Assignment 4
Defining and Solving
Reinforcement Learning Task

Ram Chandra Bhavirisetty
Yeshwanth Pabbathi

"We certify that the code and data in this assignment were generated independently, using only the tools and resources defined in the course and that We did not receive any external help, coaching or contributions during the production of this work."

PART-1

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

Theme: ' SNAKE AND LADDER ' GRID WORLD WHERE LADDERS ARE OF POSITIVE REWARD AND SNAKES ARE OF NEGATIVE REWARD.

States: {S1 = (0,0), S2 = (0,1), S3 = (0,2), S4 = (0,3), S5=(0,4), S6=(0,5),

S7 = (1,0), S8 = (1,1), S9= (1,2)..... S34 = (5,3), S35=(5,4), S36=(5,5),

Actions: {Up, Down, Right, Left}

Rewards: {-2, -4, +5, +10}

Objective: Reach the goal state with maximum reward

Number of states: 25

Number of Actions: 4

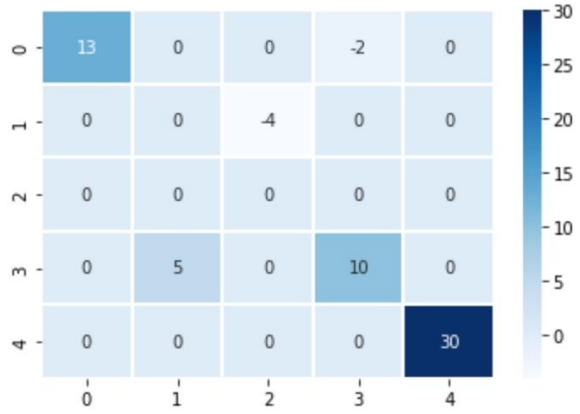
Number of Timesteps: 100

2. Provide visualisation of your environment.

```
NUMERICAL REPRESENTATION OF GRID WORLD:
START STATE [0,0]: 13
GOAL STATE [4,4]: 1

[[13.  0.  0. -2.  0.]
 [ 0.  0. -4.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  5.  0. 10.  0.]
 [ 0.  0.  0.  0. 30.]]

GRAPHICAL REPRESENTATION OF GRID WORLD:
```



3. Safety in AI: Write a brief review explaining how you ensure the safety of your environment.

Ensuring Safety of my Environment:

- The code should be secure in order to maintain the security. Reinforcement learning can be tweaked so that the code is developed in a risk-averse manner.
- Understanding the potential dangers allows me to create and implement modifications to make the application more secure.

PART-2

1. Show and discuss the results after applying SARSA to solve the environment defined in Part I.

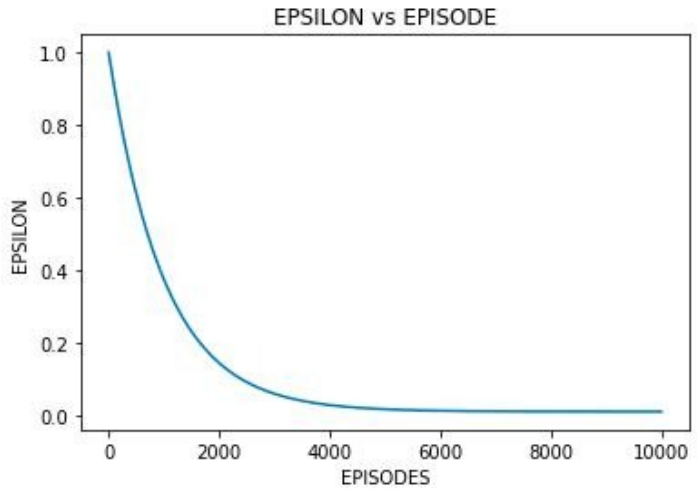
Obtained Q values after applying SARSA algorithm.

```
{0: array([2.91266388, 6.05313511, 9.5197179 , 6.77364796]),
 1: array([2.04297066, 3.73486737, 6.94103212, 2.98105156]),
 2: array([0.34376224, 1.89488519, 0.4952092 , 0.34822013]),
 3: array([0.20621   , 0.99914941, 0.13163882, 0.46658441]),
 4: array([0.06827865, 0.36009044, 0.03734174, 0.00658041]),
 5: array([0.86996254, 0.33040081, 2.05380739, 0.9433232 ]),
 6: array([ 7.39816801,  7.33058219, 14.92510518,  4.55536219]),
 7: array([ 3.67337459, 10.55448308,  3.58079459,  2.67174328]),
 8: array([0.32745345, 6.25368362, 0.91703689, 0.33737531]),
 9: array([ 0.26456926,  0.50075683, -1.59210998,  0.05698367]),
 10: array([-0.00220549,  0.90654762, -0.29471264,  0.07202207]),
 11: array([0.00748961, 1.16300747, 2.3685215 , 0.09930405]),
 12: array([ 7.61921372,  9.99854641, 16.25772532,  6.61506283]),
 13: array([ 4.34162352, 11.05902241,  5.47034632,  3.94384318]),
 14: array([-1.7764708 ,  0.95798194,  6.42194826,  1.50367002]),
 15: array([ 1.79596114,  3.11600146, -0.10004851,  0.68136954]),
 16: array([ 0.50781696, -1.39507465,  3.16402095,  0.24201587]),
 17: array([0.07655781, 0.14301424, 4.57026944, 0.21500699]),
 18: array([10.39937766, 11.4523625 , 23.32945916, 13.7607406 ]),
 19: array([ 6.0749791 , 16.30534544,  7.13182855,  7.66221449]),
 20: array([2.47058454, 3.46858714, 8.92352075, 2.19854774]),
 21: array([ 0.31782087,  2.49376084,  2.33154442, -0.71560463]),
 22: array([1.00379005, 0.54205916, 0.66578748, 0.52298171]),
 23: array([4.4966044 , 0.88971212, 6.99999518, 1.22759486]),
 24: array([12.97489795, 23.33333333, 13.19392213, 15.27605531]),
 25: array([ 9.24114772, 18.40738649,  9.23085144, 10.12148752]),
 26: array([ 3.35245177, 13.2066187 ,  5.43240793,  5.32084795]),
 27: array([4.37776043, 0.64569734, 4.62140537, 2.63439245]),
 28: array([6.43994386, 1.05907974, 6.97609523, 0.44892544]),
 29: array([ 7.        ,  2.30488176, 10.        ,  0.94426776]),
 30: array([ 9.23747018, 13.20561205, 13.21902684, 18.85704139]),
 31: array([ 2.73588855,  4.65389683,  8.92716315, 13.20751447]),
 32: array([2.07314562, 7.31341295, 0.92213764, 2.03578018]),
 33: array([1.89872359, 3.68451535, 0.42368343, 0.99696958]),
 34: array([10.        ,  1.45589672,  6.99998067,  4.86112681]),
 35: array([0., 0., 0., 0.])}
```

```
def q_update(self,state,action,state2,action2,reward):
  pred = self.Q[state,action]
  future_rwd = reward + self.gamma * self.Q[state2,action2]
  self.Q[state,action] = self.Q[state,action] + self.alpha*(future_rwd - pred)
```

Q-values are updated using this function.

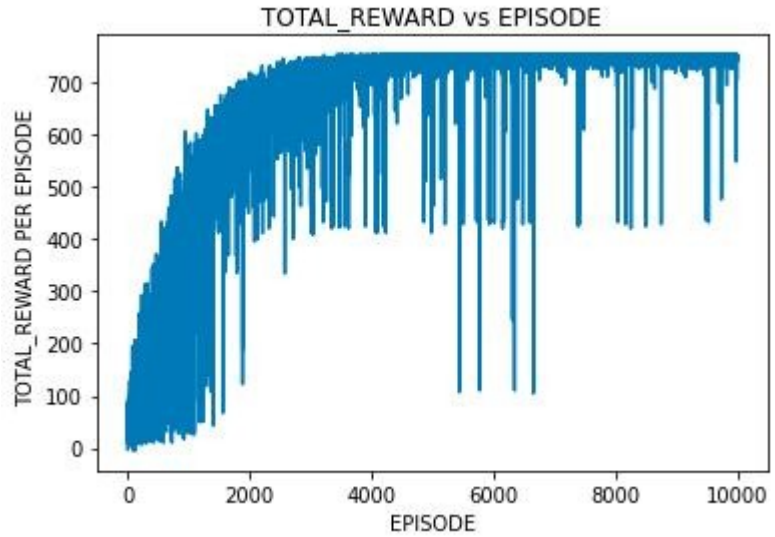
2. Provide a plot for epsilon decay




```
agent.epsilon = 0.01 + (0.99)*np.exp(-0.001*i)
```

Epsilon decay is calculated based on this equation.

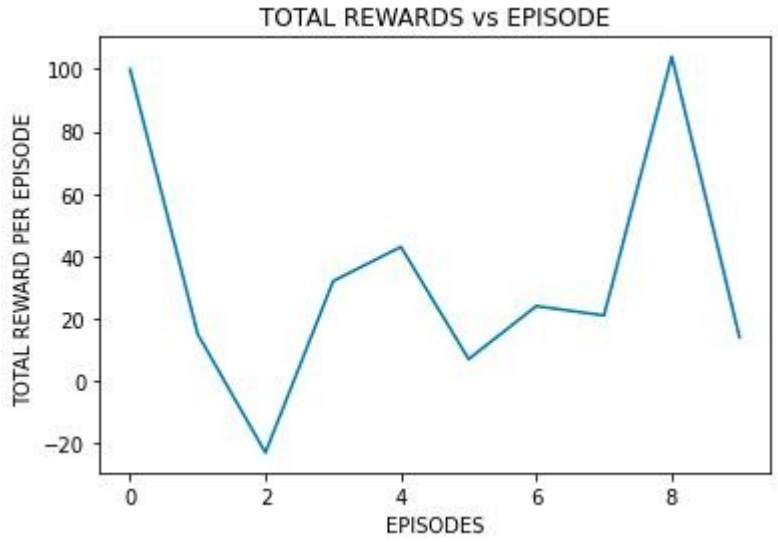
3. Provide a plot for the total reward per episode.



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

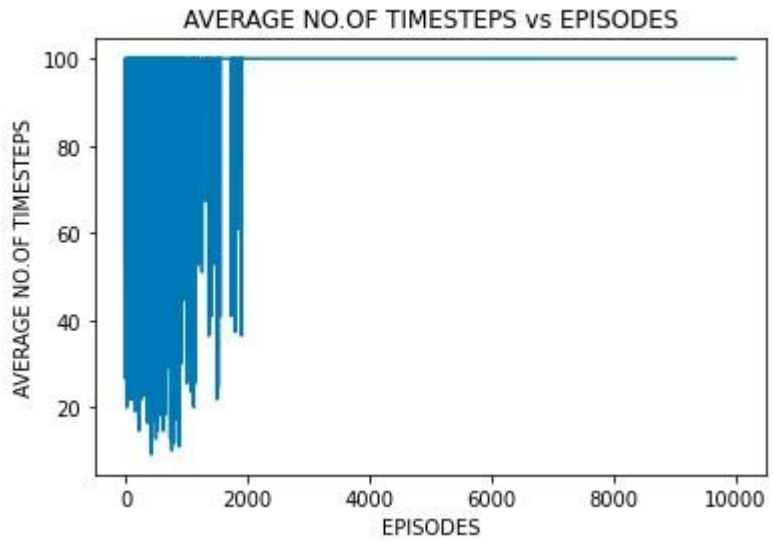
By initialising the epsilon value to '0' and updating Q-values with the values obtained in the process of training.

```
self.actions = env.action_space
self.states = env.field
self.epsilon = 0.9
self.alpha = 0.9
self.gamma = 0.95
self.Q = Q_vals #np.zeros((36,4))
```



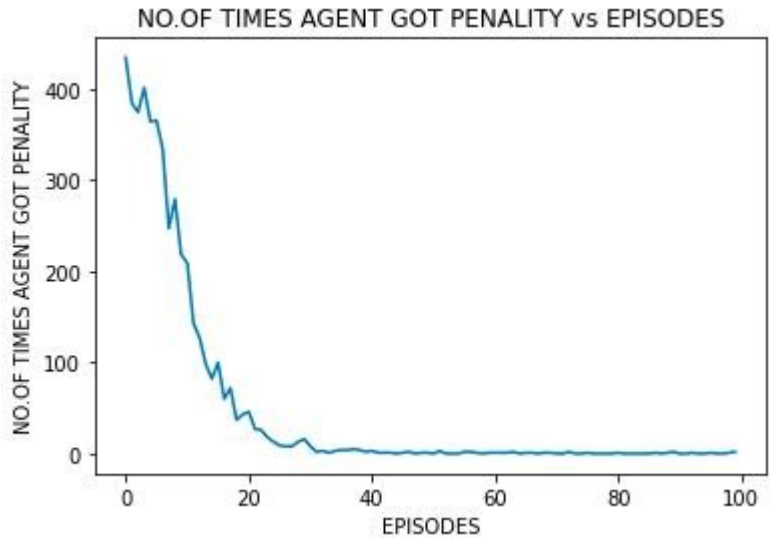
Graphical representation of total rewards with respect to obtained Q-values form training.

5. Give your interpretation of the results.

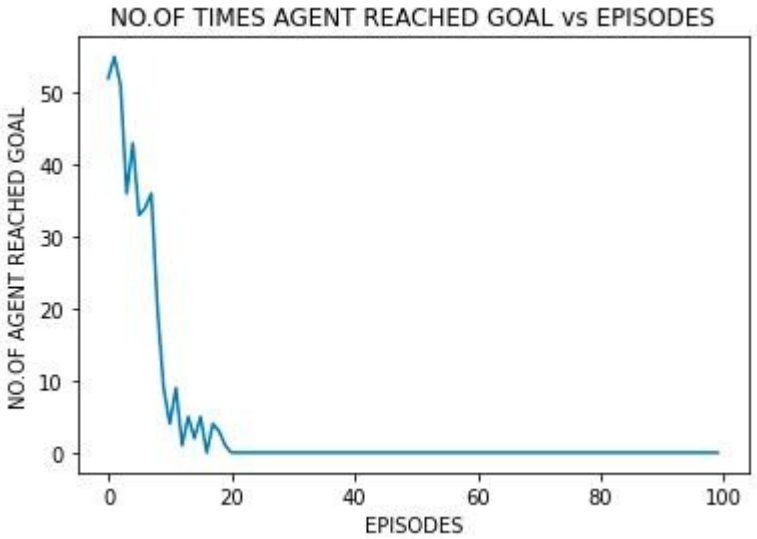


Graphical Representation for average number of time steps taken for each episode.

```
average_timesteps.append((env.step_count/env.max_timesteps)*100)
```



Graphical Representation of agent approaching negative rewards per 1000 episodes.



Graphical Representation of number of times agent reached goal per 100 episodes.

6. Briefly explain these tabular methods: SARSA and Q-learning. Provide their update functions and key features.

The main motto of these both tabular methods is updating the q-value. In SARSA Q-value is obtained by the actions performed in the current state, this method is called on-policy. In case of Q-Learning Q-value is obtained by greedy approach, this method is called off-policy.

SARSA:

$$Q(S, A) = Q(S, A) + \alpha[R + \gamma Q(S_1,A_1) - Q(S, A)]$$

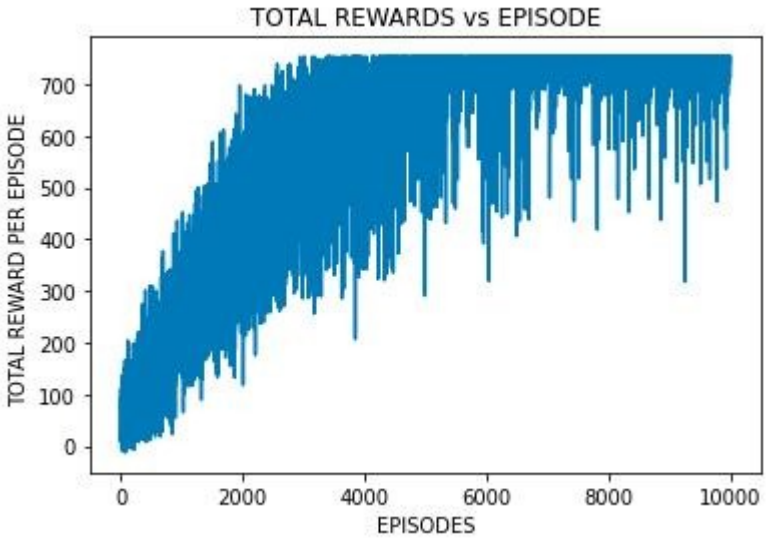
Q-Learning:

$$Q(S, A) = Q(S, A) + \alpha[R + \gamma*\max(Q(S_1,A)) - Q(S, A)]$$

7. Provide the analysis after tuning at least two hyperparameters from the list above.

Based on the analysis, we observe that the best reward is obtained when episodes = 10000, epsilon=0.9, alpha=0.9, gamma= 0.95

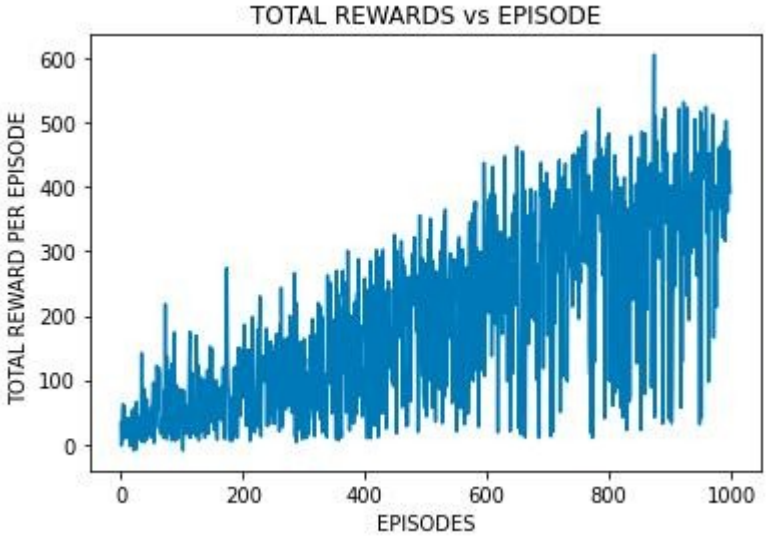
Using these values the rewards per episode graph is obtained.



8. Try at least 3 different values for each of the parameters that you choose. Provide the reward graphs and your explanation for each of the results. In total you should have at least 6 graphs and your explanations. Make your suggestion on the most efficient hyperparameters values for your problem setup.

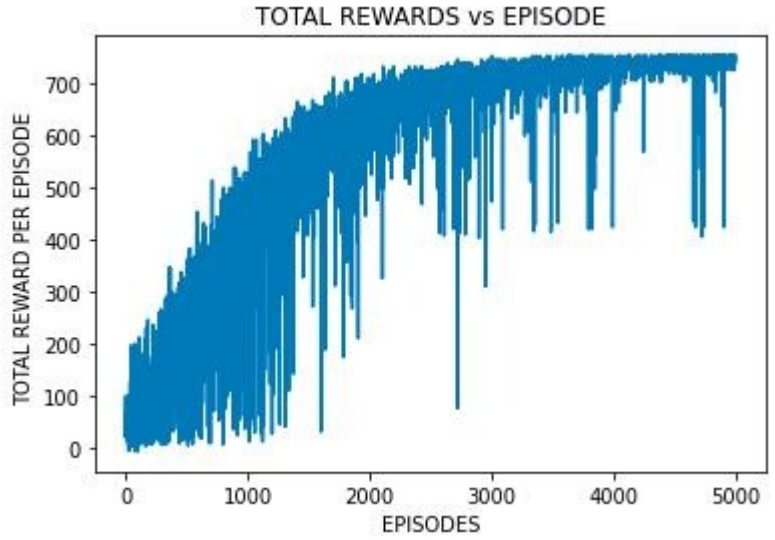
Episode Tuning:
Episodes=1000, gamma=0.7

Performing epsilon tuning, the graph represents the total rewards obtained per episode when tuning is performed for 1000 episodes. This results, minimum number of rewards for initial episodes.



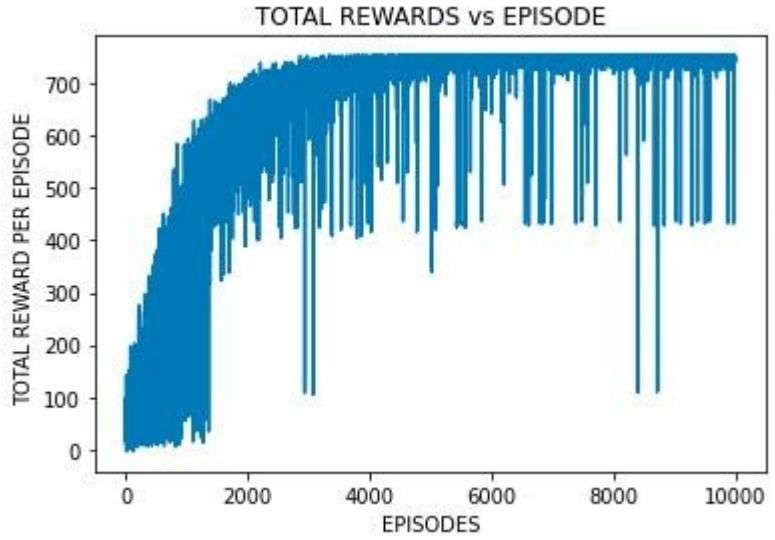
Episodes=5000, gamma=0.7

Performing epsilon tuning, the graph represents the total rewards obtained per episode when tuning is performed for 5000 episodes. In this total rewards value increased compared to previous tuning and reached a maximum reward of around 750.



Episodes=10000, gamma=0.7

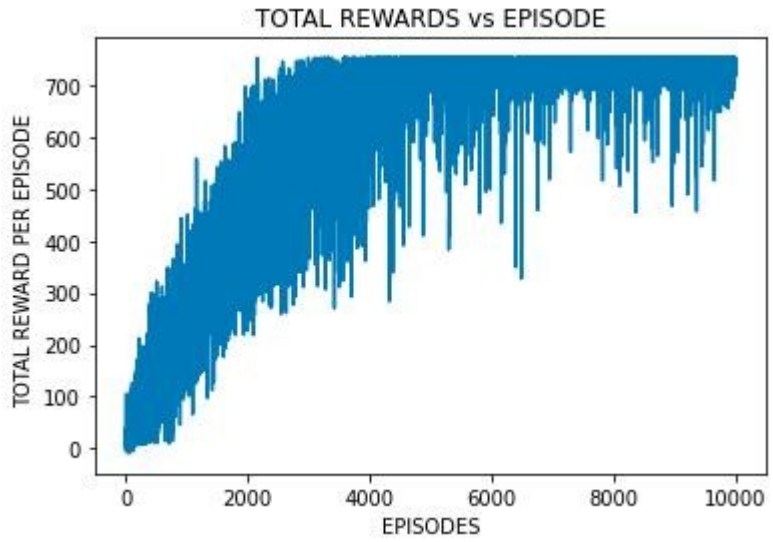
Performing epsilon tuning, the graph represents the total rewards obtained per episode when tuning is performed for 10000 episodes.In this total rewards value increased compared to previous tuning and reached a maximum reward of around 750.



Gamma Tuning:
Episodes=10000, gamma=0.95

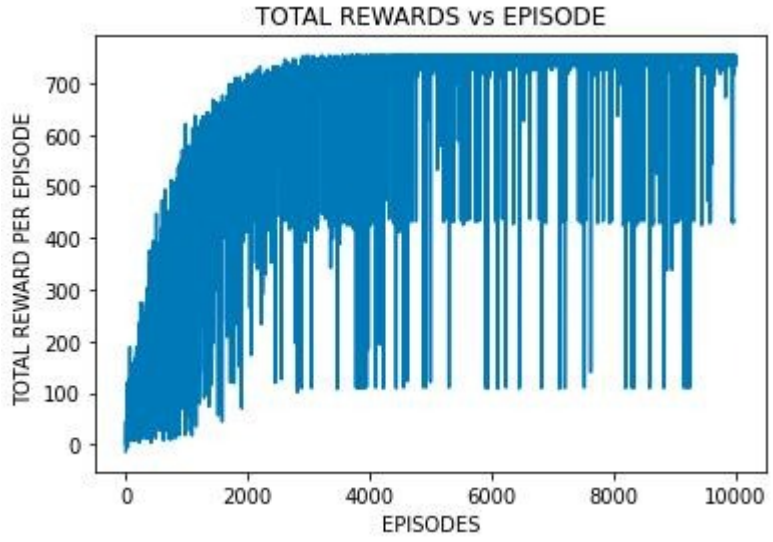
Performing gamma tuning, the graph represents the total rewards obtained per episode when tuning is performed for 10000 episodes and gamma value is 0.95. In this total

rewards value increased compared to previous tuning and reached a maximum reward of around 750.



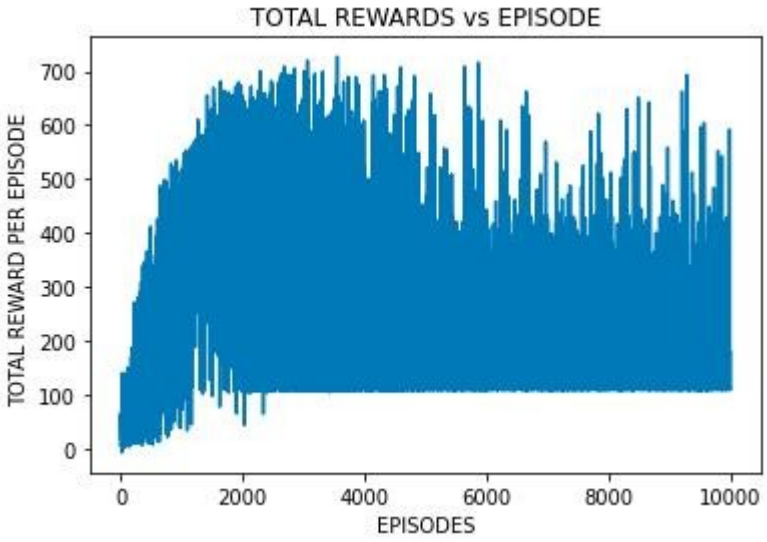
Episodes=10000, gamma=0.5

Performing gamma tuning, the graph represents the total rewards obtained per episode when tuning is performed for 10000 episodes and gamma value is 0.5. In this total rewards value increased compared to previous tuning and reached a maximum reward of around 750.



Episodes=10000, gamma=0.2

Performing gamma tuning, the graph represents the total rewards obtained per episode when tuning is performed for 10000 episodes and gamma value is 0.2. In this total rewards value increased compared to previous tuning and reached a maximum reward of 700.



The best combination among these is, when episodes= 10000 and gamma=0.95

Contribution Table -

Team Member	Assignment Part	Contribution %
Ram Chandra Bhavirisetty	Part 1	50%
Yeshwanth Pabbathi	Part 1	50%
Ram Chandra Bhavirisetty	Part 2	50%
Yeshwanth Pabbathi	Part 2	50%

References:

https://medium.com/swlh/introduction-to-reinforcement-learning-coding-sarsa-part-4-2d64d6e37617
https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292
https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/
https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/
https://towardsdatascience.com/double-q-learning-the-easy-way-a924c4085ec3

https://www.google.com/search?q=reward+graph+in+reinforcement+learning+model&sxsrf=ALiCzsbNA-FyW-Z11SO_g3hXINfowjeq6w:1651887177338&tbm=isch&source=iu&ictx=1&vet=1&fir=mVVhdljgmHc5tM%252CpR4PwAdWVkD6yM%252C_%253BzpEmLfCc4_z63M%252CsgNTSAOWgc9iEM%252C_%253BE9hgcQQMzZu7PM%252Cf90WdVNtDqE1ZM%252C_%253BxBlOtZJ8gDKZ4M%252CRTy0OHptaQ_jZM%252C_%253Brhbf4KCskB1ARM%252CvmAXlut6BVVG2M%252C_%253Br4eXDxSvZFlJfM%252CsgNTSAOWgc9iEM%252C_%253Bg8BJj3yQhgkx1M%252CrGGbt1VH_xachM%252C_%253BnXc9F0-q0z499M%252CkXIrzXyOzEQLdM%252C_%253BNnlfUMmd50JG9M%252CatIOdoZXPLjfSM%252C_%253BpZcZC5_FV0V1HM%252CvmAXlut6BVVG2M%252C_&usg=AI4_-kSelYYz5h22CC9GHYY9QxGnuJxzLQ&sa=X&ved=2ahUKEwjN2aejn8z3AhWMk4kEHQJKCNgQ9QF6BAgGEAE#imgrc=mVVhdljgmHc5tM