Assignment-3

Part I: Define an RL Environment

1. Describe the environment that you defined. Provide a set of states, actions, rewards, main objective, etc.
Answer:
Description of the Cat Quest Environment:
In the CatQuest environment, a cat navigates a grid filled with milk, fish, traps, water, and a dog, aiming to reach the fish while avoiding traps, water, and the dog. The "CatQuestEnvironment" is a custom OpenAI Gym environment designed for reinforcement learning tasks. In this grid world, a cat agent navigates through a 6x6 grid with various objects and goals. The cat's objective is to maximize cumulative rewards by reaching positions associated with positive rewards, such as milk and a high-reward fish, while avoiding positions with negative rewards, including traps, water, and a dog. The environment provides a discrete action space (up, down, left, right) and discretizes the state space into 36 possible states. Episodes can terminate when the cat reaches the fish position or after a specified number of timesteps. The custom rendering method visualizes the grid world with Matplotlib, displaying the cat, milk, fish, traps, water, and dog at their respective positions.
Environment size: Grid world of size: 6x6.
Agent: Cat is the agent and is at the starting position (0, 0) and state is 0.
Goal: Fish is the highest positive reward, as the goal at (2, 4) position.
Rewards: Below is the set of positive and negative rewards that we are using in our Cat Quest Environment.
Milk positions offer a positive reward of +3.
Fish is the final goal with highest positive reward of +10.
Traps and water position are the with low negative rewards of -2 and -1.
Dog position is considered as the final villain position with the highest negative reward of -5.
0 points for other actions.
So, Rewards are: {-5, -2, -1, 3, 10}

```
for row in self.milk_pos:
    self.reward_dict[tuple(row)] = 3
self.reward_dict[tuple([2,4])] = 10
for row in self.water_pos:
    self.reward_dict[tuple(row)] = -1
for row in self.traps_pos:
    self.reward_dict[tuple(row)] = -2
self.reward_dict[tuple([3, 3])] = -5
```

Set of states: There are 36 states in our grid world is represented by a matrix, where each cell contains one of the following values: Empty space, Water, Trap, Fish, Dog as shown below:
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) | s10 = (1, 3) | s11 = (1, 4) | s12 = (1, 5)
s13 = (2, 0) | s14 = (2, 1) | s15 = (2, 2) | s16 = (2, 3) | s17 = (2, 4) | s18 = (2, 5)
s19 = (3, 0) | s20 = (3, 1) | s21 = (3, 2) | s22 = (3, 3) | s23 = (3, 4) | s24 = (3, 5)
s25 = (4, 0) | s26 = (4, 1) | s27 = (4, 2) | s28 = (4, 3) | s29 = (4, 4) | s30 = (4, 5)
s31 = (5, 0) | s32 = (5, 1) | s33 = (5, 2) | s34 = (5, 3) | s35 = (5, 4) | s36 = (5, 5)

Start state: The cat is the S1 state that is (0, 0) at the beginning of our quest.
Terminating state: The termination status would be, the cat catches a fish which is the final goal positioned at (2, 4).

Actions:

The possible actions in our grid are: {Up, Down, Right, Left}.

Main Objective:

In the Cat Quest environment, the main objective is for the cat to navigate a grid world in search of a rewarding fish while avoiding obstacles like traps, water, and a dog. The cat receives positive rewards for finding milk but faces negative rewards like traps, water, or the dog. The aim is to train the cat in such a way that it can reach the fish by maximizing rewards and avoiding negative consequences.

2. Provide visualization of your environment.

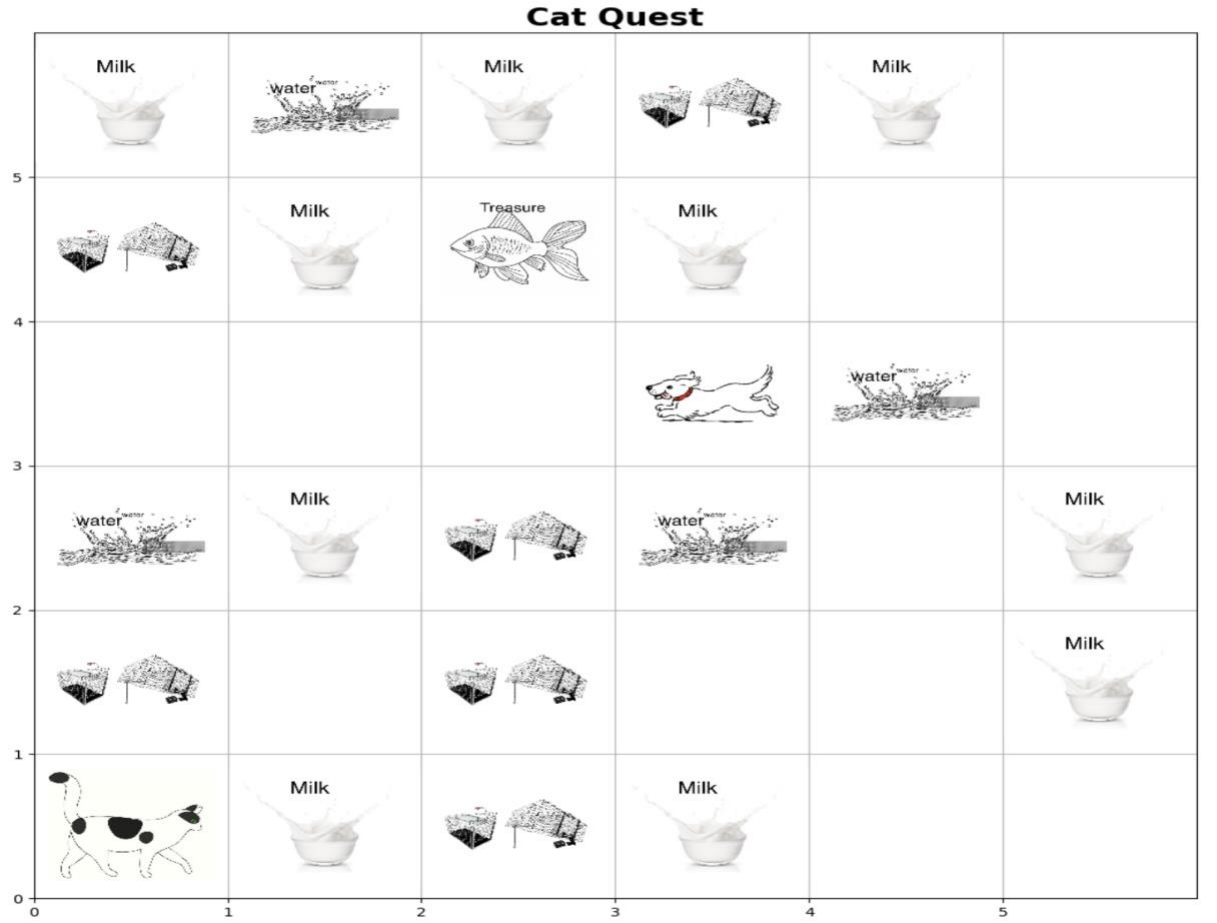
Answer:

Possible Directions are {0: ‘up’, 1: ‘down’, 2: ’left’, 3:’right’ }

Visualization of the Cat Quest Environment when agent(cat) is the stationary position:

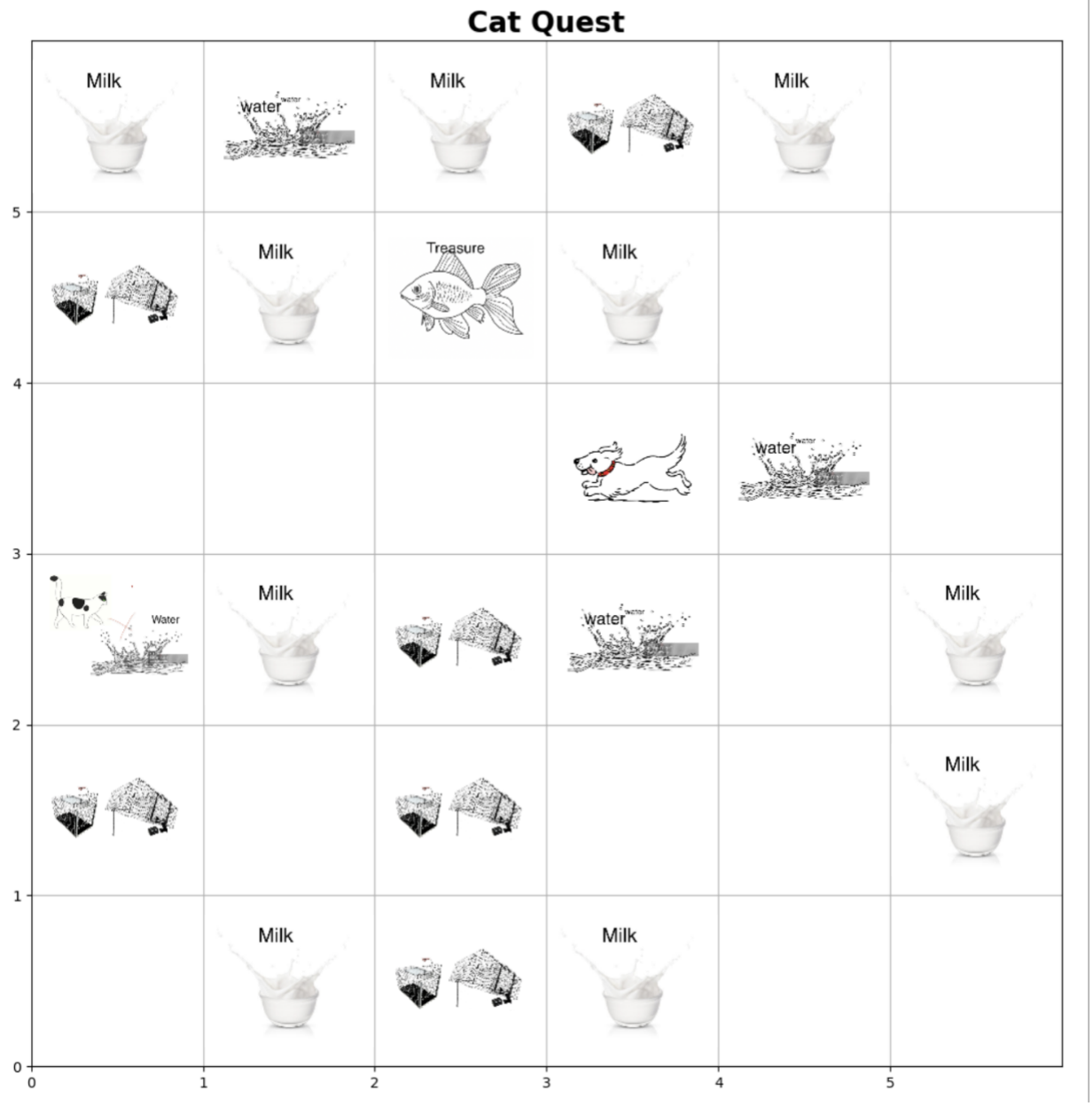
```
action_dict = {0:"up", 1:"down", 2: "left", 3:"right"}
cat_quest = CatQuestEnvironment(env_type = 'deterministic', n = 36,max_steps = 100)
cat_quest.render(plot = True)
```

```
Starting state of the cat is [0, 0]
Current action: Cat is at stationary position not taking any action
**********
```



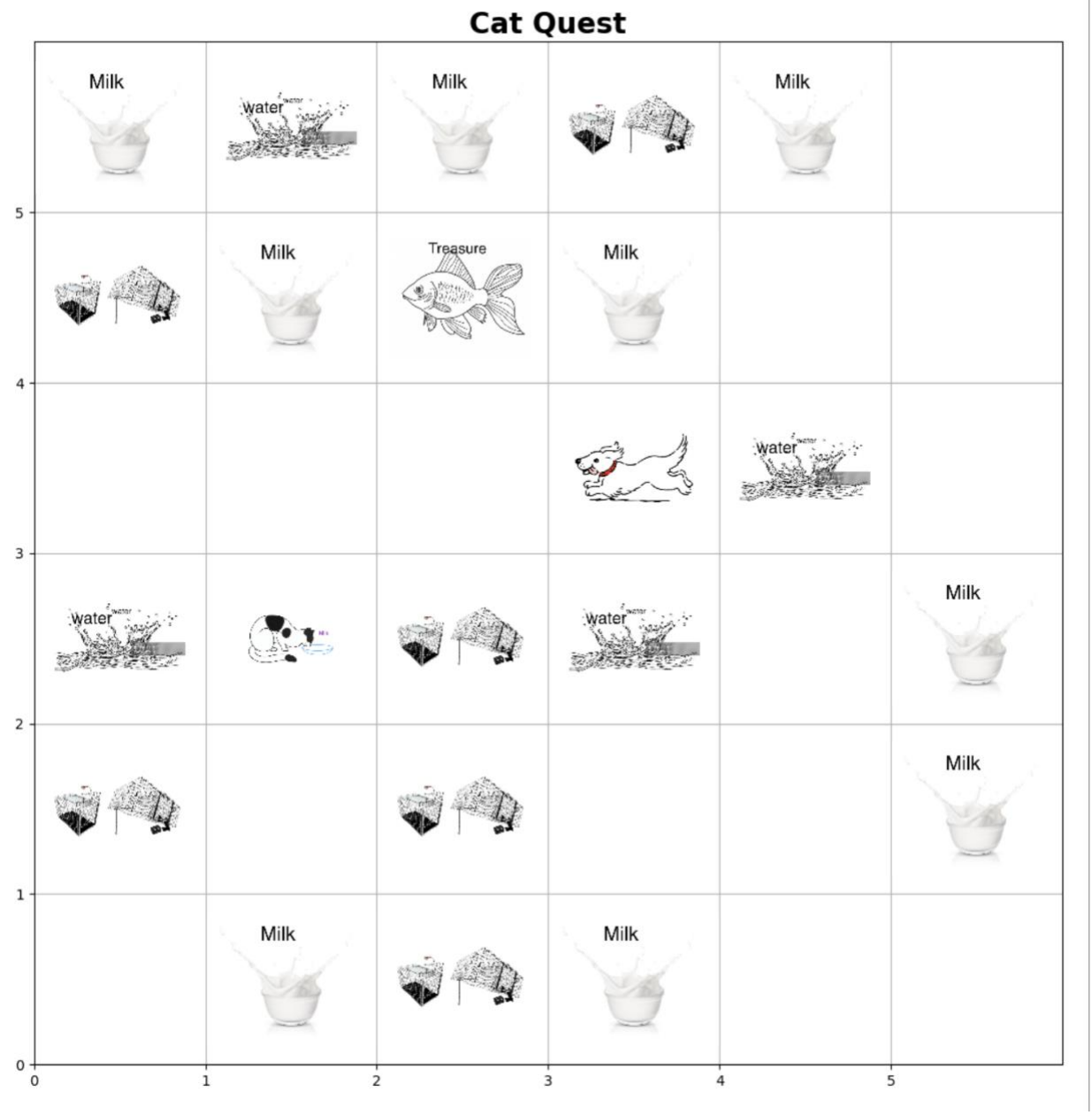
Other few scenario visualizations:
Visualization of the environment when cat falls into the water:

```
**************************************************************************************************************************
Cat Current Position 2, Action: left, Reward: 0, Goal Reached: False, trauncated: True
```



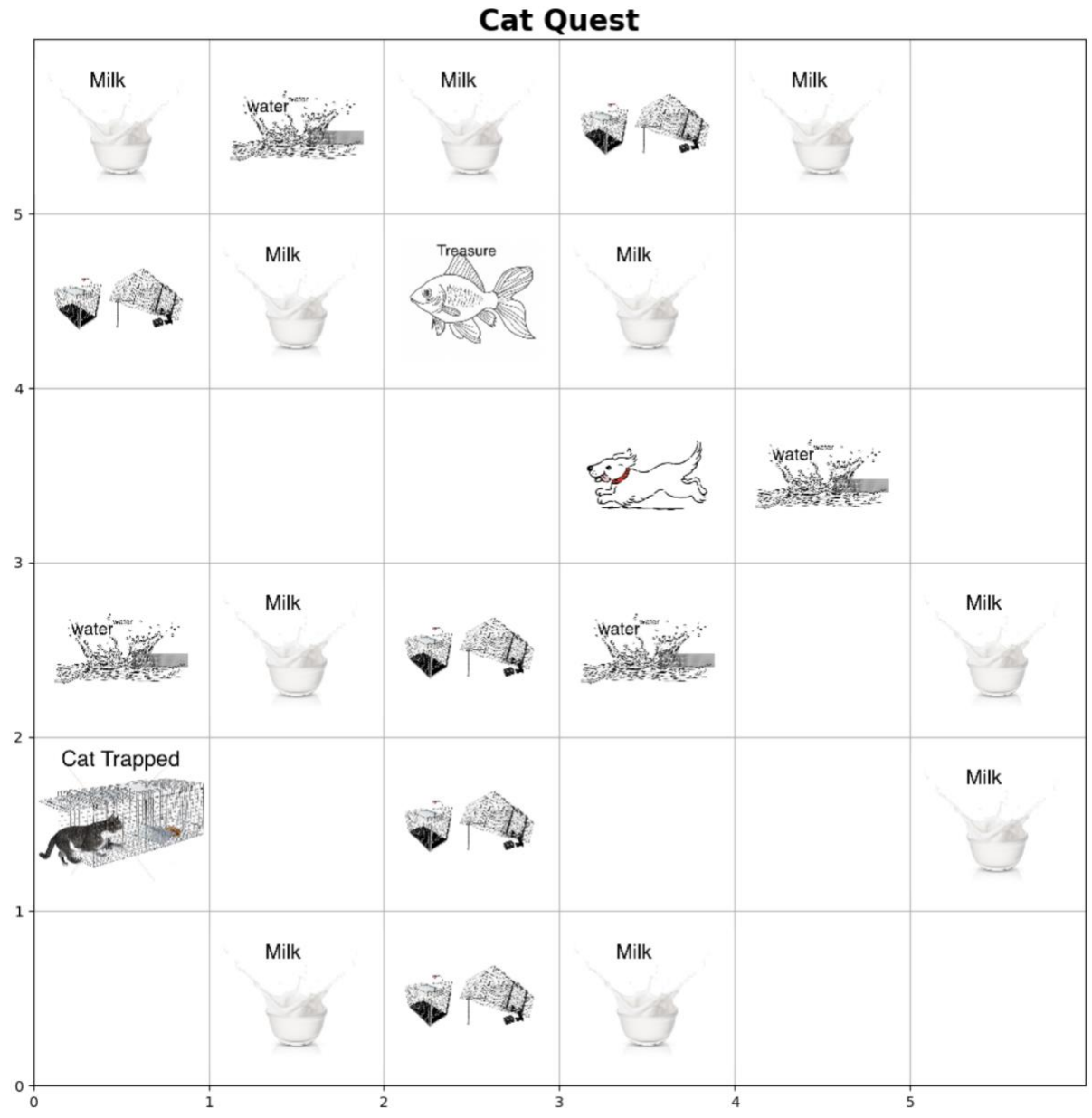
Visualization of the environment when cat drinks milk:

```
**********************************************************************************************************************************
Cat Current Position 8, Action: right, Reward: 3, Goal Reached: False, trauncated: True
```



Visualization of the environment when cat falls in trap:

Cat Current Position 1, Action: left, Reward: -2, Goal Reached: False, trauncated: True



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.,

Answer:

Agent chooses only actions that are allowed:

The action space of the agent is constrained using a discrete action space, limiting the agent to four permissible actions: up, down, left, and right.

```
action = cat_quest.action_space.sample()
```

In our environment, we used *cat_quest.action_space.sample()* so that the agent only chooses the actions that are allowed. it randomly samples an action from the defined action space. The action_space attribute of the Cat Quest environment is set to *spaces.Discrete(4),* indicating a discrete action space with four possible actions (up, down, left, right).

```
action_dict = {0:"up", 2:"left", 1: "down", 3:"right"}
if action == 3: #right
    self.cat_pos[0] += 1
if action == 2: #left
    self.cat_pos[0] -= 1
if action == 0: #up
    self.cat_pos[1] += 1
if action == 1: #down
    self.cat_pos[1] -= 1
```

Agent is navigating within defined state-space:

```
self.cat_pos = np.clip(self.cat_pos, 0, 5)
```

The agent's navigation within the defined state-space in the CatQuest environment is done by the np.clip function applied to the cat's position after each action. *self.cat_pos = np.clip(self.cat_pos, 0, 5).* The np.clip function restricts the cat's position to be within the specified grid boundaries (0 to 5 in both dimensions) of the 6x6 grid. This ensures that the agent's position remains within the defined state-space, preventing it from going beyond the grid boundaries.

```
if np.all((self.cat_pos >=0) & (self.cat_pos <= 5)):
    truncated = True
else:
    truncated = False
```

The agent is navigating within a state-space that is constrained to the range [0, 5]. The state-space represents the possible positions of the cat, and the conditions self.cat_pos >= 0 and self.cat_pos <= 5 ensure that the cat's position is within the allowed boundaries.

The variable truncated is set to True in this case, indicating that the agent (cat) is successfully navigating within the specified state-space. On the other hand, if any element of self.cat_pos falls outside the defined range, truncated is set to False, signifying that the agent has deviated from the allowed state-space.

Part II: SARSA && Part III: Double Q-learning

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?

Answer:

SARSA and Double Q learning can be used to solve problems of reinforcement learning.

SARSA(State-Action-Reward-State-Action):

SARSA, which stands for State-Action-Reward-State-Action, is a reinforcement learning algorithm used to solve Markov Decision Processes (MDPs). It belongs to the class of tabular methods and is designed to learn an optimal policy by updating Q-values in a Q-table based on the agent's experiences in an environment. SARSA is an on-policy algorithm, meaning it learns the Q-values and policy for the current policy it is following, and it often employs an exploration-exploitation strategy to balance the exploration of new actions and the exploitation of the current best-known actions. The algorithm's update function considers the current action and the next action the agent would take in the next state, making it suitable for tasks where the agent's actions influence its future states.

Update function of SARSA:

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$$

1. Q (S, A): Current estimated cumulative future rewards for action A in state S.
2. Q (S', A'): Estimated cumulative future rewards for the agent's next move in the next state S' with action A'.
3. R: Reward received by the agent for taking action A in state S.
4. α (Alpha): Learning rate
5. γ (Gamma): Discount factor

Key Features:

1. Q-Table Representation:

- SARSA maintains a Q-table, storing the expected cumulative future rewards for taking a particular action in a given state.

2. On-Policy Learning:

- SARSA is an on-policy algorithm, meaning it learns the Q-values and policy for the current policy it is following. The update function considers the current action and the next action.

3. Exploration-Exploitation Strategy:

- Typically employs an exploration strategy, such as epsilon-greedy, to balance exploration of new actions and exploitation of the current best-known actions.

Advantages:

1. Convergence Guarantee:

- SARSA is guaranteed to converge to the optimal policy for finite MDPs under certain conditions, providing stability in learning.

2. Ease of Implementation:

- SARSA is relatively simple to implement, making it suitable for introductory reinforcement learning studies.

3. On-Policy Stability:

- On-policy learning contributes to stability, ensuring that the algorithm learns values consistent with the current policy.

Disadvantages:

1. Overestimation Bias:

- SARSA may suffer from overestimation bias, particularly in environments with high variance in rewards, potentially leading to suboptimal policies.

2. Limited to Tabular Representation:

- SARSA relies on a tabular representation of the state-action space, limiting its applicability to problems with discrete and small to moderate-sized state spaces.

3. Not Suitable for Continuous Spaces:

- The discrete nature of SARSA makes it unsuitable for problems with continuous state or action spaces.

Double Q Learning:

Double Q Learning is an enhancement to traditional Q-learning designed to address overestimation bias. It uses two sets of Q-values, alternating between them during updates. This approach mitigates optimism in value estimates, providing a more accurate assessment of action values. Double Q Learning improves stability and performance in reinforcement learning tasks by offering a balanced approach to action selection and evaluation.

Update Function (Double Q-Value Update):

$$\text{With 0.5 probabilility:}$$
$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S',a)\big) - Q_1(S,A)\Big)$$
$$\text{else:}$$
$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S',a)\big) - Q_2(S,A)\Big)$$

- $Q1(S,A)$ and Q2(S,A)$Q2(S,A)$ are the Q-values for action A in state S for the first and second sets of Q-values, respectively.
- R is the reward received by the agent for taking action *a* in state *S*.
- α is the learning rate.
- γ is the discount factor.

Key Features:
1. Two Sets of Q-Values: Double Q-learning maintains two sets of Q-values, typically denoted as Q1 values and Q2 values.
2. Decoupling Action Selection and Value Estimation: Unlike traditional Q-learning, Double Q-learning decouples the selection of the action from the estimation of its value. This helps to reduce overestimation bias.
3. Alternating Updates: The algorithm alternates between updating Q1 and Q2, ensuring that each set of Q-values is used for action selection half of the time and updated with the other set's values.
4. Stabilizing Learning: By using one set of Q-values for action selection and the other for value estimation during updates, Double Q-learning provides a more stable and less biased estimate of the true Q-values.

Advantages:
1. Reduced Overestimation Bias: Double Q-learning addresses the issue of overestimation bias commonly observed in traditional Q-learning algorithms. This bias can lead to suboptimal policies and inaccurate value estimates.
2. More Stable Learning: Decoupling action selection and value estimation contributes to a more stable learning process, especially in environments with high variance or uncertainty in rewards.
3. Improved Policy Quality: The reduction in overestimation bias often results in improved policy quality, leading to better decision-making by the agent.

Disadvantages:
1. Increased Complexity: Double Q-learning is more complex than traditional Q-learning, requiring the maintenance and update of two sets of Q-values. This increased complexity may make implementation and understanding slightly more challenging.
2. Memory Requirements: Storing and updating two sets of Q-values increases memory requirements compared to algorithms with a single Q-table. This may be a concern in memory-constrained environments.
3. Not Always Necessary: While Double Q-learning addresses overestimation bias, it might not provide significant benefits in all scenarios. In some environments, the additional complexity may not be justified, and simpler methods like Q-learning or SARSA may suffice.

2. Show and discuss the results after:

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

Answer:

to the Cat quest environment. The results are,

```
Q_table = np.zeros((cat_quest.space.n, cat_quest.action_space.n))
inital_Q = Q_table.copy()
```

```
print(f"Intial Q table values:\n{inital_Q}\n{'*'*20}Trained Q table values{'*'*20}\n{Q_table}")
```

```
Intial Q table values:
[[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.]
 [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.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
********************Trained Q table values
```

```
********************Trained Q table values********************
[[299.48239047 299.90168193 299.9938556  299.55599654]
 [270.43670975 299.83899005 299.57670096 274.59213204]
 [ 99.75047785 289.18280217 191.93559184 151.70732476]
 [ 16.36747053 160.84694771  44.26629629  28.69511353]
 [ 25.03745989  17.42405974  14.97129596  32.13272133]
 [ 22.21988979  16.86784318  22.54875868  28.50969065]
 [275.0892695  299.56221926 299.85875817 279.32172381]
 [153.98945404 286.15956487 287.73959988 152.40215565]
 [ 47.46256874 211.27621973 168.74921632  55.05077777]
 [ 27.18781305  83.33321195  27.38257278  26.03884809]
 [ 32.61461256  27.60776204  18.48336576  35.07843885]
 [ 30.24313253  32.82256284  22.35027348  32.8983593 ]
 [156.96187919 223.56715222 294.13639459 116.52110126]
 [ 55.39681172 165.89748016 216.59381082  70.83616091]
 [ 18.88645473  87.10516205  97.08537503  21.47662351]
 [ 43.25857391  22.9077723   24.11998665  12.24232015]
 [ 38.0648576   36.03836588  30.77872218  25.47793194]
 [ 38.74925678  43.32795845  32.00520572  24.76935761]
 [ 77.42090688 109.44099805 129.92464974 145.98505973]
 [ 31.0178657   83.01328915  70.24939633  99.32106699]
 [  7.42048383  57.43791538  26.32437652  49.17107485]
 [ 25.33785326  13.0181155   22.1104226    8.94103054]
 [ 26.76812146  10.84675058  39.15783605  19.2432673 ]
 [ 23.34570661  31.72879912  33.41243235  20.35313356]
 [112.3276887  153.47498507 125.09563081 164.89026847]
 [ 63.42916118 129.17153529  69.30860232 134.93180585]
 [ 18.37489628  96.62660889  26.54308391  77.53329519]
 [  8.6510704   45.94479966   8.66703062  20.36334235]
 [ 19.20731728   5.84807674  24.997708    17.92000104]
 [ 21.3835942   14.2811813   19.7482387   21.12891213]
 [155.31855366 164.61299852 153.68815003 170.95775923]
 [116.34529607 161.57985045 113.48511728 157.79833403]
 [ 55.50123612 133.32596913  55.9412241   81.67694326]
 [ 19.63745357  75.16758859  15.00694759  27.40315708]
 [ 19.19761373  34.08926715  16.38080056  20.06404608]
 [ 19.50197853  18.5315612   19.52212864  20.50420696]]
```

Q_table is a matrix where the rows correspond to different states, and the columns correspond to different actions. The above Q table has 36 rows and 4 columns where the rows represent the states of our grid and the columns represent the four actions which are up, down, right, left.

```
Q_table[state,action] = Q_table[state,action]+ learning_rate * (
    reward + gamma * Q_table[state_, action_] - Q_table[state,action])
```

Each entry in the matrix (Q_table[state,action]*Q_table*[*state*,*action*]) represents the learned value of taking a specific action in a specific state.
The trained Q-table signifies the agent's capacity to differentiate between distinct states and actions, showcasing its proficiency in accurately predicting expected outcomes.

Graphs

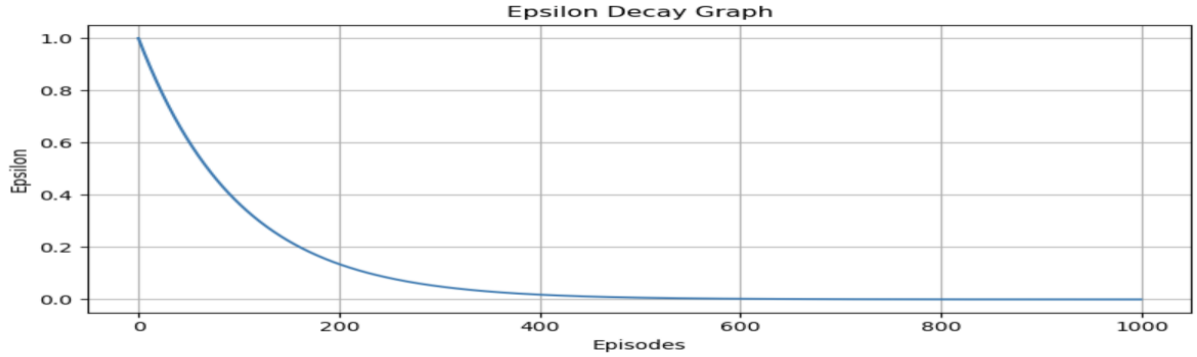
```
def graphs(total_episodes, SARSA_scores, epsilon_List):
    fig, axs = plt.subplots(2, 1, figsize=(8, 8))
    axs[0].plot(SARSA_scores)
    axs[0].set_title('Total Reward for the graph')
    axs[0].set_xlabel('Episodes')
    axs[0].set_ylabel('Scores')
    axs[0].grid()
    axs[1].plot(epsilon_List)
    axs[1].set_title('Epsilon Decay Graph')
    axs[1].set_xlabel('Episodes')
    axs[1].set_ylabel('Epsilon')
    axs[1].grid()
    plt.tight_layout()
    plt.show()
```

```
graphs(total_episodes, SARSA_scores, epsilon_List)
```

The above code helps us to plot the graphs for the Epsilon Decay and total Reward graph for the SARSA algorithm.

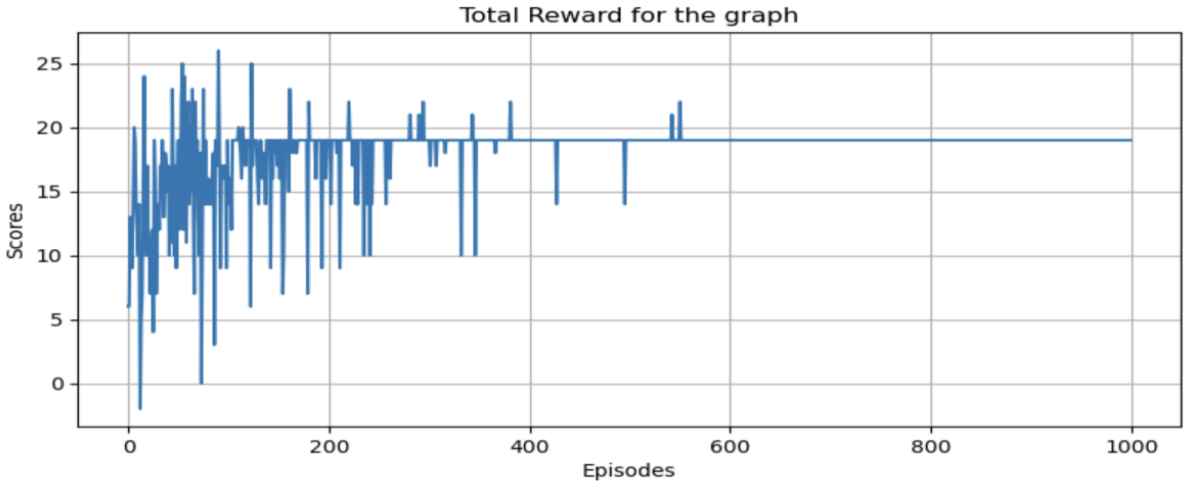
Epsilon Decay Graph:

The epsilon value exhibits a hyperbolic decay pattern, acting as the key parameter in balancing exploration (opting for a random action) and exploitation (choosing the best-known action) within the realm of reinforcement learning.

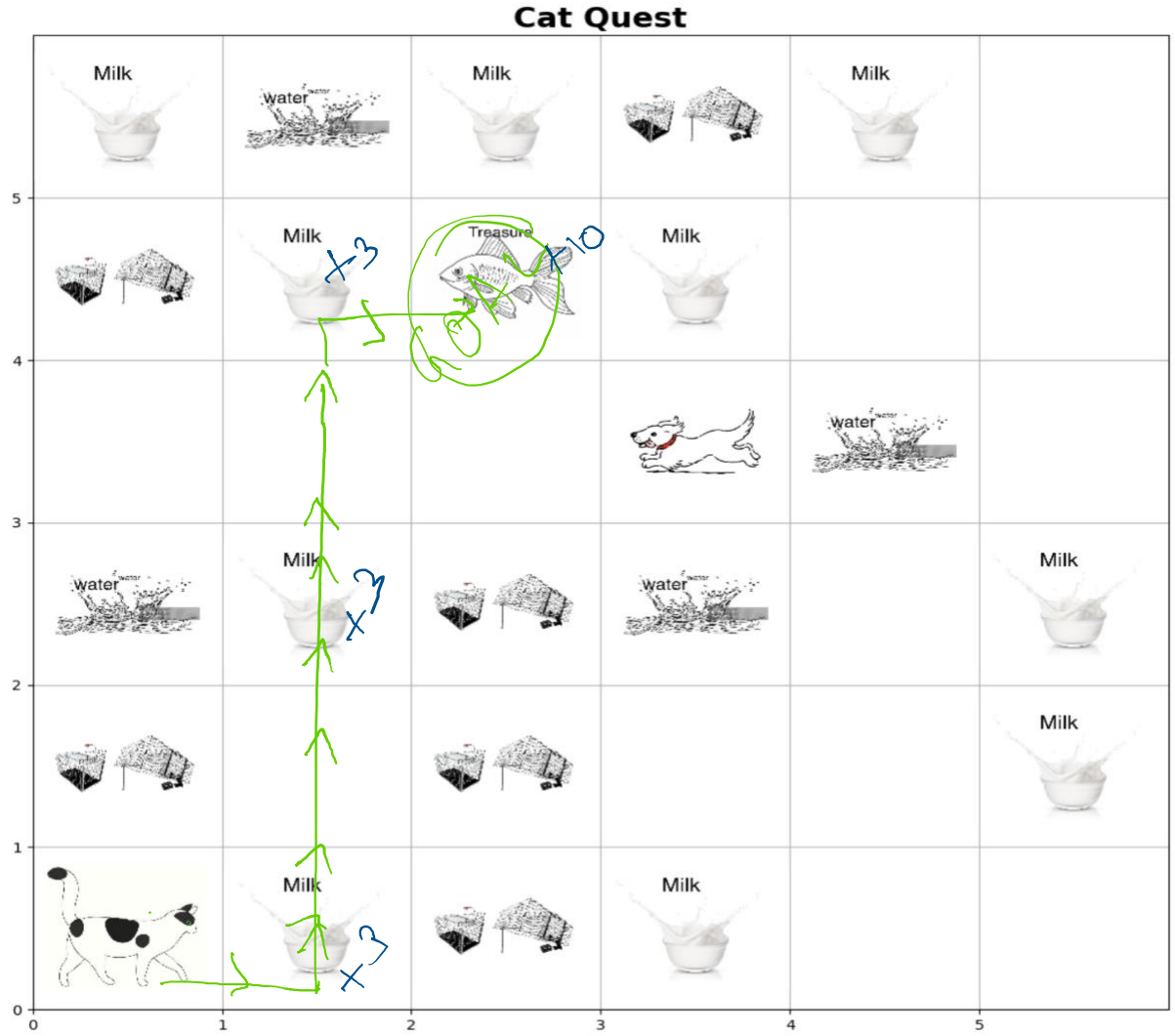


In this case, the calculated epsilon value is 0.995. Across 1000 episodes, epsilon gradually decreases according to the formula epsilon = max(epsilon * 0.995, 0.01), facilitating a shift from exploration to exploitation effectively—a crucial element in reinforcement learning.

Total Reward Graph:



The chart suggests that in the beginning, the agent tends to explore by taking random actions to understand the environment. As time progresses and the agent gains familiarity with the surroundings, it shifts towards exploitation, increasingly relying on the information stored in the Q-table to inform its decision-making process. Thus, which makes the agent to choose more optimal path by maximizing the result in the total number of max steps.
With the increase in the number of episodes, the reward increases with decreasing in the variance show casing that our agent cat is learning in a better way n navigating in the environment.
Below is the path chosen by the cat in order to reach the fish from the S1 to the treasure location S17. The path is as shown below:



The optimal Path of the cat is shown above: with a maximum reward of 19.
SARSA Base model required parameters as follows:
cat_quest = CatQuestEnvironment(env_type = 'deterministic', n = 36,max_steps = 100)
total_state = cat_quest.space.n

total_action = cat_quest.action_space.n
epsilon=0.01
total_episodes = 700
learning_rate = 0.1
gamma = 0.99
Isgreedy = False
Q_table = np.zeros((cat_quest.space.n, cat_quest.action_space.n))

• 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.
Answer:
In Double Q learning, we have two Q table. Initial and trained value of Q1 table is as shown below:

```
print(f"Intial Q1 table values:\n{inital_Q1}\n{'*'*20}Trained Q1 table values{'*'*20}\n{Q1_table}")
```

```
Intial Q1 table values:
[[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.]
 [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.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
********************Trained Q1 table values********************
[[-2.73340274e-01  2.81753344e+00  2.06273941e+00  6.87100000e+00]
 [-5.38015102e-02  3.48758157e+00  5.25094818e-01  1.28839088e+00]
 [ 9.73946647e-02  1.94579437e-01  3.06222855e-01  3.06310172e+00]
 [-2.20324268e-01  1.56416376e-01  2.79335442e-01  2.60604534e+00]
 [ 1.01812296e+00  1.31128929e-02  4.93536343e-02  4.64914580e+00]
 [-7.14581000e-03  4.08196764e-01 -4.76441000e-03  0.00000000e+00]
 [ 5.53000000e+00  9.63839205e-01  1.87913853e+00 -8.71749037e-01]
 [ 7.90000000e+00  1.32973507e+00 -7.82234395e-02 -6.00768059e-01]
 [ 7.00000000e+00  3.01458135e+00  4.00669330e-01 -5.46737999e-01]
 [ 1.00000000e+01  1.06506041e+00  2.74583446e-01  1.59504121e+00]
 [ 1.65551614e-01  4.32017610e+00 -2.91600876e-02  1.00000000e+01]
 [ 0.00000000e+00  1.87887427e+00  7.37208850e-01  3.00000000e-01]
```

```
 [-6.50406149e-01  6.11452551e-01  2.26067885e+00  1.32640129e+00]
 [-6.48997127e-01  3.00641876e-01  2.58537381e+00  1.10059280e-01]
 [ 3.07362351e+00 -1.32980186e-01  6.35382012e-02 -2.25393672e-01]
 [ 9.35389181e+00  1.51472087e-01  9.74591196e-01 -1.24297480e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  1.90000000e+00  0.00000000e+00  0.00000000e+00]
 [ 1.78606597e-01  3.76167038e-01  5.86411458e-01  2.18613104e-01]
 [-3.92781153e-01  4.76900362e-01 -1.26077573e-01  1.74493740e-01]
 [-8.15100000e-01  8.21345229e-02 -2.63359896e-01  2.23675891e-02]
 [ 1.32708700e+00 -1.16329952e-01  6.52378300e-01 -4.56415070e-01]
 [ 9.31000000e-03 -5.00000000e-01  5.21703100e+00  1.89000000e-02]
 [ 1.76890000e-02  1.09213000e+00  0.00000000e+00  0.00000000e+00]
 [ 2.59283539e-01  3.75848304e-02  3.67220005e-01  3.81269058e-02]
 [ 2.91225595e-02  3.83600059e-02  6.37371140e-03  1.07878894e+00]
 [-2.73923600e-01  0.00000000e+00 -2.21451898e-01  2.14461339e-01]
 [ 3.81026100e-02  0.00000000e+00  1.77740500e-01  1.60143209e-02]
 [ 1.02010061e+00 -3.31870000e-02  7.27387000e-01  0.00000000e+00]
 [ 0.00000000e+00  1.89000000e-02  0.00000000e+00  0.00000000e+00]
 [ 5.94700048e-01  8.00126548e-02  2.19629470e-02  8.40256432e-02]
 [ 5.35094066e-01  8.49921747e-02  2.35360784e-01  1.73614678e-01]
 [ 1.49056239e-01  5.69083539e-01  7.51424961e-02  7.39029653e-02]
 [ 0.00000000e+00  5.69865582e-01 -3.85073186e-01  3.59254628e-02]
 [ 0.00000000e+00  1.60143209e-02  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]
```

Initial and trained value of Q2 table is as shown below:

```
print(f"Intial Q2 table values:\n{inital_Q2}\n{'*'*20}Trained Q2 table values{'*'*20}\n{Q2_table}")
```

```
Intial Q2 table values:
[[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.]
 [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.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
********************Trained Q2 table values********************
[[-2.35647118e-01  2.16981844e+00  1.96964924e+00  6.87100000e+00]
 [-2.57101209e-01  3.07601339e+00  5.56987640e-01  1.23571181e+00]
 [ 2.23381197e-01  1.11114393e-01  3.39252150e-01  2.84391782e+00]
 [-5.10263496e-01 -4.00407140e-02  1.80318558e-01  4.51983720e+00]
 [ 4.66346362e-01  9.31943302e-02  1.73915781e-01  5.36790853e+00]
 [ 0.00000000e+00 -2.76023843e-02  1.42354800e-03 -1.00000000e-01]
 [ 5.53000000e+00  1.41302704e+00  2.44087032e+00 -7.05501631e-01]
 [ 7.90000000e+00  2.14609443e+00 -1.09345186e-01 -3.82444527e-01]
 [ 7.00000000e+00  2.78316556e+00  2.46884892e-01 -4.10876195e-01]
 [ 1.00000000e+01  2.02350071e+00  1.15081524e+00  4.44847708e+00]
 [-7.48530610e-02  2.92400561e+00  4.38158073e-01  1.00000000e+01]
 [ 0.00000000e+00  4.12995051e+00  0.00000000e+00  1.00884700e+00]
 [-3.23691558e-01  8.65060505e-01  1.32661162e+00  9.23217913e-01]
 [-7.39626915e-01  3.61533179e-01  2.71253880e+00  1.67136041e-01]
 [ 4.13291174e+00 -4.38388007e-02  2.05000624e-01 -9.10755990e-02]
 [ 8.33228183e+00  1.37072373e-01  0.00000000e+00 -1.95304034e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  1.90000000e+00  2.10000000e-02 -2.00000000e-01]
 [ 1.47670482e-01  3.06626544e-01  5.72622556e-01  1.15647416e-01]
 [-4.83725512e-01  4.51437735e-01  6.68885892e-02  2.01534394e-01]
 [-8.65528900e-01  1.08714968e-01 -5.77330696e-01  0.00000000e+00]
 [ 1.27095130e+00  0.00000000e+00  4.91805658e-01 -1.72900000e-01]
 [-2.00000000e-01 -4.24100000e-01  4.68559000e+00  0.00000000e+00]
 [ 0.00000000e+00  1.33000000e-01  0.00000000e+00  0.00000000e+00]
 [ 2.31116882e-01  3.46767946e-02  4.94046421e-01  1.33785585e-01]
 [ 4.33214481e-04  1.09861282e-02  2.84369631e-02  1.19208553e+00]
 [-2.71000000e-01  3.13873727e-01  1.81786806e-03  3.12186098e-01]
 [ 1.89000000e-02  0.00000000e+00 -9.00790000e-01  6.66481245e-02]
 [ 5.44323000e-01  0.00000000e+00  2.03387437e+00  0.00000000e+00]
 [ 0.00000000e+00  7.21215859e-02 -4.41580000e-01  0.00000000e+00]
 [ 6.90046290e-01  5.98030727e-02  2.53114108e-02  2.02302030e-02]
 [ 6.80774780e-01  2.09188289e-01  3.27849560e-01  3.48437608e-01]
 [ 1.53403620e-01  5.15515151e-01  2.32411178e-02  1.34983735e-01]
 [ 0.00000000e+00  3.16537973e-01 -1.00000000e-01  2.11318699e-02]
 [ 0.00000000e+00  3.75610245e-02  1.87504163e-01  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  3.00000000e-01  0.00000000e+00]]
```

Graphs

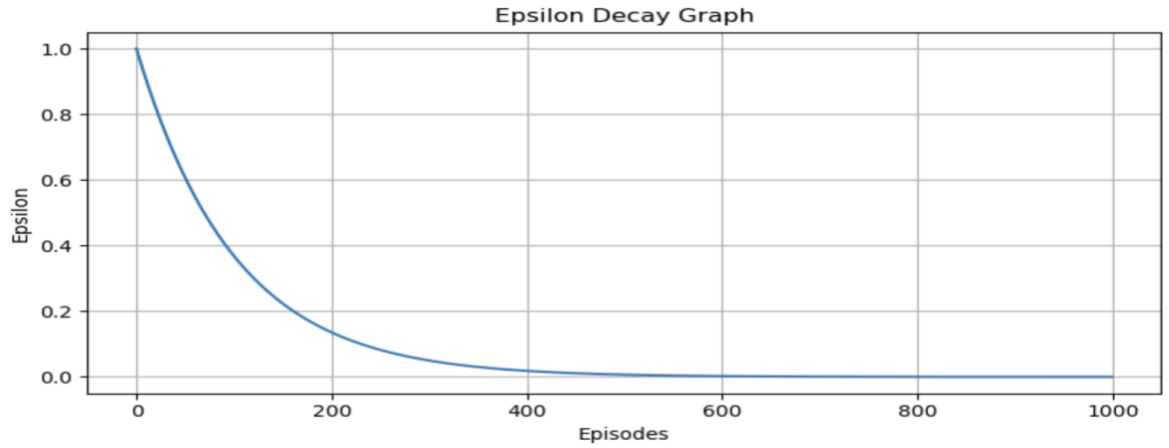
```
def graphs(total_episodes, SARSA_scores, epsilon_List):
    fig, axs = plt.subplots(2, 1, figsize=(8, 8))
    axs[0].plot(SARSA_scores)
    axs[0].set_title('Total Reward for the graph')
    axs[0].set_xlabel('Episodes')
    axs[0].set_ylabel('Scores')
    axs[0].grid()
    axs[1].plot(epsilon_List)
    axs[1].set_title('Epsilon Decay Graph')
    axs[1].set_xlabel('Episodes')
    axs[1].set_ylabel('Epsilon')
    axs[1].grid()
    plt.tight_layout()
    plt.show()
```

```
graphs(total_episodes, DQL_scores, epsilon_List)
```

The above code helps us to plot the graphs for the Epsilon Decay and total Reward graph for the Double Q Learning algorithm.

Epsilon Decay Graph:

The epsilon value exhibits a hyperbolic decay pattern, acting as the key parameter in balancing exploration (opting for a random action) and exploitation (choosing the best-known action) within the realm of reinforcement learning.

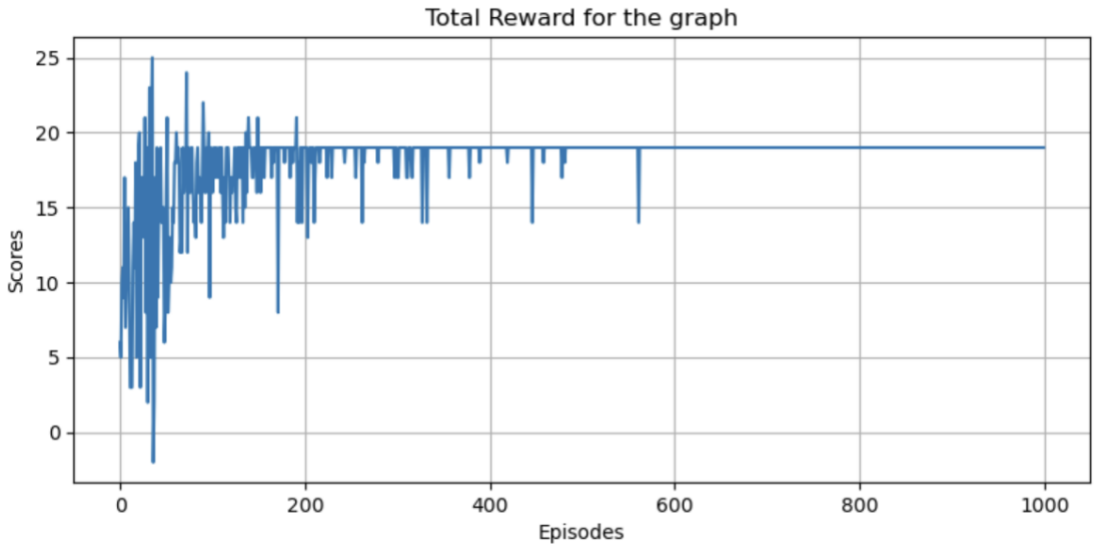


In this case, the calculated epsilon value is 0.995. Across 1000 episodes, epsilon gradually decreases according to the formula epsilon = max(epsilon * 0.995, 0.01), facilitating a shift from exploration to exploitation effectively—a crucial element in reinforcement learning.

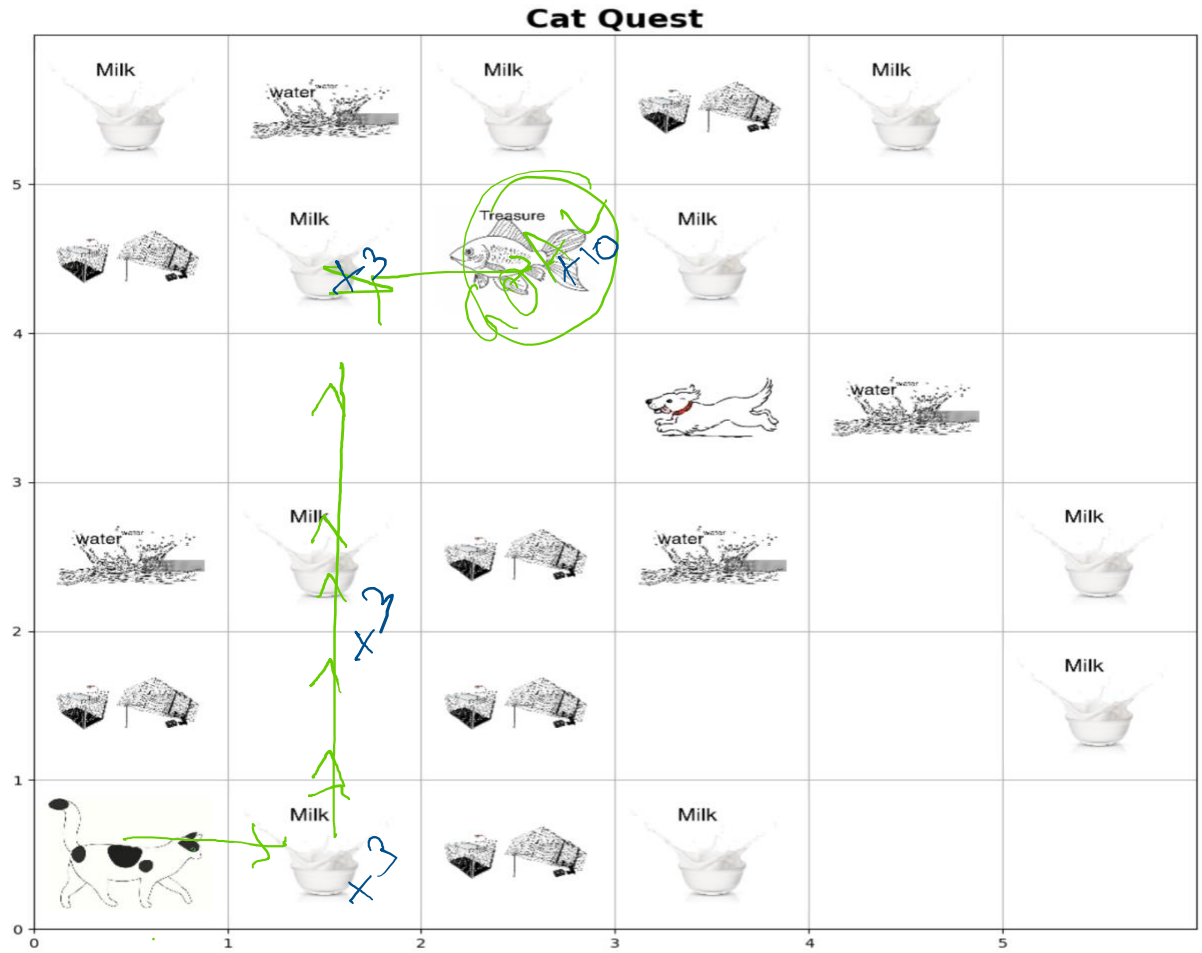
Total Reward Graph:

The chart suggests that in the beginning, the agent tends to explore by taking random actions to understand the environment. As time progresses and the agent gains familiarity with the surroundings, it shifts towards exploitation, increasingly relying on the information stored in the Q-tables to inform its decision-making process. Thus, which makes the agent to choose more optimal path by maximizing the result in the total number of max steps.

With the increase in the number of episodes, the reward increases with decreasing in the variance show casing that our agent cat is learning in a better way n navigating in the environment.



Below is the path chosen by the cat in order to reach the fish from the S1 to the treasure location S17. The path is as shown below:



The optimal Path of the cat is shown above: with a maximum reward of 19.

The cumulative reward graph observed during the training of Double Q-Learning exhibits a pattern of SARSA. Initially, there is notable variance in the rewards, with a tendency toward negative values, signifying the early learning phase of the agent. During this phase, the agent makes mistakes and subsequently learns from them. However, as episodes progress, there is a noticeable increase in cumulative rewards, accompanied by a reduction in variance. This trend suggests that the agent is adapting and developing a more effective strategy for navigating the environment.

Double Q Learning Base model required parameters as follows:
cat_quest = CatQuestEnvironment(env_type = 'deterministic', n = 36,max_steps = 100)
total_state = cat_quest.space.n
total_action = cat_quest.action_space.n
epsilon=0.01
total_episodes = 700
learning_rate = 0.1
gamma = 0.99
Isgreedy = False
Q1_table = np.zeros((cat_quest.space.n, cat_quest.action_space.n))
Q2_table = np.zeros((cat_quest.space.n, cat_quest.action_space.n))

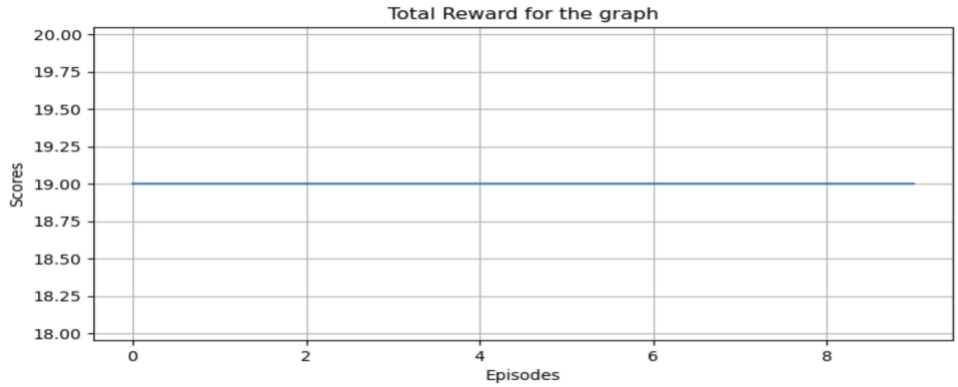
• Provide the evaluation results for both SARSA and Double Q-learning. Run your environment for at least 10 episodes, where the agent chooses only greedy actions from the learned policy. Plot should include the total reward per episode.

Greedy evaluation results for the SARSA algorithm:
In the environment by using the learned policy we tried to plot first 10 steps of the agent where the cat chooses only the greedy action.

Agent choosing only greedy actions from the learned policy SARSA Algorithm

```
Isgreedy = True
gamma = 0.99
total_episodes = 10
cat_quest = CatQuestEnvironment(env_type = 'deterministic', n = 36,max_steps = 100)
epsilon_List, SARSA_scores_greedy, Q_table = SARSA_algorithm(cat_quest,epsilon, 200, Q_table, Isgreedy, gamma, learn
```

```
graphs(total_episodes, SARSA_scores_greedy, epsilon_List)
```



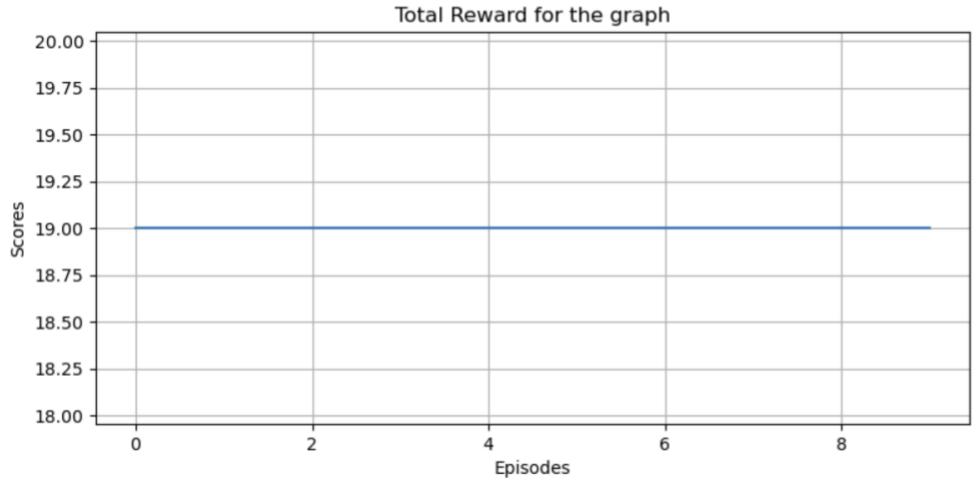
Here, we can see that the maximum reward for the first 10 episodes from the learned Q table is around 19 in SARSA algorithm.

Greedy evaluation results for the Double Q learning algorithm:
In the environment by using the learned policy we tried to plot first 10 steps of the agent where the cat chooses only the greedy action.

Agent choosing only greedy actions from the learned policy Double Q Learning

```
Isgreedy = True
gamma = 0.90
total_episodes = 10
cat_quest = CatQuestEnvironment(env_type = 'deterministic', n = 36,max_steps = 100)
epsilon_List, DQL_scores_greedy, Q1_table, Q2_table= DoubleQ_learning_algorithm(cat_quest,epsilon, 150, Q1_table, Q2
```

```
graphs(total_episodes, DQL_scores_greedy, epsilon_List)
```



Here, we can see that the maximum reward for the first 10 episodes from the learned Q1 and Q2 table is around 19 in Double Q learning algorithm.

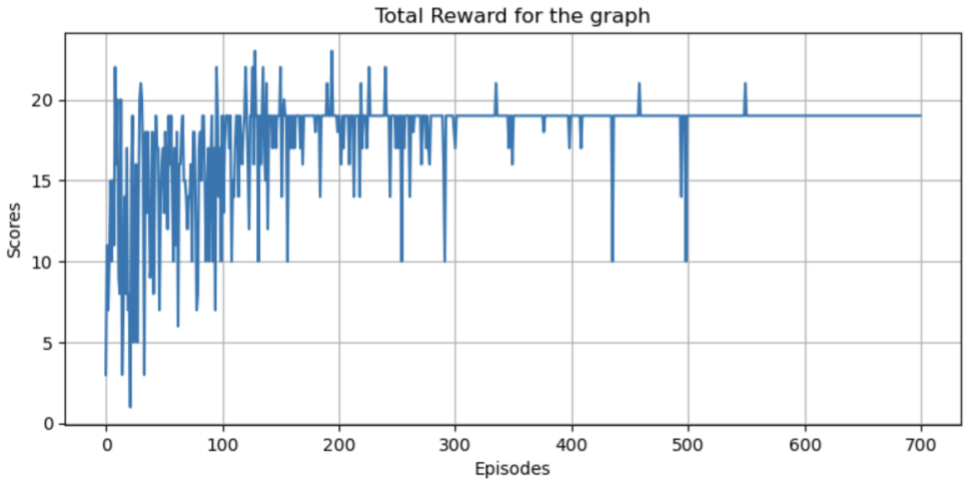
3. Provide the analysis after tuning at least two hyperparameters from the list above. Provide the reward graphs and your explanation for each of the results. In total, you should have at least 6 graphs for each implemented algorithm and your explanations. Make your suggestion on the most efficient .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

Answer:

Hyperparameter Tuning for SARSA:

Hyperparameter 1: Discount Factor => 0.40

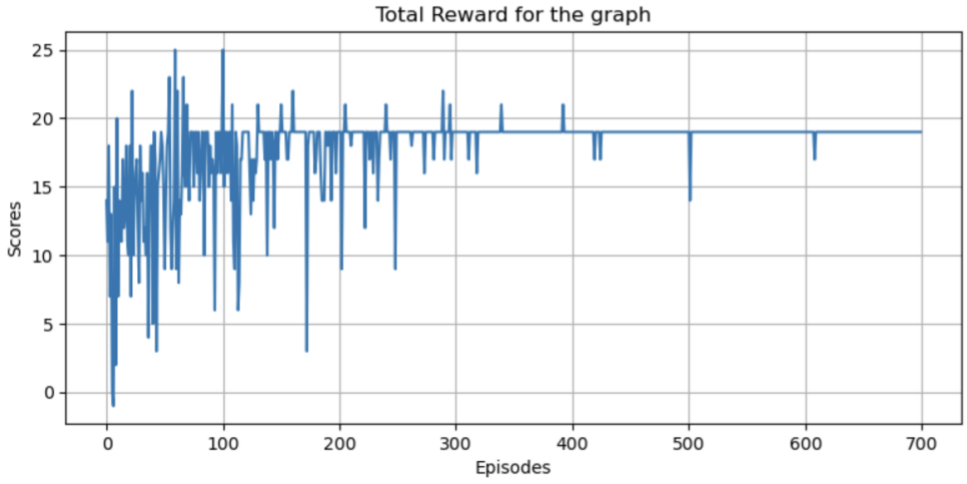
```
****************************** Evaluation Results : Discount Factor value is 0.4 ******************************
```



Above graph suggests that our algorithm used the discount factor that is the gamma value of 0.4. From the graph there are many fluctuations in reward values. But as the episodes increases the fluctuations decreased, thus creating a constant value of 19 that is the optimal cumulative score of the cat agent from the starting position to the end position. So, for our agent to make good decision for the discount factor 0.4 we need to increase more episodes to get optimal rewards and reach the destination.

Hyperparameter 2: Discount Factor => 0.65

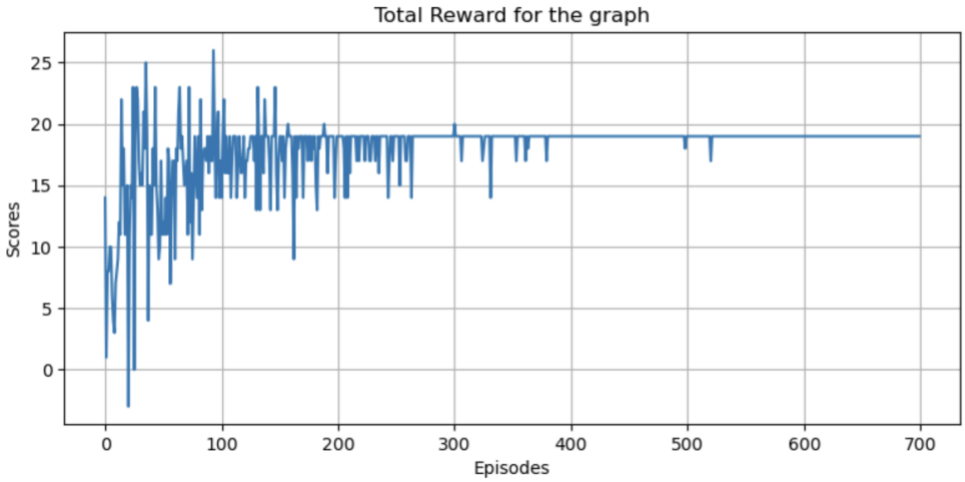
```
****************************** Evaluation Results : Discount Factor value is 0.65 ******************************
```



Above graph suggests that our algorithm used the discount factor that is the gamma value of 0.65. Compared to the gamma value of 0.4, our agent reaches the destination in less steps and more frequently with a constant value of 19 points. As the episodes increases, the fluctuations are decreased. Thus, our agent performs better compared to that of the previous discount factor value with more consistency.

Hyperparameter 3: Discount Factor => 0.80

```
****************************** Evaluation Results : Discount Factor value is 0.8 ******************************
```

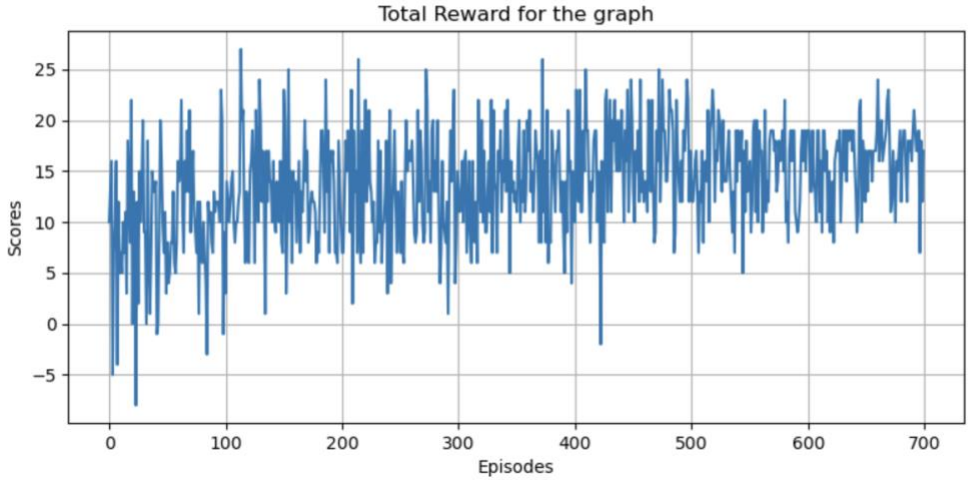


Above graph suggests that our algorithm used the discount factor that is the gamma value of 0.80. Compared to other two gamma values that is the discount factor, discount factor of 0.80 performs a lot better. Fluctuations are very less, and consistency of our agent performance is a lot better.

Suggestions: For the given model, the gamma (discount factor) significantly impacts the learning process. A gamma value of 0.99, being the highest and just below 1, proves to be optimal for reaching the optimal reward path. Comparatively, lower gamma values such as 0.4, 0.65, 0.80, and 0.9 may compromise long-term rewards. Therefore, for this specific model, maintaining a discount factor of 0.99 is recommended to ensure robust and effective learning.

Hyperparameter 4: Decay Rate => 0.001

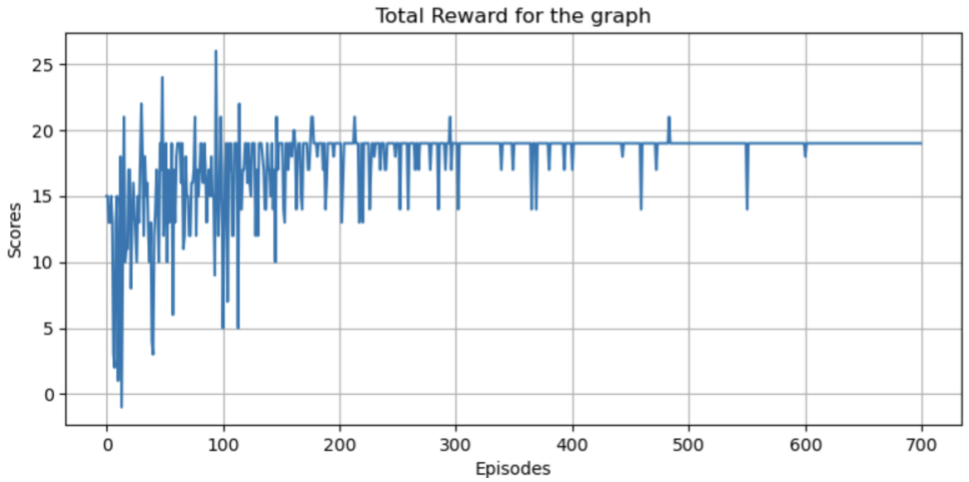
```
****************************** Evaluation Results : Decay Factor is 0.001 ******************************
```



Above graph shows that the total reward per episodes, when the decay factor 0.001. We can see that our agent had over 700 episodes, still the agent hasn't reached the optimal reward and something the agent reached the destination but not with the stable policy, so we need to run for more than 700 episodes in order to get the stable value. So we can say that the for the decay factor 0.001, we need more episodes thus increasing for training time. So, A decay factor of 0.001 will result in a slow reduction of exploration, likely leading to a gradual improvement in the score vs episode graph, with a smoother convergence over time.

Hyperparameter 5: Decay Rate => 0.009

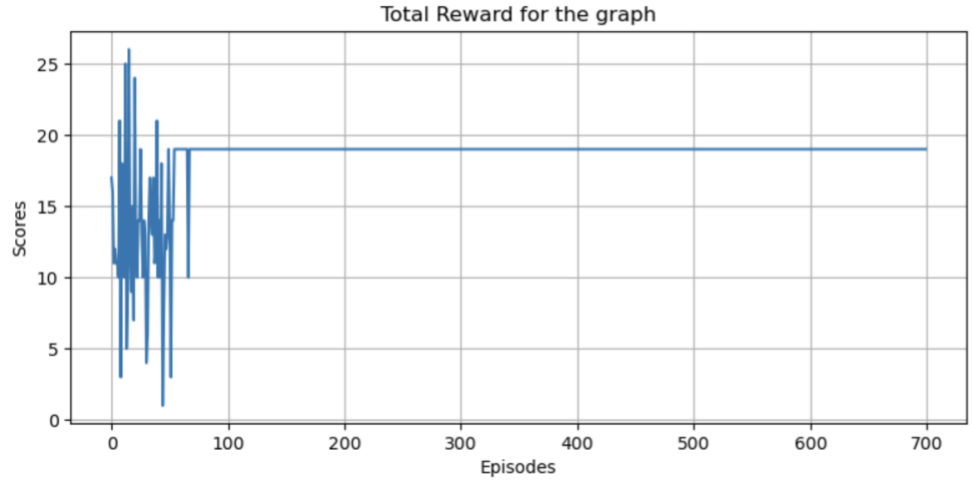
```
****************************** Evaluation Results : Decay Factor is 0.009 ******************************
```



Unlike the decay factor 0.001, 0.009 performs better compared to that, as there are few fluctuations at the beginning as the number of episodes increase, the policy is becoming more stable and consistent. With a decay factor of 0.009, there will be a moderate reduction in exploration, possibly balancing the trade-off between exploration and exploitation. The score vs episode graph may exhibit a more moderate convergence rate compared to a smaller decay factor.

Hyperparameter 6: Decay Rate => 0.005

```
****************************** Evaluation Results : Decay Factor is 0.05 ******************************
```

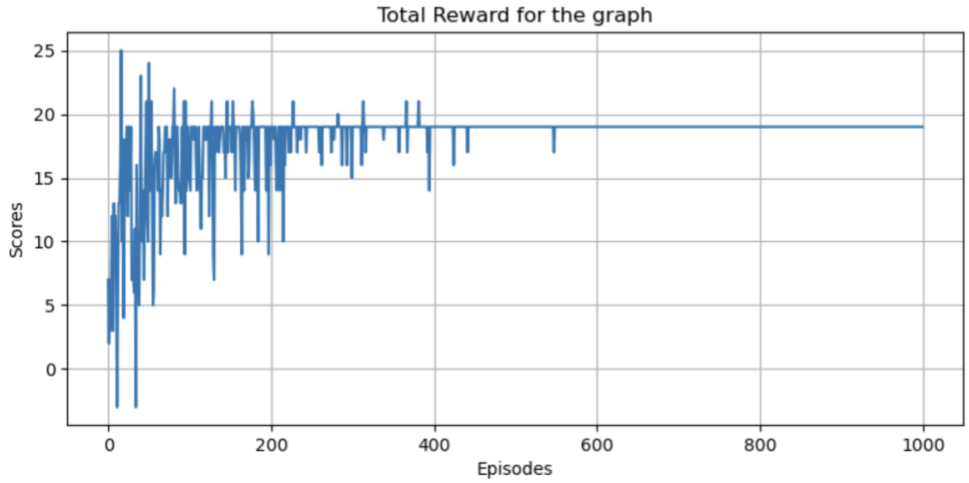


A larger decay factor of 0.05 will lead to a rapid reduction in exploration. While this may expedite initial convergence, there's a risk of prematurely settling into suboptimal policies. The score vs episode graph might show a quick but potentially less stable convergence. From the graph it is very clear that the agent has less fluctuations, and reached the optimal path, very rapidly.

Suggestion for SARSA hyperparameter tuning of Decay factor: Considering the exploration-exploitation trade-off, a decay factor of 0.01 appears to be the optimal choice for SARSA. It strikes a balance between gradual reduction in exploration and efficient convergence, stability in learning without risking settling into optimal policies. This moderate decay factor is recommended for achieving a robust and effective learning process in SARSA.

Hyperparameter Tuning for Double Q Learning:
Hyperparameter 1: Discount Factor => 0.40

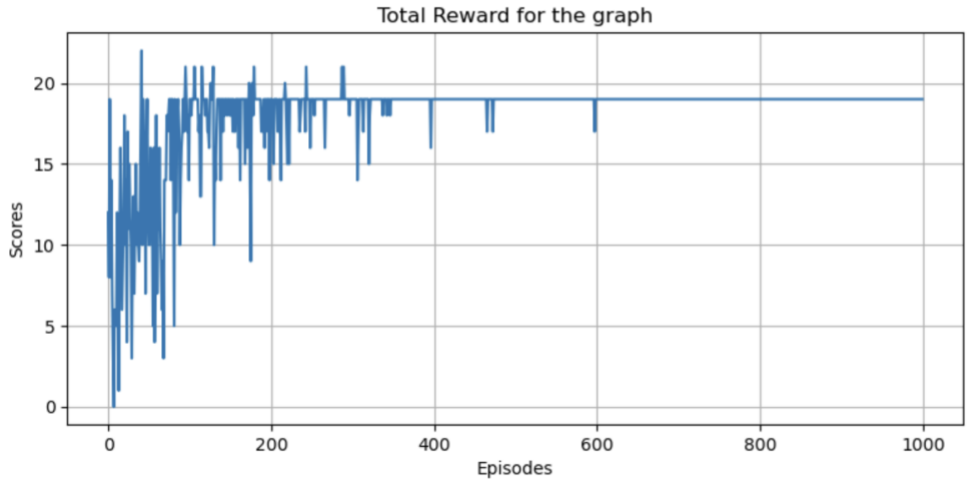
```
****************************** Evaluation Results : Discount Factor value is 0.4 ******************************
```



Above graph suggests that our algorithm used the discount factor that is the gamma value of 0.4. From the graph there are many fluctuations in reward values until the 600. After 300 episodes, the agent started reaching the optimal reward and reaching the destination As the episodes increases the fluctuations decreased, thus creating a constant value of 19 that is the optimal cumulative score of the cat agent from the starting position to the end position. So, for our agent to make good decision for the discount factor 0.4 we need to increase more episodes to get optimal rewards and reach the destination.
For the better comparison purpose, I have used the 0.4 gamma value, so it will be easy to compare the SARSA and Double Q learning. Compared to that of SARSA, Double Q learning algorithms helps to achieve the results in less episodes.
.

Hyperparameter 2: Discount Factor => 0.65

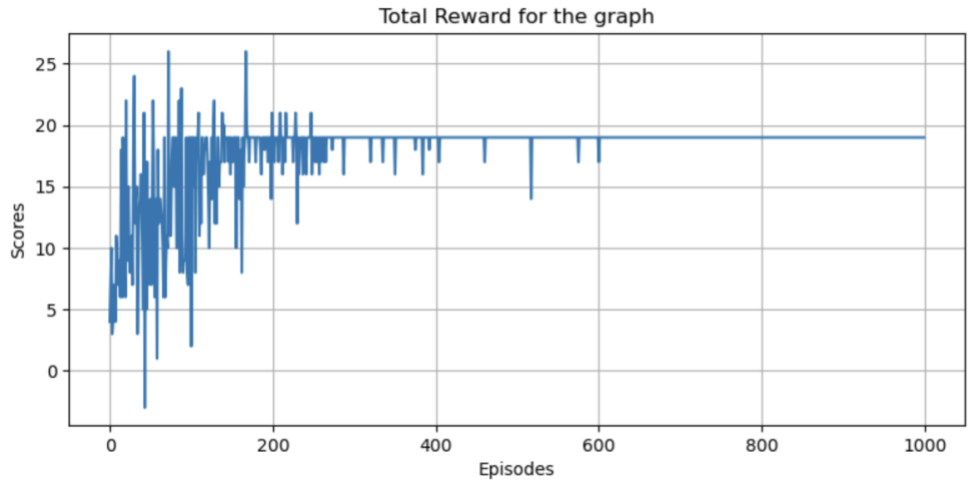
```
****************************** Evaluation Results : Discount Factor value is 0.65 ******************************
```



Above graph suggests that our algorithm used the discount factor that is the gamma value of 0.65. Compared to the gamma value of 0.4, our agent reaches the destination in less steps and more frequently with a constant value of 19 points. At the beginning the rewards were more than 20, but after the 100 episodes, the agent learned more and with the help of the double Q tables, the agent performs more better as the episodes increases, the fluctuations are decreased. Thus, our agent performs better compared to that of the previous discount factor value with more consistency.

Hyperparameter 3: Discount Factor => 0.80

```
****************************** Evaluation Results : Discount Factor value is 0.8 ******************************
```



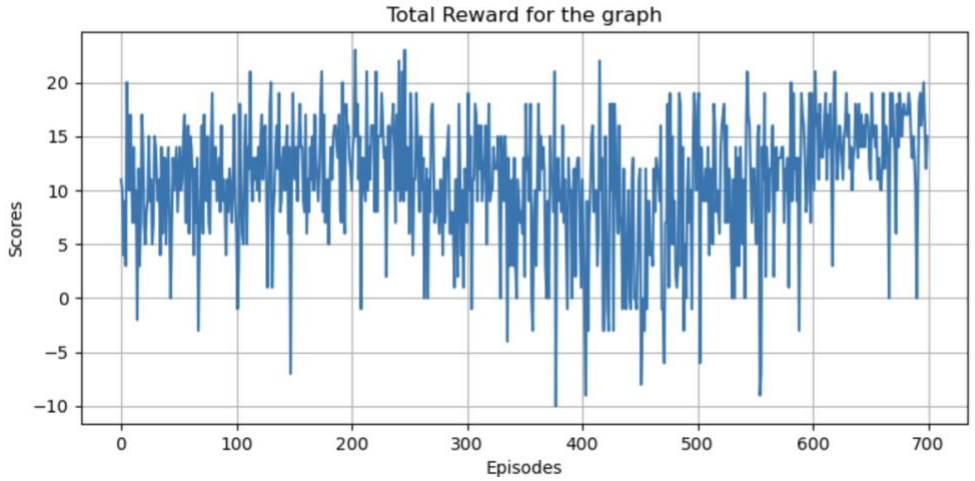
Above graph suggests that our algorithm used the discount factor that is the gamma value of 0.80. Compared to other two gamma values that is the discount factor, discount factor of 0.80 performs a lot better. Fluctuations are very less, and consistency of our agent performance is a lot better.

Suggestions: In both SARSA and Double Q Learning algorithms, a noticeable trend emerges as the gamma values and episodes increase: the Double Q Learning

consistently outperforms SARSA, showcasing incremental improvements in agent performance. Specifically, for the base model with a gamma of 0.99, the highest among considered values, it attains an optimal reward path surpassing all other gamma values. This consistent superiority suggests that a discount factor of 0.99 is the most effective choice, indicative of stable policies and exceptional results in both SARSA and Double Q Learning algorithms.

Hyperparameter 4: Decay Rate => 0.001

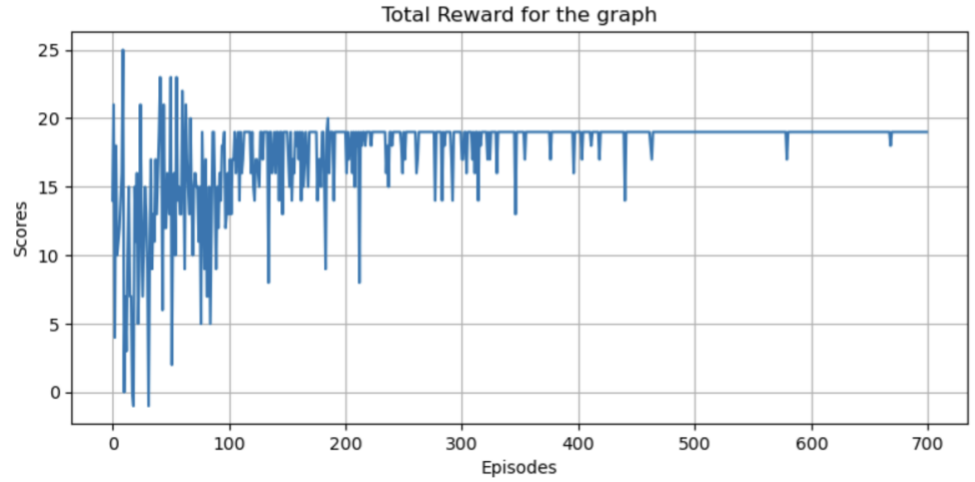
```
****************************** Evaluation Results : Decay Factor is 0.001 ******************************
```



The total reward per episode graph reveals that, with a decay factor of 0.001, the agent, after over 700 episodes, has not yet reached the optimal reward consistently. At times, the agent reaches the destination, but without a stable policy. This suggests that a decay factor of 0.001 requires more episodes for stable convergence, resulting in an extended training time. The slow reduction in exploration may lead to a gradual improvement in the score vs episode graph.

Hyperparameter 5: Decay Rate => 0.009

```
****************************** Evaluation Results : Decay Factor is 0.009 ******************************
```

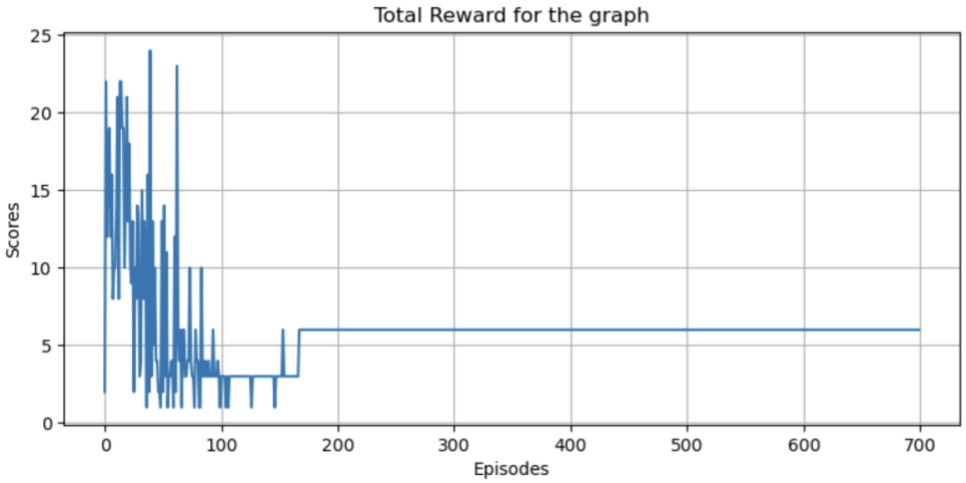


In contrast to the decay factor of 0.001, a decay factor of 0.009 exhibits better performance, with fewer fluctuations as episodes increase. The policy becomes more

stable and consistent over time. This moderate reduction in exploration strikes a balance, possibly enhancing the trade-off between exploration and exploitation. The score vs episode graph displays a more moderate convergence rate compared to a smaller decay factor.

Hyperparameter 6: Decay Rate => 0.005

```
****************************** Evaluation Results : Decay Factor is 0.05 ******************************
```



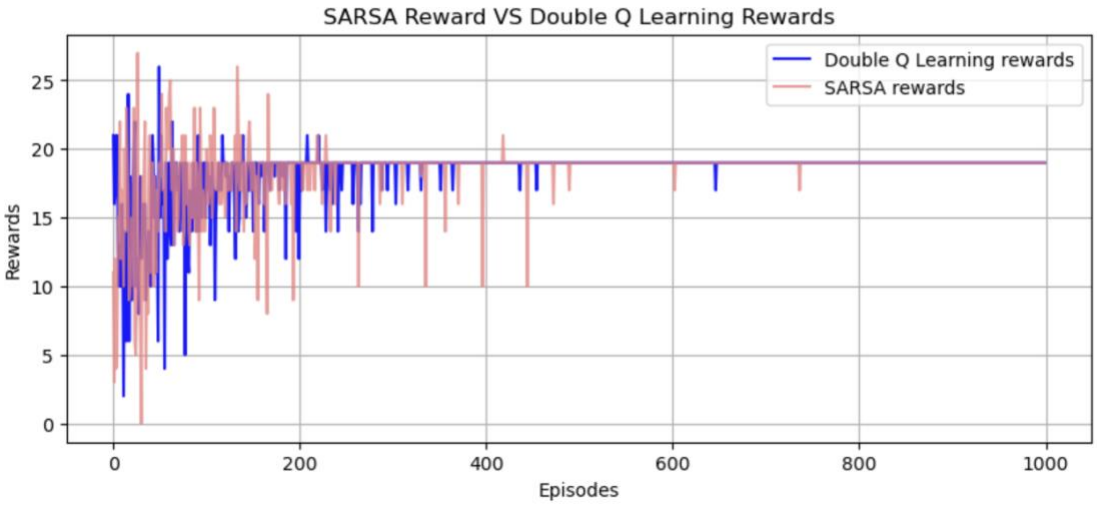
With a larger decay factor of 0.05, the agent experiences rapid reduction in exploration, leading to quick convergence. The graph shows fewer fluctuations, and the agent reaches the optimal path swiftly. There's a potential risk of prematurely settling into suboptimal policies, and the convergence may be less stable as shown in the graph. As the optimal values showing is 6 which is wrong. The trade-off between speed and stability is evident, as the score vs episode graph exhibits rapid but potentially less stable convergence.

Suggestion: Choosing a decay factor of 0.01 proves to be a balanced option for both SARSA and Double Q Learning, offering stability and effective convergence. While SARSA performs satisfactorily, Double Q Learning benefits from a moderate balance between exploration and exploitation. Alternative decay factors, such as 0.001, extend training time for smoother convergence, while 0.009 strikes a more effective balance with moderate convergence and enhanced stability. However, a larger decay factor of 0.05 accelerates convergence but raises concerns about premature settling into suboptimal policies, impacting stability. Tailoring the decay factor to the specific learning environment is crucial, considering factors like convergence speed, stability, and training time.

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.

Answer:

```
dynamicsRewardsGraph(DQL_scores, SARSA_scores)
```



The above graph is plotted to show the comparison of rewards collected by both algorithms SARSA vs Double Q-Learning during the training phase over 1000 episodes.

In the initial phase upto 200 episodes the agent explores the environment so the rewards fluctuate in both algorithms, but the rewards collected by SARSA at this phase is comparatively higher than DQ learning.

As the number of episodes increases, both the algorithms shift from exploration to exploitation phase nearly after 300 episodes. It can be observed at episode 500 that Sarsa gained more rewards at the initial phase whereas, DQ-learning gained stability and optimal path earlier than SARSA.

In the above graph, which is the SARSA rewards vs Double Q learning rewards per thousand episodes, it is clear that the agent reaches the destination very quickly with Double Q Learning compared to that of the SARSA. Initially, both algorithms undergo fluctuations in cumulative rewards, a common occurrence during the exploration phase. As the episode increases, the learning curves for both algorithms stabilize, signaling that each algorithm is converging toward a consistent and stable policy.

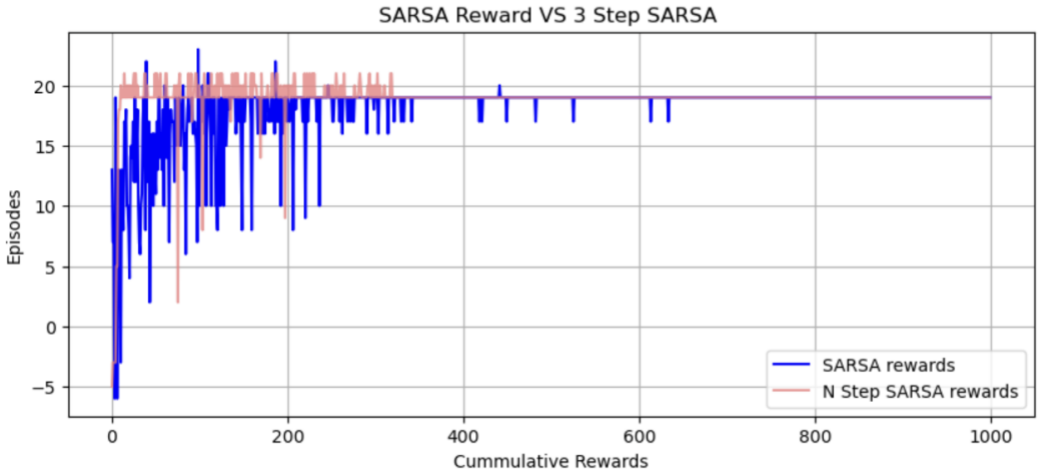
Bonus Task

n-step Bootstrapping

Answer: In this task, we have performed the 3 step bootstrapping SARSA
Below is the dynamic reward graph for the SARSA algorithm and 3 step bootstrapping SARSA algorithm:

```
: def dynamicsRewardsSGraph(SARSA_scores, SARSA_n_scores):
      plt.figure(figsize = (10, 4))
      plt.plot(SARSA_scores, color = 'blue', label = 'SARSA rewards')
      plt.plot(SARSA_n_scores,  color = 'lightcoral', alpha = 0.8, label = 'N Step SARSA rewards')
      plt.xlabel('Cummulative Rewards')
      plt.ylabel('Episodes')
      plt.title('SARSA Reward VS 3 Step SARSA ')
      plt.grid()
      plt.legend()
      plt.show()


  dynamicsRewardsSGraph(SARSA_scores,SARSA_n_scores)
```



Comparison: In 3-step bootstrapping SARSA, the agent incorporates information from three consecutive state-action pairs, allowing for a quicker learning convergence with a stable policy achieved in 300 episodes. In contrast, traditional SARSA relies on information from single state-action pairs, leading to a slower learning process, requiring 650 episodes to attain a stable policy. The increased temporal scope in 3-step bootstrapping enables more efficient learning and policy convergence.

Contribution Table

Team Member	Assignment Part	Contribution (%)
Yashwanth Chennu (ychennu)	1, 2, 3, Bonus, Report	**50%**
Amrutha Pullagummi (Amruthap)	1, 2, 3, Bonus, Report	**50%**