CSE 574 - ASSIGNMENT 3 - REPORT

Part 1:

1. Describe the environment that you defined. Provide a set of states, actions, rewards, main objective, etc.

An Overview of Our Environment
We’ve designed a grid-based world, a 4x4 grid, for our environment.

States - In our environment, there are 16 discrete states. Each state corresponds to a unique position on this grid. We chose to represent these states as single integers for ease of handling.

Actions - Our agent can perform four actions: moving up, down, left, or right, to navigate through the grid.

Rewards - We’ve designated certain grid positions to offer positive rewards. These positions provide 5 points, except for the goal position, which offers 10 points. There are also specific positions that have negative rewards, marked with a penalty of -5 points. The main target for the agent is the goal position at the bottom-right corner, which gives a higher reward of 10 points upon reaching it.

Main Objective - The primary goal we set for the agent is to navigate through the grid to reach the goal position. While doing so, the agent should maximize its total reward by collecting positives and avoiding negatives.

Episode Dynamics - We’ve set a limit of 10 steps for each episode in our environment.
An episode concludes either when the agent successfully reaches the goal position or when it hits the maximum step limit.

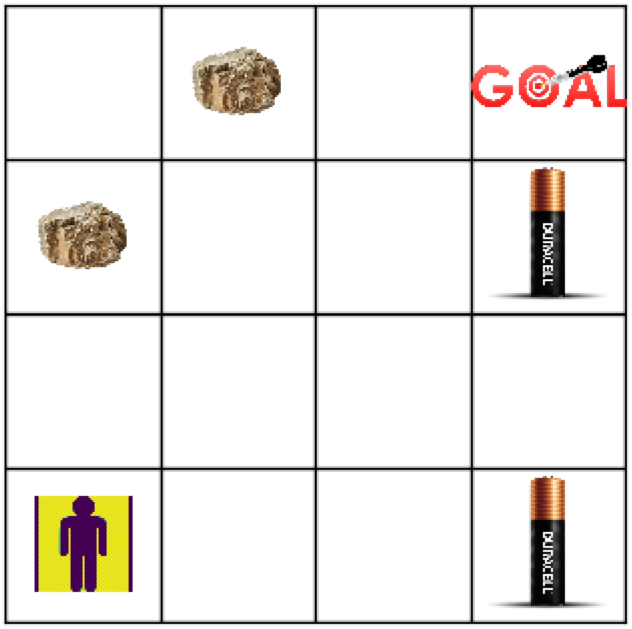
Rendering - We included a rendering function in our environment that visually displays the grid, the agent’s current position, the locations of rewards, and the goal position.

Initialization and Reset - Initially, the agent starts at the top-left corner of the grid, which is position (0,0). We’ve also provided a reset function to bring the agent back to this starting position.

Additional Functionalities - Our environment includes methods to update the agent’s position (‘stp’), to acquire the current state (‘_gs’), and to render the grid (‘rndr’).

2. Provide visualization of your environment.

Visualization of our LawnmowerGridWorld environment:



We can see the 4x4 grid that constitutes our environment. The agent's starting point is at the top-left corner, depicted by an icon resembling a person. The bottom-right corner marks our goal, clearly labeled 'GOAL'. Positive rewards that the agent aims to collect are illustrated as batteries, and the negative rewards to avoid are shown as rocks. The grid layout indicates the potential moves our agent can make—up, down, left, or right—reflecting the action space we've defined.

3. Safety in AI: Write a brief review (~ 5 sentences) explaining how you ensure the safety of your environment. E.g. how do you ensure that the agent chooses only actions that are allowed, that agent is navigating within defined state-space, etc

In our LawnmowerGridWorld game, we make sure our AI agent plays safely by using a few rules. We only let it pick from four moves: up, down, left, or right. This is set up using something called 'spaces.Discrete' from the Gymnasium library. If the agent tries to go off the grid, we have a special clip in the code that puts it back inside, so it always stays in the play area. We keep track of how many moves the agent has made with a step counter to stop it from going around in circles. And we've set up rewards and penalties to nudge the agent to make smarter, safer choices.

Part 2:

1. Briefly explain the tabular methods that were used to solve the problems. Provide their update functions and key features. What are the advantages/disadvantages?

<u>**SARSA Algorithm**</u>

Update Function:

In SARSA, we update our Q-table using this formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma Q(s',a') - Q(s,a)]\ Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma Q(s',a') - Q(s,a)]$$

Here, s and s’ are the states we’re in before and after we take an action, a and a’ are the actions we take in those states, r is the reward we get for taking action a in state s, alpha is our learning rate, which helps us balance new information with what we’ve already learned, gamma is our discount factor, which helps us weigh the importance of future rewards compared to immediate ones.

Key Features:

Learning From Our Policy- In SARSA, we learn from the very actions we take, including when we try something new.

Epsilon-Greedy Strategy- We use this strategy to mix up taking random actions (to learn more about our environment) and taking the best action we know (to make the most of what we’ve learned).

Reducing Exploration Over Time: As we learn more, we reduce our random exploration, indicated by the decrease in epsilon, our exploration rate.

Advantages:

We find SARSA to be more stable and less risky because it learns from the actions we’re actually taking, including our explorations.

We’re less likely to make big mistakes in our learning because we stick closely to our actual policy. So SARSA is reliable.

Disadvantages:

We might learn more slowly with SARSA because it includes exploratory steps that may not always be efficient.

It is not always ideal in risky situations because when exploration can lead to negative outcomes, learning from every single exploratory step might not be the best approach for us.

Double Q-learning

Update Function:

In Double Q-learning, we maintain two separate Q-tables, Q1 and Q2. The way we update each table is as follows:

We choose an action a using a policy that considers both Q1 and Q2. With a 50% chance, we update Q1 using the values from Q2:

$$Q1(s,a) \leftarrow Q1(s,a) + \alpha\, [r + \gamma Q2(s', argmaxaQ1(s',a)) - Q1(s,a)]$$

Otherwise, we update Q2 using the values from Q1**:

$$Q2(s,a) \leftarrow Q2(s,a) + \alpha\, [r + \gamma Q1(s', argmaxaQ2(s',a)) - Q2(s,a)]$$

Here, s is the current state, a is the action we choose, r is the reward, s' is the next state, and alpha and gamma are the learning rate and discount factor, respectively.

Key Features:

Two Tables (Q1 and Q2)- We use two sets of learnings to avoid being too optimistic about the value of actions.

Epsilon-Greedy Strategy- We balance exploring new actions and exploiting what we already know works, gradually focusing more on exploitation as we learn.

Epsilon Decay- We decrease the rate of exploration over time, encouraging more reliance on what we've learned.

Advantages:

It is less overly optimistic because we reduce the tendency to overestimate action values, which is a common issue in simpler Q-learning methods.

We tend to learn more consistently, especially in environments with unpredictable outcomes.

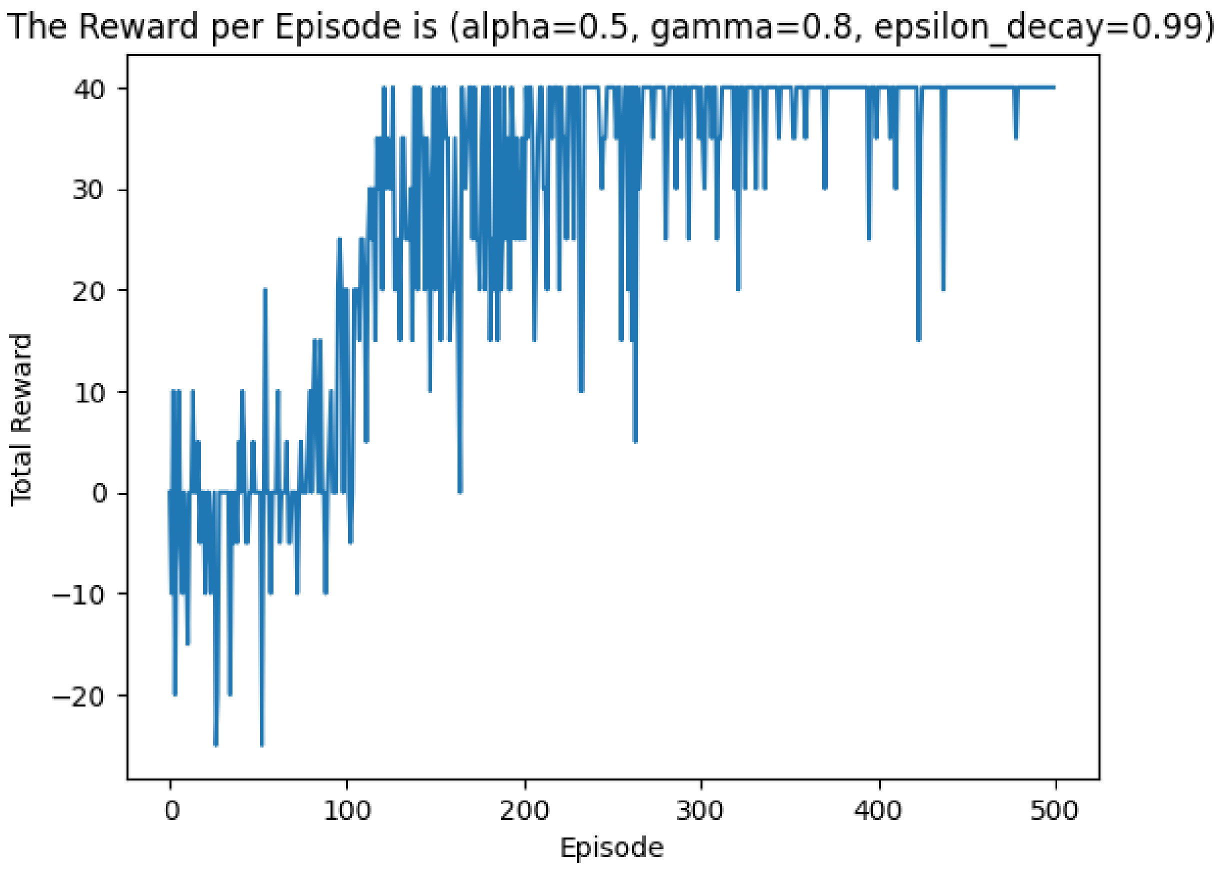
Disadvantages:

We require more computational resources and time due to the two Q-tables.

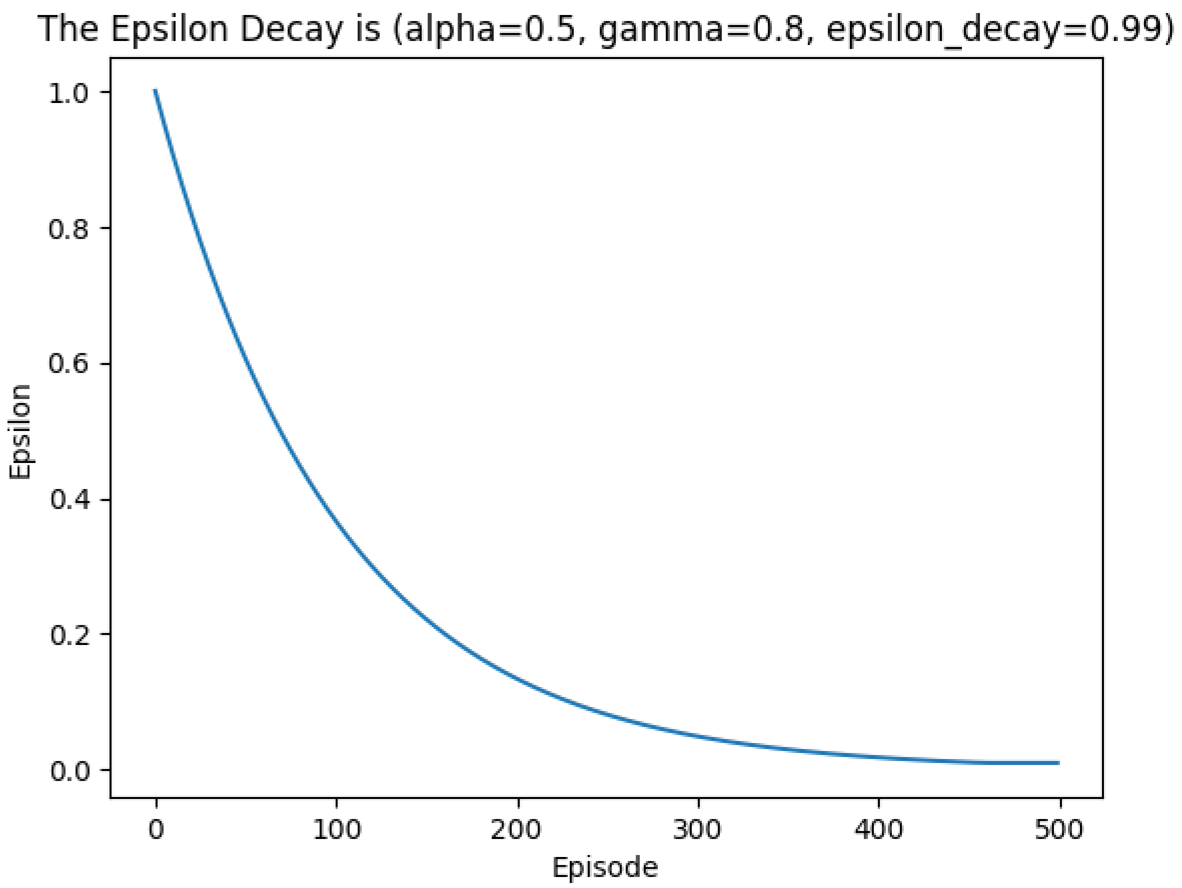
It might take us longer to determine the best actions, so slower learning.

2. Show and discuss the results after:

• Applying SARSA to solve the environment defined in Part 1. Include Qtable. Plots should include epsilon decay and total reward per episode.



As can be seen from the above plot, the Total Rewards increased with an increase in the number of episodes indicating that the Agent is learning and improving its performance over time.



The above plot shows that Epsilon decreased exponentially as the training progressed. At the end of the training, the epsilon value smoothly decreased to a very small value close to 0. This suggests that the agent explored actions more in the initial stages of training and slowly shifted to exploiting its learnings more in the later stages, which eventually led to higher rewards.

Initial Q-Table:

```
{0: array([0, 0, 0, 0]), 1: array([0, 0, 0, 0]), 2: array([0, 0, 0, 0]), 3: array([0, 0, 0, 0]), 4: array([0, 0, 0, 0]),
5: array([0, 0, 0, 0]), 6: array([0, 0, 0, 0]), 7: array([0, 0, 0, 0]), 8: array([0, 0, 0, 0]), 9: array([0, 0, 0, 0]),
10: array([0, 0, 0, 0]), 11: array([0, 0, 0, 0]), 12: array([0, 0, 0, 0]), 13: array([0, 0, 0, 0]), 14: array([0, 0,
0, 0]), 15: array([0, 0, 0, 0])})
```

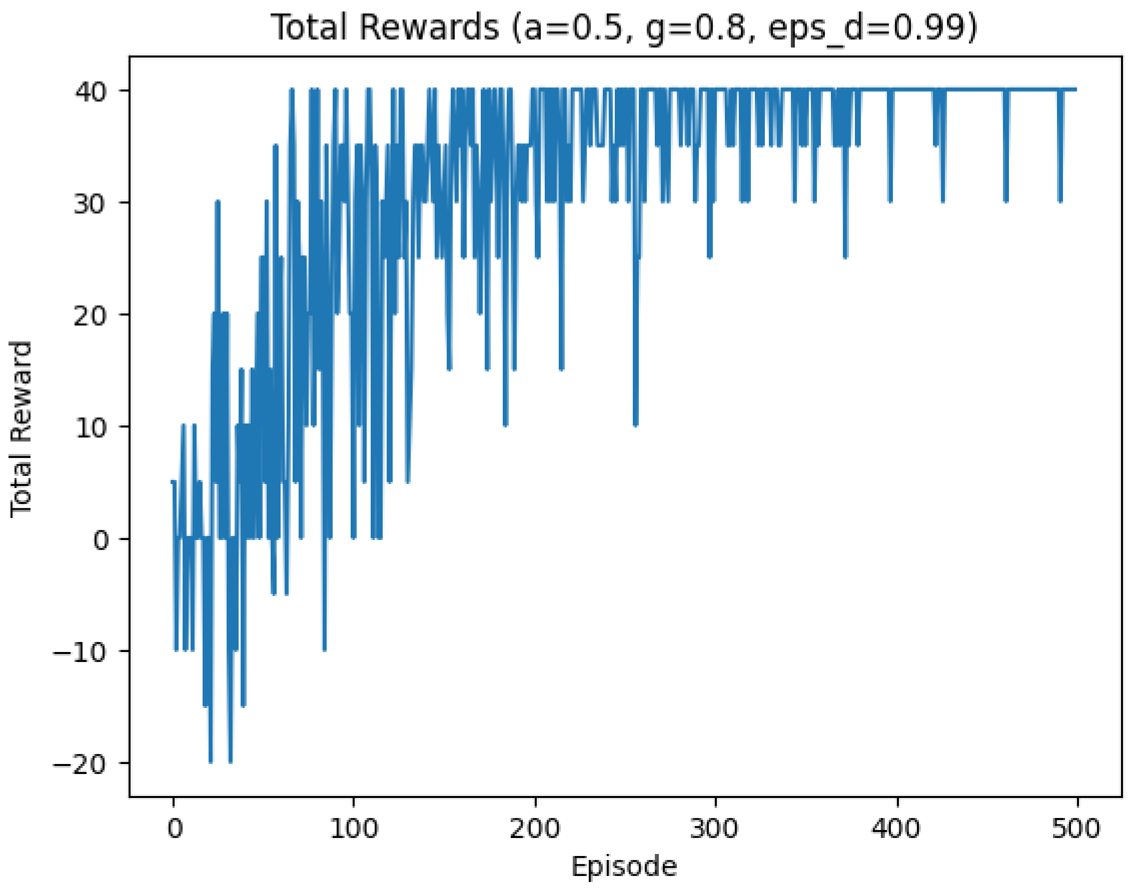
Before training, the initial Q-Table contained only zero values.

Trained Q-Table (alpha=0.5, gamma=0.8, epsilon_decay=0.99):

```
{0: array([2, 3, 7, 3]), 1: array([ 3,  4, 11,  2]), 2: array([ 3,  6, 16,  5]), 3: array([ 7, 16, 13,  7]), 4: array([-
2,  4,  0,  0]), 5: array([-2,  7,  0,  1]), 6: array([7, 0, 2, 1]), 7: array([11,  8,  6,  3]), 8: array([-1, -1,  2, -6]),
9: array([-2,  0,  5, -4]), 10: array([ 6,  0, 11,  2]), 11: array([9, 5, 7, 3]), 12: array([ 0, -7, -6,  0]), 13:
array([-4, -2,  5,  0]), 14: array([ 0,  0,  9, -3]), 15: array([0, 0, 0, 0])})
```

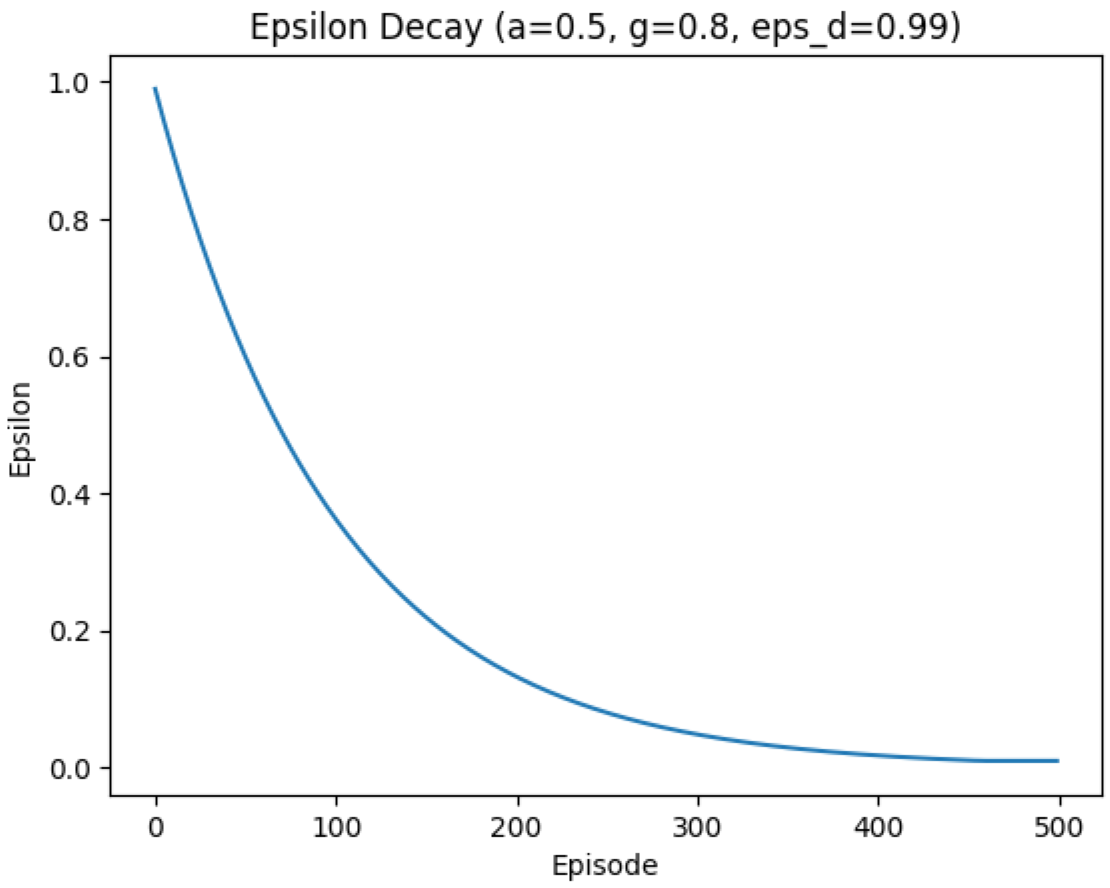
After training, we see that the Q-Table got updated with values based on the rewards it received at each of the 16 states for each of the 4 actions (up,down,left,right).

• 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.



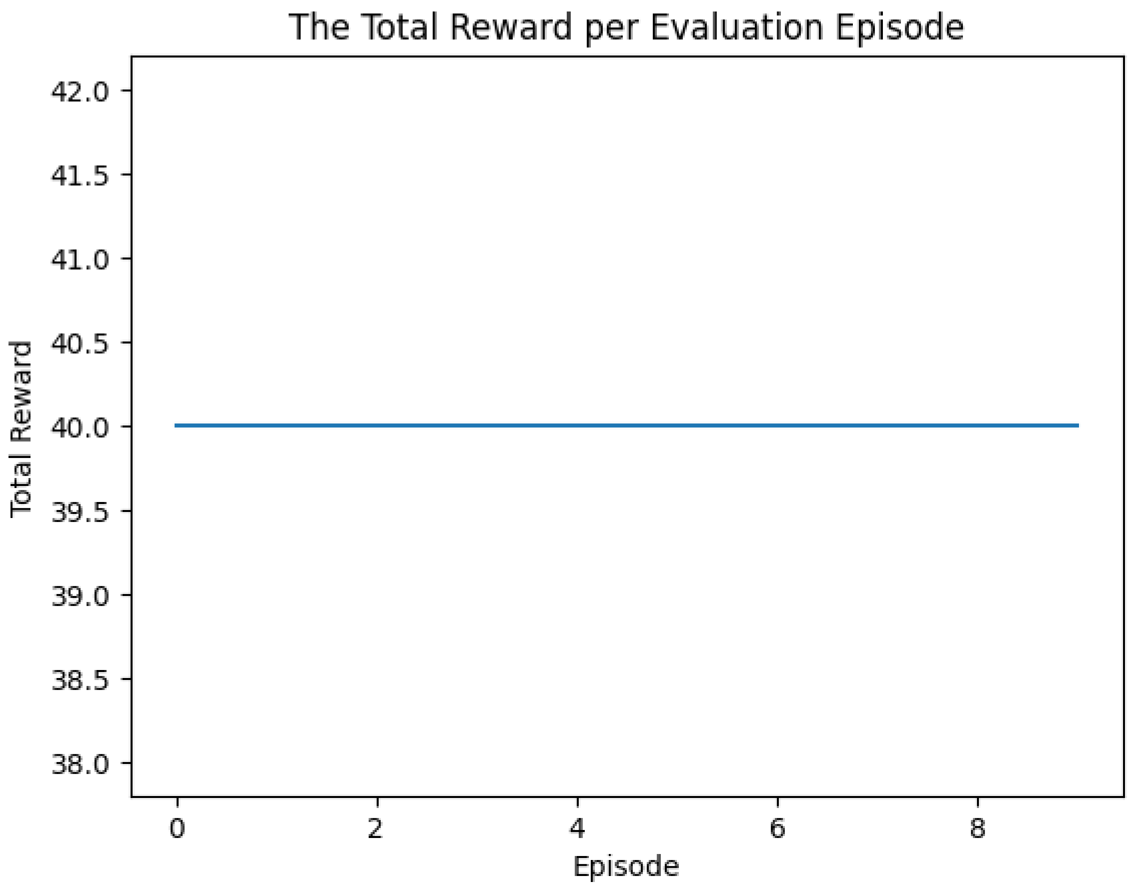
As can be seen from the above plot, using Double Q-Learning as well, the Total Rewards increased with an increase in the number of episodes indicating that the Agent is learning and improving its performance over time.

When this plot is compared with the plot obtained upon using SARSA, we can see that the Double Q-Learning resulted in a faster convergence as max Total Reward was achieved in less than 100 episodes when compared to about 140 episodes in SARSA.

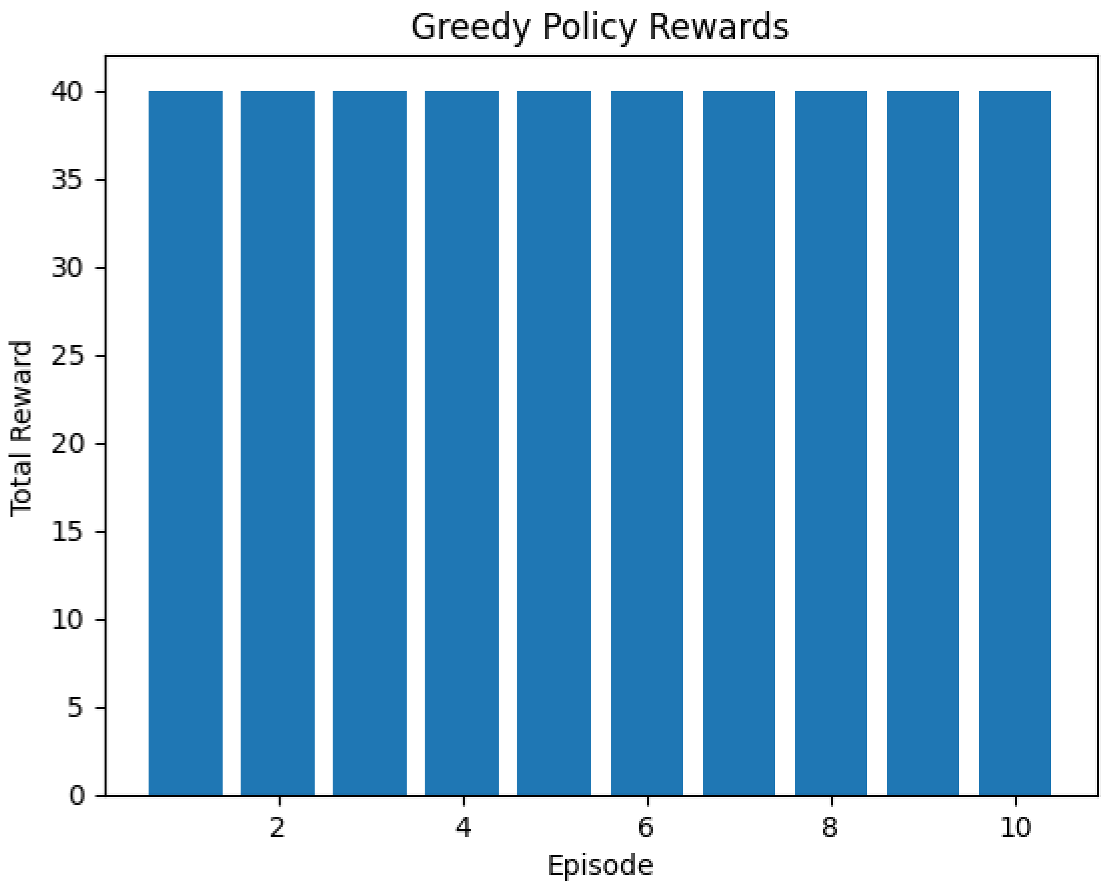


The above plot shows that epsilon decreased exponentially as the training progressed. At the end of the training, the epsilon value smoothly decreased to a very small value close to 0, suggesting that the agent explored actions more in the initial stages of training and slowly shifted to exploiting its learnings more in the later stages, which eventually led to higher rewards.

• 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.



When the environment was run for 10 episodes such that the agent chooses only greedy actions from the learned policy, it can be observed from the above plot that the total reward obtained at each epoch is 40.



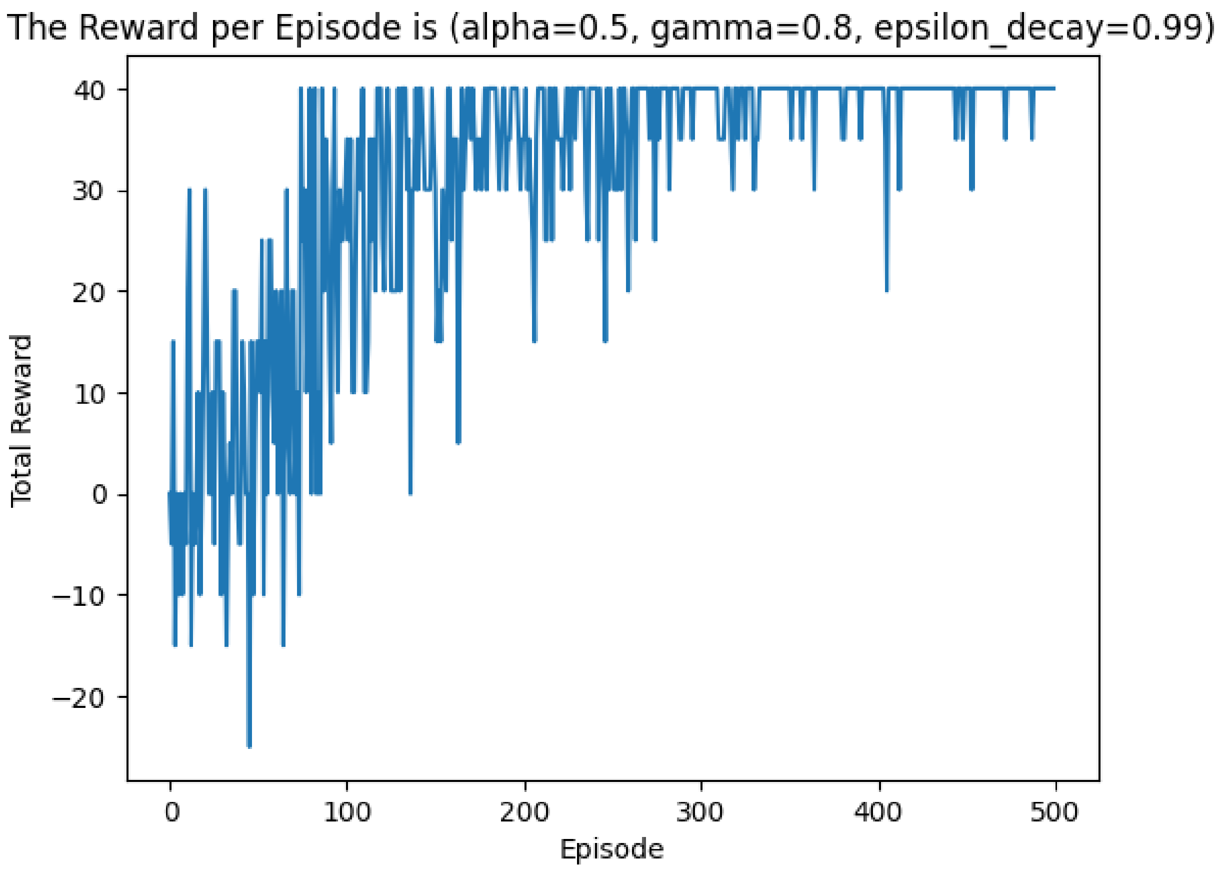
The total reward at each episode is 40 when the agent chooses only greedy actions from the learned policy.

3. 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.

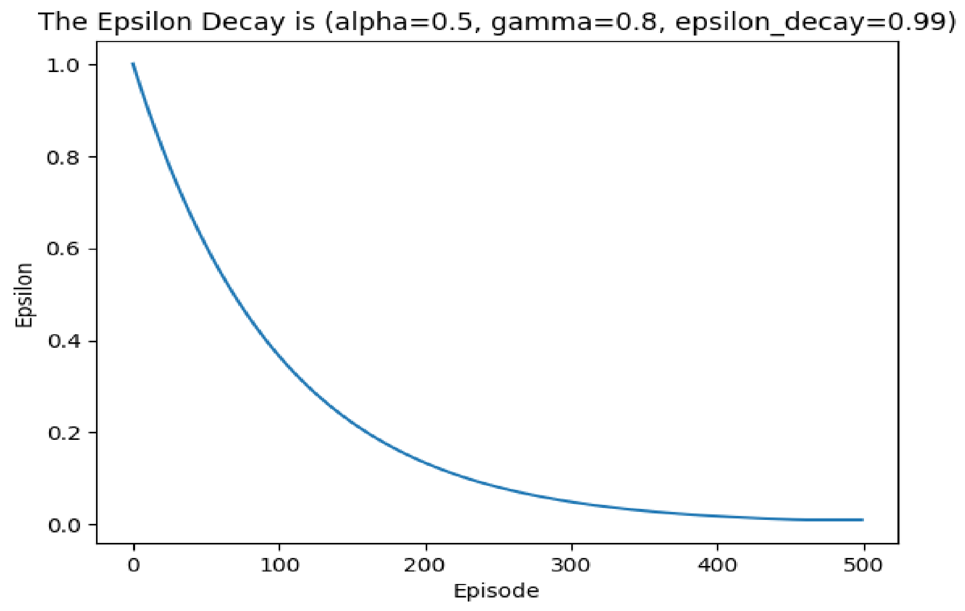
Different hyperparameters chosen -
a = 0.5, g = 0.8 , eps = 0.99
a = 0.5, g = 0.8 , eps = 0.995
a = 0.5, g = 0.8 , eps = 0.999

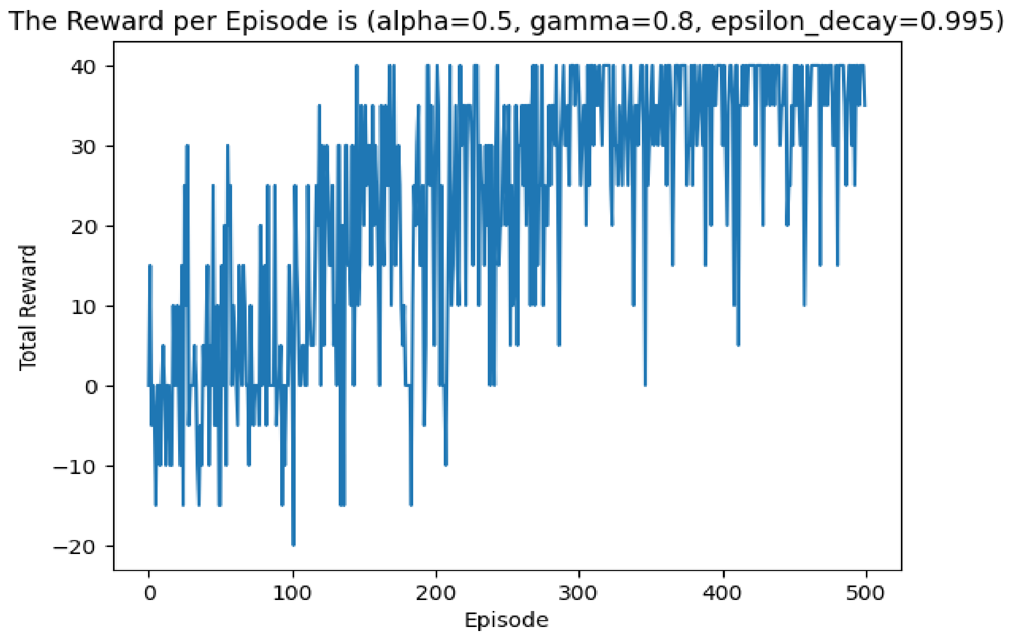
a = 0.5, g = 0.9 , eps = 0.99
a = 0.5, g = 0.9 , eps = 0.995
a = 0.5, g = 0.9 , eps = 0.999

SARSA

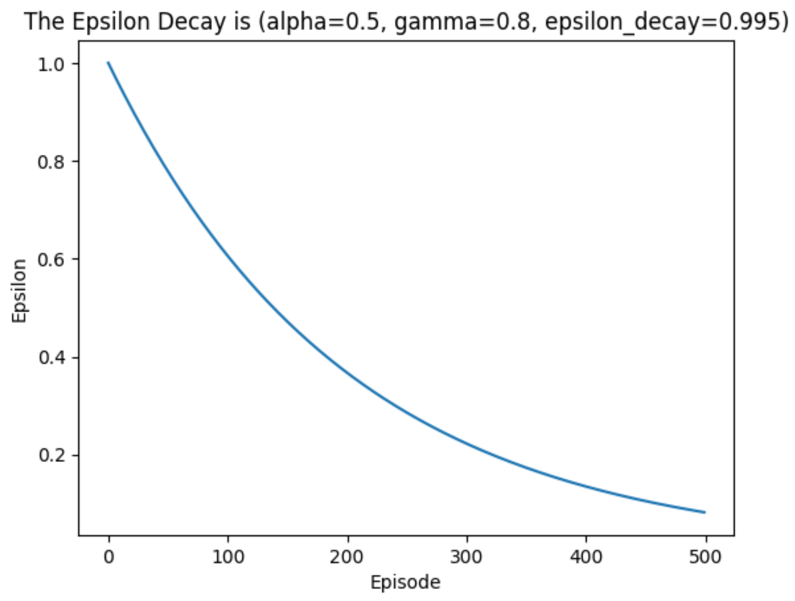


When alpha=0.5, gamma=0.8 and epsilon_decay=0.99, we can see a steep linear increment in the total reward over time. Max Total reward (40) was achieved around the 100th episode.

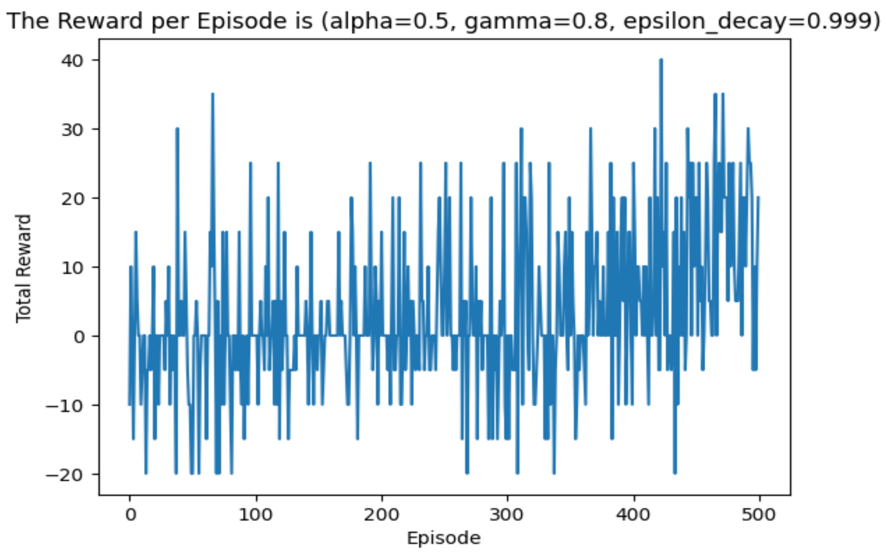




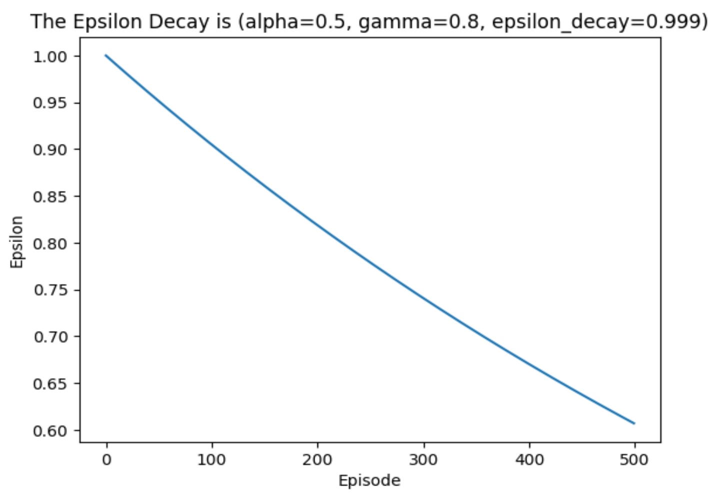
When alpha=0.5, gamma=0.8 and epsilon_decay=0.95 were used, the increment in the total reward was still linear over time, but the increment was not as steep when compared to the environment having epsilon_decay= 0.99.



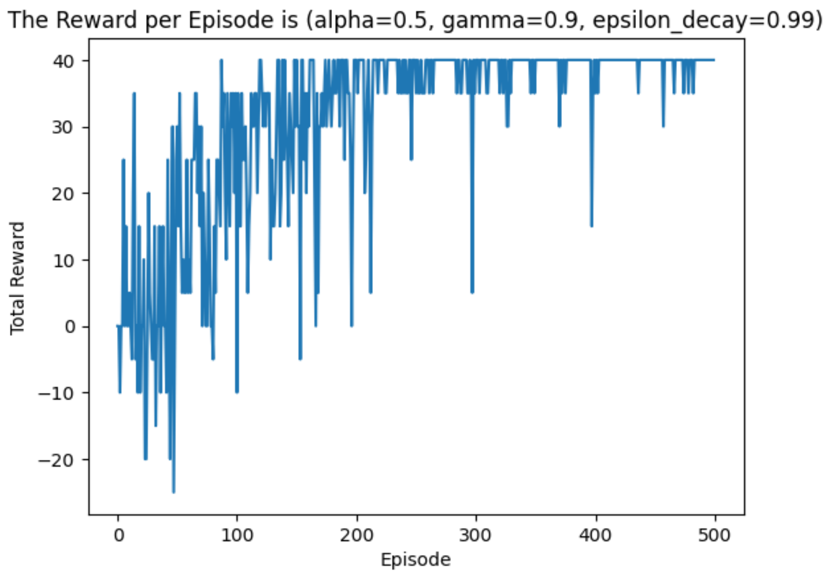
Looking at the above plot, the epsilon has consistently decreased over time (to nearly 0) indicating that the agent explored actions more in the initial stages of training and slowly shifted to exploiting its learnings more in the later stages, which eventually led to higher rewards. Using this combination of hyperparameters, we do not see any abrupt shift in the agent from exploration to exploitation.



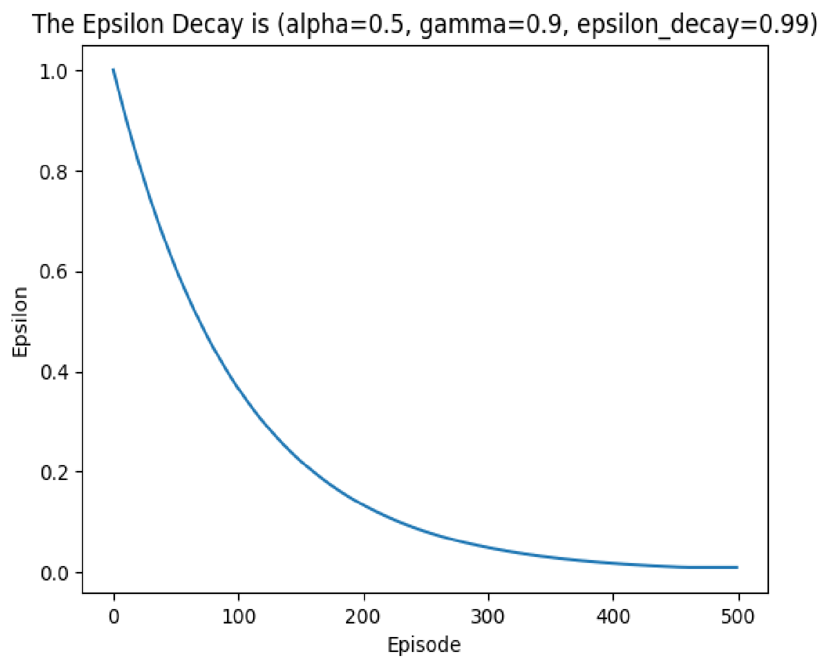
This combination of hyperparameters did not result in a linear increment in the Total Reward over the episodes. It shows a lower performance when compared to the previous combinations of hyperparameters.



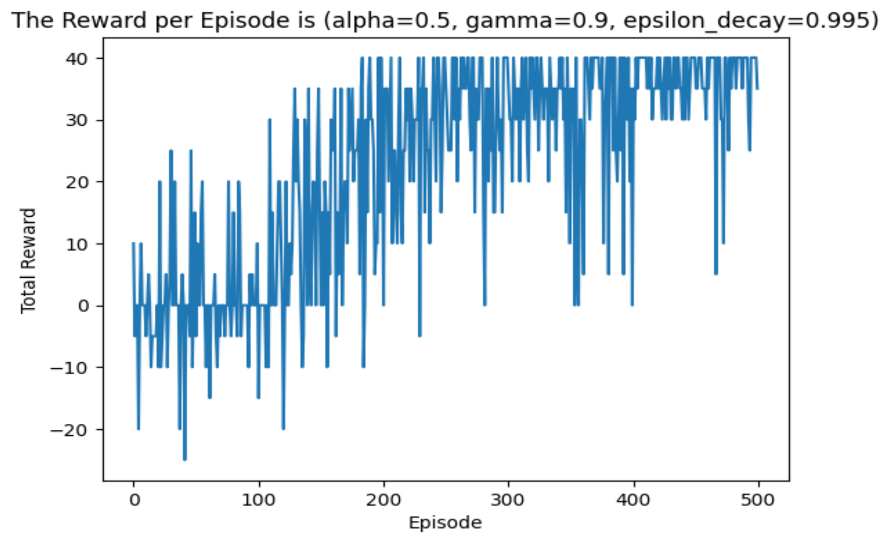
The environment hasn't converged well as the epsilon value decreased only up to 0.6 in 500 episodes, which indicates that the agent did not learn well and had explored actions for a long duration.

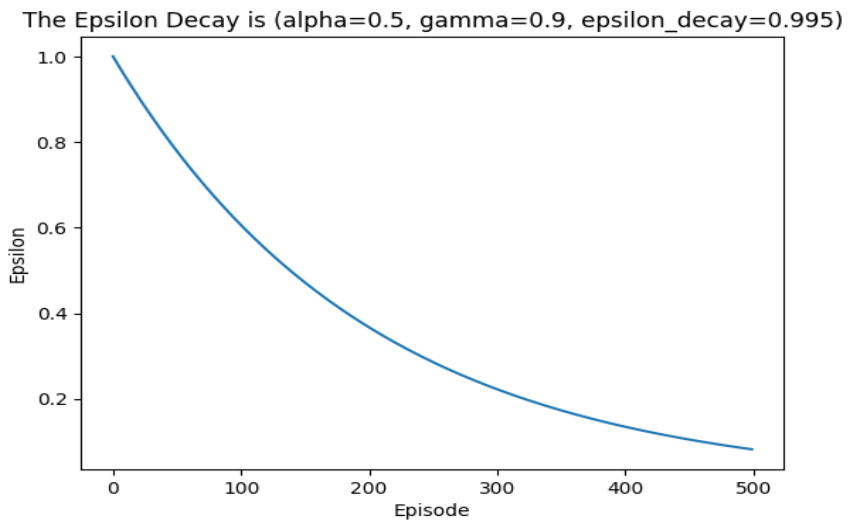


When alpha=0.5, gamma=0.8 and epsilon_decay=0.95 were used, the total reward increased linearly over time.



The above plot shows that epsilon decreased exponentially as the training progressed. At the end of the training, the epsilon value smoothly decreased to a very small value close to 0, suggesting that the agent explored actions more in the initial stages of training and slowly shifted to exploiting its learnings more in the later stages, which eventually led to higher rewards.





The above plot shows that epsilon decreased exponentially as the training progressed, suggesting that the agent explored actions more in the initial stages of training and slowly shifted to exploiting its learnings more in the later stages, which eventually led to higher rewards.

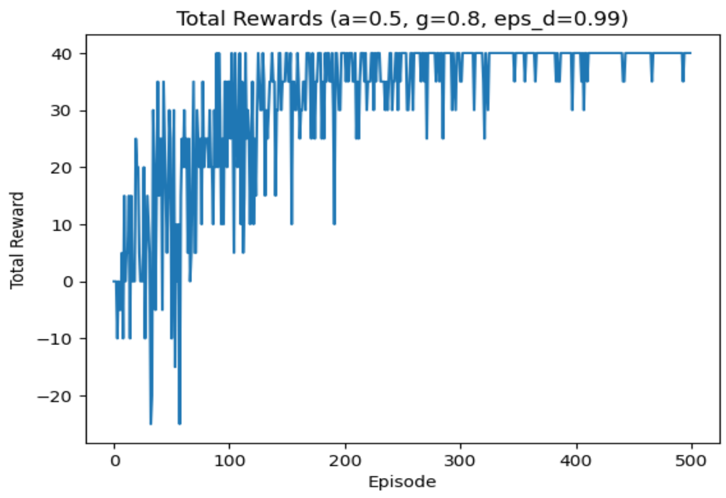
Make your suggestion on the most efficient hyperparameters values for your problem setup.

Based on the above plots, alpha=0.5, gamma=0.8 and epsilon_decay=0.99 gave the best results as the total rewards increased linearly over time and achieved max Total Reward at the earliest when compared to the other set of hyperparameters.

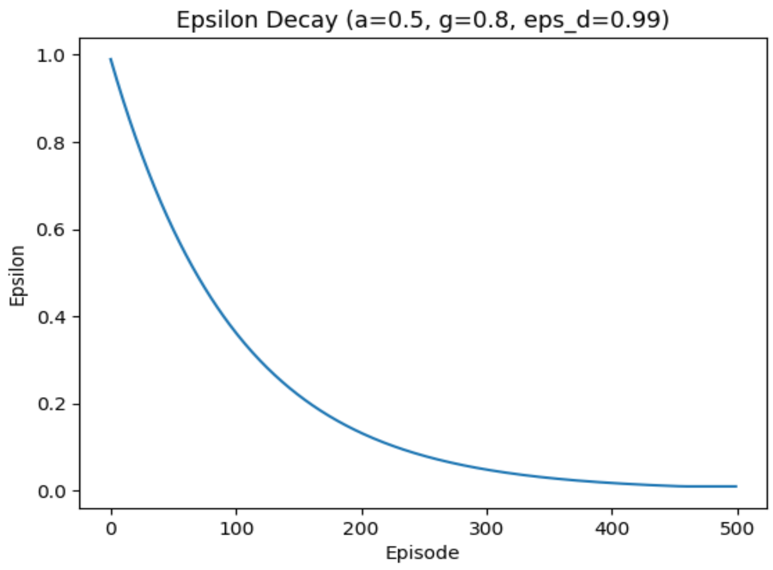
Double Q Learning

Different hyperparameters chosen -
a = 0.5, g = 0.8 , eps = 0.99
a = 0.5, g = 0.8 , eps = 0.995
a = 0.5, g = 0.8 , eps = 0.999

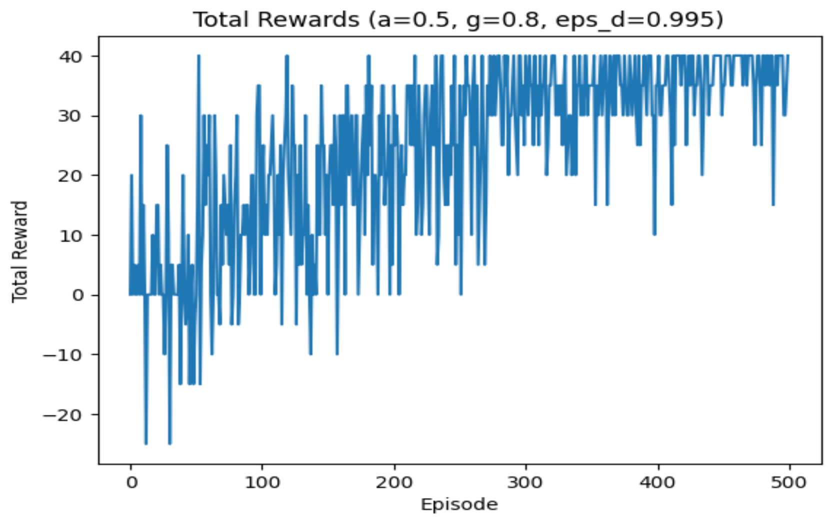
a = 0.5, g = 0.9 , eps = 0.99
a = 0.5, g = 0.9 , eps = 0.995
a = 0.5, g = 0.9 , eps = 0.999



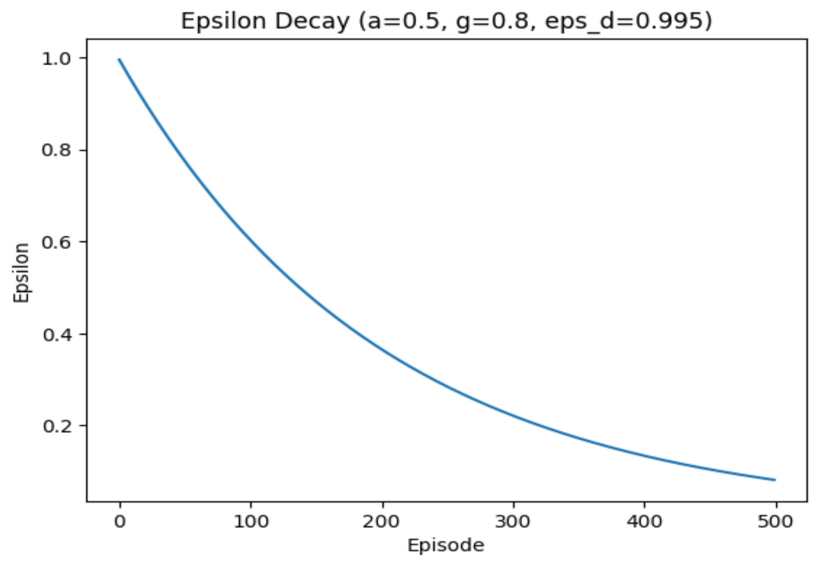
When alpha=0.5, gamma=0.8 and epsilon_decay=0.99, we can see a steep linear increment in the total reward over time. Max Total reward (40) was achieved around the 100th episode.



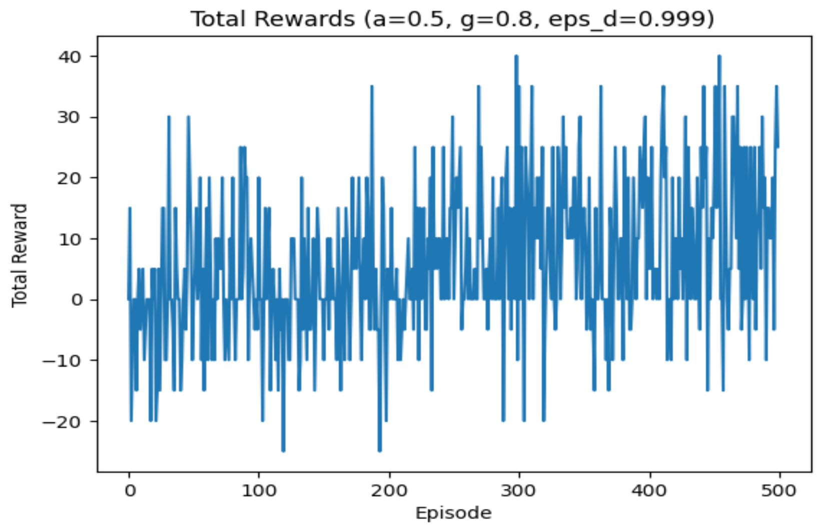
The above plot shows that epsilon decreased exponentially as the training progressed. At the end of the training, the epsilon value smoothly decreased to a very small value close to 0, suggesting that the agent explored actions more in the initial stages of training and slowly shifted to exploiting its learnings more in the later stages, which eventually led to higher rewards.



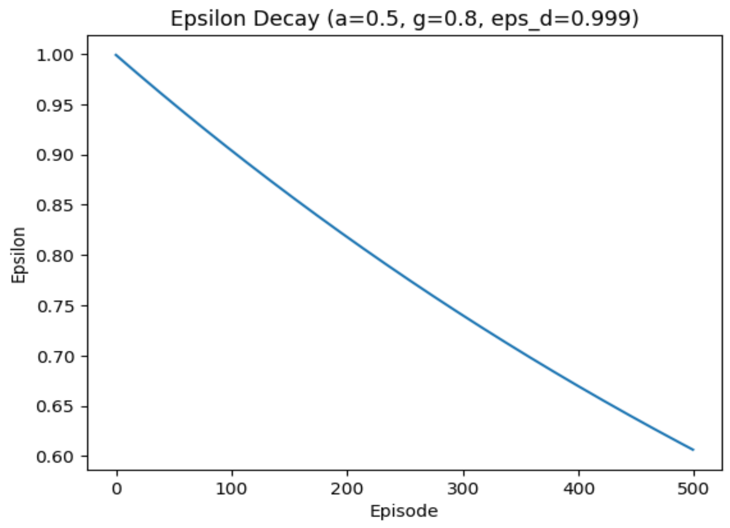
When alpha=0.5, gamma=0.8 and epsilon_decay=0.995, we see that the total reward increased linearly over time, and max Total reward (40) was achieved consistently beyond the 200th episode.



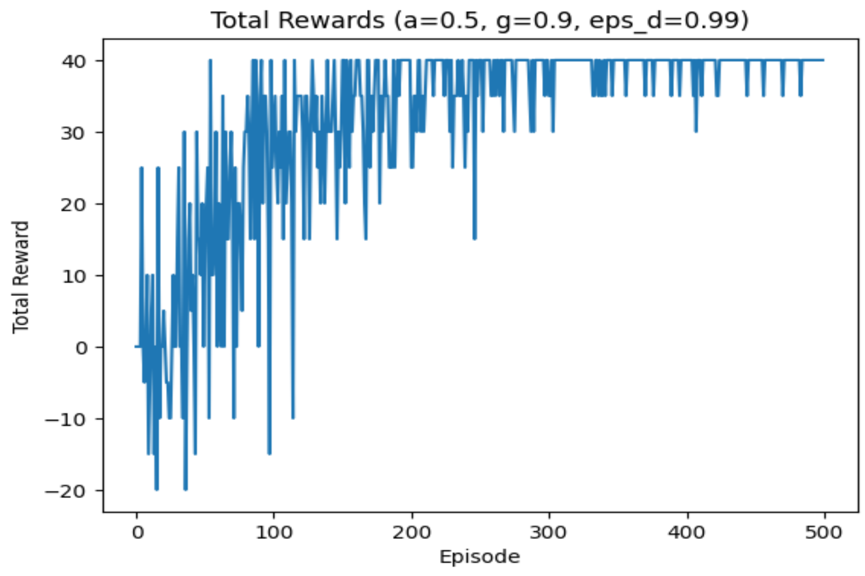
Looking at the above plot, the epsilon has consistently decreased over time indicating that the agent explored actions more in the initial stages of training and slowly shifted to exploiting its learnings more in the later stages, which eventually led to higher rewards. Using this combination of hyperparameters, we do not see any abrupt shift in the agent from exploration to exploitation.



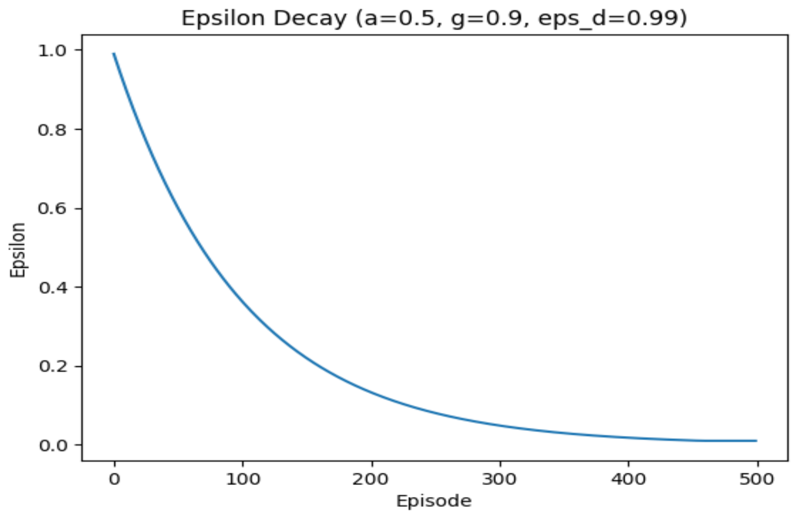
This combination of hyperparameters did not result in notable increment in the Total Reward over the episodes. It shows a lower performance when compared to the previous combinations of hyperparameters.



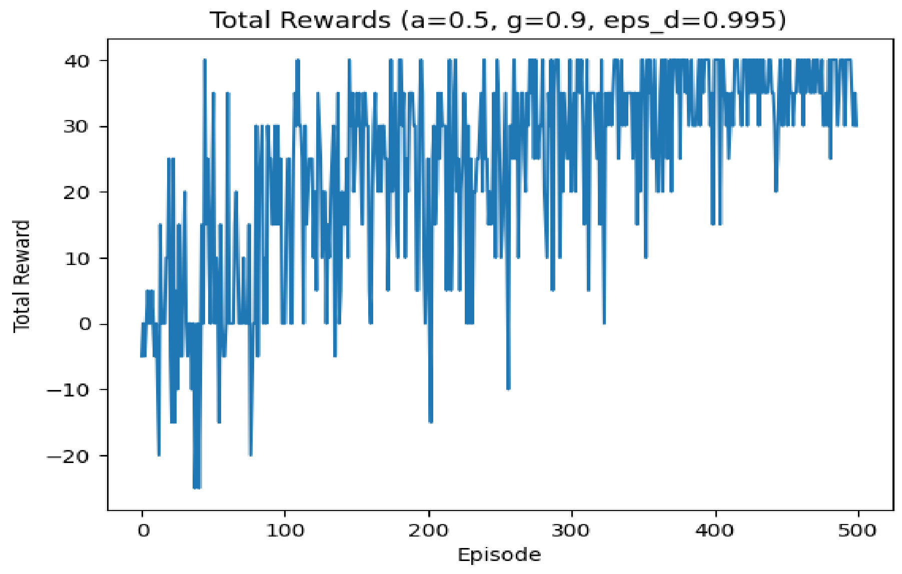
The environment hasn't converged well as the epsilon value decreased only up to 0.6 in 500 episodes, which indicates that the agent did not learn well and had explored actions for a long duration.



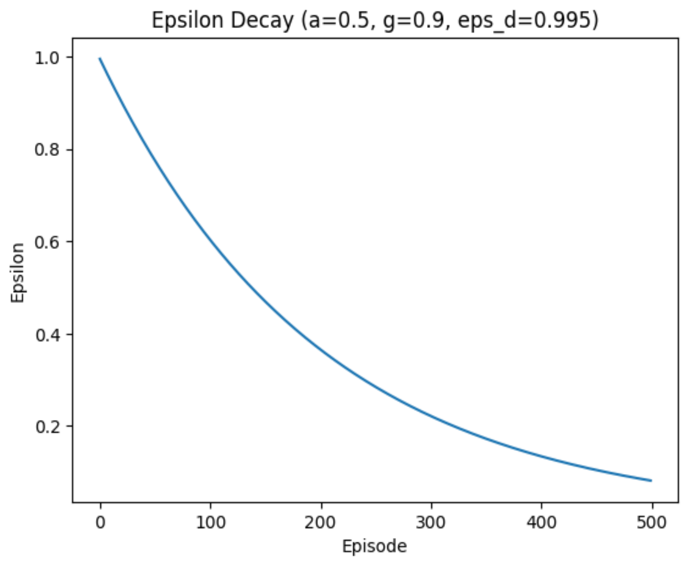
When alpha=0.5, gamma=0.9 and epsilon_decay=0.99, we can see a steep linear increment in the total reward over time. Max Total reward (40) was achieved almost consistently beyond the 100th episode.



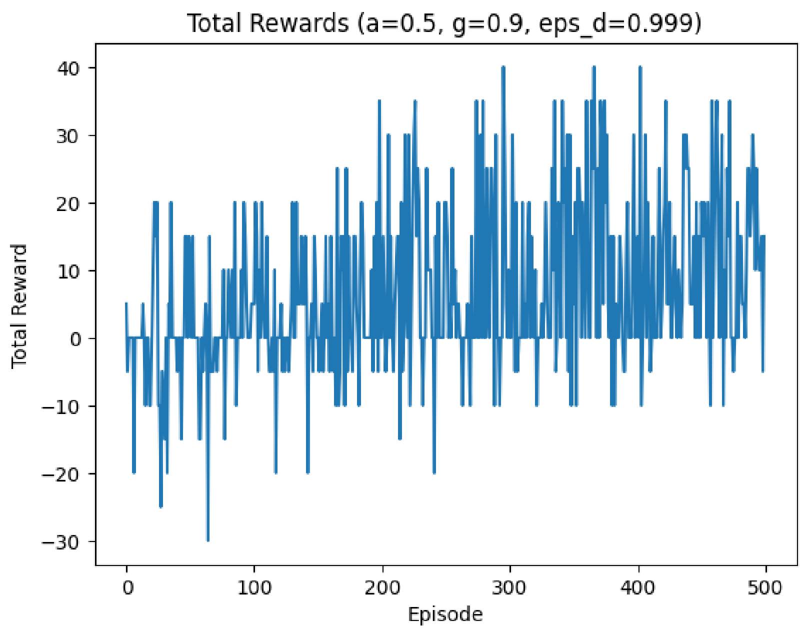
The above plot shows that epsilon decreased exponentially as the training progressed. At the end of the training, the epsilon value smoothly decreased to a very small value close to 0, suggesting that the agent explored actions more in the initial stages of training and slowly shifted to exploiting its learnings more in the later stages, which eventually led to higher rewards.



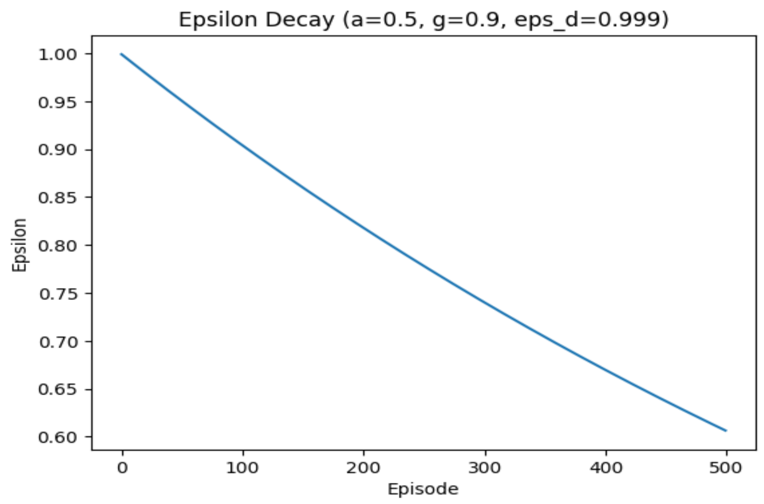
From the above plot, we can see that the total reward increased linearly over time.



The above plot shows that epsilon decreased exponentially as the training progressed, suggesting that the agent explored actions more in the initial stages of training and slowly shifted to exploiting its learnings more in the later stages, which eventually led to higher rewards.



This combination of hyperparameters did not result in notable increment in the Total Reward over the episodes, and also did not achieve the maximum total reward . It shows a lower performance when compared to the previous combinations of hyperparameters.

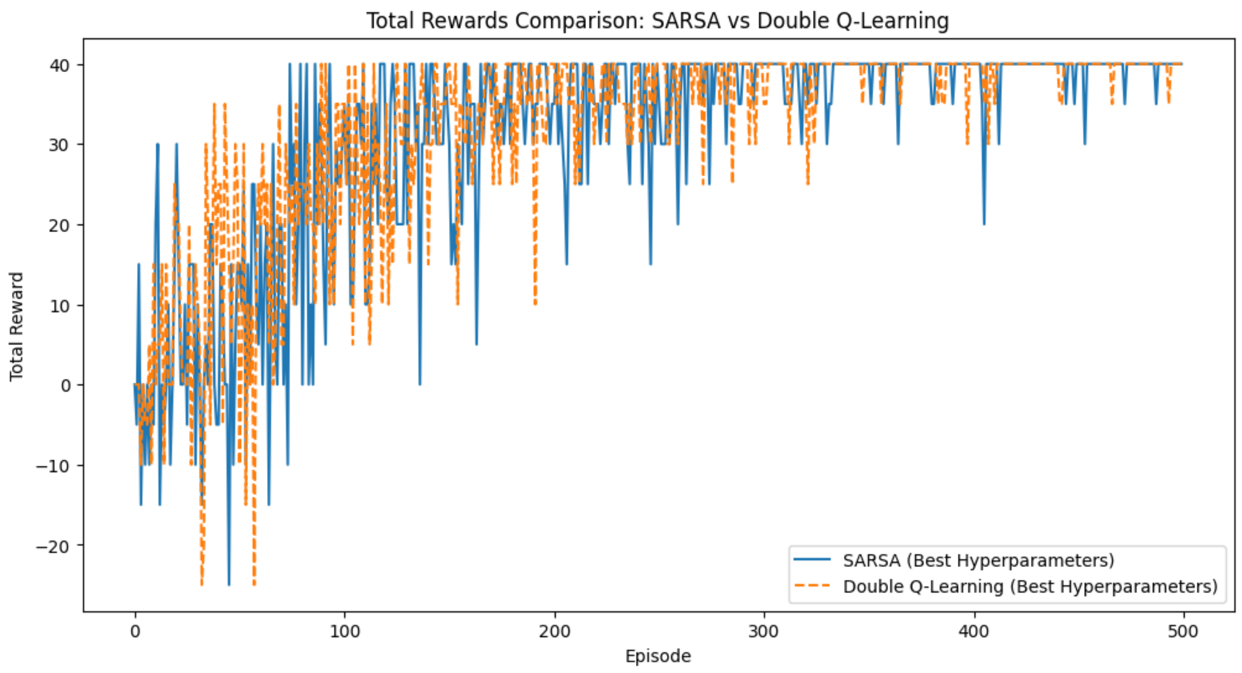


The environment hasn’t converged well as the epsilon value decreased only up to 0.6 in 500 episodes, which indicates that the agent did not learn well and had explored actions for a long duration.

Make your suggestion on the most efficient hyperparameters values for your problem setup.

Based on the above plots, alpha=0.5, gamma=0.8 and epsilon_decay=0.99 gave the best results as the total rewards increased linearly over time and achieved max Total Reward at the earliest when compared to the other set of hyperparameters.

4. Compare the performance of both algorithms on the same environment (e.g. show one graph with two reward dynamics) and give your interpretation of the results.



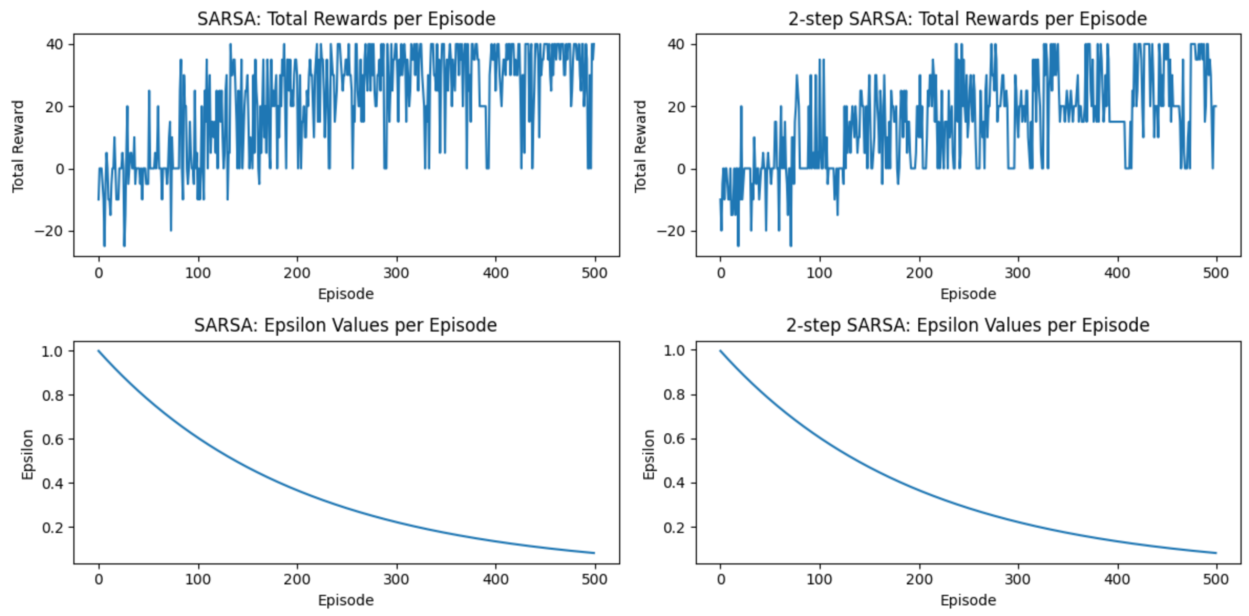
Interpretation of the results:

While the best hyperparameters for both SARSA and Double-Q learning algorithms gave very good results, Double-Q learning performed slightly better in terms of accumulating higher rewards at a lower number of episodes.

Bonus

Task 1

SARSA vs 2-STEP SARSA:



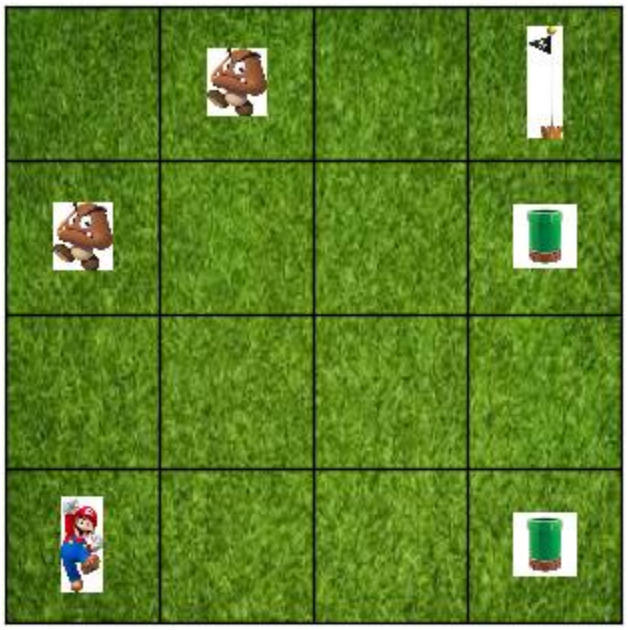
Comparison:

Looking at the above plots, SARSA algorithm has performed better than 2-step SARSA algorithm in terms of cumulative rewards.

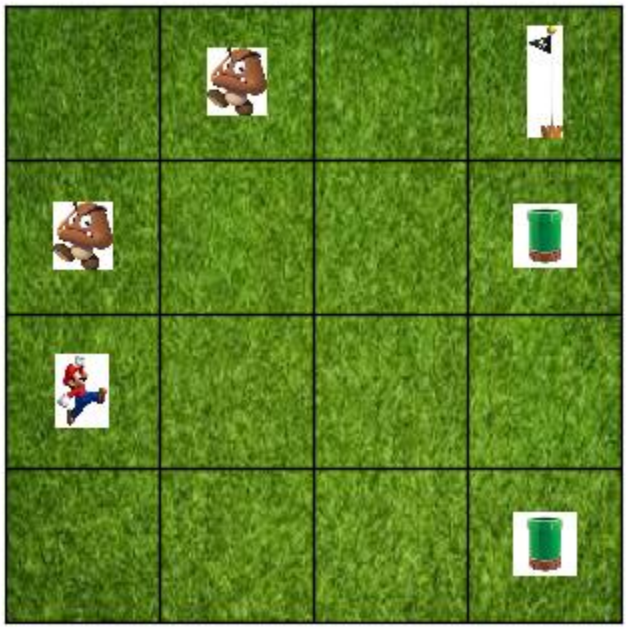
<u>Task 2</u>

For background I've chosen grass
For the agent two states - stationary mario and moving mario
For the negative reward I have chosen an enemy in mario
For the positive reward I have chosen a green pipe in mario
For the goal I have chosen the goal post from mario

When agent is not moving:



When agent is moving:



References

1. https://www.gymlibrary.dev/
2. https://matplotlib.org/stable/contents.html
3. https://numpy.org/doc/stable/
4. https://docs.python.org/3/library/collections.html
5. https://docs.python.org/3/library/itertools.html
6. https://tqdm.github.io/
7. https://docs.python.org/3/library/random.html
8. https://docs.python.org/3/
9. https://docs.python.org/3/library/functions.html
10. https://docs.python.org/3/library/stdtypes.html
11. https://docs.python.org/3/tutorial/controlflow.html
12. https://docs.python.org/3/library/filesys.html
13. https://docs.python.org/3/tutorial/errors.html

Contribution

Team Member	**Assignment Part**	**Contribution %**
Kritik, Mohith	1	50, 50
Kritik, Mohith	2	50, 50
Kritik, Mohith	3	50, 50
Kritik, Mohith	Bonus task 1	50, 50
Kritik, Mohith	Bonus task 2	50, 50