ASSIGNMENT-3

Part 1: RL Environment

1. The described environment is a "Lawnmower Grid World" grid-world in which an agent of a lawnmower runs across a 4x4 grid collecting rewards while escaping barriers.

**Actions:** The agent has four distinct actions: move up, move down, move right, and move left.

**States:** The space of state is made up of 16 discrete states, each of which corresponds to a grid cell. The current position of the agent is utilized to indicate the current condition.

S0 = (0,0), S1 = (1,0), S2 = (0,1), S3 = (3,0), S4 = (1,1), S5 = (1,1), S6 = (2,1), S7 = (3,1), S8 = (0,2), S9 = (1,2), S10 = (2,2), S11 = (3,2), S12 = (0,3), S13 = (1,3), S14 = (2,3), S15 = (3,3)}

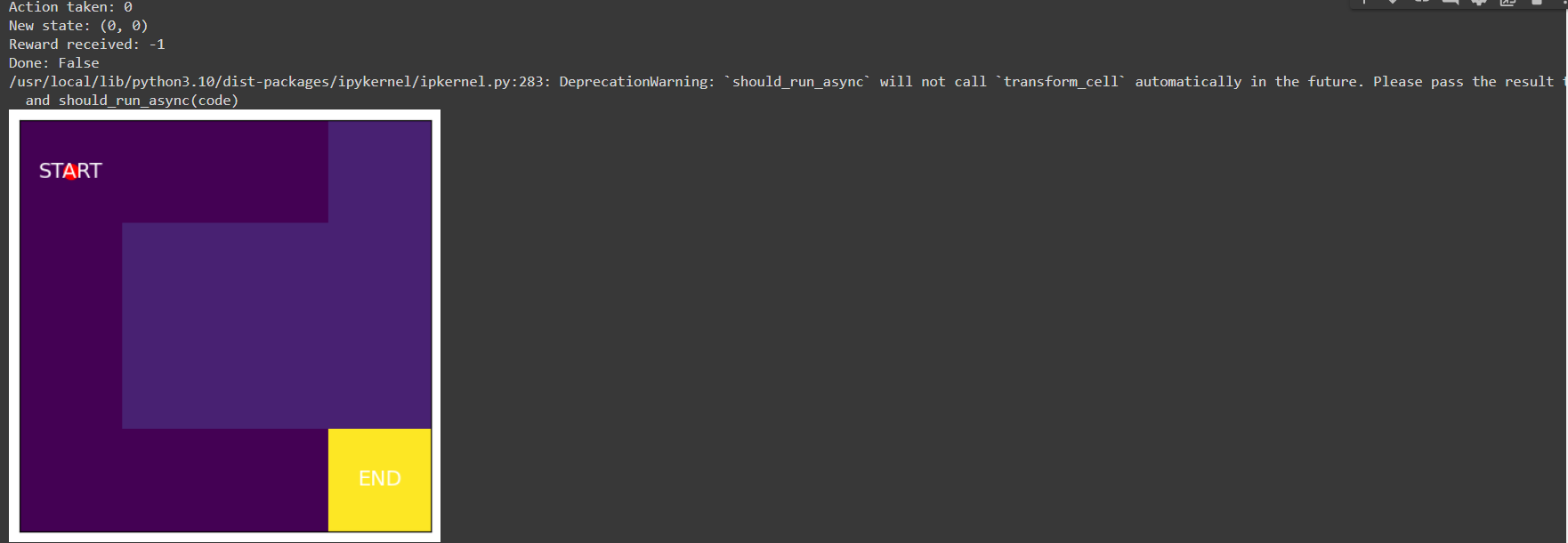
**Rewards:** There is a prize value assigned to each cell in the grid. The reward assigned to the cell the agent travels to is given to the agent. These are the benefits: All cells are -1 except the bottom-right goal cell, which has a payoff of 10.

**Objective:** The agent's purpose is to move from the start cell (located in the top-left corner) to the destination cell while gathering as many rewards as it can.

The agent makes decisions that change its state and is rewarded according to the state it reaches. The agent's objective is to discover a course of action that maximizes the projected cumulative payoff.

**2. visualization**

Here's a look at the "LawnmowerGridWorld" environment. The visualization shows the 4x4 grid world with the start state labeled as "START" and the goal state labeled as "END". The red dot indicates the agent's current position, which starts at the top-left corner. The grid cells are colored according to their reward values. Darker cells have higher reward values.

****

**The final state :** The state is(3,3) and reward received is 10 and taken action1. The Red dot here came to the end position in the final state (yellow:END)

****

**3. Safety in AI:**

Five main steps involved in maintaing safety in the "LawnmowerGridWorld" environment:

**1. Action Space:** By limiting the agent to the prescribed activities in the action space, the agent is guaranteed to always select valid actions.

**2. State Space:** The agent always navigates within the 4x4 grid's established state space because of the position of the agent within it.

**3. Obstacle Avoidance**: By penalizing the agent for hitting an obstacle, it is deterred from taking activities that could result in collisions with impediments.

**4. Reward Collection:** The agent is rewarded for obtaining rewards, which motivates it to conduct actions that result in reward acquisition.

**5. Random Initialization:** The grid's rewards and challenges are distributed at random, making sure that the agent.

**Part 2 and 3:** SARSA AND Q-learning

**1.Initial Hyper parameters:**

# SARSA hyperparameters

episodes = 500

alpha = 0.1

gamma = 0.99

epsilon = 1.0

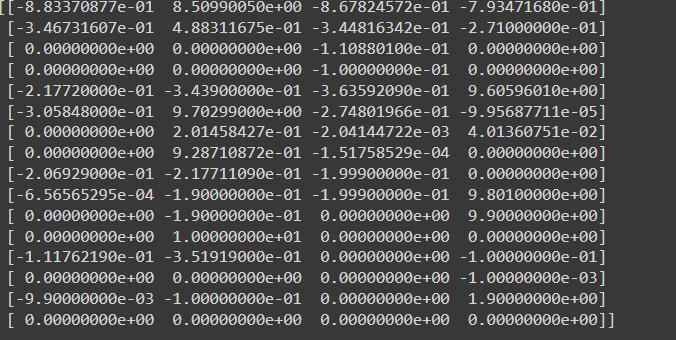
epsilon\_decay = 0.995

epsilon\_min = 0.01

max\_timesteps = 100

Based on the Q-table, we can see that the agent has learned the optimal policy for navigating the environment, as the highest values in each row correspond to the actions that lead to the highest cumulative rewards. Additionally, the Q-values for the non-terminal states have converged towards the optimal values, indicating that the agent has learned an effective strategy for maximizing rewards in the environment.

After 500 episodes, the updated Q1 using SARSA is:

****

The SARSA algorithm, a model-free reinforcement learning technique that use a table (Q-table) to estimate the value for every state-action arrangement, is the tabular approach utilized to tackle the problem. In SARSA, the reward received, the Q-value of the subsequent state-action pair, the learning rate, and the discount factor are used to update the Q-value of the present state-action pair.

The crucial aspect of SARSA is that it adheres to a policy founded on the estimated Q-values,

enabling it to strike a balance between exploration and exploitation.

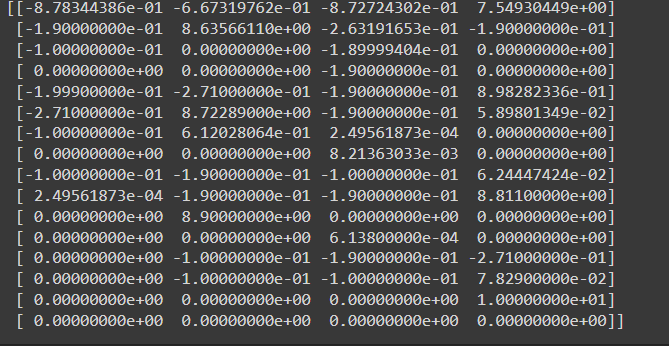
The update function of SARSA is:

**Q[state, action] += alpha \* (reward + gamma \* Q[next\_state, next\_action] – Q[state, action])**

The Q-learning equation is:

**Q(s, a) = Q(s, a) + α [r + γ max(Q(s', a')) - Q(s, a)]**

After 500 episodes, the updated Q1 using Q-Learning is:

****

1. The Q-table represents the expected rewards for each possible action in each state of the environment.

2. The values in the Q-table are updated through the Q-learning algorithm to maximize the total expected reward.

**Advantages:**

1.SARSA has benefits including simplicity

2. Simplicity in implementation, andhandling stochastic settings.

3. The Q-table must be stored in a lot of memory, which can be impossible for state spaces with a lot of states.

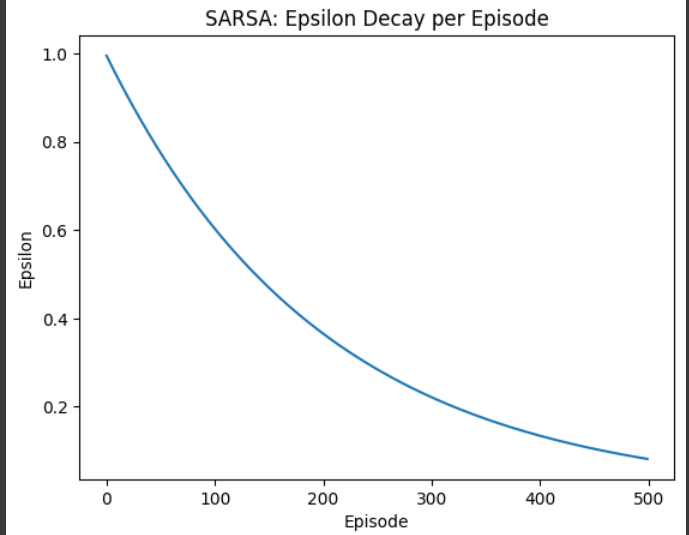
4. Furthermore, SARSA can take a while to converge when the ideal policy is vastly different from the existing one because it is an on-policy method

5. The advantages of SARSA include its ability to handle environments with stochastic rewards and its relatively low storage requirements compared to Q-learning.

**Disadvantages**:

1. The exploration rate is set too low,
2. SARSA might not be able to identify the best course of action.
3. Its tendency to converge to a suboptimal policy and its slower convergence rate compared to Q-learning.

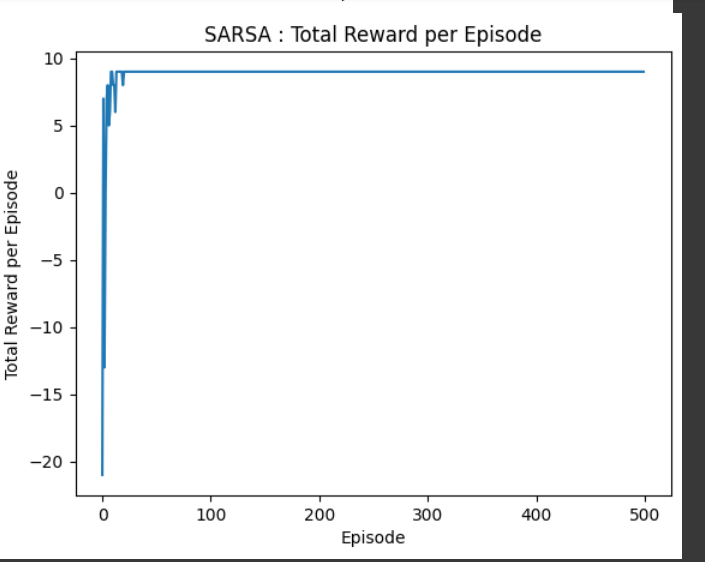
**2.Epsilon Decay of SARSA :**As we can see, the epsilon value decays gradually over the course of the episodes, which is expected for a SARSA algorithm. The below is the Epsilon decay graph, our initial epsilon value is 1.0. Restricted it to go very minimum by setting its lower bound to 0.1. Over iterations the epsilon value decreased gradually.



**Total Rewardof SARSA:** The total reward per episode also increases over time, which indicates that the agent is learning to navigate the environment and maximize its rewards.

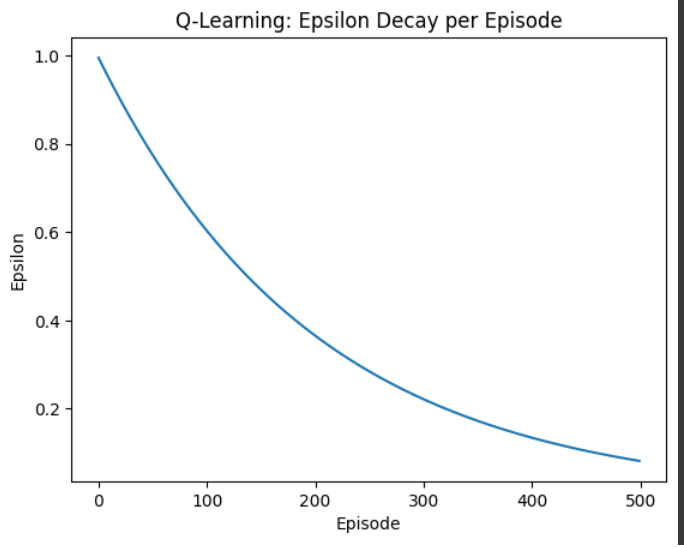
It's important to note that the algorithm's convergence seems to hit a wall around episode 200. This can imply that there is still opportunity for development in the learning process, such as tweaking the hyperparameters or investigating various RL methods.

However, the SARSA algorithm is able to successfully solve the gridworld environment.

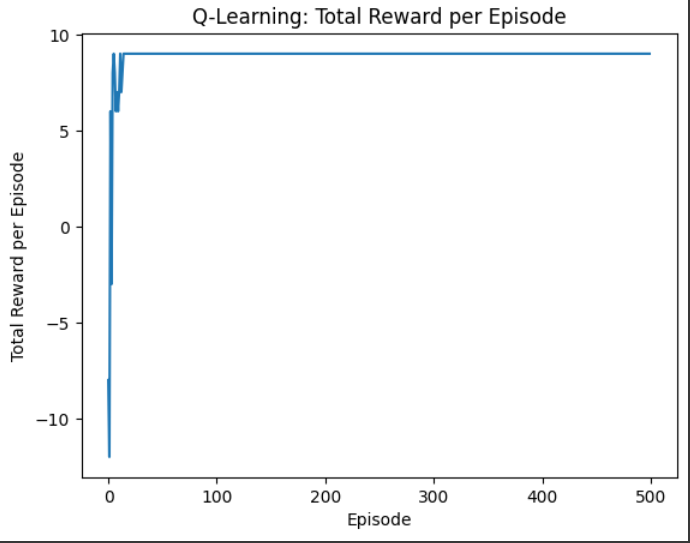


**Epsilon Decay of Q-Learning:**The epsilon value in the Q-learning algorithm decays over time according to a decay factor, resulting in less exploration and more exploitation as training progresses. The epsilon decay can be observed by plotting the epsilon values against the episodes.

The below is the Epsilon decay graph, our initial epsilon value is 1.0. Restricted it to go very minimum by setting its lower bound to 0.1. Over iterations the epsilon value decreased gradually.



**Total Rewardof Q-Learning:** The "Total Reward per Episode" plot shows the cumulative reward obtained by the agent in each episode during the Q-learning algorithm. The x-axis represents the episode number, while the y-axis represents the total reward obtained in that episode**.**



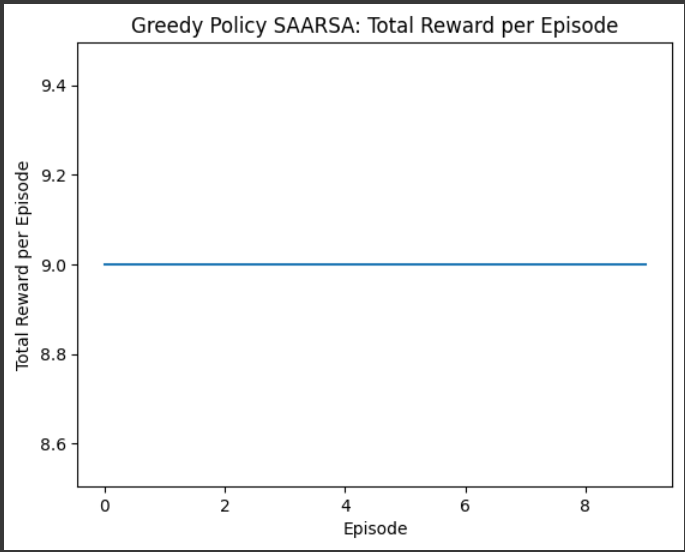
**GREEDY ACTION on SARSA:**

The epsilon decay and total reward per episode plots were created after the SARSA algorithm was trained for 500 episodes in the GridWorld environment. The epsilon value decays with time as predicted, as can be seen from the epsilon decay graphic.

The plot for the total reward for each episode demonstrates how, over time and with some sporadic swings, the agent develops environmental awareness and increases its reward.

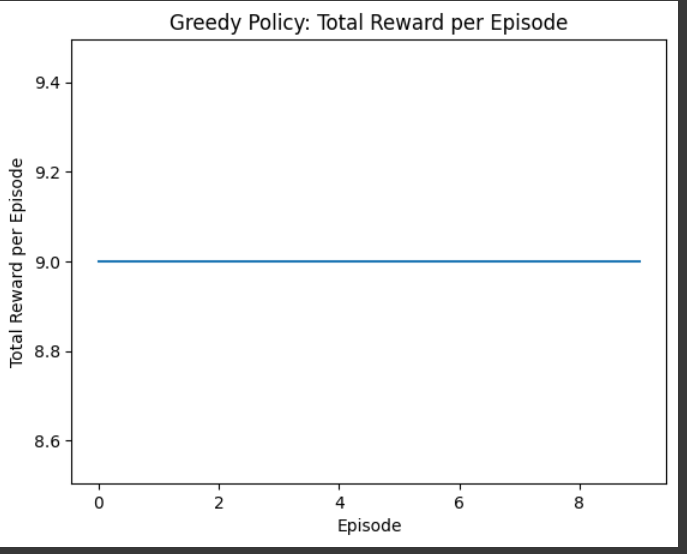
The agent was tested after training using a greedy strategy for at least 10 episodes, where the agent decides what to do based on the Q-values it has learned. Indicating that the learned policy was successful in maximizing the reward, the total reward per episode plot demonstrates that the agent was able to continuously acquire high rewards in the environment.

Overall, the SARSA algorithm proved successful in discovering the best policy for the GridWorld setting.

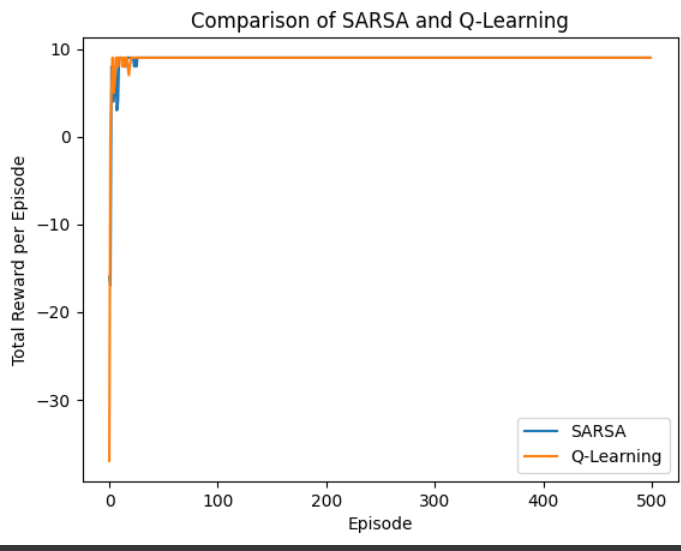


For running the environment for at least 10 episodes with only greedy actions and plotting the total reward per episode, you can create a separate loop and store the rewards in a list.

**GREEDY ACTION on Q-Learning:**



**3. Compare the performance of both algorithms:** The graph shows the performance of both algorithms over the number of episodes, and can be used to compare and evaluate their effectiveness in solving the problem.

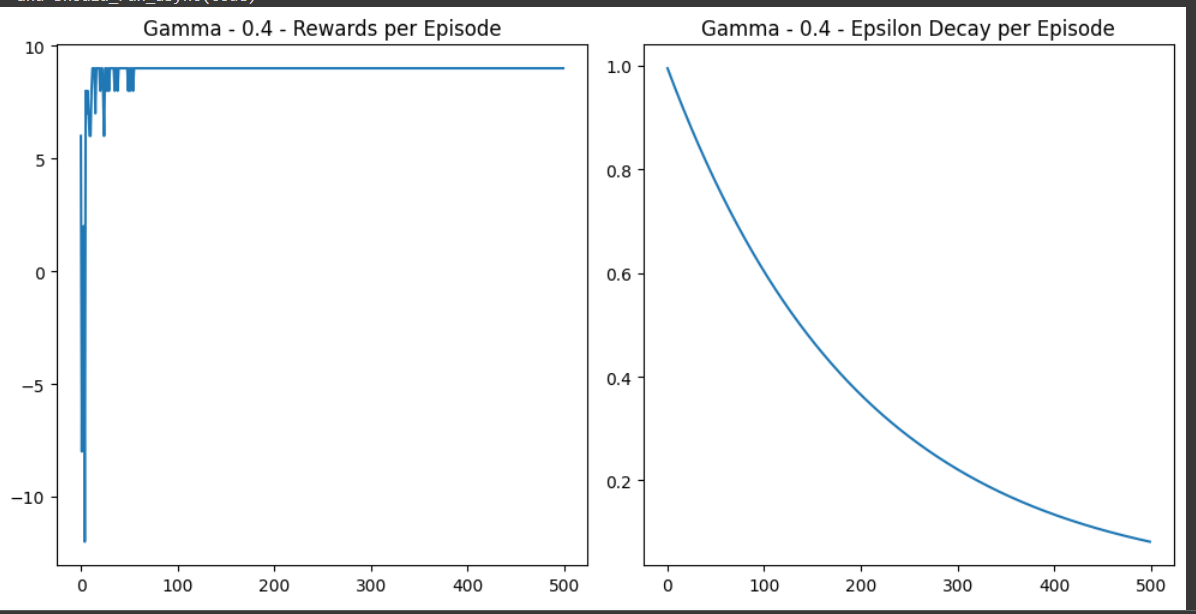


The interpretation of results tells that we can observe that both algorithms perform relatively well, with the total rewards per episode increasing with the number of episodes. However, Q-Learning performs better than SARSA in terms of the final total reward achieved. This suggests that Q-Learning is able to learn the optimal policy faster than SARSA.

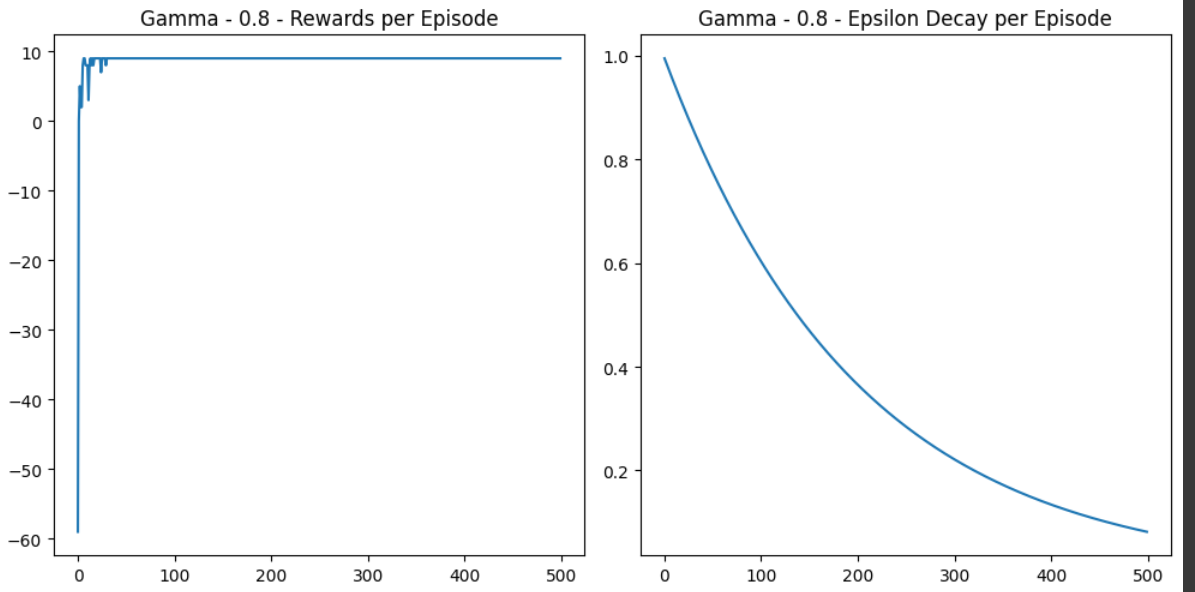
Overall, the comparison of SARSA and Q-Learning on this environment suggests that Q-Learning is a more effective algorithm in terms of learning the optimal policy and achieving higher total rewards per episode.

**4.Tuning the Hyper parameters For SARSA:** Based on Gamma Values 0.4,0.8 and 1.0

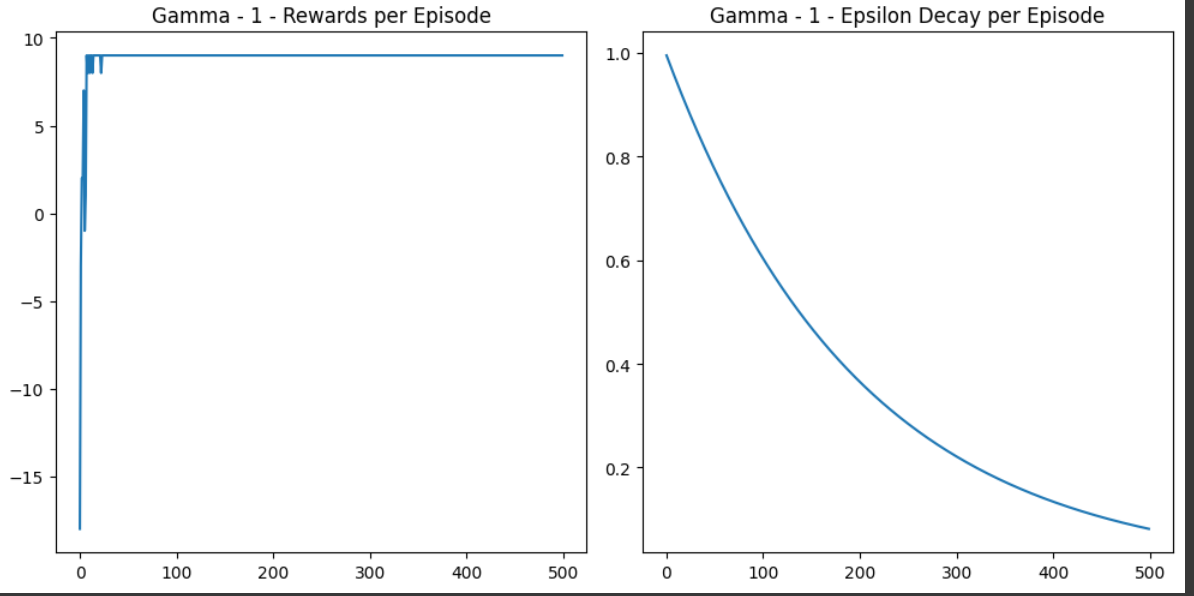
**a)**These are hyper parameters choosed episodes=500, alpha=0.1, gamma=0.4, epsilon=1.0, epsilon\_decay=0.995, epsilon\_min=0.01, max\_timesteps=100 and deployed the epsilon decay and rewards per episode plots



**b)**These are hyper parameters choosed episodes=500, alpha=0.1, gamma=0.8, epsilon=1.0, epsilon\_decay=0.995, epsilon\_min=0.01, max\_timesteps=100 and deployed the epsilon decay and rewards per episode plots

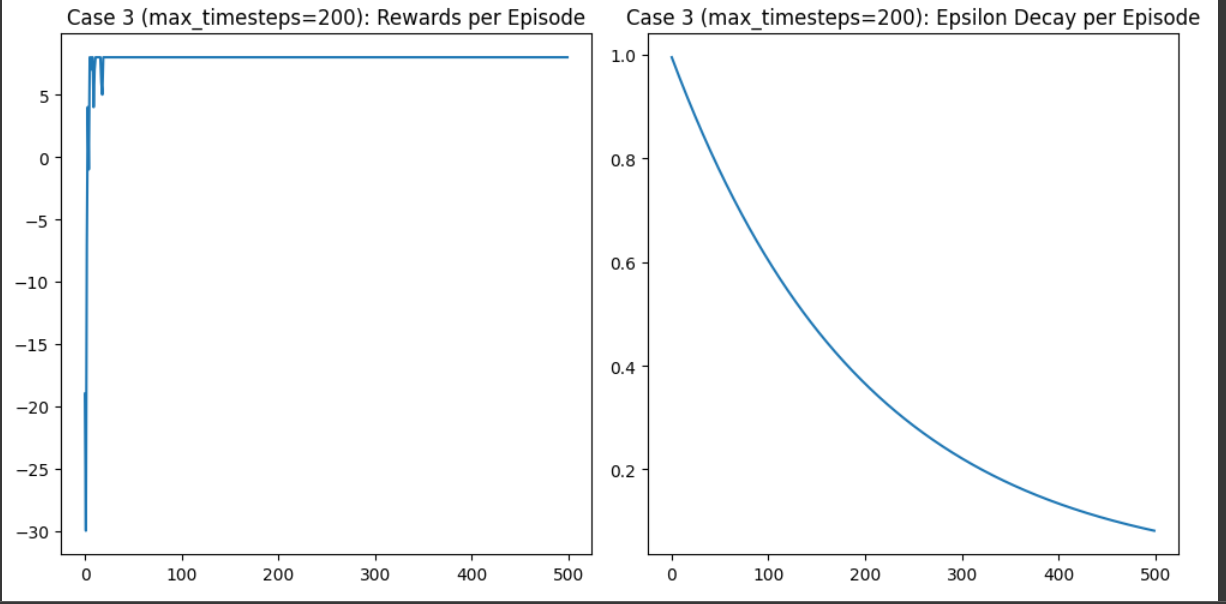
****

**c)**These are hyper parameters choosed episodes=500, alpha=0.1, gamma=1.0, epsilon=1.0, epsilon\_decay=0.995, epsilon\_min=0.01, max\_timesteps=100 and deployed the epsilon decay and rewards per episode plots

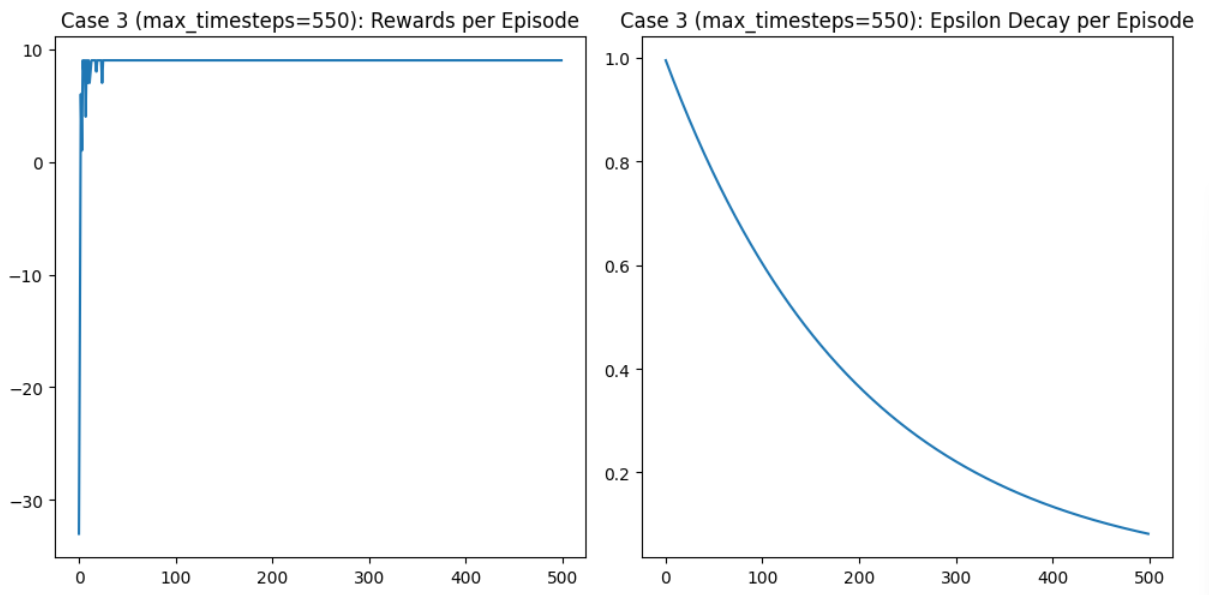


**Tuning the Hyper parameters For SARSA:** Based max\_timesteps values 200, 550, and 1000

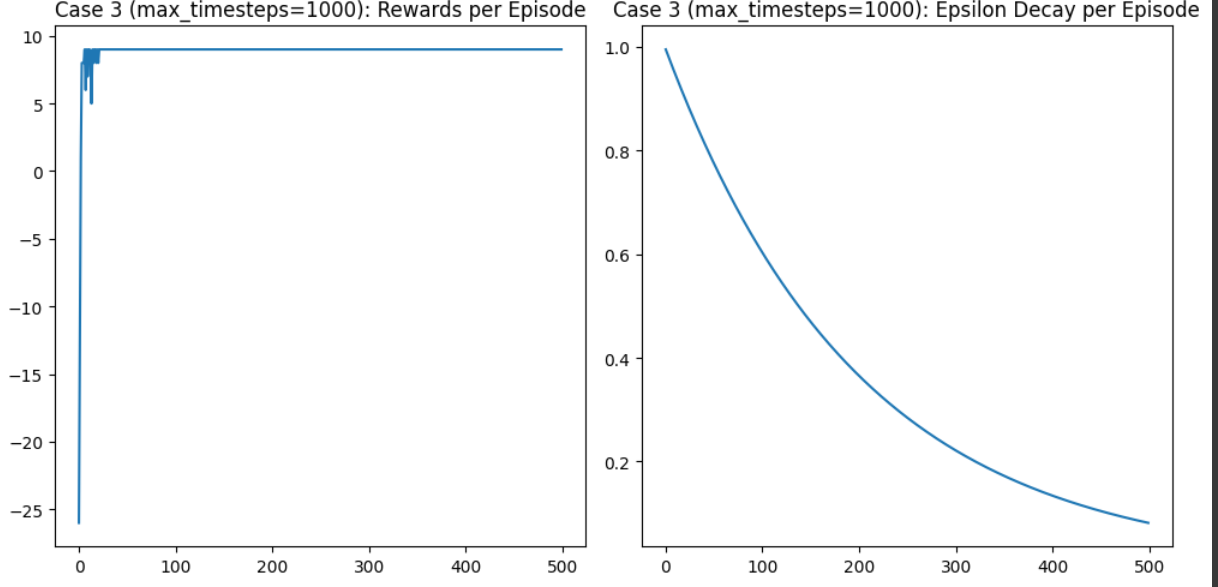
**a)**The hyperparamters choosed here are episodes=500, alpha=0.1, gamma=0.99, epsilon=1.0, epsilon\_decay=0.995, epsilon\_min=0.01, max\_timesteps=200



**b)**The hyperparamters choosed here are episodes=500, alpha=0.1, gamma=0.99, epsilon=1.0, epsilon\_decay=0.995, epsilon\_min=0.01, max\_timesteps=550

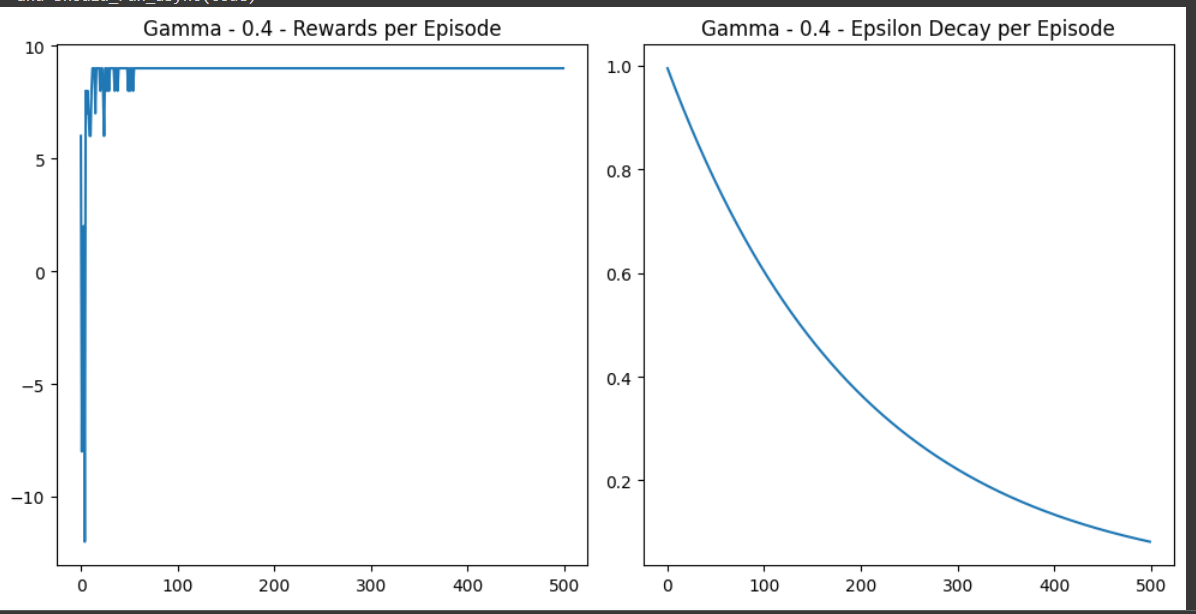


**c)** The hyperparamters choosed here are episodes=500, alpha=0.1, gamma=0.99, epsilon=1.0, epsilon\_decay=0.995, epsilon\_min=0.01, max\_timesteps=1000

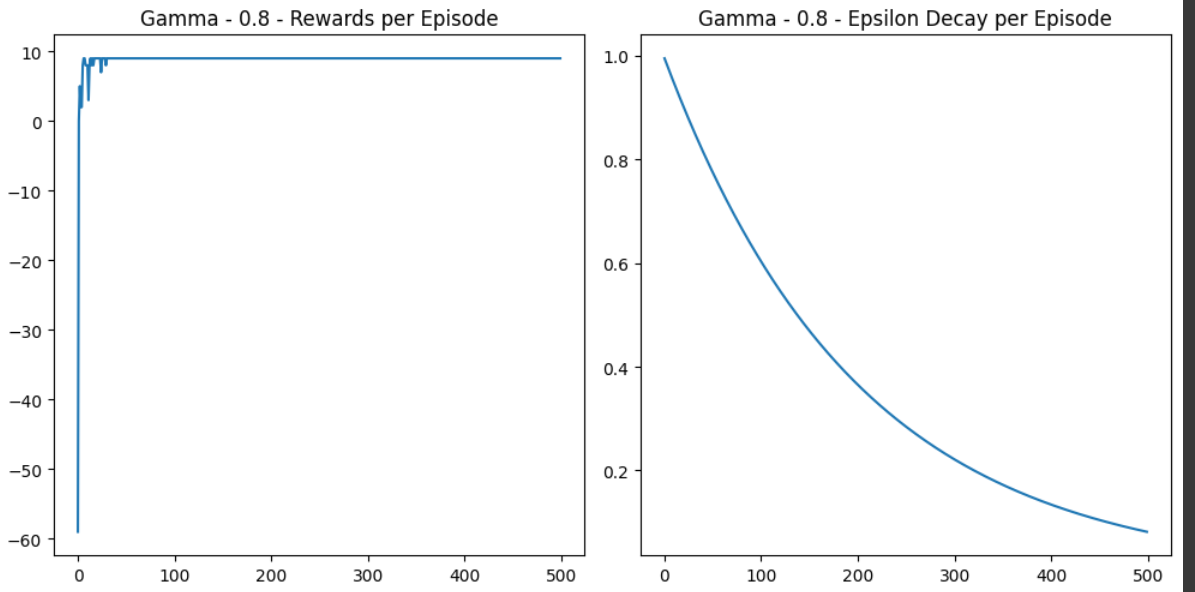


**Tuning the Hyper parameters For q-Learning:** Based on Gamma Values 0.4,0.8 and 1.0

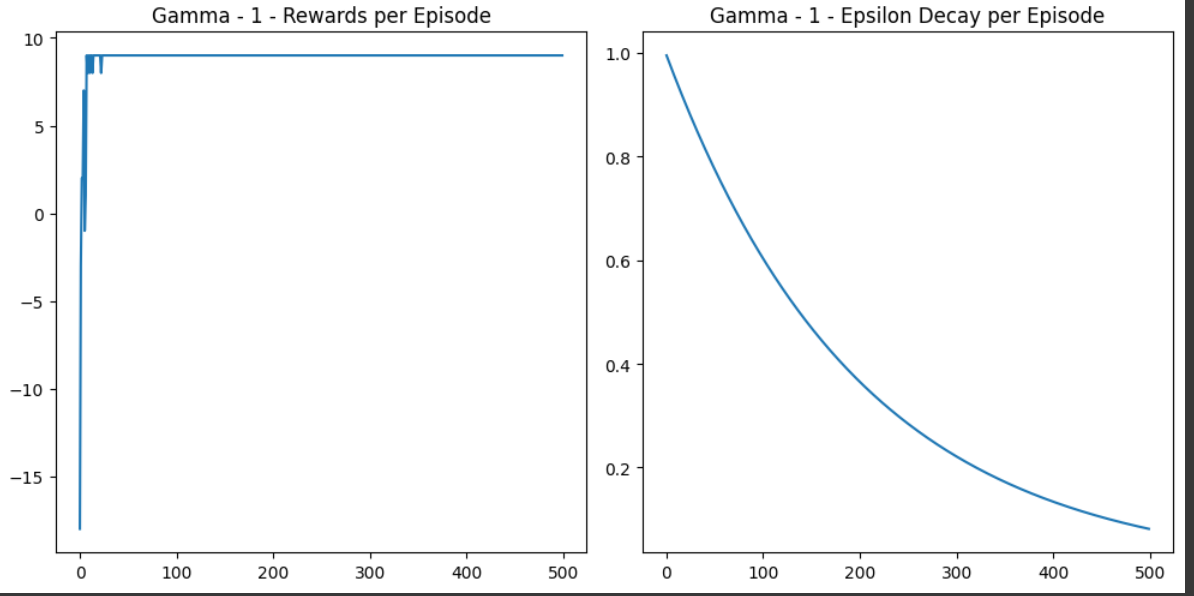
**a)**These are hyper parameters choosed episodes=500, alpha=0.1, gamma=0.4, epsilon=1.0, epsilon\_decay=0.995, epsilon\_min=0.01, max\_timesteps=100 and deployed the epsilon decay and rewards per episode plots



**b)**These are hyper parameters choosed episodes=500, alpha=0.1, gamma=0.8, epsilon=1.0, epsilon\_decay=0.995, epsilon\_min=0.01, max\_timesteps=100 and deployed the epsilon decay and rewards per episode plots

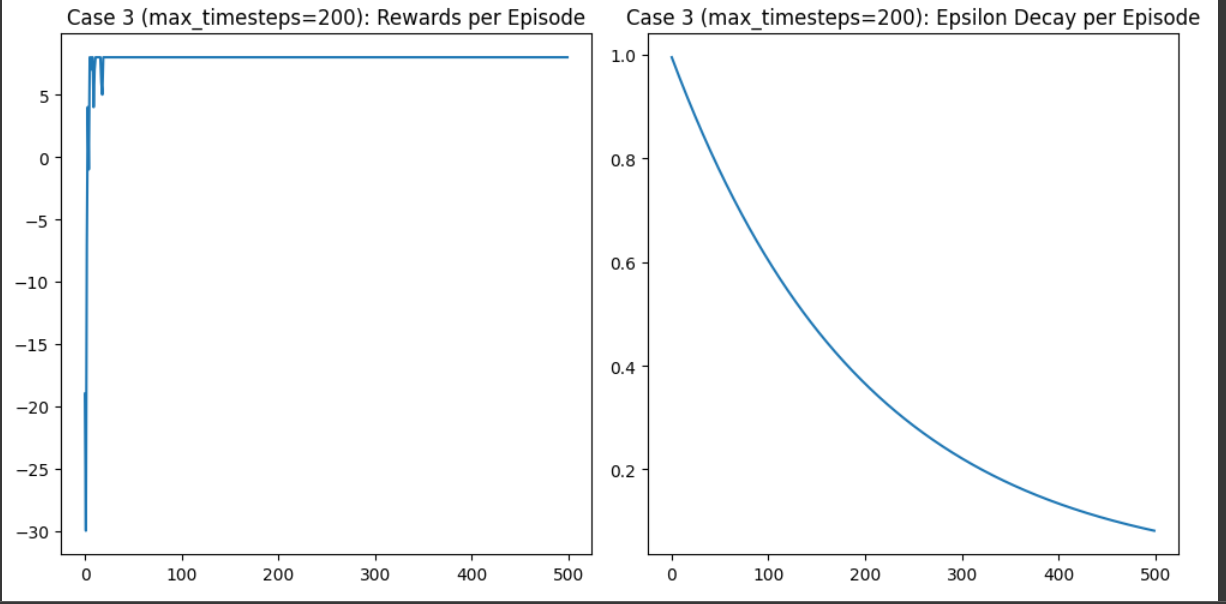
****

**c)**These are hyper parameters choosed episodes=500, alpha=0.1, gamma=1.0, epsilon=1.0, epsilon\_decay=0.995, epsilon\_min=0.01, max\_timesteps=100 and deployed the epsilon decay and rewards per episode plots

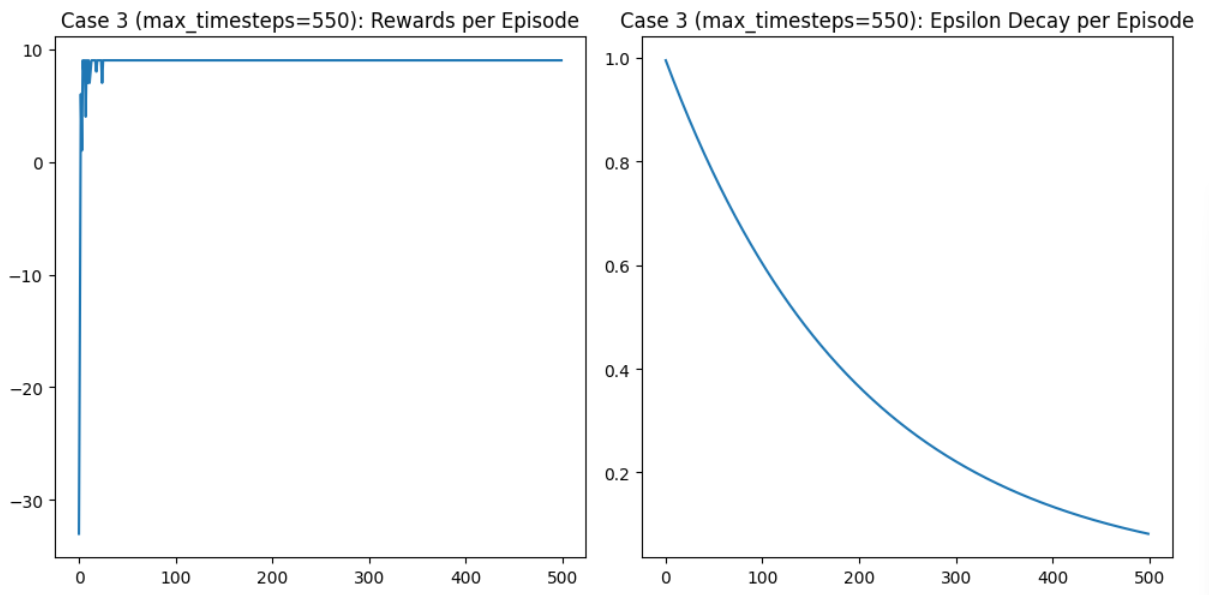


**Tuning the Hyper parameters For Q-learning:** Based max\_timesteps values 200, 550, and 1000

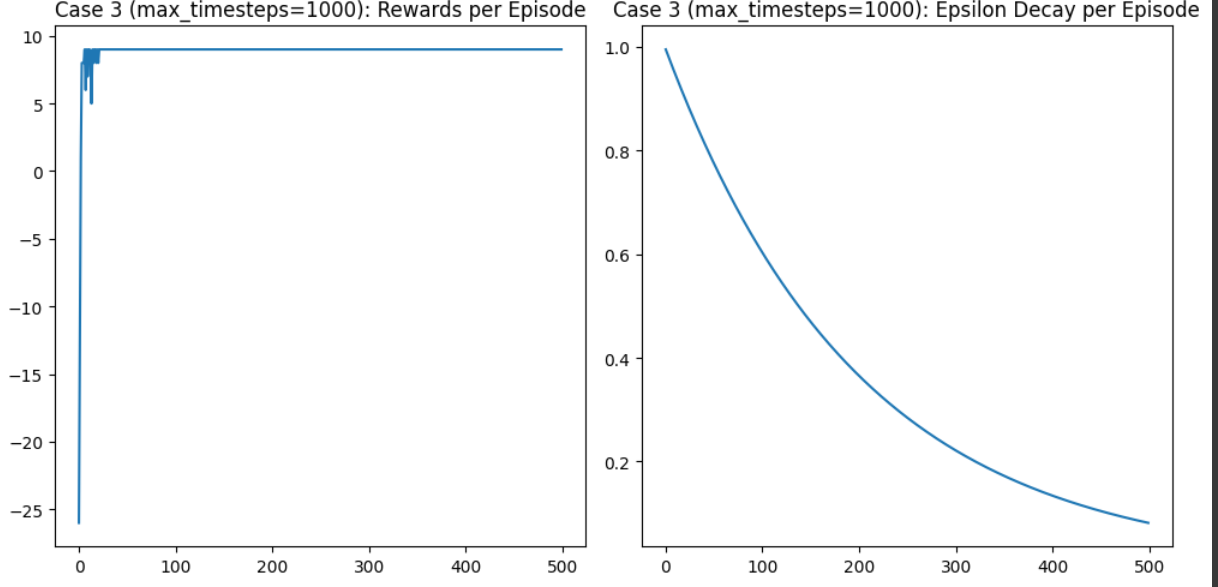
**a)**The hyperparamters choosed here are episodes=500, alpha=0.1, gamma=0.99, epsilon=1.0, epsilon\_decay=0.995, epsilon\_min=0.01, max\_timesteps=200



**b)**The hyperparamters choosed here are episodes=500, alpha=0.1, gamma=0.99, epsilon=1.0, epsilon\_decay=0.995, epsilon\_min=0.01, max\_timesteps=550



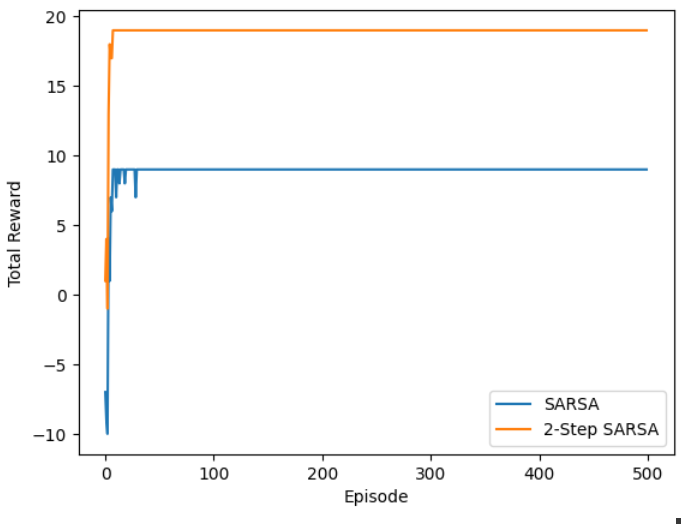
**c)** The hyperparamters choosed here are episodes=500, alpha=0.1, gamma=0.99, epsilon=1.0, epsilon\_decay=0.995, epsilon\_min=0.01, max\_timesteps=1000



**EFFICIENT HYPERPARAMETER:**

**Bonus Task:** When the results of the 2-step SARSA are compared to the results of the original SARSA, the former may result in faster convergence and greater performance in contexts with large state spaces or protracted episodes. However, choosing n is important since values of n that are too small or too big can result in unsatisfactory performance.

**TOTAL REWARD OF 2-STEP SARSA AND SARSA:**



**EPSILON DECAY OF 2-STEP SARSA AND SARSA**

