methods used for problem-solution

SARSA

SARSA is a tabular Q-learning method, where the agent tries to learn a Q-table to estimate the value to do a particular action in a particular state

1. The function choose_action is used to select the action considering the current state and Q-table value.
2. And with epsilon probability, it chooses a random action which is said to be exploration. And with probability (1-epsilon), it chooses the action with the highest Q-value for the current state.
3. The update function updates the Q-values based on the SARSA update rule as shown below

   Q[state, action] = Q[state, action] + alpha * (target - predict)
4. The doSARSALearn function implements the SARSA Learning process for a particular number of episodes. First, it initializes the Q-table with zeroes, and in each episode, the SARSA updates the Q-table considering how the agent interacts with the environment.

   Advantages:

   1. As we know SARSA is an on-policy algorithm which means it learns or updates the Q-table with the action policy returns. So, this is useful in cases where we need the agent to learn while exploring.
   2. It is a simple algorithm to understand which is easy to implement.
   3. The SARSA is more conservative and will learn higher negative rewards existing close to the optimal path and will learn not to choose it by exploring.

4. SARSA can be easily combined with neural networks, which makes it to learn complex policies in high-dimensional environments

Disadvantages

1. It is sensitive to Hyperparameters such as learning rate, alpha, gamma
2. It has a slow convergence rate in larger environments. It is slow to converge to optimal policy.
3. There is a chance of overfitting if the learning rate is too high

Double Q-Learning

Double Q-learning is another tabular method where the agent tries to learn two Q-tables to estimate the value to do a particular action in a particular state.
In our implementation:

1. The function choose_action is used to select the action considering the current state and both Q-table values.
2. And with epsilon probability, it chooses a random action which is said to be exploration. And with probability (1-epsilon), it chooses the action with the highest Q-value sum from both tables
3. The update function will update the Q table chosen with a 0.5 probability
    a. The selected Q table will be updated with the function
        i. Q[state, action] = Q[state, action] + alpha * (target - predict)
        ii. Where predict = Q[state, action]
        iii. And target = reward + gamma*Q'(s', $\text{argmax}_{a'}$(Q(s', a')))
4. The doDoubleQLearn function implements the Double Q-Learning process for a particular number of episodes. First, it initializes the Q-tables with zeroes, and in each episode, the loop will update the chosen Q-table considering how the agent interacts with the environment.

Advantages of Double Q-learning:

1. Q-learning is model-free. It doesn't require knowledge of the complete environment model.
2. It converges to an optimal policy quickly compared to SARSA
3. It can handle large state stationary environments with good results.

Disadvantages of Double Q-learning:

1. It is very slow when the number of states is high in number and rewards are sparse.
2. The Q-learning is unreliable since the Q-learning is an off-policy algorithm and it learns from a policy that is not the target policy trying to be optimal.
3. Q-learning gives very bad results in non-stationary environments.

1) **Using SARSA to solve the environment:**

Parameters used:

- Epsilon = 1
- Episode count = 500
- Learning rate = 0.1
- Discount factor = 0.95
- Epsilon decay = 0.009

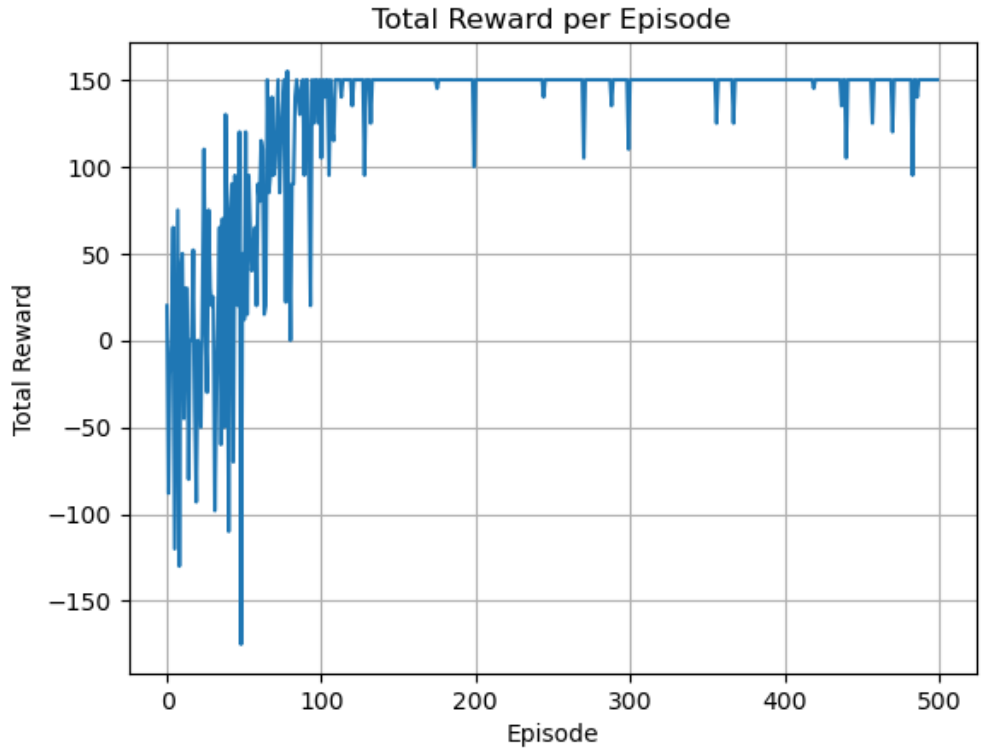
- **Initial Q-table:**

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

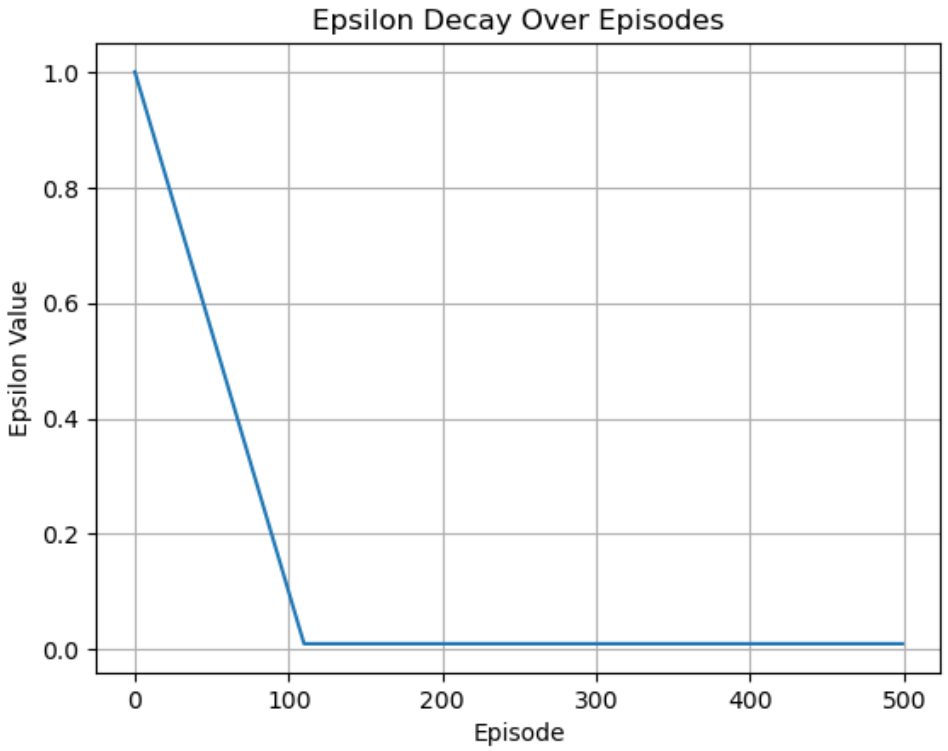
- **Trained Q table:**

```
Trained Q table
[[240.99217364  33.23588432  24.33943653  17.57555306]
 [ 17.4652885   79.44053518 260.40482485 103.75886827]
 [ 40.67071743   7.25919246   2.75155544   3.34418304]
 [ 10.2646215   36.75979189   3.59011948  16.80421736]
 [157.17839503  -9.40343752  -5.97172762   0.28532577]
 [ 60.19122375  59.35942776  46.60387299 260.61494611]
 [  1.67442175 129.40870982  -1.08620998   7.23410101]
 [  1.32920373   1.83365207   5.052513     0.        ]
 [ -3.49304973  -3.77237023  -5.8807217   -7.36115312]
 [ -5.93697874  -0.88541132   0.65859143 124.85018561]
 [ -1.523       -2.56954738  -3.00587271  -2.9657831 ]
 [  0.           0.           0.           0.        ]]
```

- **Total Rewards per Episode Plot**



- **Epsilon Decay Over Episodes**



2) **Using Double-Q Learning to solve the environment:**
Parameters used:

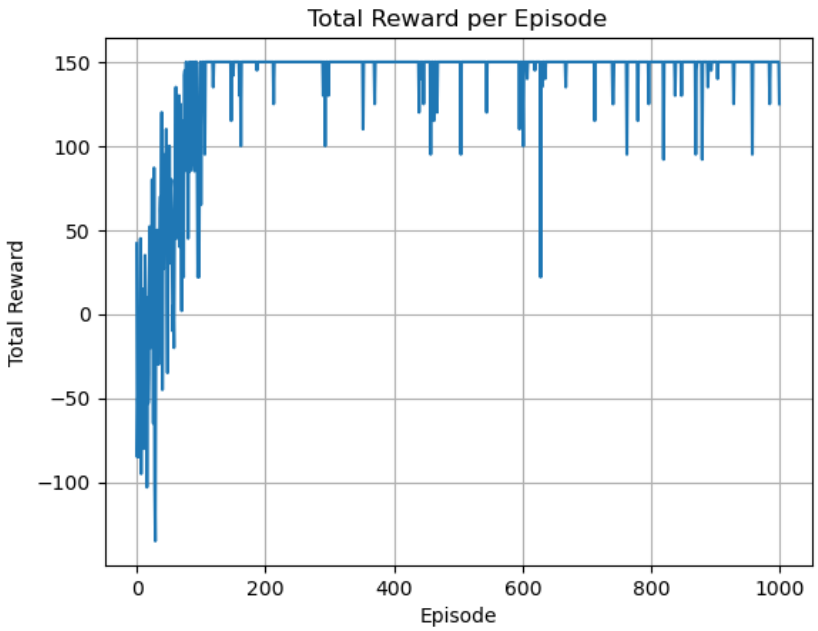
- Epsilon = 1
- Episode count = 1000
- Learning rate = 0.1
- Discount factor = 0.95
- Epsilon decay = 0.009

- Trained Q tables

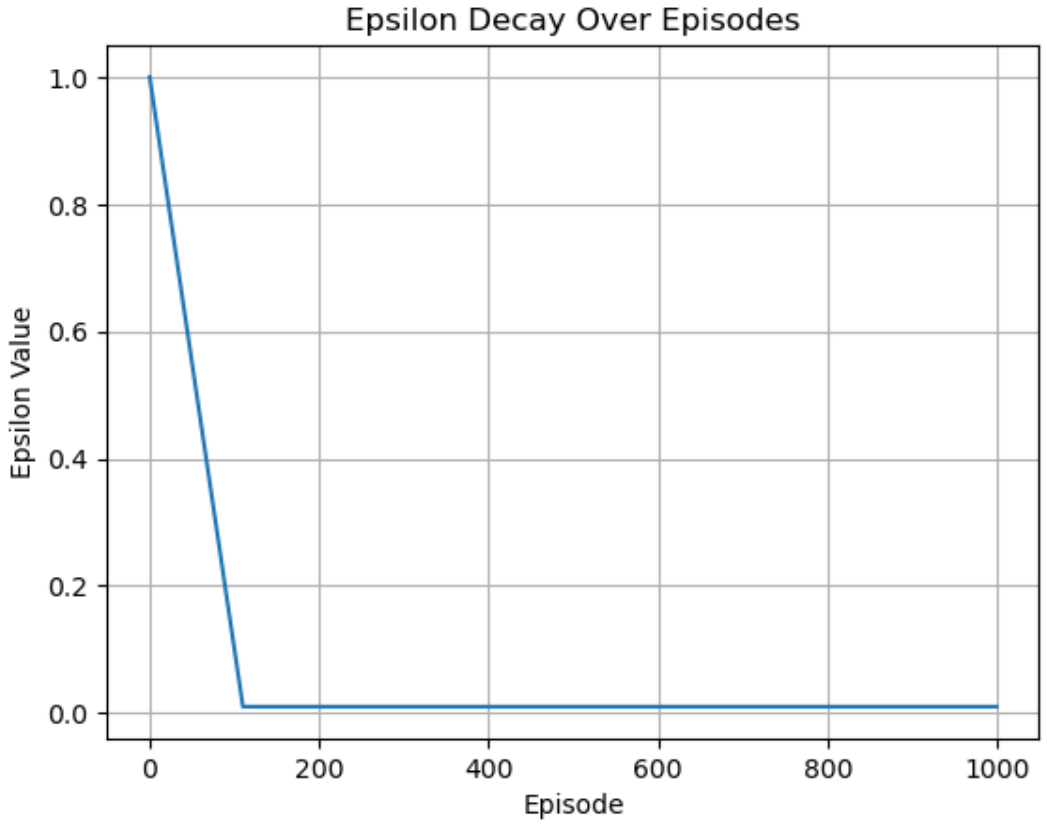
```
Trained Q table 1
[[289.17   2.2    7.27   7.86]
 [ 11.65 152.24 291.57 159.3 ]
 [ 19.23   0.49   2.57   2.57]
 [  2.02   4.53  10.58   1.46]
 [159.34  -6.06  -2.39  -0.65]
 [ 73.25  29.94  16.12 306.16]
 [  1.74 137.72   0.71   2.5 ]
 [  4.54   3.3   13.21  -0.5 ]
 [ -2.81  -2.71  -3.52  -0.42]
 [  2.09   1.43   0.87  46.75]
 [ -0.47   0.94  -1.5    1.05]
 [  0.     0.     0.     0.  ]]


Trained Q table 2
[[276.69  52.55  20.16  48.2 ]
 [ 15.81 154.57 304.92  77.73]
 [ 14.11   1.21   2.17   5.81]
 [  3.53   0.    14.51   3.63]
 [127.34  -3.76  -5.41   1.51]
 [ 28.92  59.64  20.77 290.03]
 [  1.02 166.72   0.5    1.35]
 [  1.4    1.55   9.77   3.96]
 [ -2.85  -2.77  -2.81  -1.97]
 [ -2.7   -1.74   0.49 109.61]
 [  0.64   2.18   0.     0.  ]
 [  0.     0.     0.     0.  ]]
```

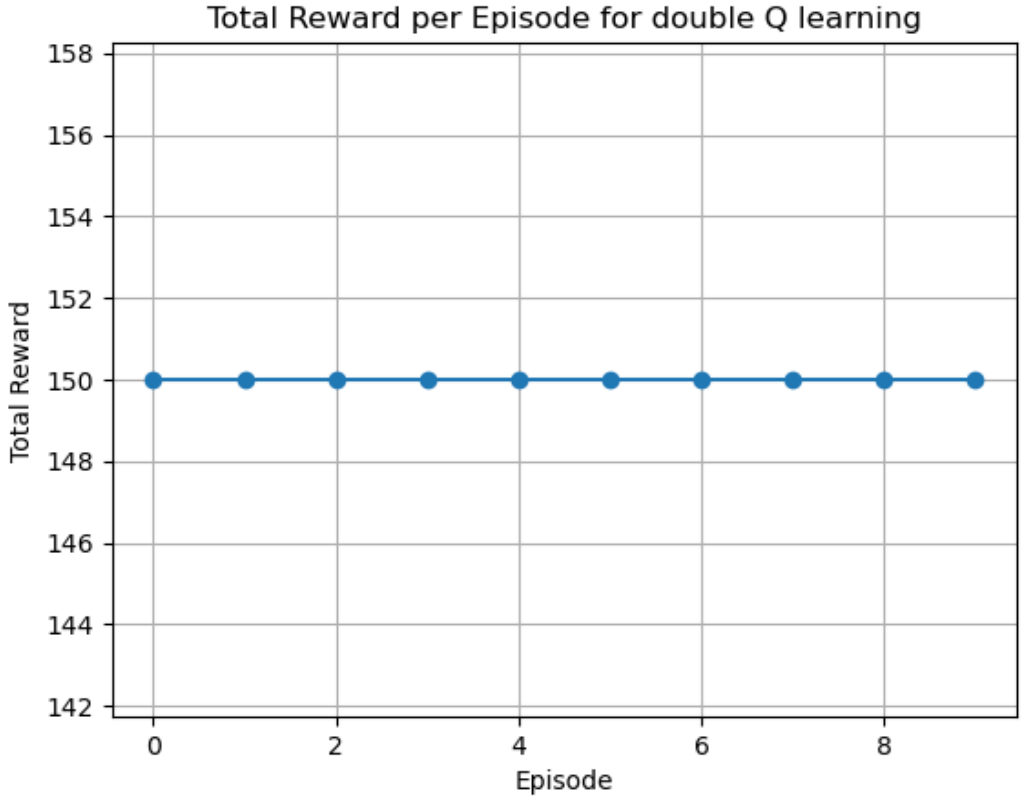
- **Total Rewards per Episode Plot**



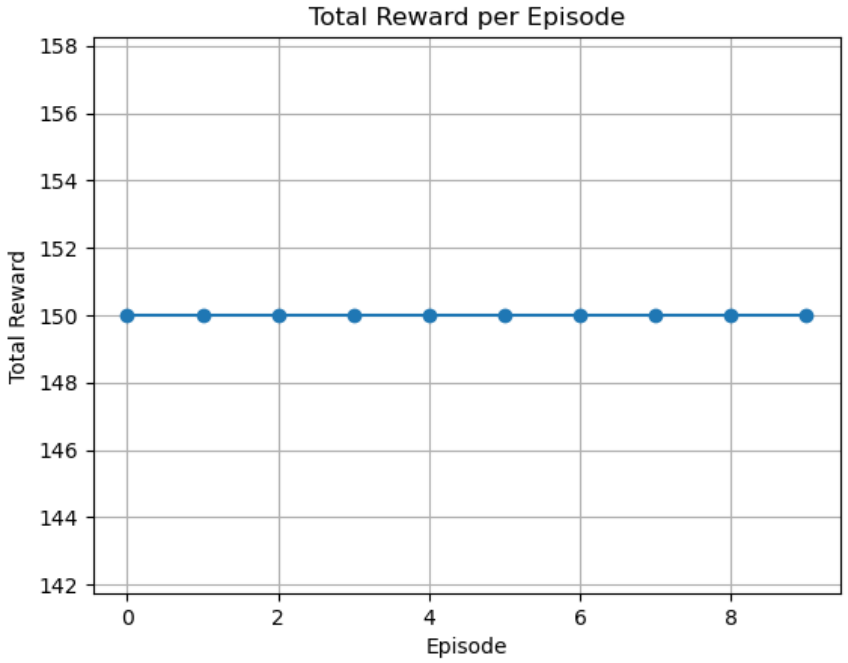
- **Epsilon Decay Over Episodes**



- **The agent taking only Greedy actions:**

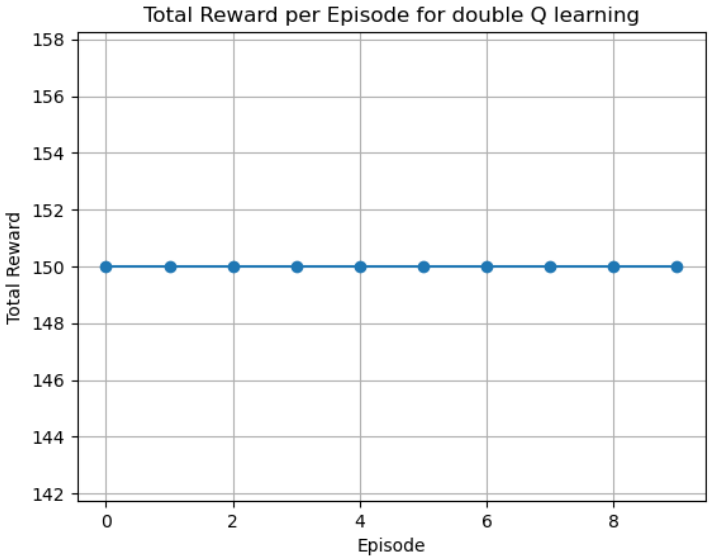


3. **SARSA Learning evaluation**
   - **Running the agent with actions chosen greedily from the learned policy**
   - Total Rewards per episode plot:



   - It can be seen from the above graph, that the agent only chose the steps that led to the reward of 150. The maximum reward the agent was able to achieve with the current learned policy is 150.
   - For all 10 episodes, the reward was 150.

4. **Double Q- Learning evaluation**
   - **Running the agent with actions chosen greedily from the learned policy**
   - Total Rewards per episode plot:



- Here also we can see that the total reward in each of the 10 episodes the agent was able to secure was 150.

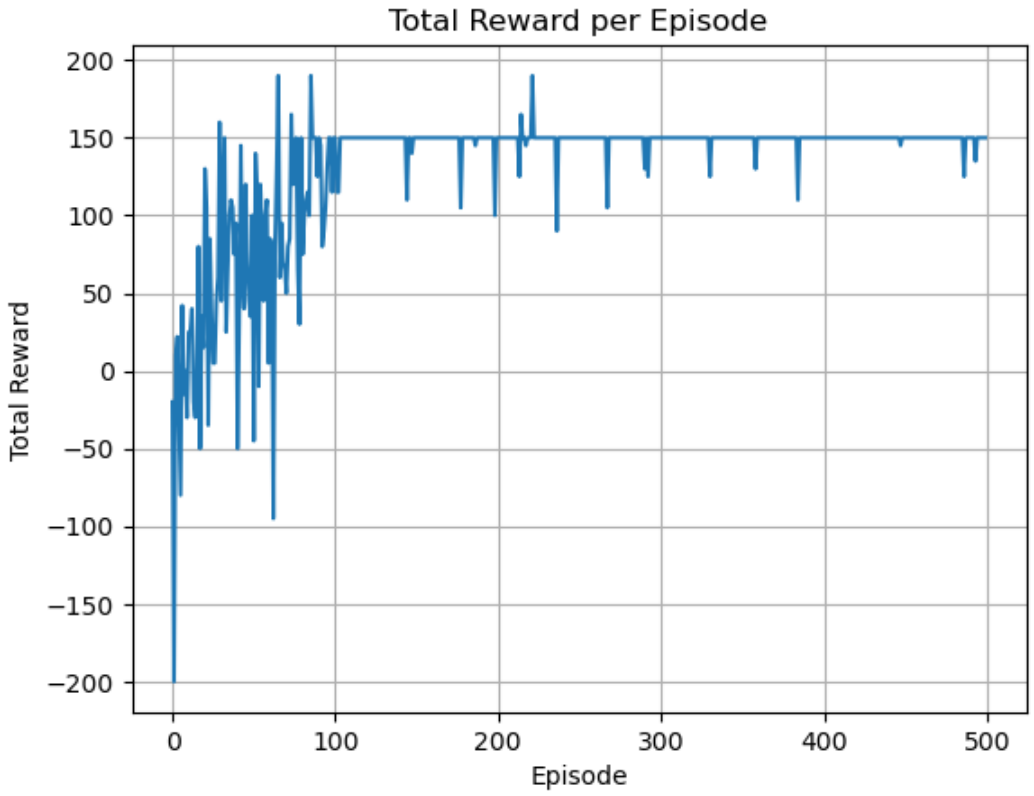
Hyperparameter tuning for SARSA

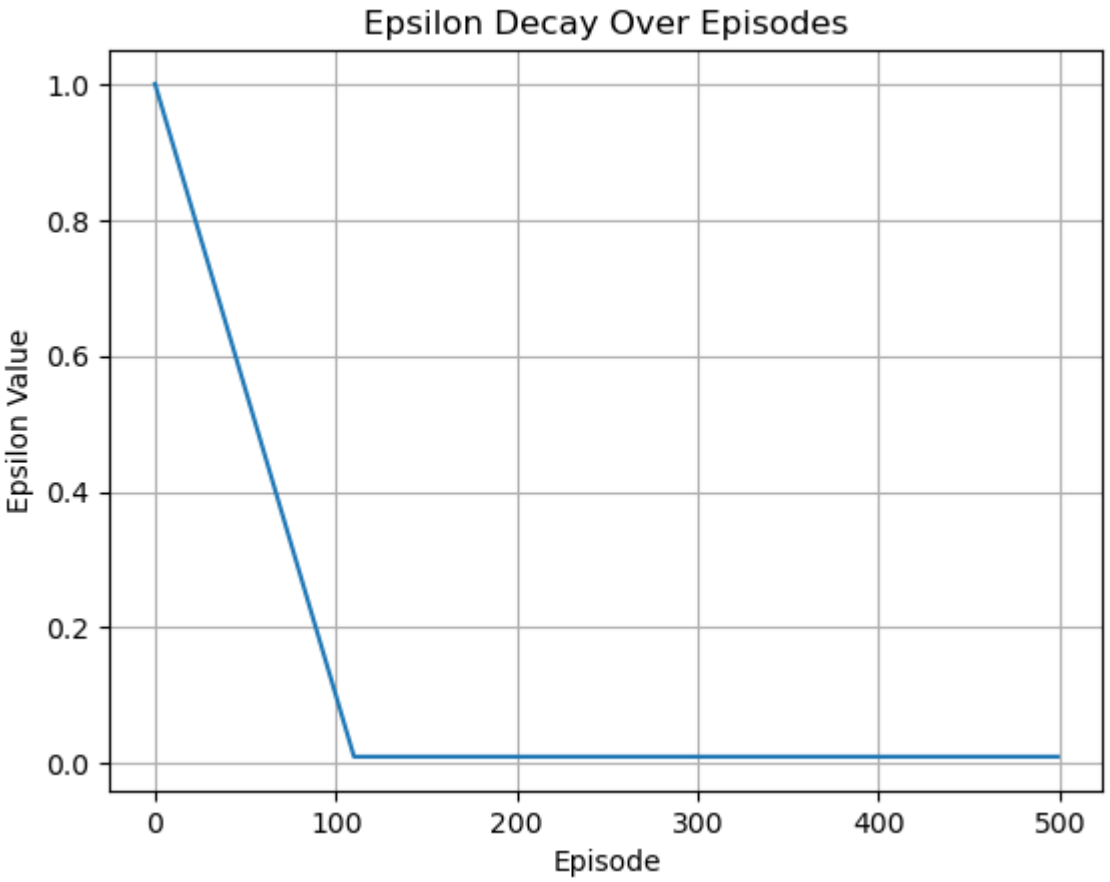
- For the tuning, we choose gamma which is the discount factor, and epsilon decay.
- We tried three different values for both parameters separately. Following are the details of that experiment
- The gamma value is reduced from 0.95 to 0.80 this means the agent is now checks for immediate rewards more and agent likely to explore new states and actions, even they are risky
- When we compare plots of gamma 0.95 and 0.80 the total rewards per episode is higher with gamma value 0.80 this is because agents are now valuing immediate rewards and willingly to take actions that lead to immediate rewards, even though they don't lead to higher total reward in the long term.

1. **Gamma = 0.80**

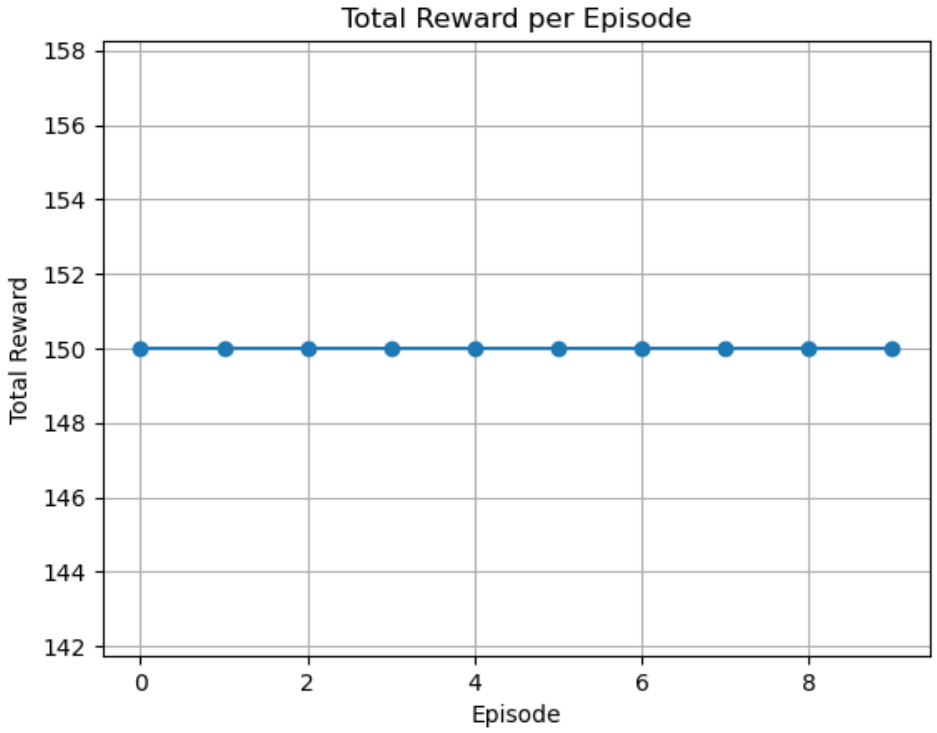
   Parameters used:

   - Epsilon = 1
   - Episode count = 500
   - Learning rate = 0.1
   - Discount factor = 0.80
   - Epsilon decay = 0.009





- With gamma as 0.80, we got the total reward per episode when the agent takes greedy steps from the learned policy as



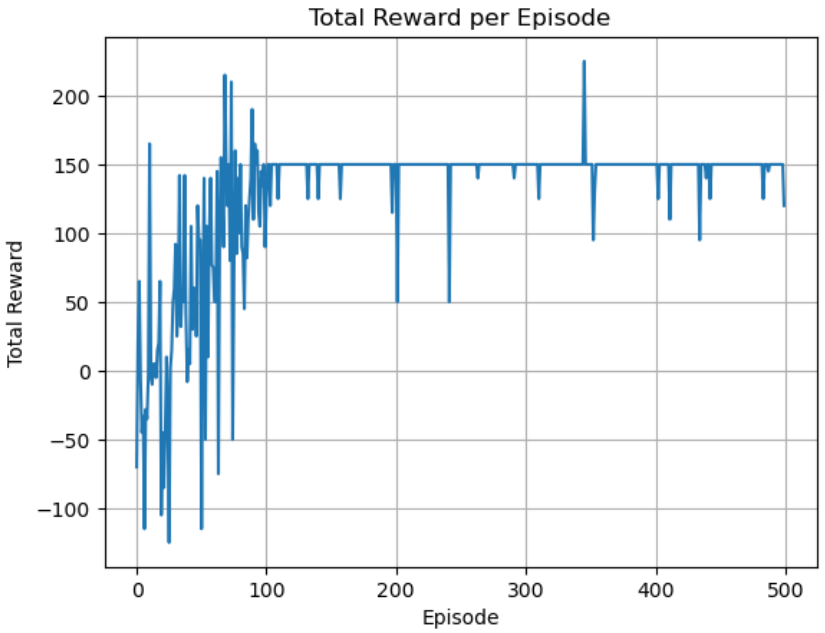
Observation:

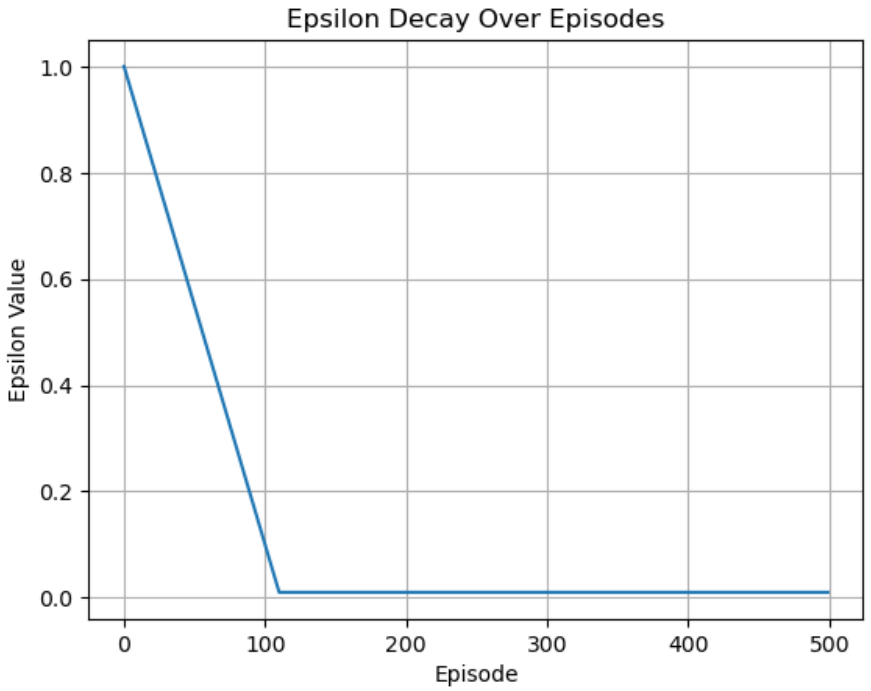
- It can be seen that the maximum total reward achieved was 150.

2. **Gamma = 0.4**

   Parameters used:

   - Epsilon = 1
   - Episode count = 500
   - Learning rate = 0.1
   - Discount factor = 0.40
   - Epsilon decay = 0.009





- With gamma as 0.4, we got the total reward per episode when the agent takes greedy steps from the learned policy as

-

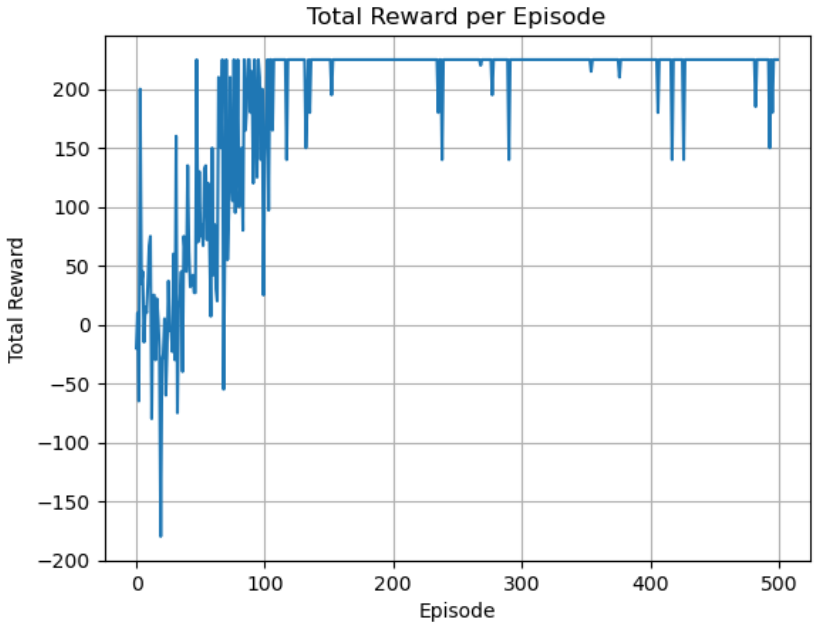
Observation:

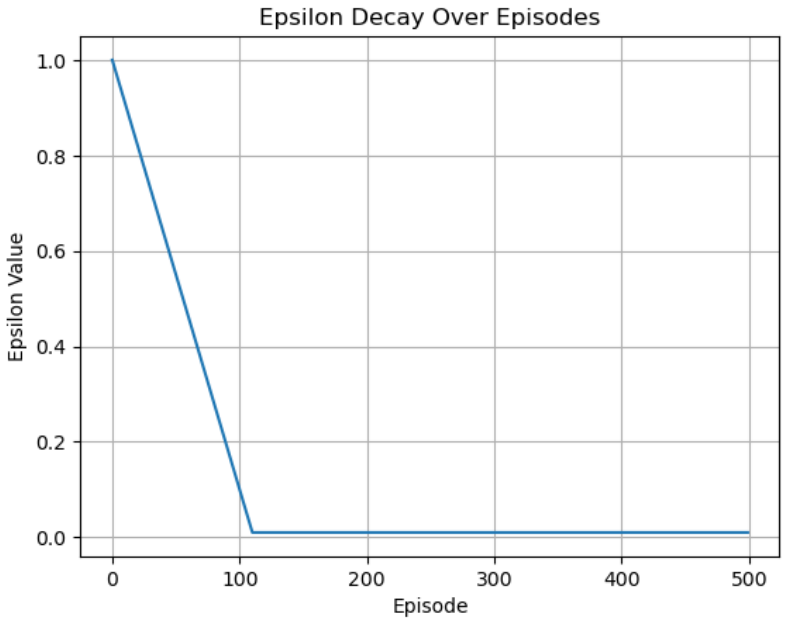
- It can be seen that the maximum total reward achieved was again 150. No change from the previous try with a gamma value of 0.80

3. Gamma = 2

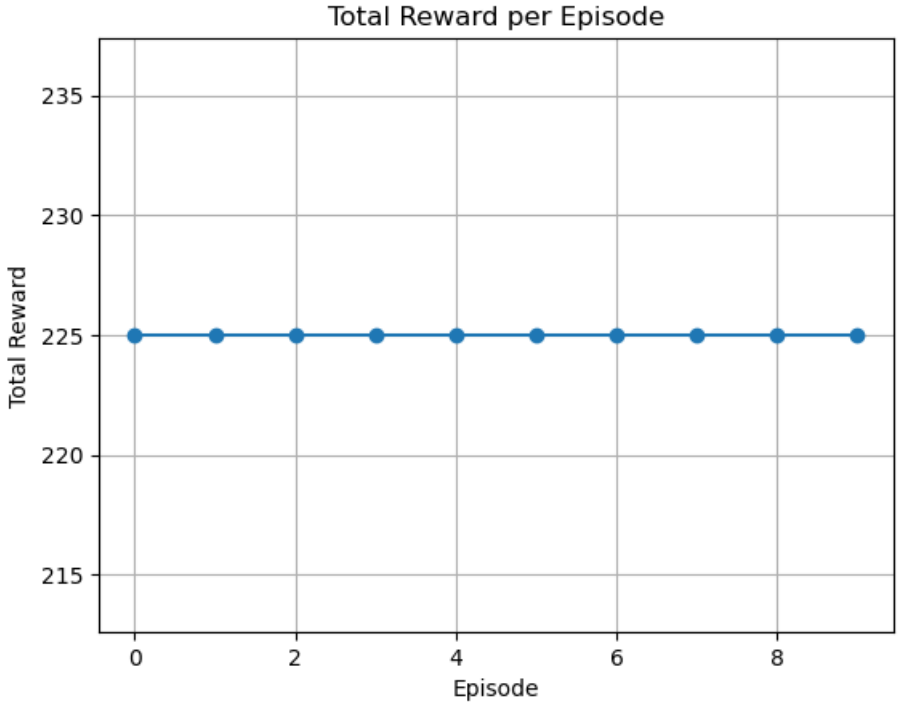
Parameters used:

- Epsilon = 1
- Episode count = 500
- Learning rate = 0.1
- Discount factor = 2
- Epsilon decay = 0.009





- With gamma as 2, we got the total reward per episode when the agent takes greedy steps from the learned policy as



-
  - It can be seen that the maximum total reward achieved was improved to 225.
  - A significant change from previous tries was achieved.

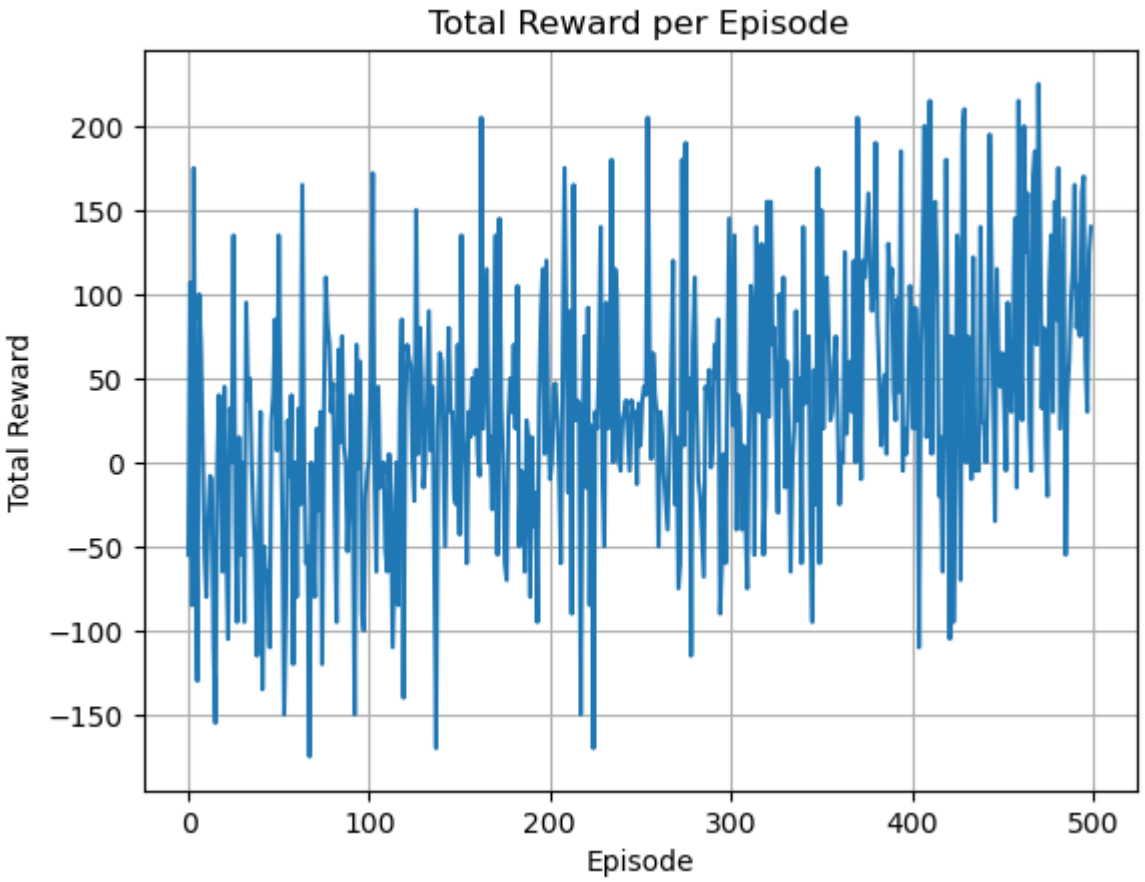
The different gamma values we chose are 0.8, 0.4, 2. Out of these gammas with 2 we got the highest total reward which is 225.

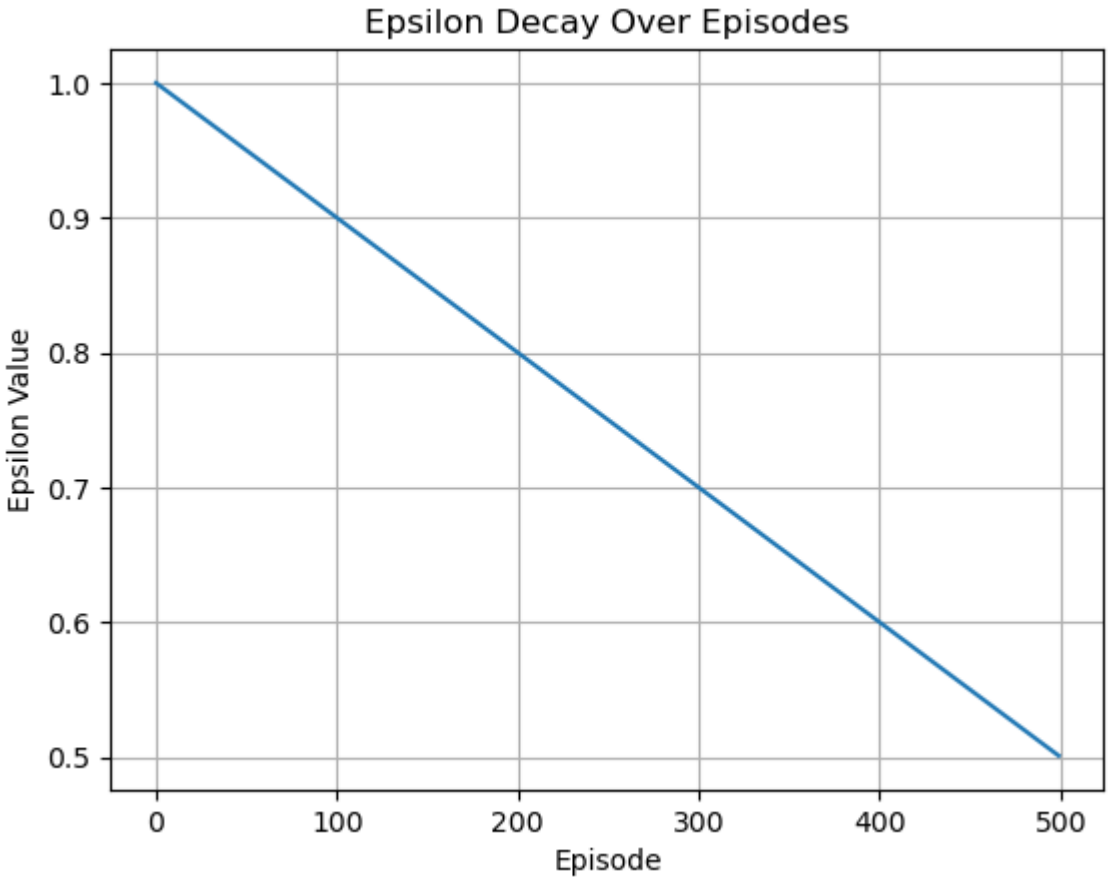
The increased reward is due to the agent trying to explore new states and actions. With the gamma =2 the agent values future rewards more heavily. Even though there

is risk, the agent tries to explore new states and actions which leads to high rewards in the longer term. So this makes discovery of new and better policies.

- Epsilon decay was the second parameter we chose for tuning

1. **Epsilon decay = 0.001**
   Parameters used:
   - Epsilon = 1
   - Episode count = 500
   - Learning rate = 0.1
   - Discount factor = 0.80
   - Epsilon decay = 0.001



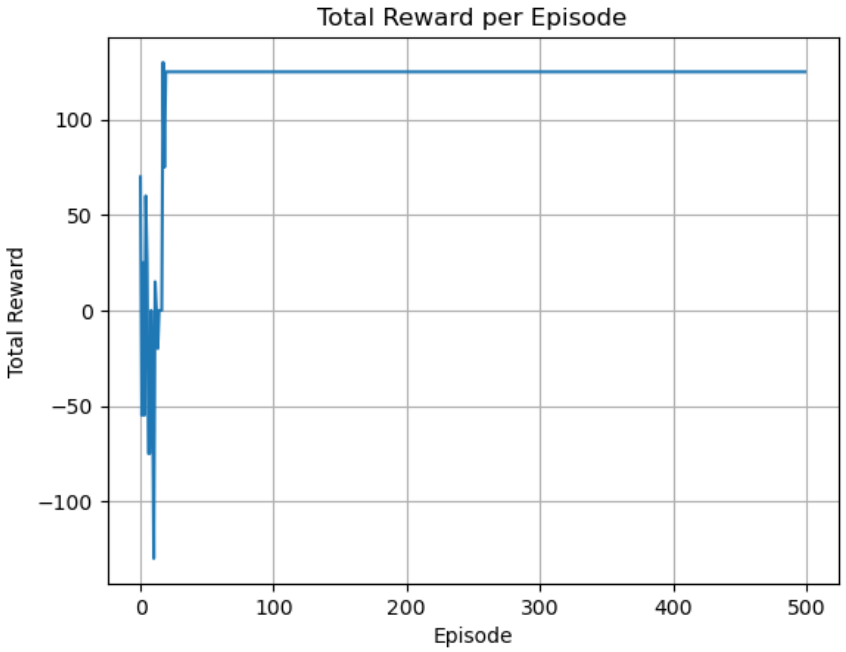


- With Epsilon decay = 0.001, we got the total reward per episode when the agent takes greedy steps from the learned policy as

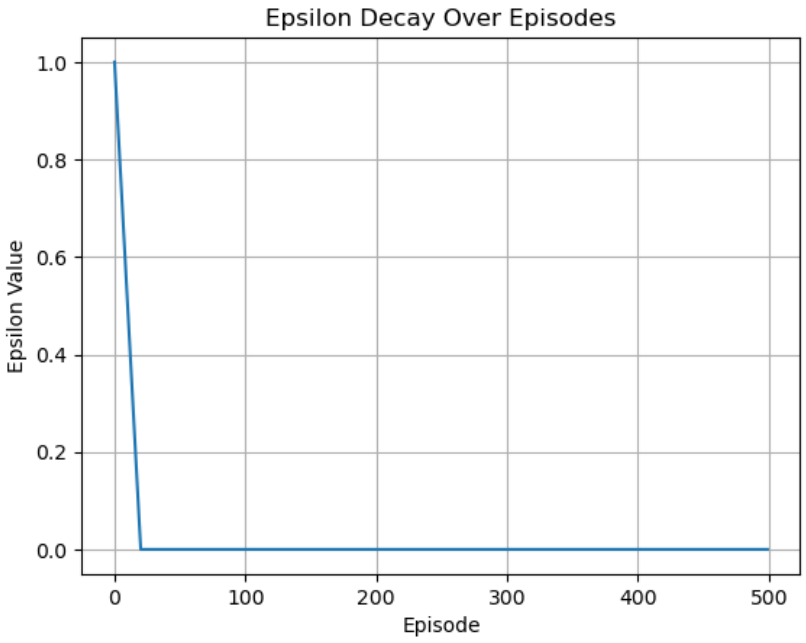
Observation:

- The result obtained was better compared to previous trials. We got a maximum total reward of 225. Which is greater than the 150 we were getting in previous trials.

2. **Epsilon decay = 0.05**
   Parameters used:
   - Epsilon = 1
   - Episode count = 500
   - Learning rate = 0.1
   - Discount factor = 0.80
   - Epsilon decay = 0.05

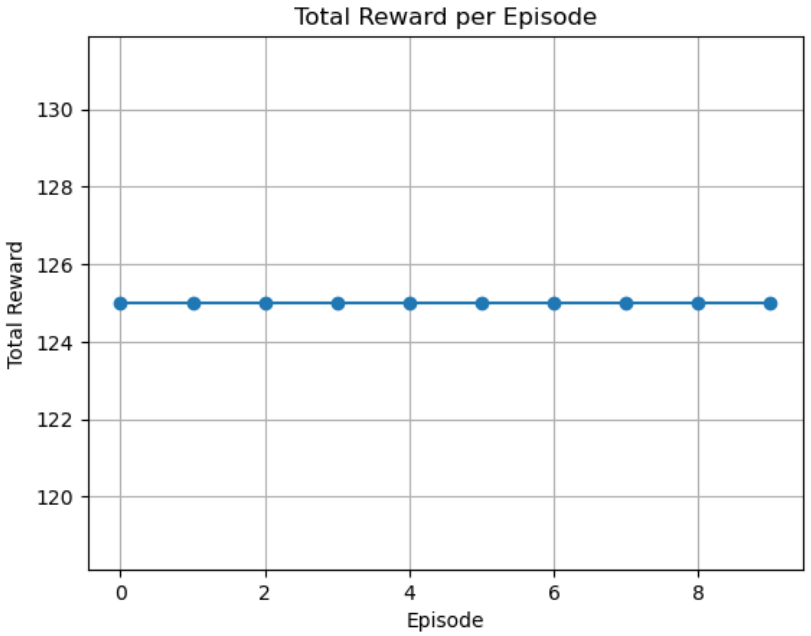


-



-

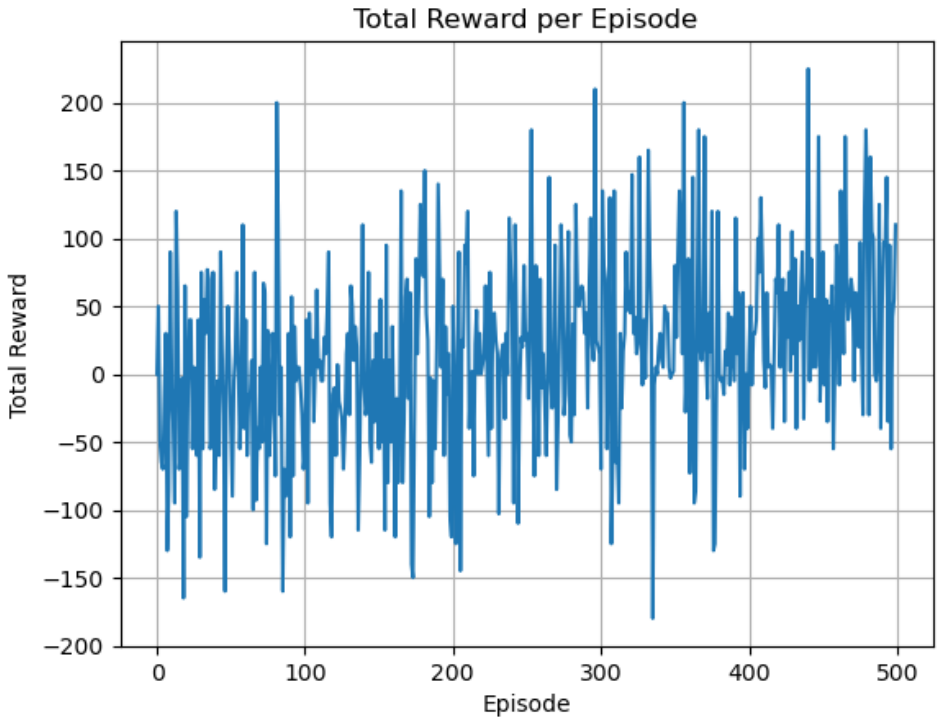
- With Epsilon decay = 0.05, we got the total reward per episode when the agent takes greedy steps from the learned policy as

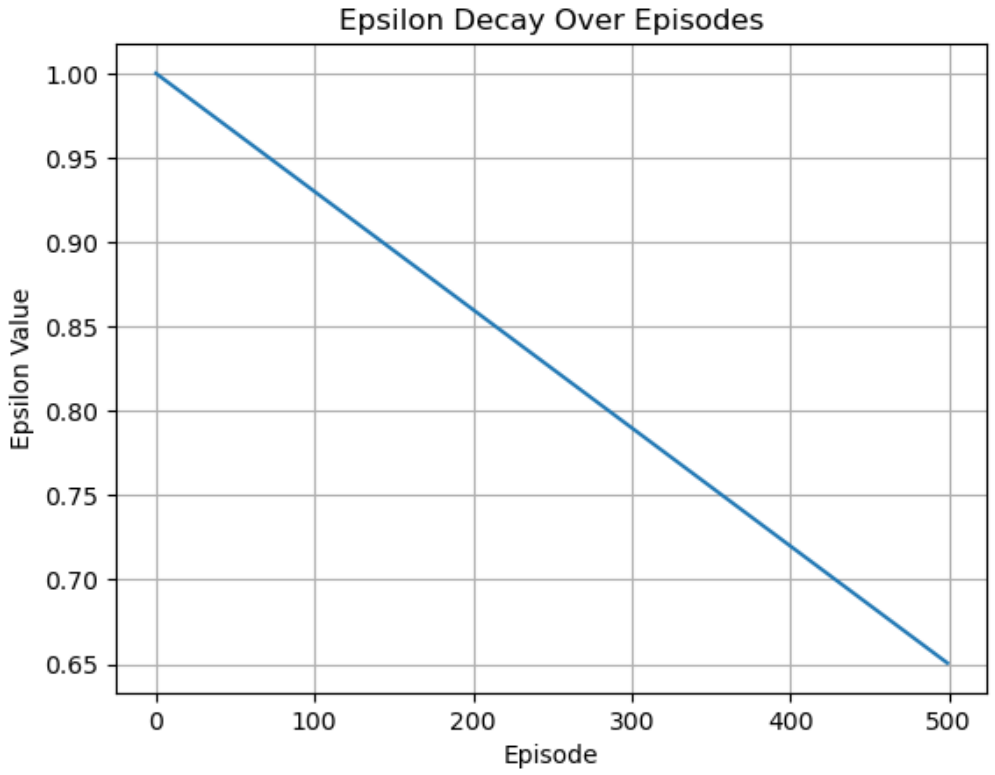


Observation:

- The maximum total reward obtained is much less compared to previous trials. It was around 125 which is less than 150 which we were getting almost all the time. So, it can be concluded that this value for epsilon decay does not result in optimal results.

3. **Epsilon decay = 0.0007**
   Parameters used:
   - Epsilon = 1
   - Episode count = 500
   - Learning rate = 0.1
   - Discount factor = 0.80
   - Epsilon decay = 0.007





- With Epsilon decay = 0.0007, we got the total reward per episode when the agent takes greedy steps from the learned policy as

Observation:

- The total maximum reward we obtained in this try is 225 which is one of the maximum we obtained.

- From all the above experiments on the SARSA model the best parameter set we found was with parameters with values gamma=0.8 and epsilon decay=0.007, where we got the best result.

  Parameters used:

  - Epsilon = 1
  - Episode count = 500
  - Learning rate = 0.1
  - Discount factor = 0.80
  - Epsilon decay = 0.0007

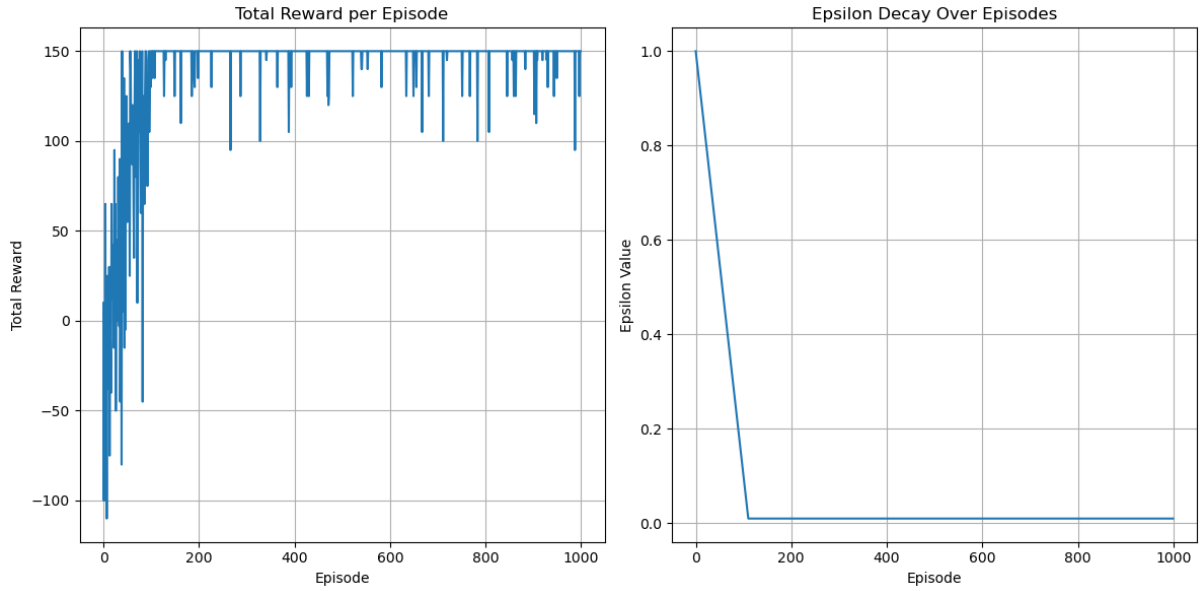
- We are choosing this set of parameters since they gave the best result for our SARSA implementation.

  Epsilon decay is typically related to the exploration-exploitation trade-off in algorithms. When epsilon decay is 0.0007 there is an increase in exploration, which allows the agent to discover more favorable actions leading to better rewards. The higher epsilon decay means less exploration and more dependent on learned policy. The decrease in reward is due to learned policy not being enough, and the agent needs more exploration to choose better actions.

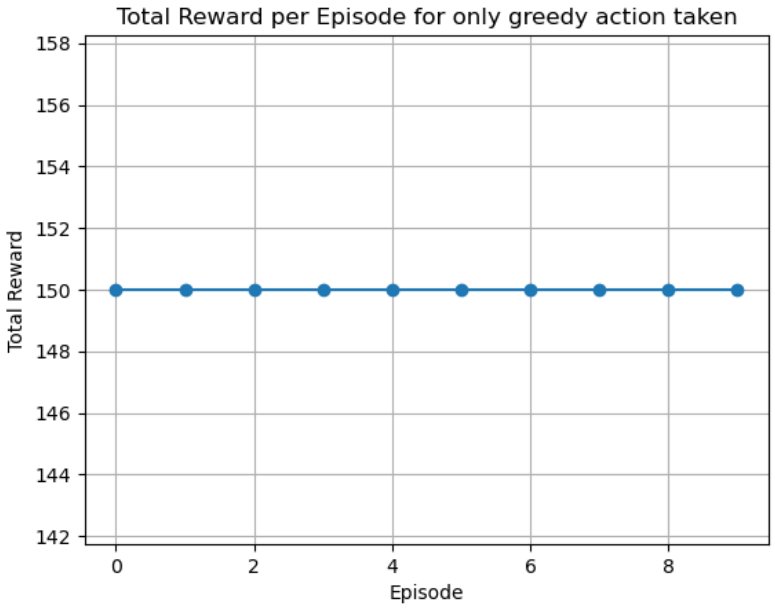
Hyperparameter tuning for Double Q-Learning

- For the tuning, we choose gamma which is the discount factor, and epsilon decay.
- We tried three different values for both parameters separately. Following are the details of that experiment

1. **Gamma = 0.85**
   Parameters used:
   - Epsilon = 1
   - Episode count = 1000
   - Learning rate = 0.1
   - Discount factor = 0.85
   - Epsilon decay = 0.009



- With Gamma = 0.85, we got the total reward per episode when the agent takes greedy steps from the learned policy as



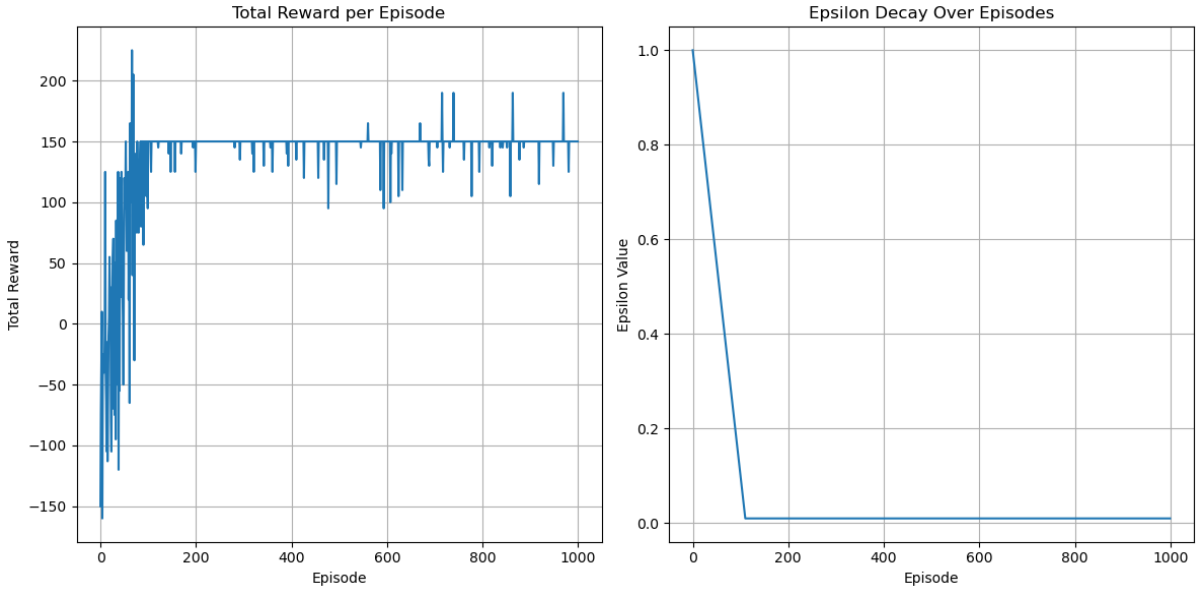
Observation:

- The max total reward the agent was able to achieve when the agent only chose greedy actions from the learned policy is 150

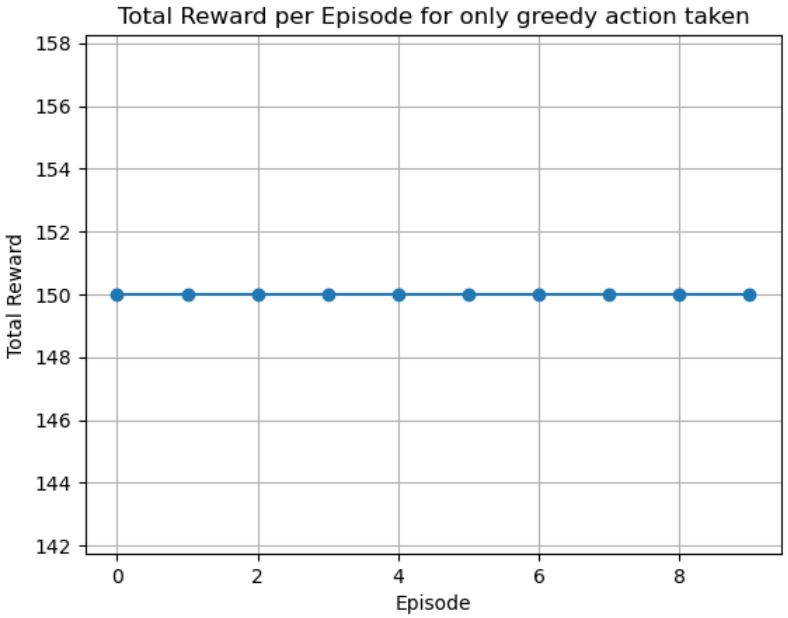
2. **Gamma = 0.5**

Parameters used:

- Epsilon = 1
- Episode count = 1000
- Learning rate = 0.1
- Discount factor = 0.5
- Epsilon decay = 0.009



- With Gamma = 0.5, we got the total reward per episode when the agent takes greedy steps from the learned policy as



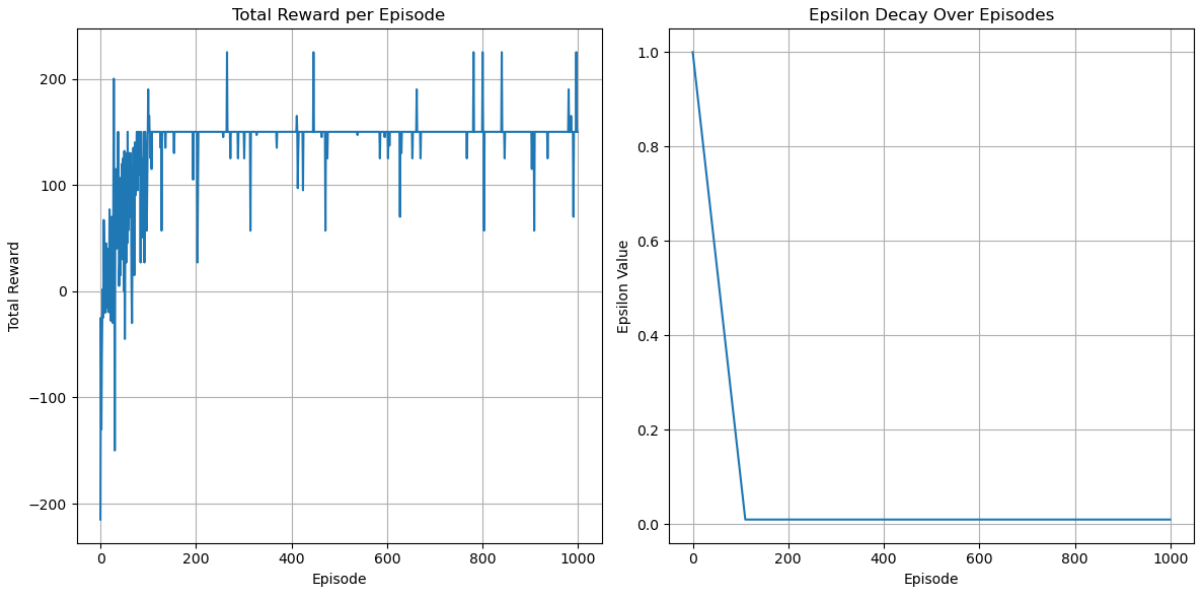
Observation:

- The max total reward the agent was able to achieve when the agent only chose greedy actions from the learned policy is again 150

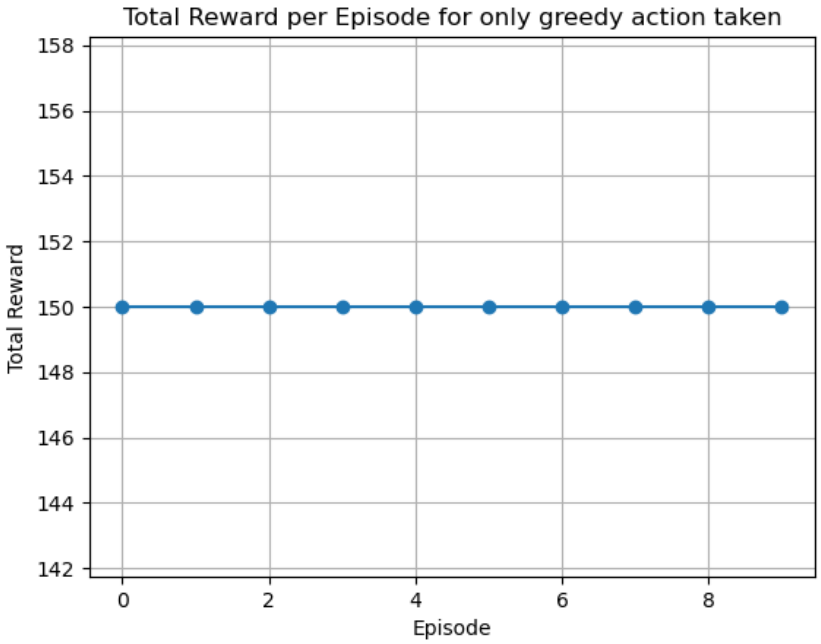
3. Gamma = 1.5

Parameters used:

- Epsilon = 1
- Episode count = 1000
- Learning rate = 0.1
- Discount factor = 1.5
- Epsilon decay = 0.009



- With Gamma = 1.5, we got the total reward per episode when the agent takes greedy steps from the learned policy as

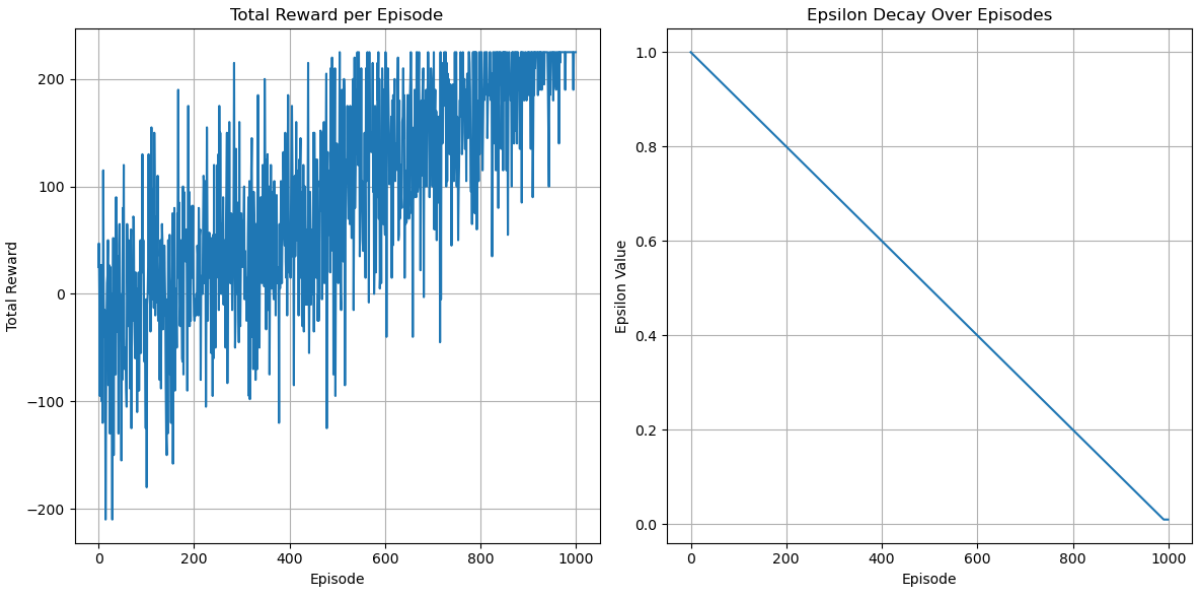


  Observation:
  - The max total reward the agent was able to achieve when the agent only chose greedy actions from the learned policy is 150 in this parameter set also.

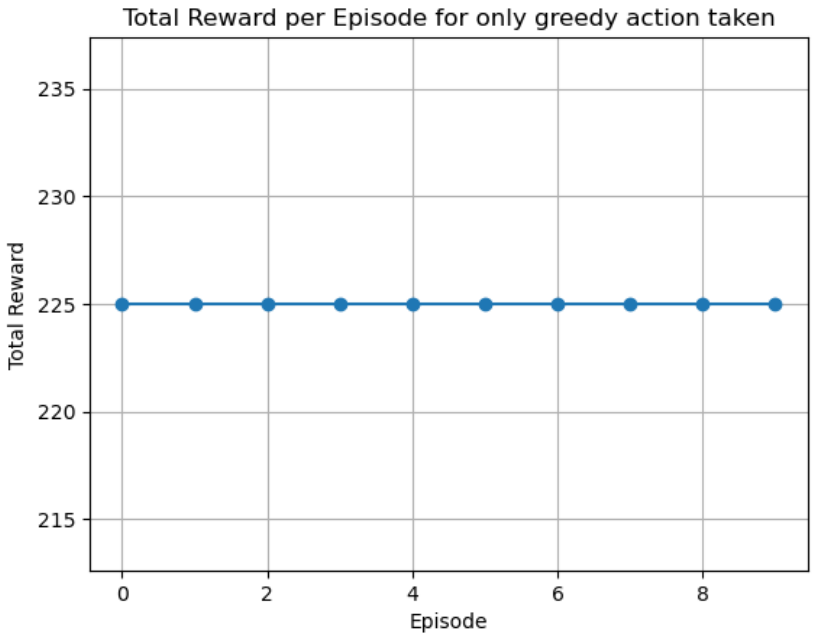
- For the gamma, we tried three different values ranging from 1.5 to 0.5 so that the agent now checks for immediate rewards more and agent likely to explore new states and actions, even if they are risky
- We observed that the total reward was almost the same for all the case. It might be because of the saturation of the rewards. Even though we are changing the gamma value we weren't getting any better rewards with the change in value.

- Epsilon decay was the second parameter we chose for tuning
  1. **Epsilon decay = 0.001**

     Parameters used:
     - Epsilon = 1
     - Episode count = 1000
     - Learning rate = 0.1
     - Discount factor = 0.9
     - Epsilon decay = 0.001



- With Epsilon decay = 0.001, we got the total reward per episode when the agent takes greedy steps from the learned policy as

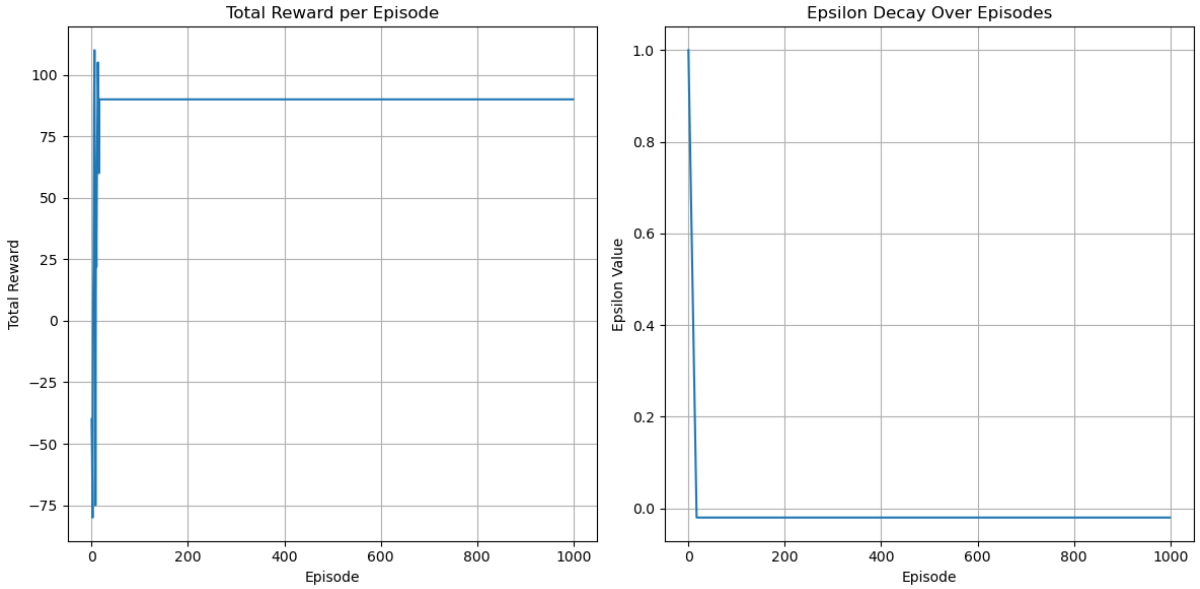


Observation:

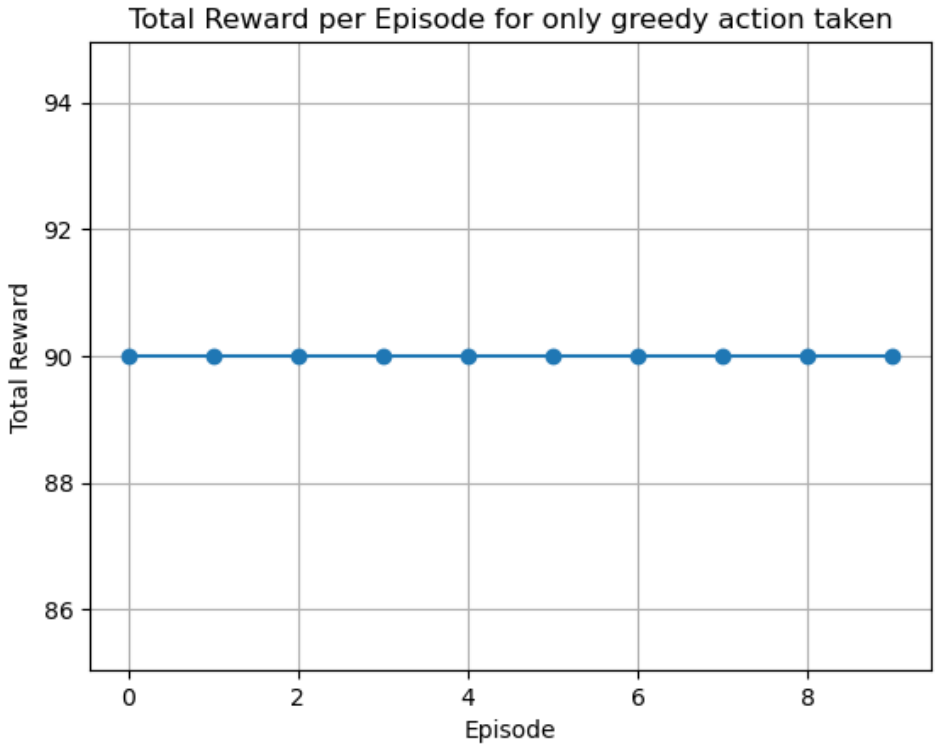
- We can see that the agent got an improved total reward of 225. This is an improvement from previous scores.

2. **Epsilon decay = 0.06**
   Parameters used:
   - Epsilon = 1
   - Episode count = 1000

- Learning rate = 0.1
- Discount factor = 0.9
- Epsilon decay = 0.06



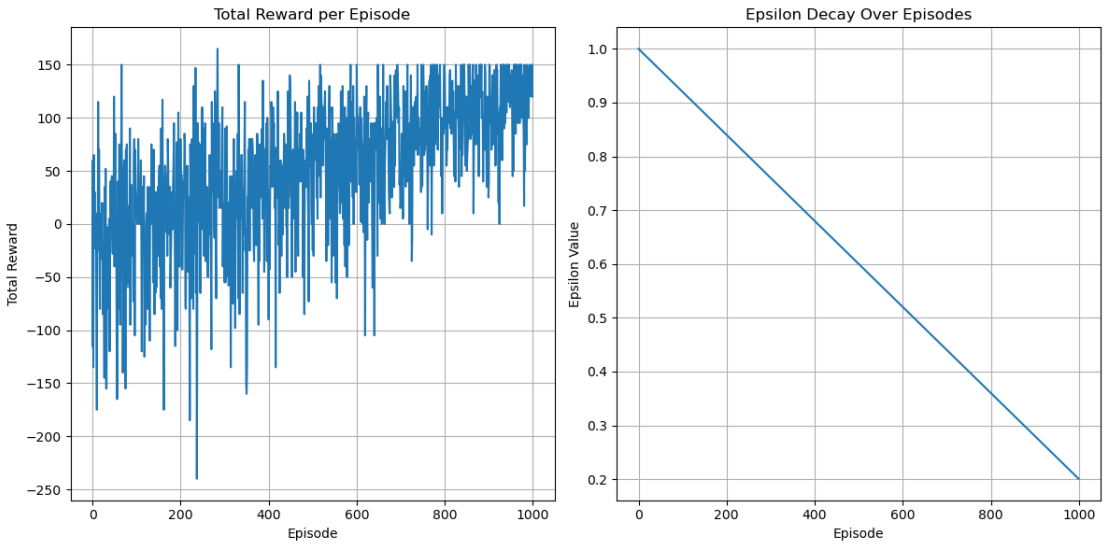
- With Epsilon decay = 0.06, we got the total reward per episode when the agent takes greedy steps from the learned policy as



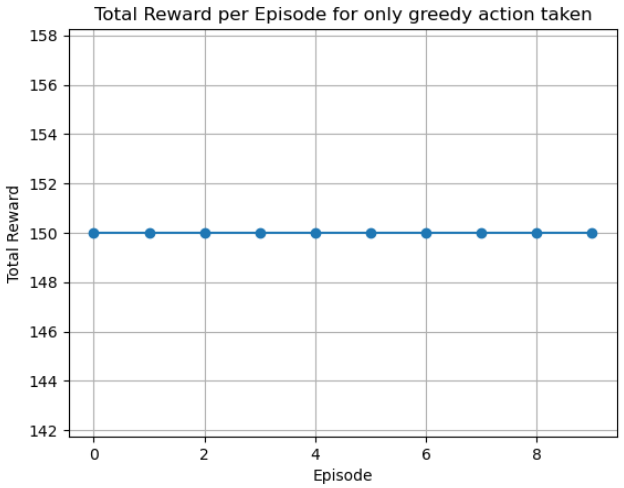
Observation:

- Compared to previous trials the result we got here is very bad. The total reward the agent was able to achieve was 90 only.

3. **Epsilon decay = 0.0008**
   Parameters used:
   - Epsilon = 1
   - Episode count = 1000
   - Learning rate = 0.1
   - Discount factor = 0.9
   - Epsilon decay = 0.0008



- With Epsilon decay = 0.0008, we got the total reward per episode when the agent takes greedy steps from the learned policy as



Observation:

    - This is also not an impressive result. Here we only got a total reward max of 150.

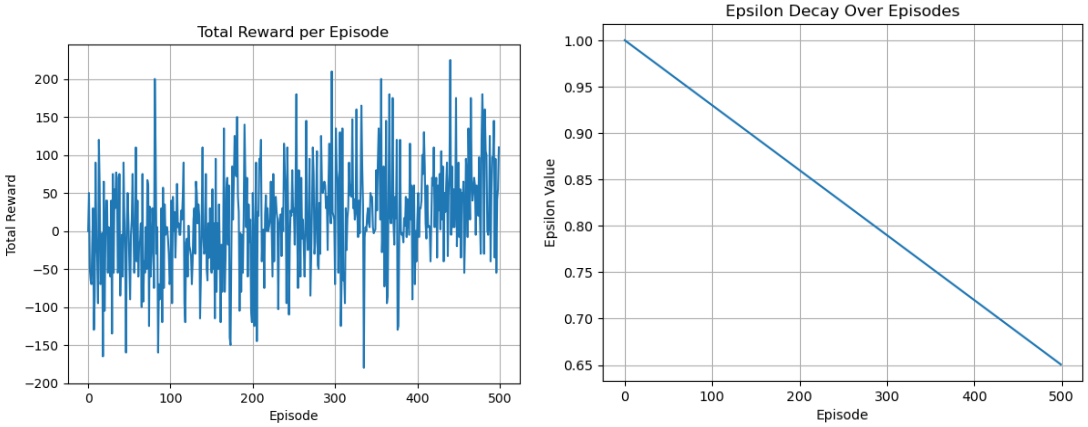
- We tried three different epsilon decay values. The lowest we tried was **0.0008.** In this case, the agent reached a maximum of 150 rewards while learning. In the other two cases, we tried giving low values like 0.06 and 0.001. In these two, the 0.001 gave a better result than 0.0008. Since the epsilon took time to decay and exploration kept on happening.
- On examining the 0.06 case we saw that the epsilon decayed to the minimum value early and the agent didn't do the exploration anymore from there and learning didn't happen. That's why we got a very bad result of 75.
- The case when the epsilon decay is 0.001 was having the perfect balance between the decay of epsilon and learning of the Q table, that's why we got the better result of 225 among others.
- From all the above experiments on the Double Q-Learning model, the best parameter set we found was that with parameters: gamma=0.9 and epsilon decay=0.001, we got the best result of total reward 225.

    Parameters used:
    - Epsilon = 1
    - Episode count = 1000
    - Learning rate = 0.1
    - Discount factor = 0.9
    - Epsilon decay = 0.001

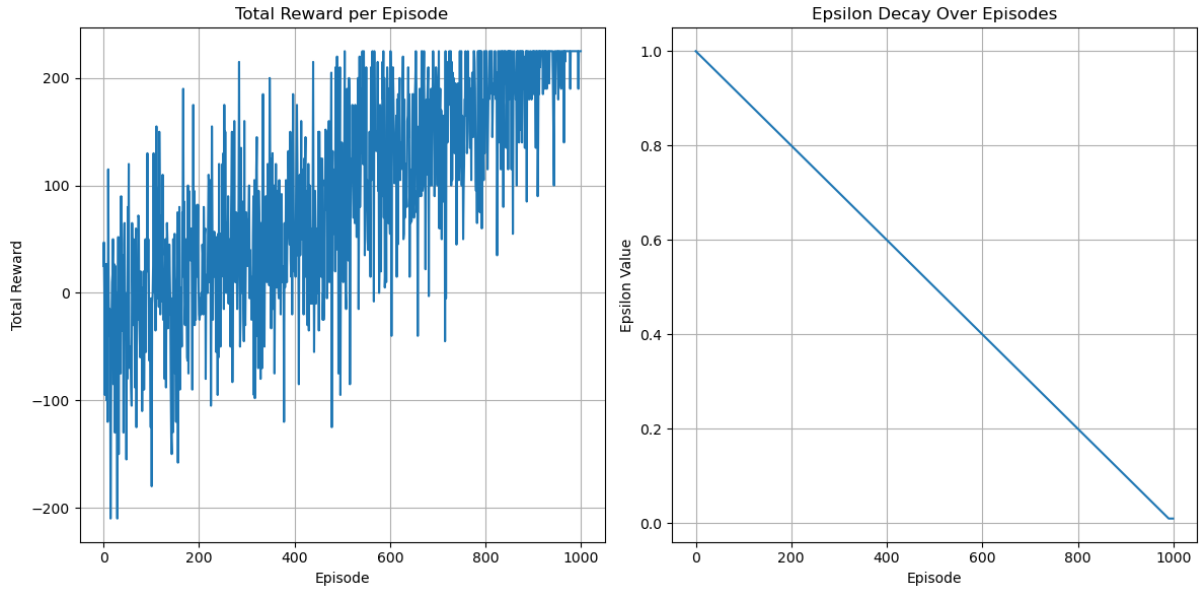
- We are choosing this set of parameters since they gave the best result for our Double Q-Learning implementation.

Comparison of SARSA and Double Q-learning in the same environment

- **SARSA best model rewards per episode graph**



- **Double Q--Learning rewards per episode graph**



As we see the final results of both Learnings are similar, Double Q-Learning has a higher average reward per episode than SARSA, On the other hand SARSA has lower average reward per episode.

1. In Q-Learning the maximum reward per episode is approximately around 230 but in SARSA it is around 200
2. From the plots, we can say that with Double Q learning the agent learns more quickly than SARSA and reaches max reward per episode quickly
3. It is more stable than SARSA and does not have the same spikes in reward per episode
4. Whereas in SARSA the agent tries to learn more slowly than Double Q-learning and reaches its maximum reward per episode later.
5. It is less stable than Double Q-Learning and has more spikes in reward per episode

In summary Double Q-Learning is a better algorithm than SARSA based on shown plots.

References:

1. **https://ubuffalo-my.sharepoint.com/personal/avereshc_buffalo_edu/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Favereshc%5Fbuffalo%5Fedu%2FDocuments%2F2023%5FFall%5FAI%2F%5Fpublic%2FCourse%20Materials%2FRL%20Environment%20Visualization%20by%20Nitin%20Kulkarni&ga=1**

2. **https://ubuffalo-my.sharepoint.com/personal/avereshc_buffalo_edu/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Favereshc%5Fbuffalo%5Fedu%2FDocument**

s%2F2023%5FFall%5FML%2F%5Fpublic%2FCourse%20Materials%2FRL%20Environment%20Demo&ga=1

3. https://optuna.org/

Contribution		
Dharma. Acha	**dharmaac**	**50%**
Gokul	**gokulpul**	**50%**