REPORT

We have defined an environment called LawnmoverEnv where the lawnmover will travserve through different cells in the grid.

There are total 16 different states since the environment since it is a 4*4 grid. {S1 = (0,0), S2 = (0,1), S3 = (0,2),

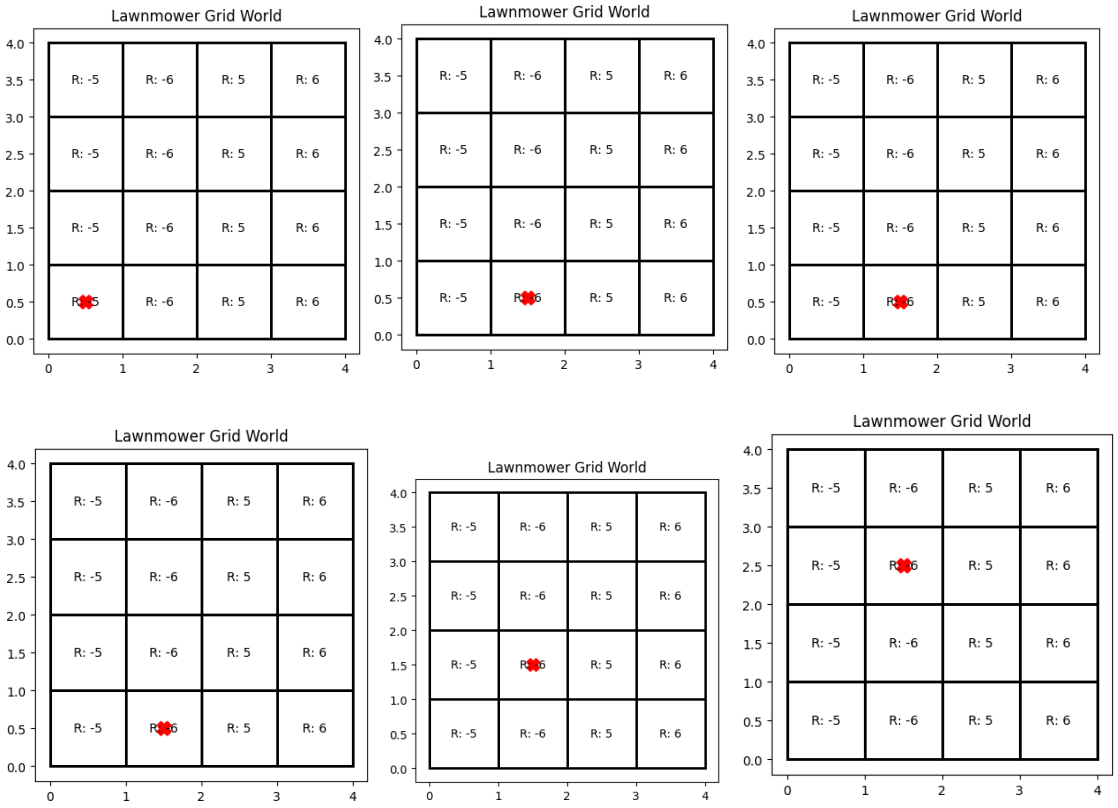
S4 = (0,3), S5 = (1,0), S6 = (1,1), S7 = (1,2), S8 = (1,3), S9 = (2,0), S10 = (2,1), S11 = (2,2), S12 = (2,3), S13 = (3,0), S14 = (3,1), S15 = (3,2), S16 = (3,3)}.

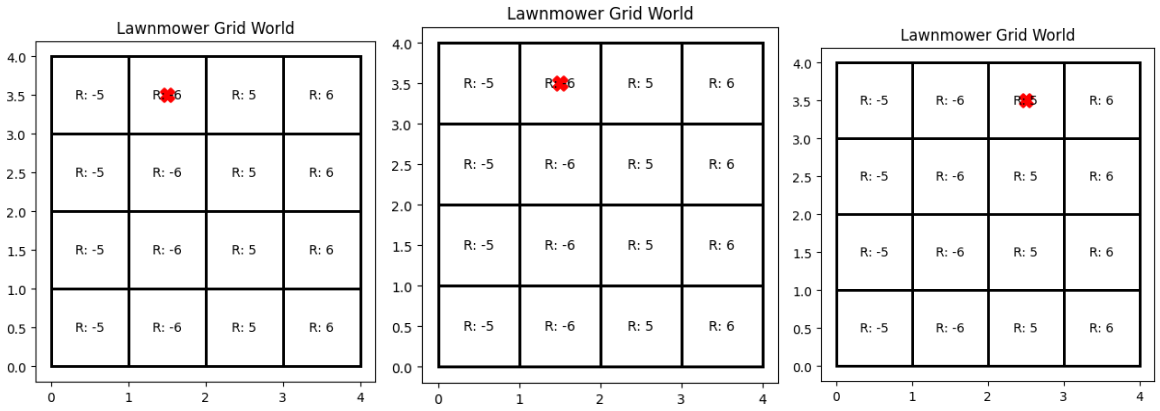
These are the states we have defined. The agent here will take 4 actions which we defined as up, down, right , left.

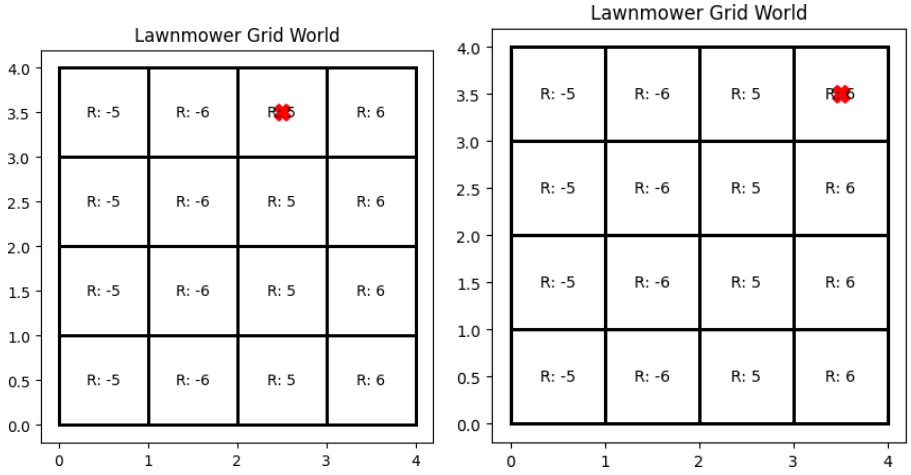
The rewards are coordinated to the particular cells in the grid. We have taken 4 rewards -5, -6 ,+5 ,+6.

The main objective is to reach the goal state with maximum reward i.e., +6 in our case. The goal state is (3,3).

1. Provide visualization of your environment.







Here the initial state was (0,1) and the goal state is (3,3) . It has reached goal state after 11 time steps.

2. Safety in AI:

The environment is defined in such a way that agent can take 4 actions, Up, down, left, right,. This ensures that it goes in these 4 directions which we have mentioned in our step function. If any other action is done other than these, it will automatically get redirected to one of these 4 actions. In step method while updating agent state based upon agent action our Lawnmoverenv checks the noundaries of the grid to ensure that the agent stays or moves inside the grid itself. We have also used pre defined rewards for each cell which ensures that agent doesn't get any random rewards and acts according to our specified inputs. This way we have ensured that the agents navigates in a proper way according to the predefined conditions.

The tabular methods that were used to solve the problems

I have used two methods named, SARSA(State Action Reward State Action) and

Double Q learning method. In SARSA we have one initialise one Q table and in

Double Q learing we initiliase 2 Q tables.

Update functions for SARSA:

In update function of SARSA we have updated Q table. In update function, we have indexed our states, by getting our current and next q values from our q table. We have updated the q value for the current state action pair based on the SARSA, that will take immediate reward into coonsideration and it will determine rewards in future.

The below screenshot is our update function

```
def update_q_table(self, state, action, reward, next_state, next_action):
  #using sarsa update q table
    state_index = self.env.states.index(state)
    next_state_index = self.env.states.index(next_state)

    current_q = self.q_table[state_index, action]
    next_q = self.q_table[next_state_index, next_action]
    updated_q = current_q + self.alpha * (reward + self.gamma * next_q - current_q)
    self.q_table[state_index, action] = updated_q
```

- During training the model, SARSA uses epsilon greedy strategy for selecting an action.
- It has only one Q table
- Considering agents experiences, SARSA updates Q values.

Advantages for SARSA:

- Can be easily implemented
- Highly compatible with different models like neural networks

Disadvantages for SARSA:

- It is not robust to different types of hyperparameter values
- Cannot be applicable to larger states

Update function for Double Q learning:

In update function in Double Q learning, we have updated Q tables since we have 2 Q tables in this method. In update function, the updation is done on one of the two Q tables during the consideration of the maximum Q value from the other table. There is not any pre defined criteria to choose which Q table need to be considered , that can be completely random.

```
def update_q_tables(self, state, action, reward, next_state):
    state_index = self.env.states.index(state)
    next_state_index = self.env.states.index(next_state)
    y=True
    z=False
    if random.choice([y, z]):
        max_next_action = np.argmax(self.q1_table[next_state_index, :])
        self.q1_table[state_index, action] += self.alpha * (
            reward + self.gamma * self.q2_table[next_state_index, max_next_action] -
            self.q1_table[state_index, action]
        )
    else:
        max_next_action = np.argmax(self.q2_table[next_state_index, :])
        self.q2_table[state_index, action] += self.alpha * (
            reward + self.gamma * self.q1_table[next_state_index, max_next_action] -
            self.q2_table[state_index, action]
        )
```

Key features of Double Q learning:

- It has 2 Q tables
- It helps in reducing overestimation

Advantages for Double Q learning:

- Reduces overestimation bias
- More stability
- More accurate

Disadvantages of Double Q learning:

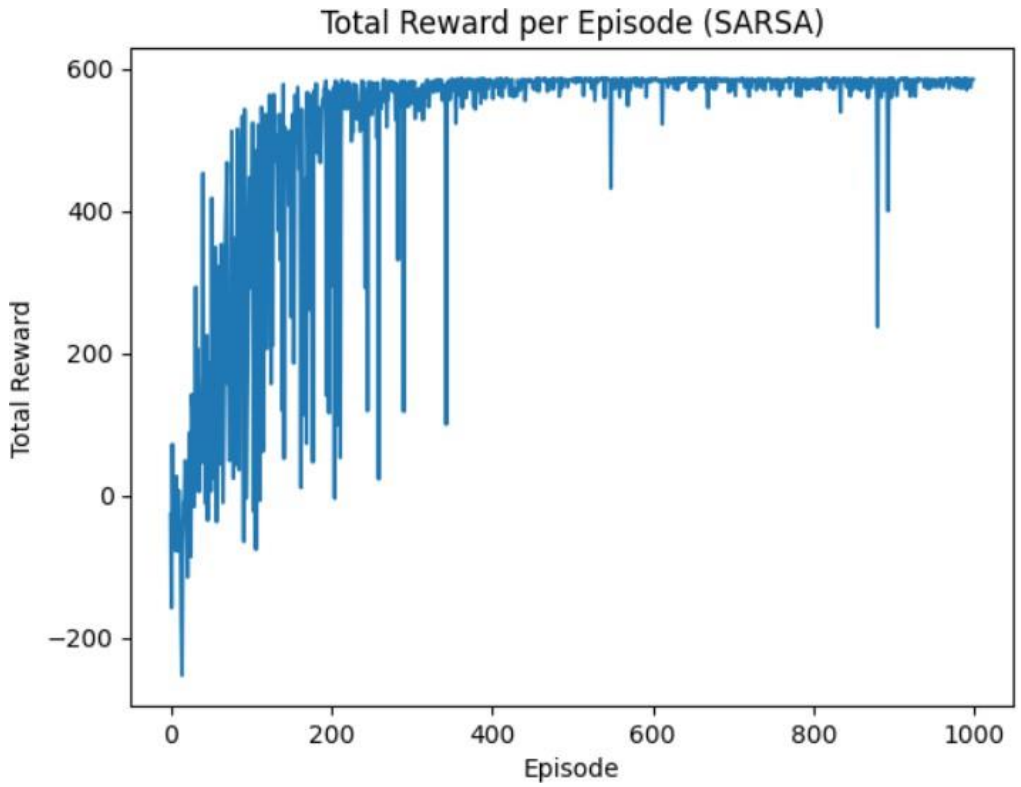
- More complex structure
- Not benefitable for all types of actions

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

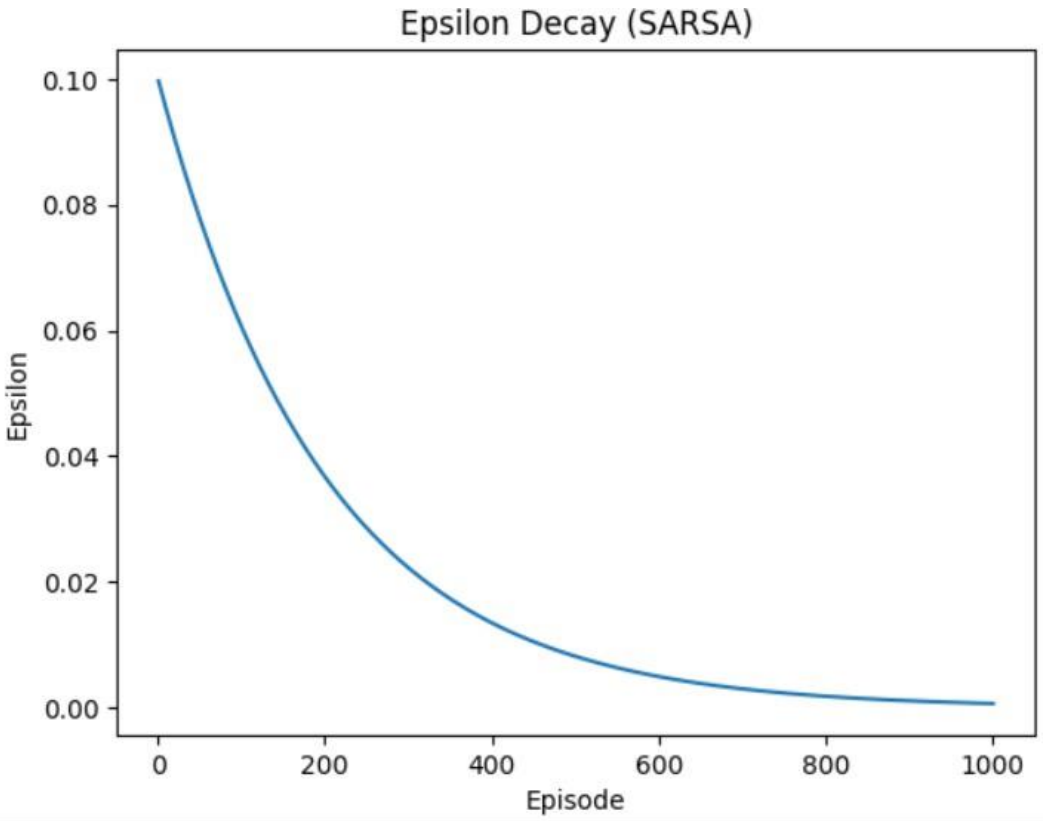
This is the initial q table which is all intialised to 0.

```
  trained q-table:
[[-4.00729043 -5.92571738 -5.75011239 -2.02730893]
 [-4.71094692 -4.19861888  2.53201011 -4.86518048]
 [ 1.73655169  3.02332637  5.25466639 -2.51931446]
 [ 3.37250423  5.58367713  5.06733765  3.2047327 ]
 [-3.4971996  -6.2160635  -4.5371079  -4.99060918]
 [-3.08419457 -3.96908725  4.8141218  -5.51550613]
 [ 3.09469833  2.02874193  6.20447883 -3.85453957]
 [ 4.66194645  5.85889661  6.39868837  3.7501746 ]
 [-4.84992257 -4.16102988 -6.6315501  -6.11026996]
 [-5.2077988  -3.23417716  3.92329606 -5.22734734]
 [ 2.34379818  2.75011802  5.85571245 -4.76582669]
 [ 4.44141628  2.0634      5.98378734  4.80105201]
 [-3.81825857 -2.3056123  -4.2964568  -3.28637578]
 [-2.9309036  -4.48715202  2.47389072 -2.72290073]
 [ 1.86030489  0.64634     2.45706    -2.09075852]
 [ 0.          0.          0.          0.        ]]
```

The above is the screenshot for the Trained Q table using SARSA. Here, the Q values have been updated on how the agent has been experiencing the environment(Lawnmover) Here, each row is a state and each column is an action. More the Q values greater is the model learning our algorithm.



The above is the graph for the total reward per episode plotted against total reward vs episode. We can observe as the number of episodes are increasing the total rewards are also increasing and after particular number of episodes it has shown a constant increase. The peak can be observed at the episode 200 where the total reward is approximately 590.



The above is a epsilon decay graph. In this we can observe that as the number of episodes are increasing the epsilon values are decreasing and this is a common behaviour shown by our agent during training this means our model is exploring well.

• Applying Double Q-learning to solve the environment defined in Part 1. Include Q-tables. Plots should include epsilon decay and total reward per episode. Include the details of the setup that returns the best results.

Ans:

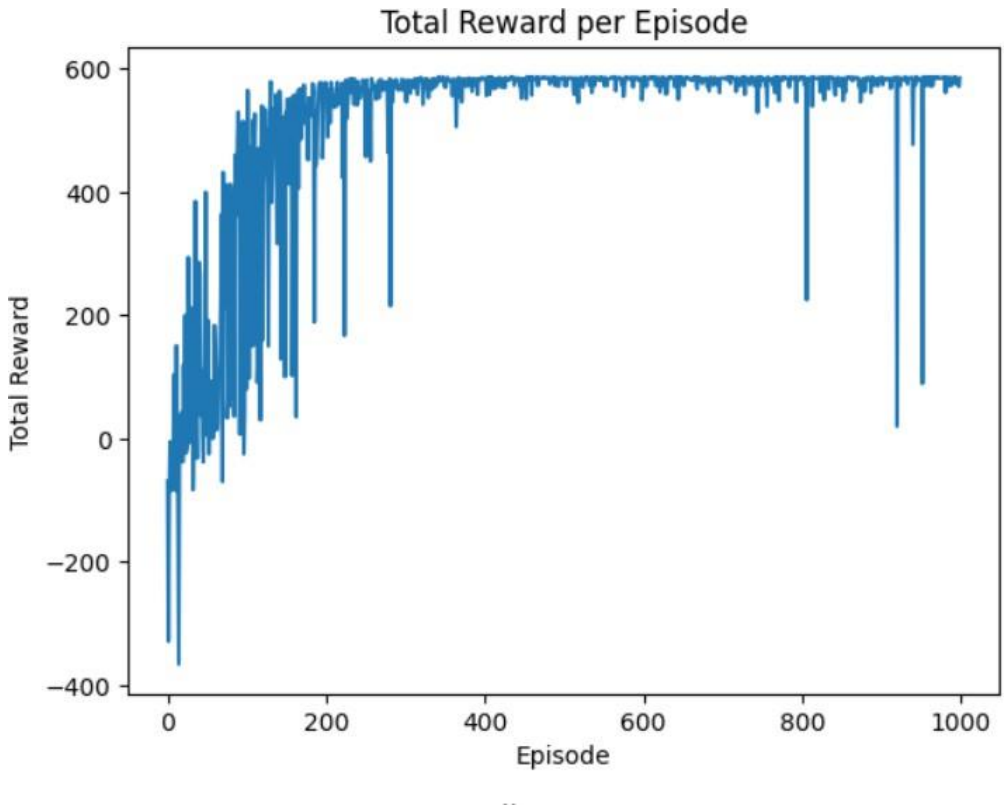
```
Initial Q-tables:
Q1:
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
Q2:
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [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.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

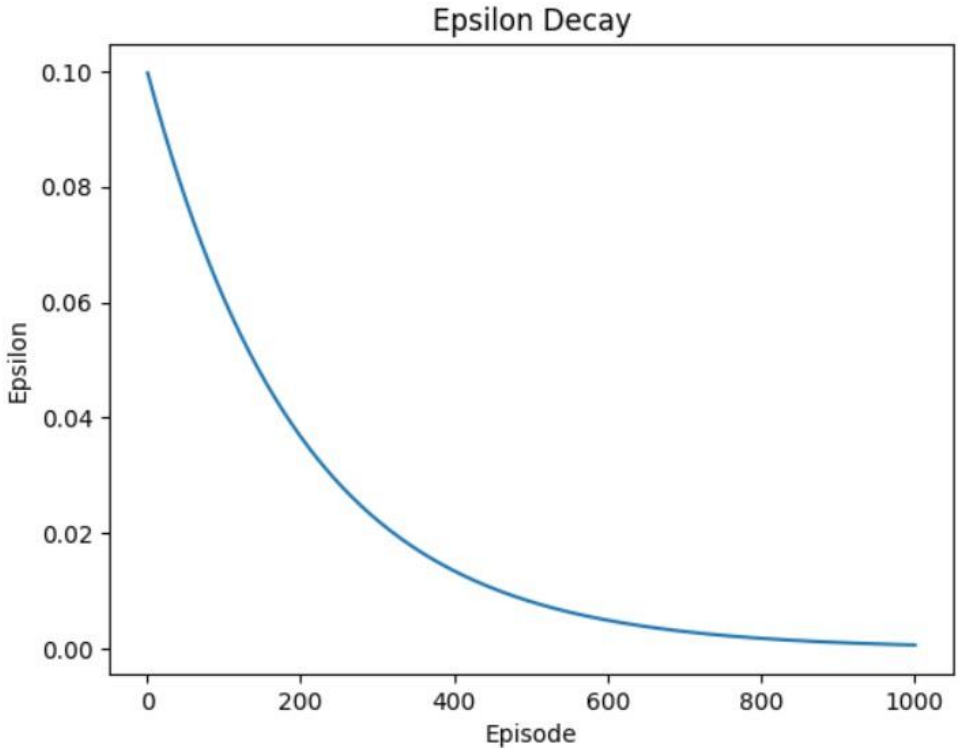
The above is the initial q table which is initialised with 0.

```
Trained Q-tables:
Q1:
[[34.36691124 29.80255289 47.1        36.8629107 ]
 [45.57104311 46.61702801 59.         36.7493773 ]
 [58.99988332 58.99758805 60.         47.09991931]
 [60.         60.         60.         59.        ]
 [21.07896407 -4.16759629 45.27565562 11.67369293]
 [39.37108189 21.87815574 58.9701686  11.87467462]
 [58.9999232  51.42202906 59.99999983 45.70587703]
 [60.         59.99768352 59.99997272 58.99964803]
 [ 2.5415541  -5.13737693 12.83340753 -5.08998063]
 [ 5.07876109  3.3627264  51.10041943 -2.56937119]
 [58.27084652 28.6993028  39.85223137 20.13265441]
 [59.99954342  5.9113147  49.80900194 49.29367029]
 [-2.11518613 -3.51726589  1.45291355 -4.19129154]
 [ 0.78948682  2.63150135 20.78765026 -4.53211151]
 [41.46335139 10.86597945  3.1302186   4.90572265]
 [ 0.          0.          0.          0.        ]]
Q2:
[[34.77286758 30.0832456  47.1        34.43612178]
 [46.1805585  45.34838426 59.         36.84835032]
 [58.99999262 58.998461   60.         47.09993551]
 [60.         60.         60.         59.        ]
 [ 6.29077129 -2.85122843 44.7880039  11.75140388]
 [31.1384244  18.62209434 58.95693733 18.49141672]
 [58.99997497 53.55255227 60.         45.04807916]
 [60.         59.99787456 59.99983456 58.99967968]
 [ 1.33297065 -5.09894591 13.04979314 -3.35507451]
 [15.06007278  2.4650752  39.77305169 -2.61495739]
 [57.73230413 25.2357867  40.34383717 15.97809779]
 [59.99965154  5.71739228 49.61674192 38.11653175]
 [-3.28610642 -4.75971611  1.65548614 -4.42842134]
 [ 4.4645755   0.98564058 23.54333664 -4.44980499]
 [39.57098384 16.6133624   5.09943219  0.97393812]
 [ 0.          0.          0.          0.        ]]
```

The above is the Trained Q tables In the above trained Q tables, Q1 and Q2, they are having high values which says that our model is learning well. Reduction of overestimation bias can be achieved.

The below is the graph for total reward per episode. In this graph, we can observe that, as the number of episodes are increasing , rewards are also getting increasing. At some point that became constant but again they started increasing with tiny differences. So, eventually it is in a increasing trend which specifies that our model is learning well.



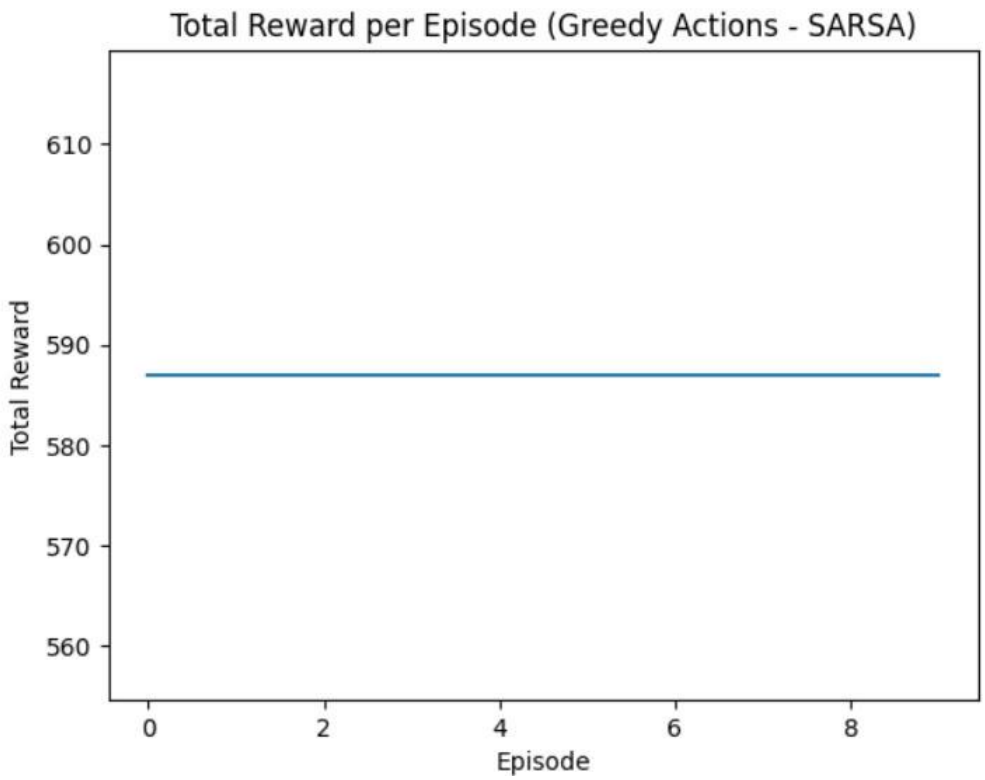


The above is the epsilon decay graph plotted for epsilon values against episode. It is in a decreasing trend. In this we can observe that as the number of episodes are increasing the epsilon values are decreasing and this is a common behaviour shown by our agent during training this means our model is exploring well.

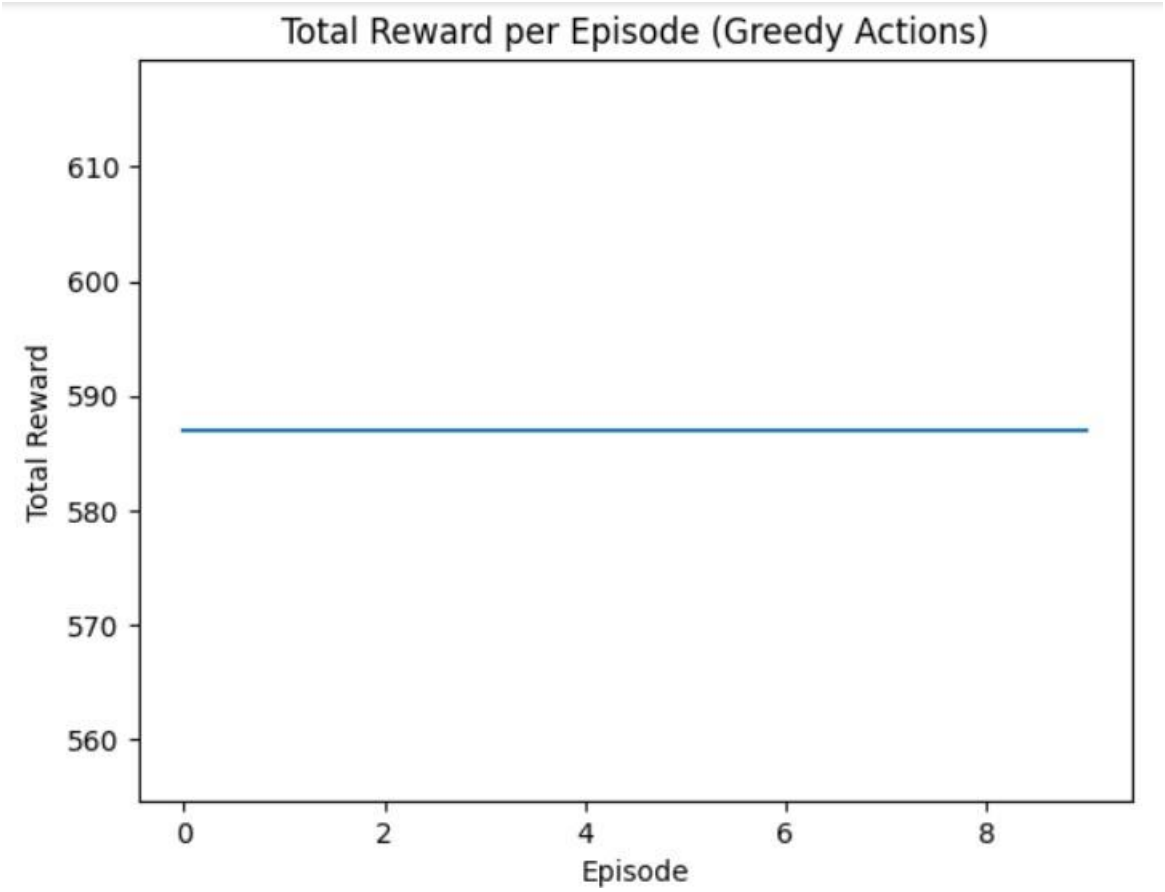
- Provide the evaluation results for both SARSA and Double Q-learning.

Run your environment for at least 10 episodes, where the agent chooses only greedy actions from the learned policy. Plot should include the total reward per episode.

Ans: SARSA



Double Q learning

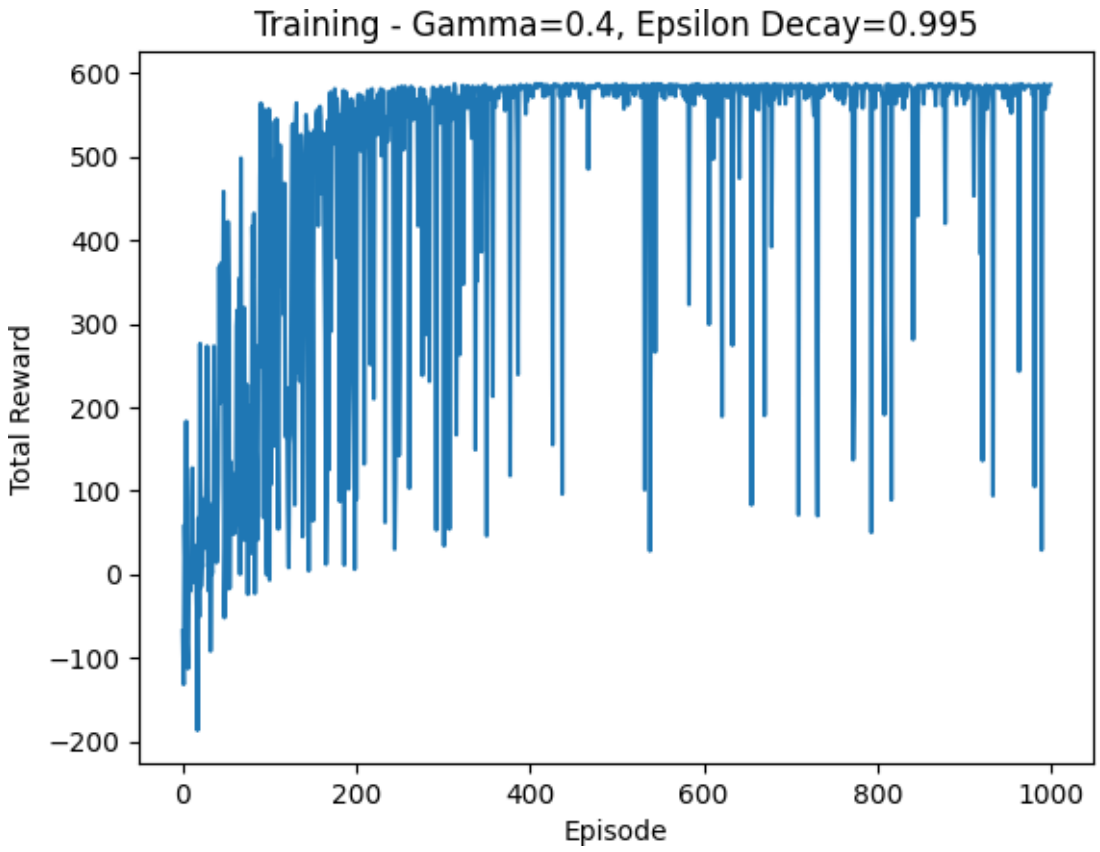


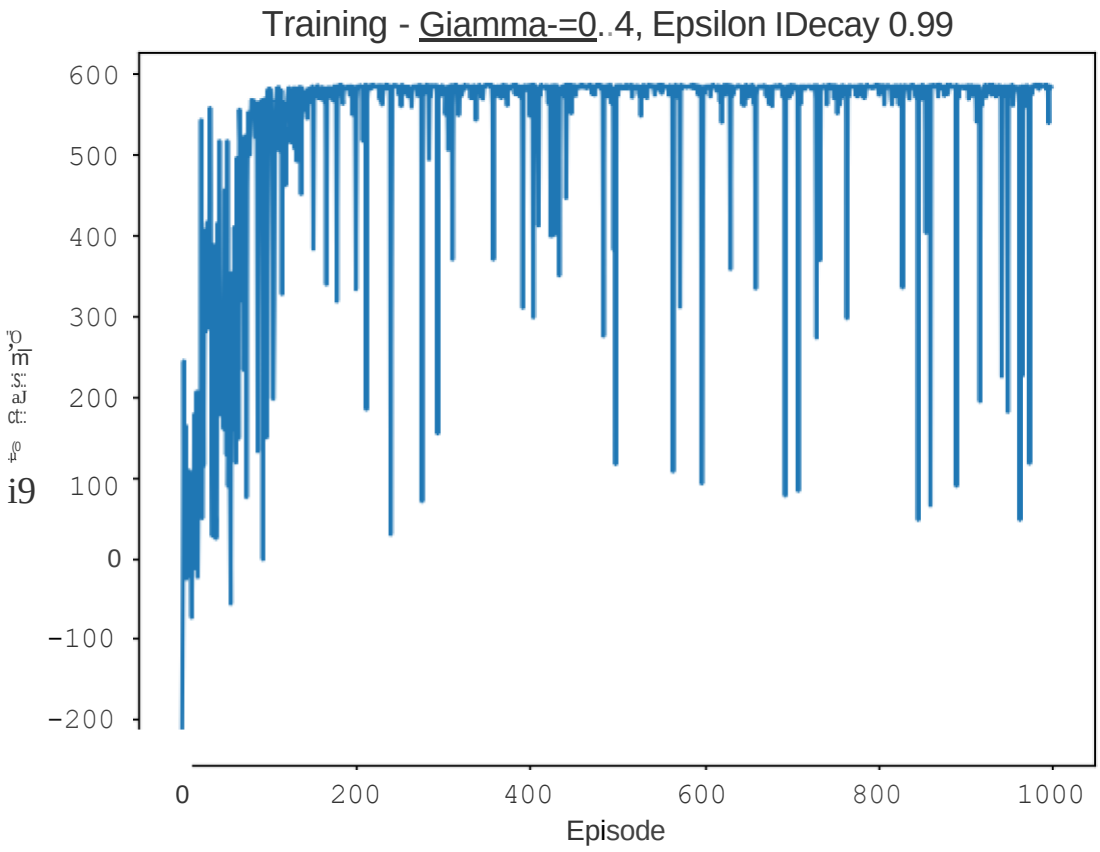
During the testing, our rewards are not changed among different number of episodes for both the methods. Here, the total rewards for both the algorithm are same since both the algorithm

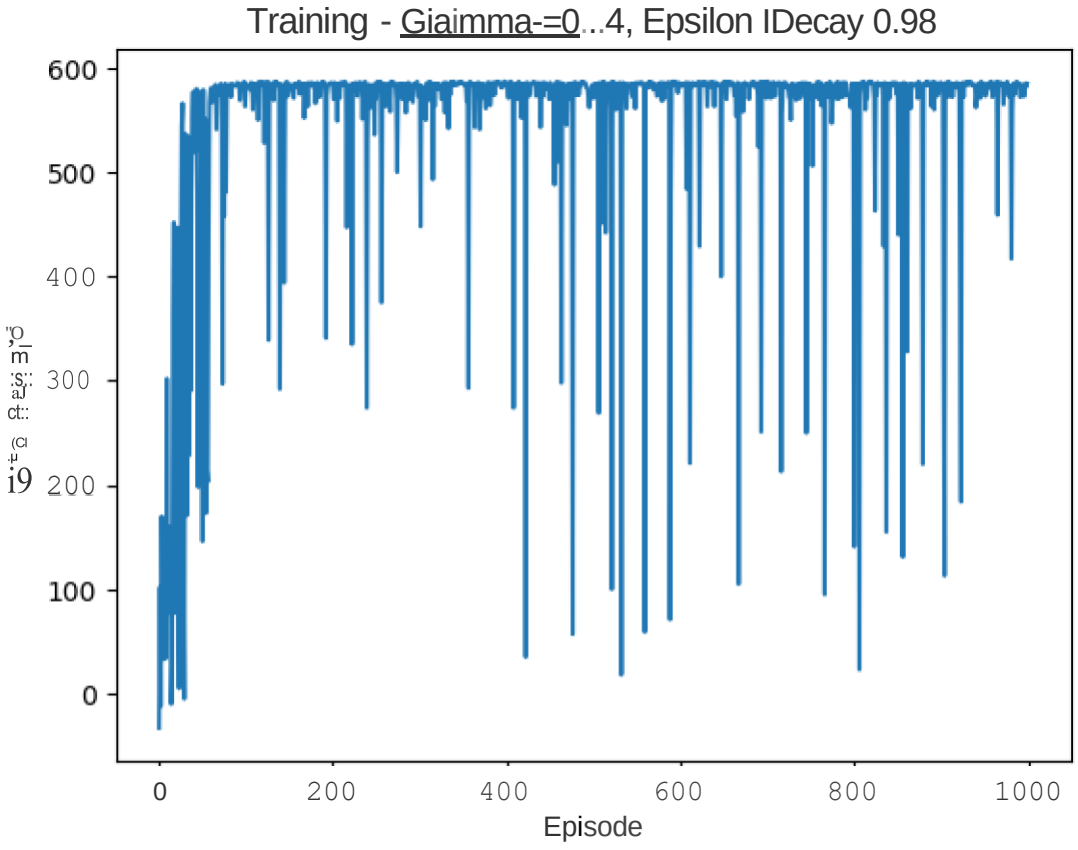
are following greedy approach to find the optimal way.

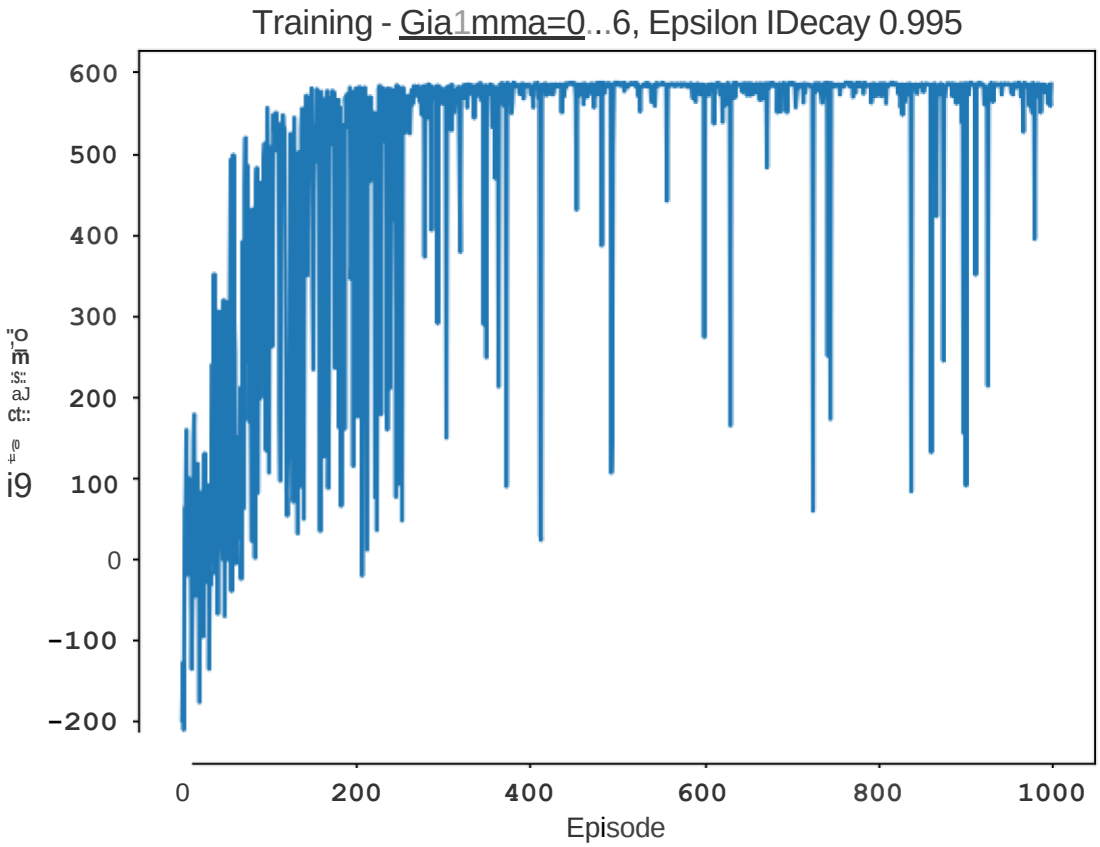
2. Provide the analysis after tuning at least two hyperparameters from the list above. Provide the reward graphs and your explanation for each of the results. In total, you should have at least 6 graphs for each implemented algorithm and your explanations. Make your suggestion on the most efficient hyperparameters values for your problem setup.

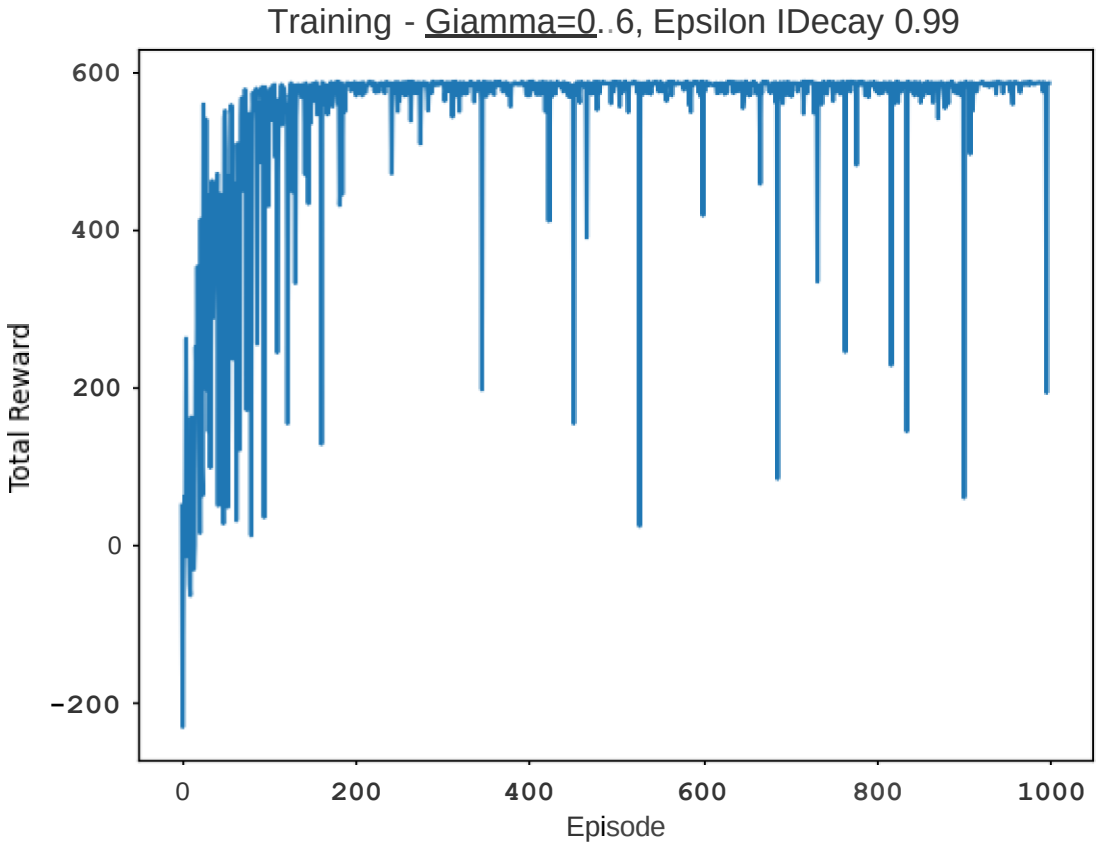
Ans:

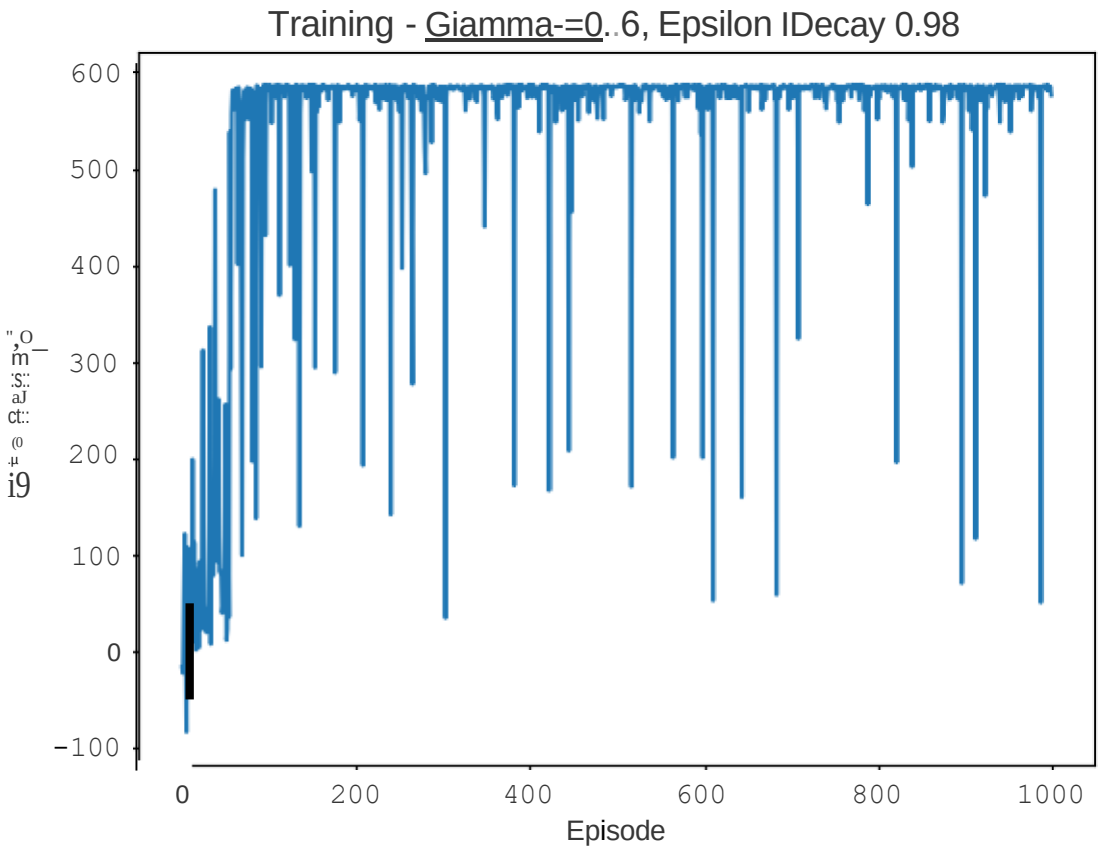


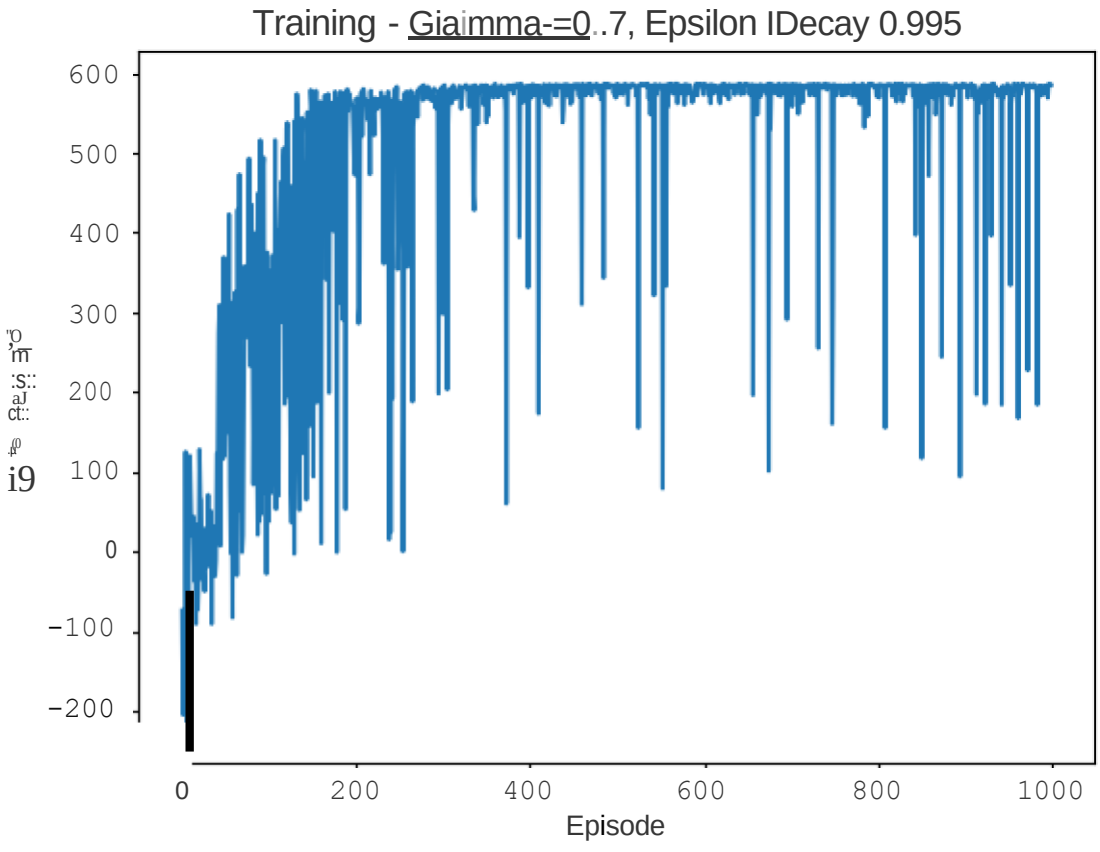

Training - Giamma-=0..4, Epsilon IDecay 0.99
600
500
400
300
200
100
0
-100
-200
0
200
400
600
800
1000
Episode

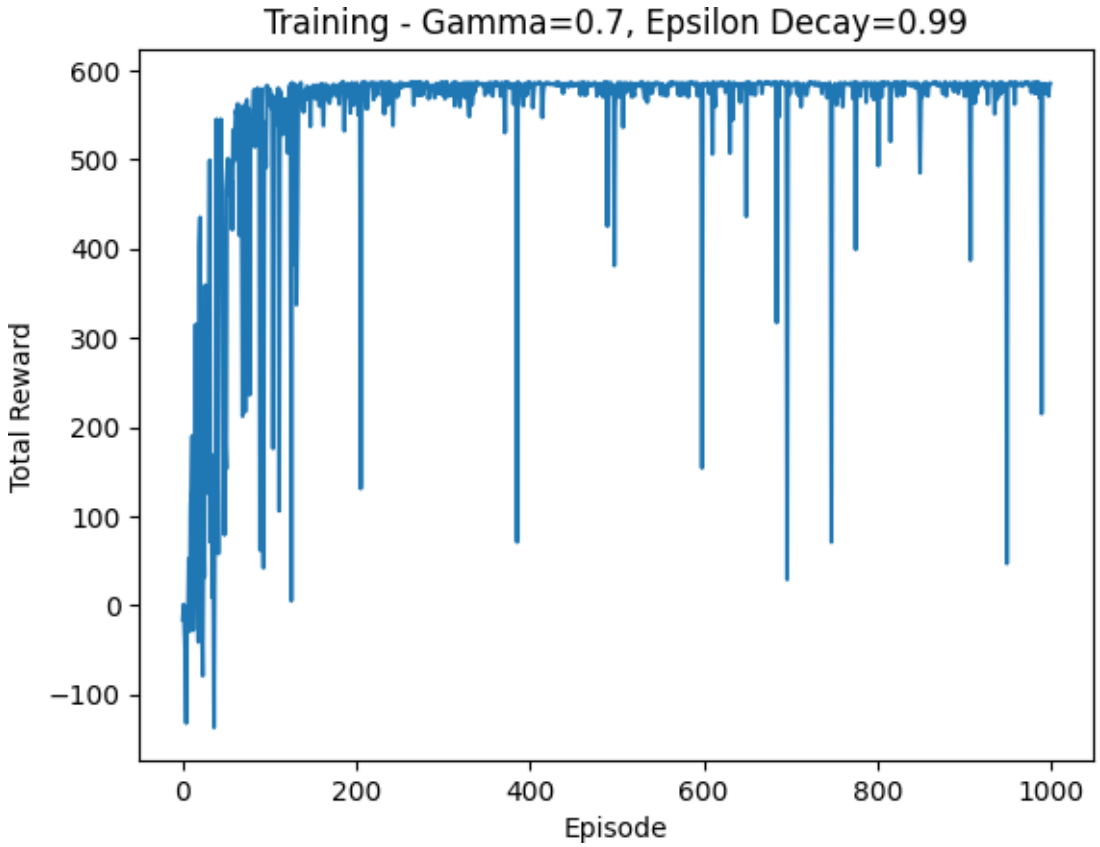

Training - Giaimma-=0...4, Epsilon IDecay 0.98
600
500
400
300
200
100
0
0
200
400
600
800
1000
Episode

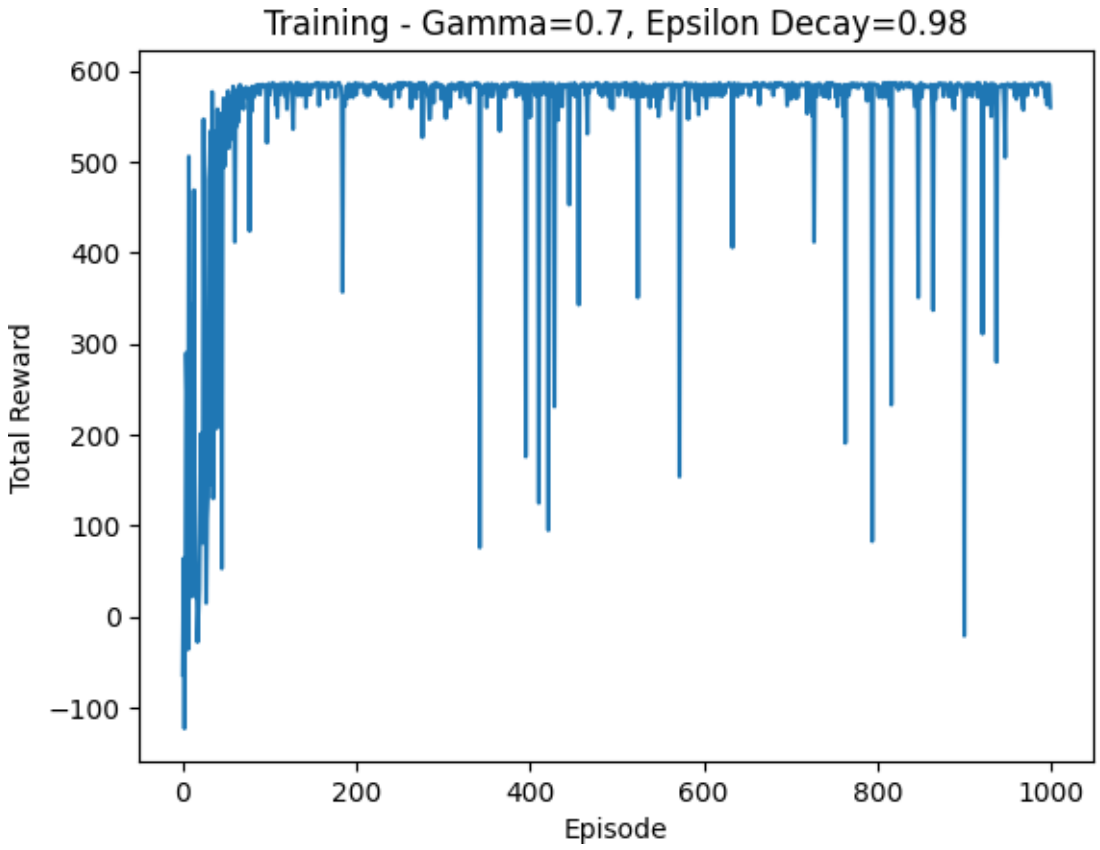

Training - Gia1mma=0...6, Epsilon IDecay 0.995
600
500
400
300
200
100
0
-100
-200
0
200
400
600
800
1000
Episode


Training - Giamma=0...6, Epsilon IDecay 0.99
Total Reward
600
400
200
0
-200
0
200
400
600
800
1000
Episode


Training - Giamma-=0..6, Epsilon IDecay 0.98
600
500
400
300
200
100
0
−100
0
200
400
600
800
1000
Episode

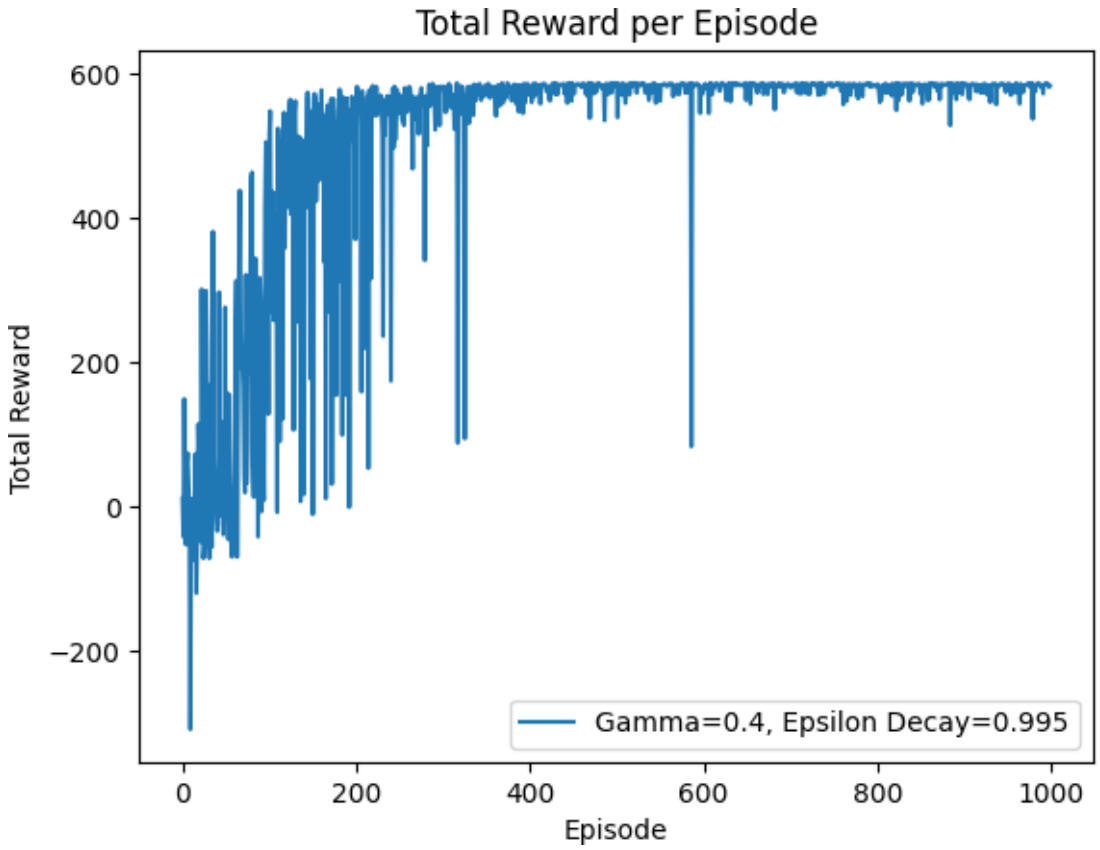

Training - Giaimma-=0..7, Epsilon IDecay 0.995
600
500
400
300
200
100
0
−100
−200
0
200
400
600
800
1000
Episode

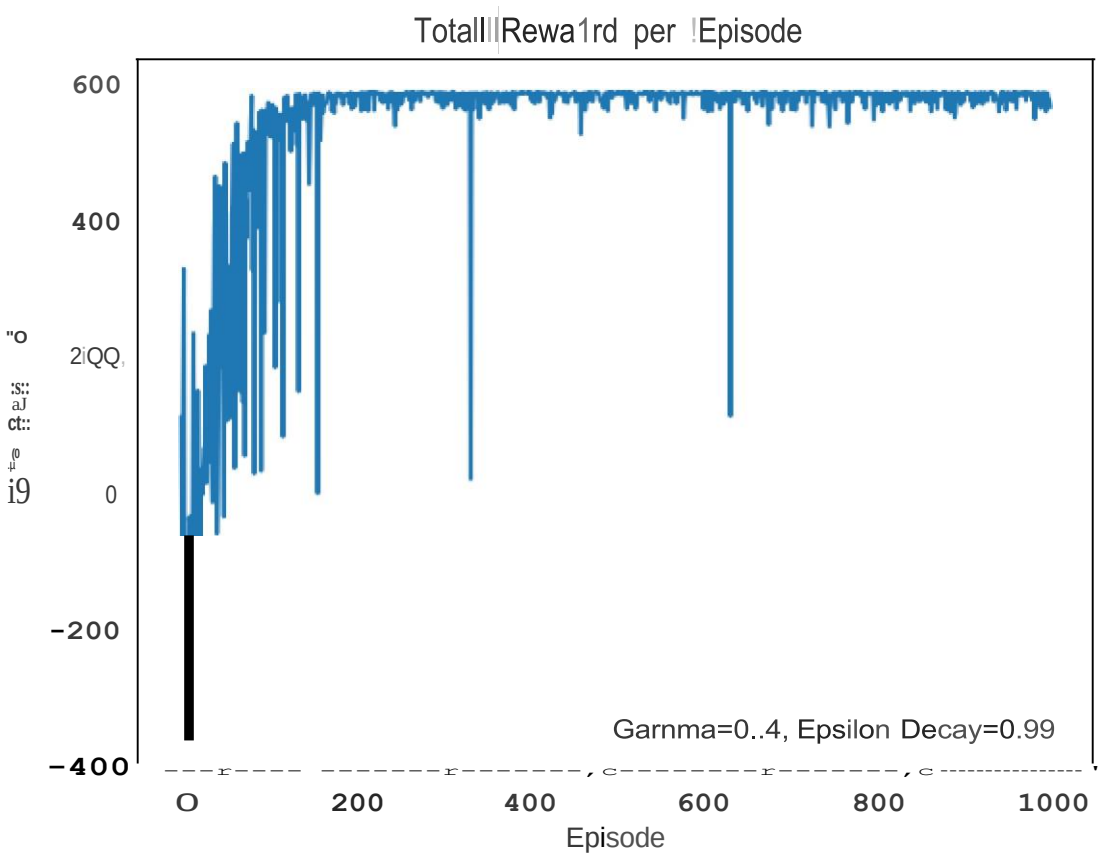


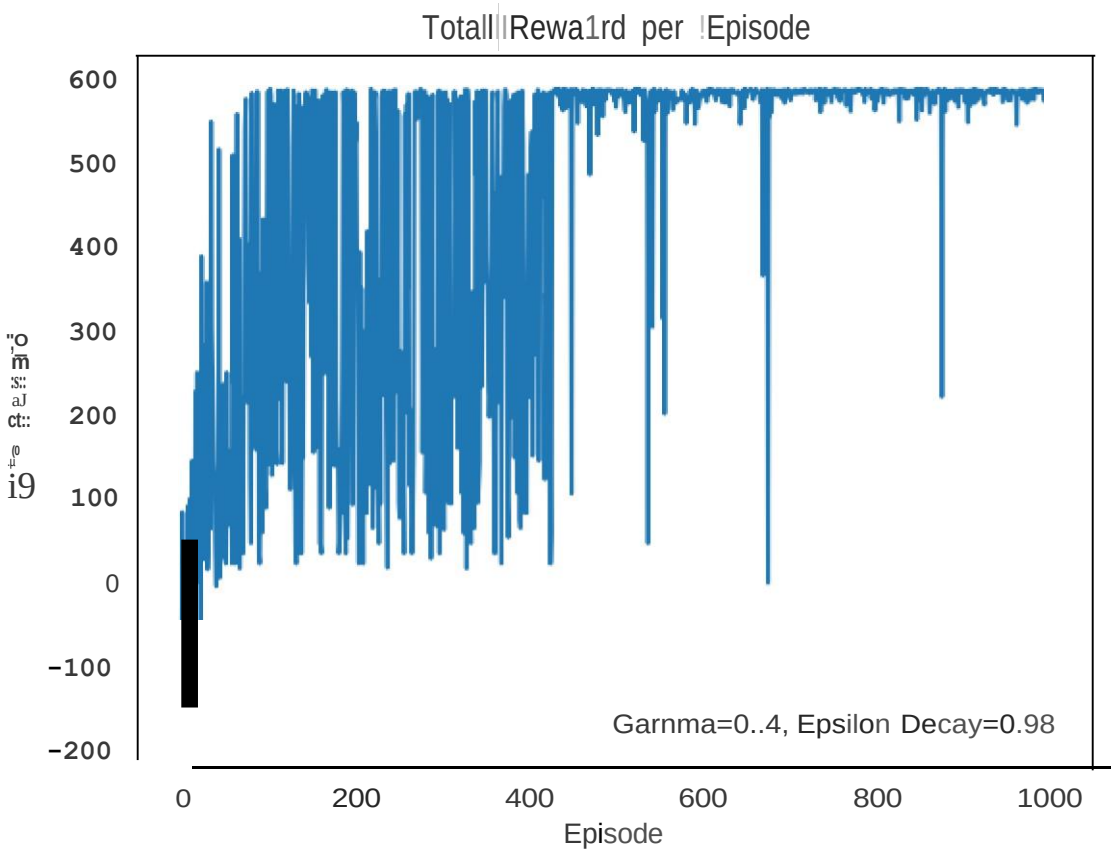


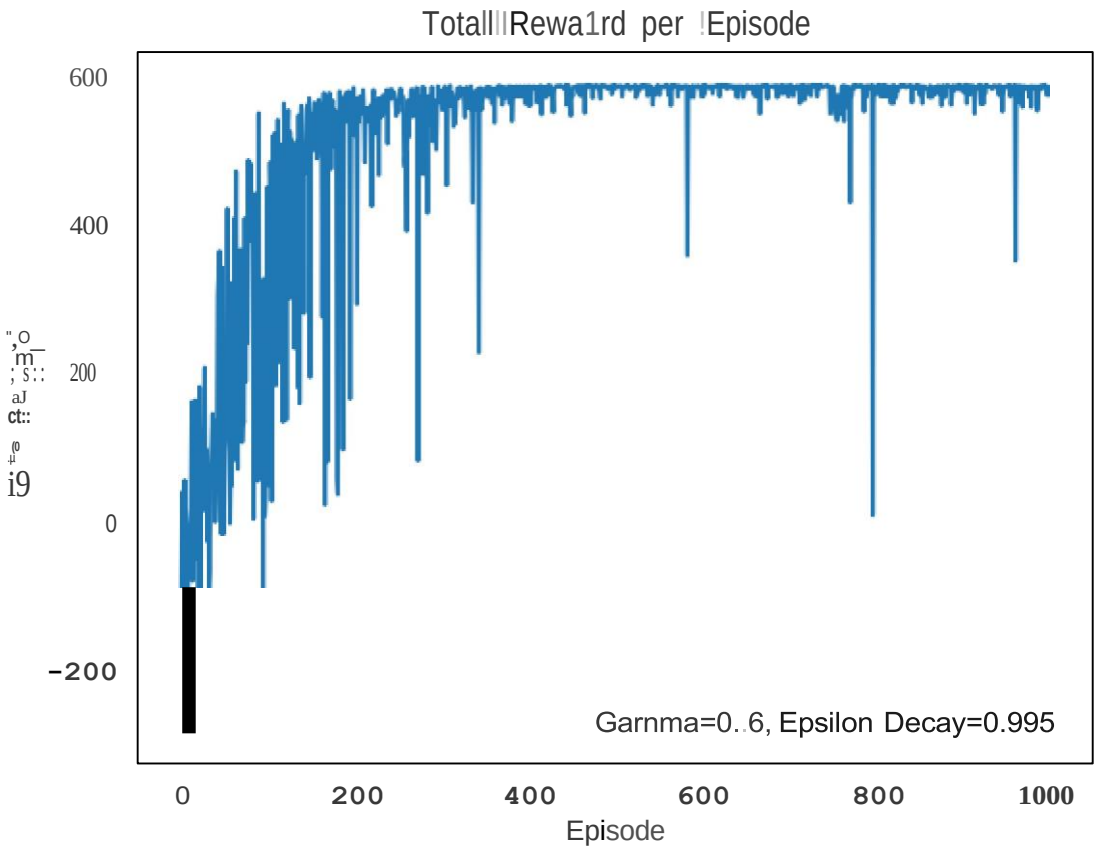
After observing the above graphs for rewards per episode, we can say that Gamma =0.7 and Epsilon decay =0.98 are the efficient hyperparameters since they are converging faster.

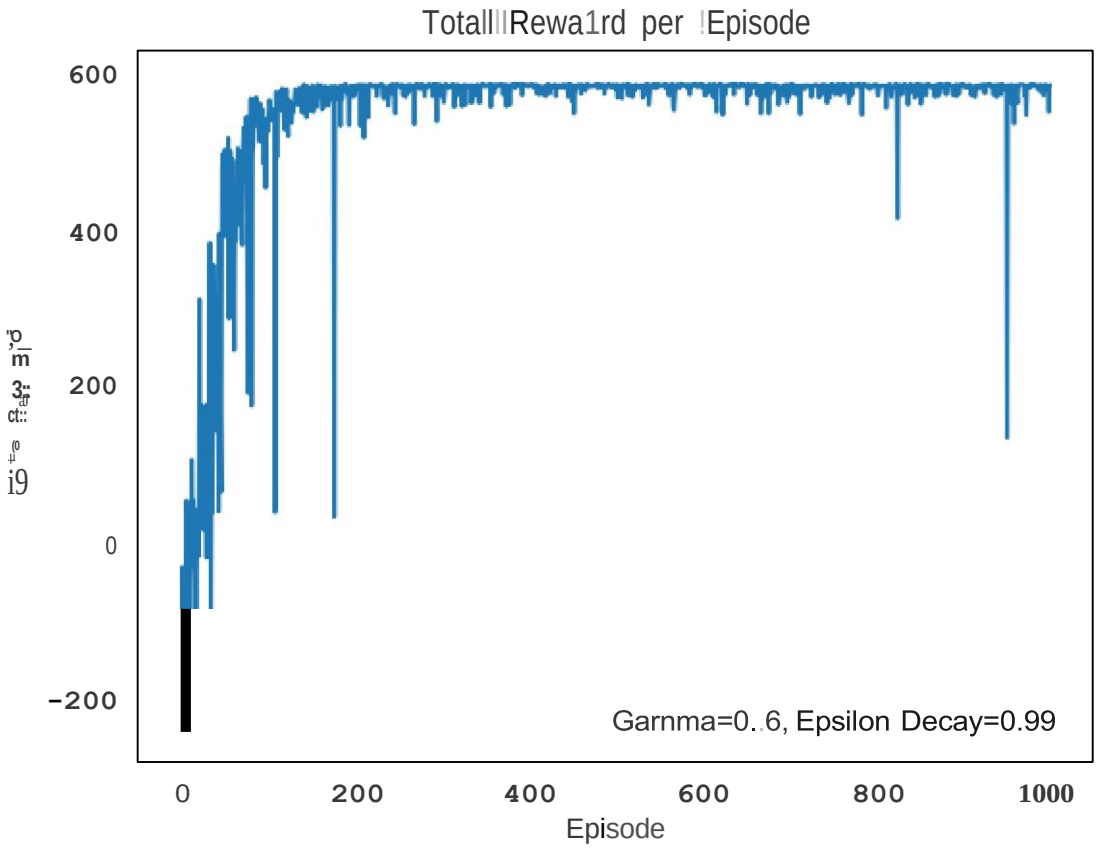
DOUBLE Q LEARNING

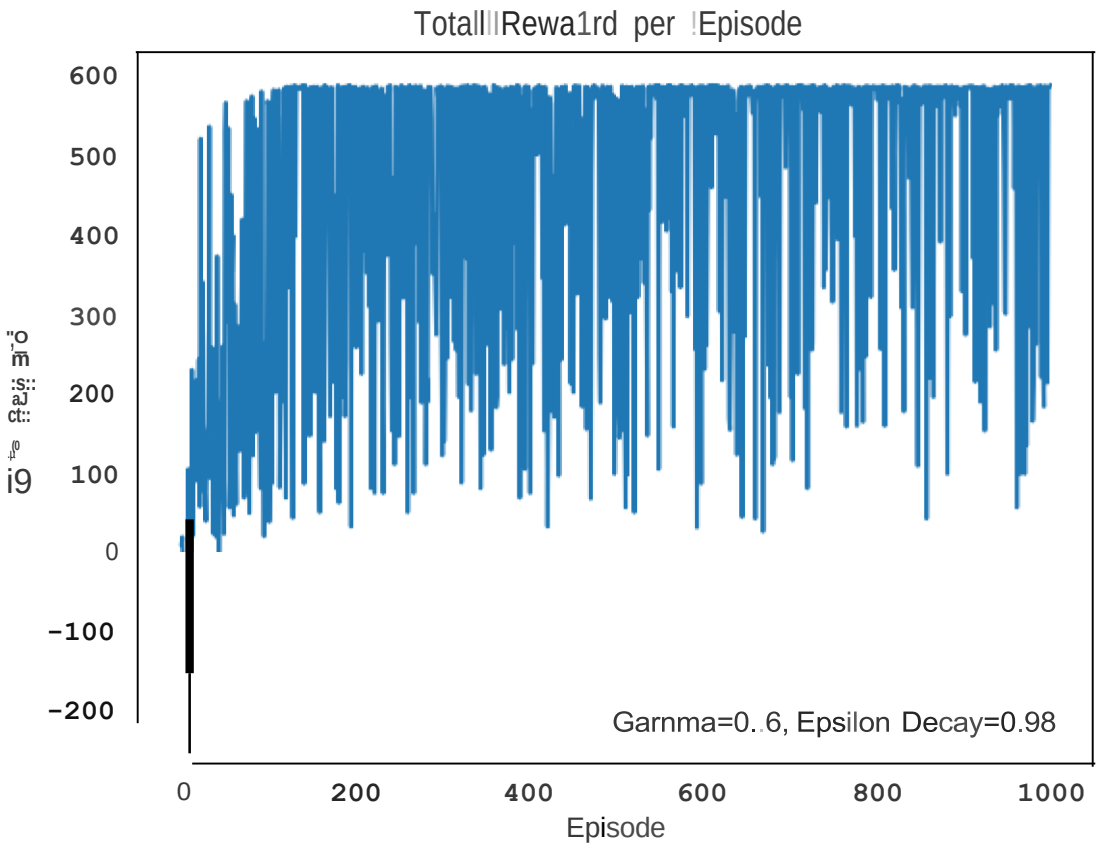


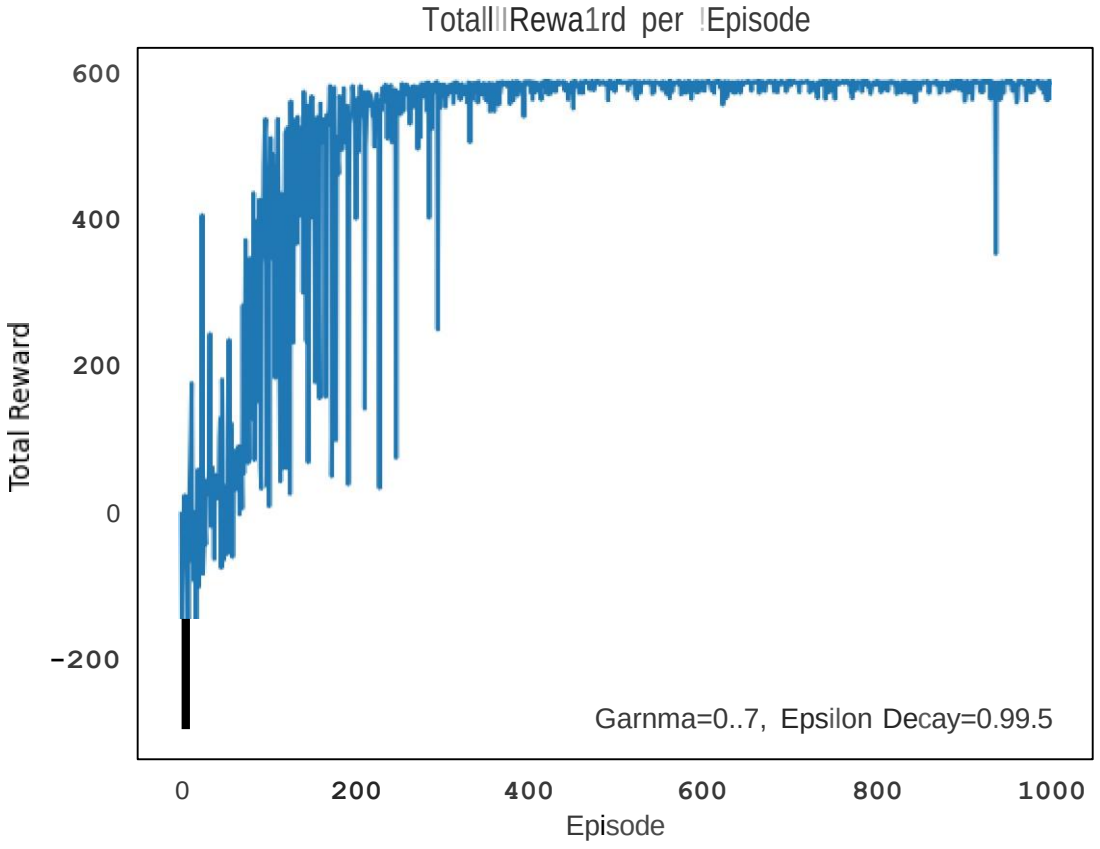

Totall Rewa1rd per Episode
600
400
200
0
-200
-400
0
200
400
600
800
1000
Episode
Garnma=0..4, Epsilon Decay=0.99

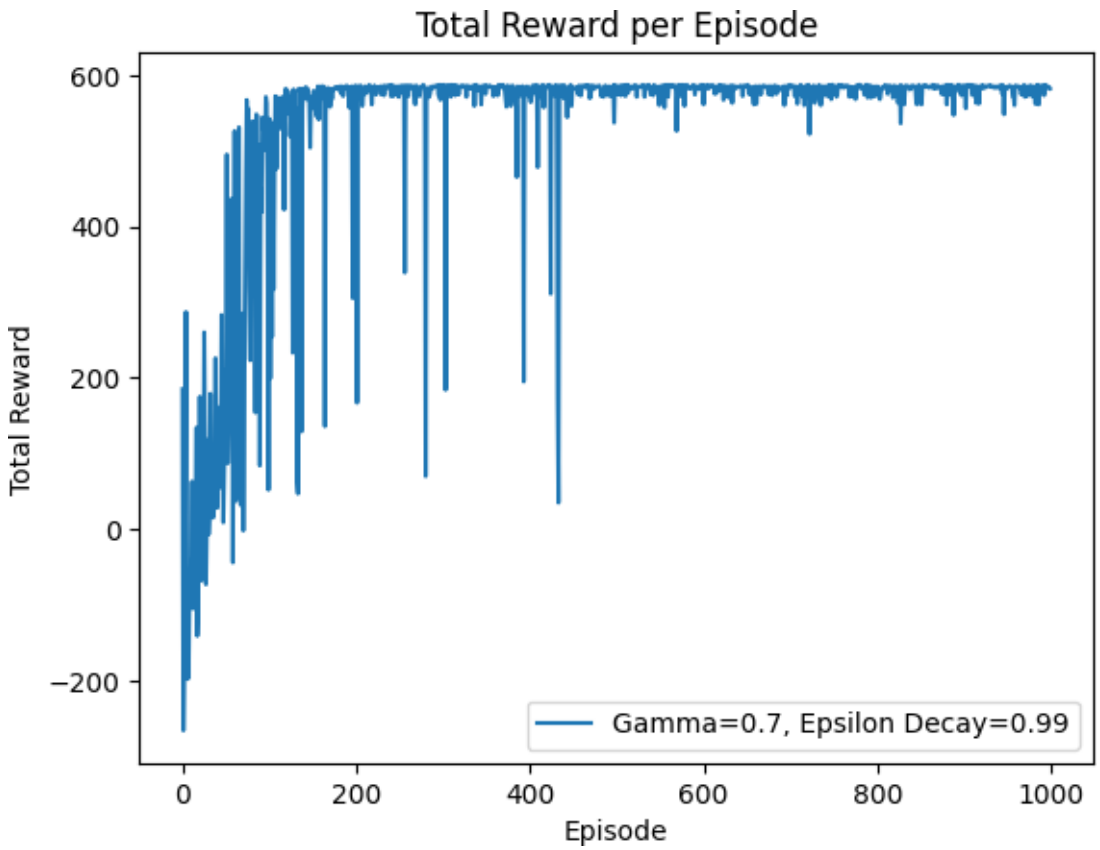

Totall Rewa1rd per Episode
600
500
400
300
200
100
0
-100
-200
0
200
400
600
800
1000
Episode
Garnma=0..4, Epsilon Decay=0.98

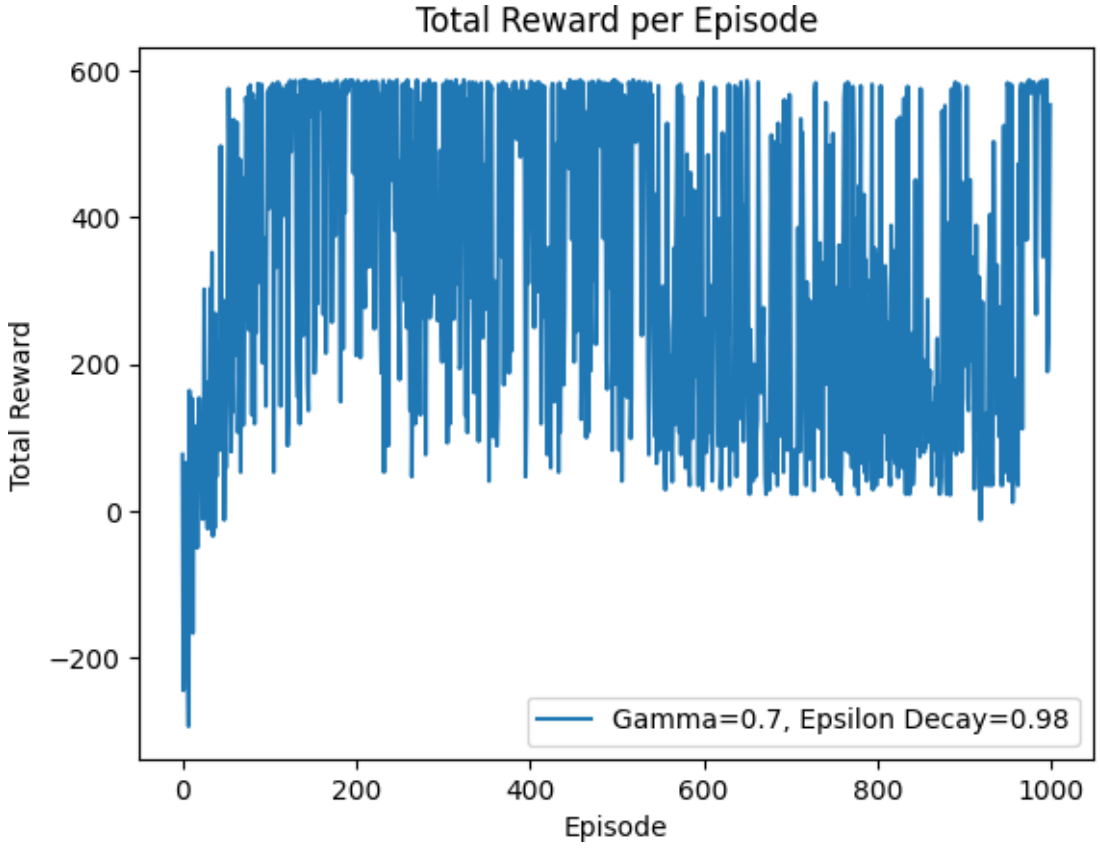

Totall Rewa1rd per Episode
600
400
200
0
-200
Garnma=0.6, Epsilon Decay=0.995
0
200
400
600
800
1000
Episode


Totall Rewa1rd per Episode
600
400
200
0
-200
Garnma=0.6, Epsilon Decay=0.99
0
200
400
600
800
1000
Episode


Totall||Rewa1rd per !Episode
600
500
400
300
200
100
0
-100
-200
Garnma=0..6, Epsilon Decay=0.98
0
200
400
600
800
1000
Episode

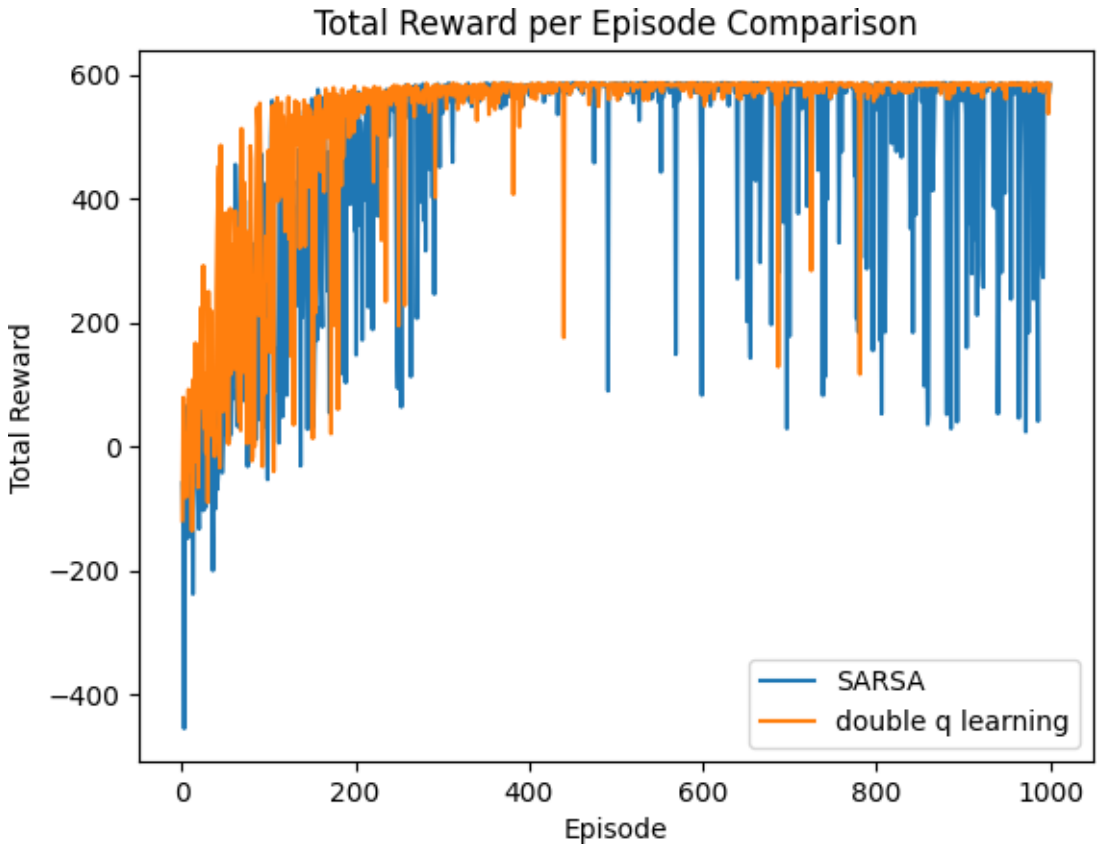

Totall||Rewa1rd per !Episode
600
400
200
0
-200
Total Reward
Garnma=0..7, Epsilon Decay=0.99.5
0
200
400
600
800
1000
Episode





After observing the above graphs for rewards per episode, we can say that Gamma =0.4 and Epsilon decay =0.98 are the efficient hyperparameters since they are converging faster.

Comparing the performance of both algorithms on the same environment



Here, we can observe thar Double Q learning is converging faster. Here we have plotted graph for total rewards against episodes and we got the convergence better for double q learning. We can say Double Q learning has performed well than SARSA.

REFERENCES

- https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf
- https://towardsdatascience.com/reinforcement-learning-with-sarsa-a-good-alternative-to-q-learning-algorithm-bf35b209e1c
- https://tcnguyen.github.io/reinforcement_learning/sarsa_vs_q_learning.html
- https://builtin.com/machine-learning/sarsa#:~:text=Jul%2031%2C%202023-,State%2Daction%2Dreward%2Dstate%2Daction%20(SARSA),to%20solve%20reinforcement%20learning%20challenges.