Assignment 1- Defining & Solving RL Environments

Name: Bhanu Chakra Sai Tarun Reddi **UB Number:** 50545060

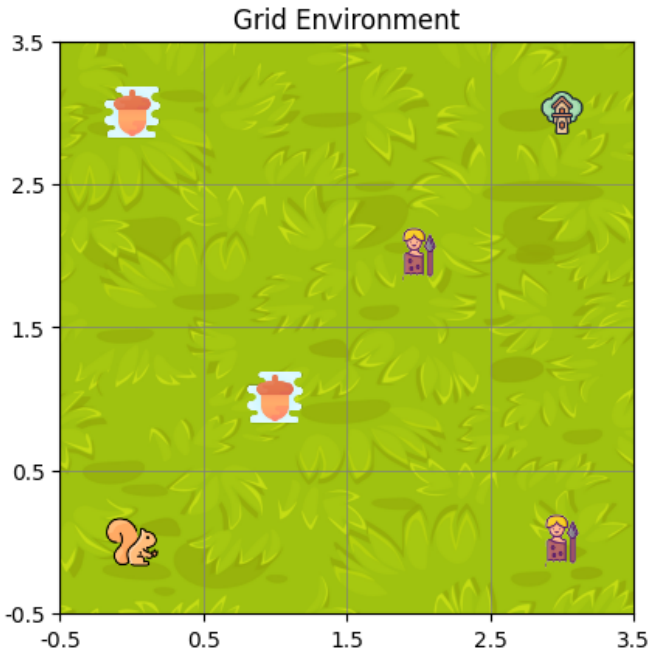
UB IT: bhanucha

This report provides insights into the first assignment, which focuses on defining and solving reinforcement learning (RL) environments. In this assignment, I outline the RL agents, examine the logic and functionality of our environment and agent, and employ tabular methods such as SARSA and Q-learning to train our agent to maximize rewards. This document is an initial submission for the checkpoint of assignment 1, detailing the tasks outlined in sections one and two.

PART 1 - Define an RL Environment

Squirrel Maze

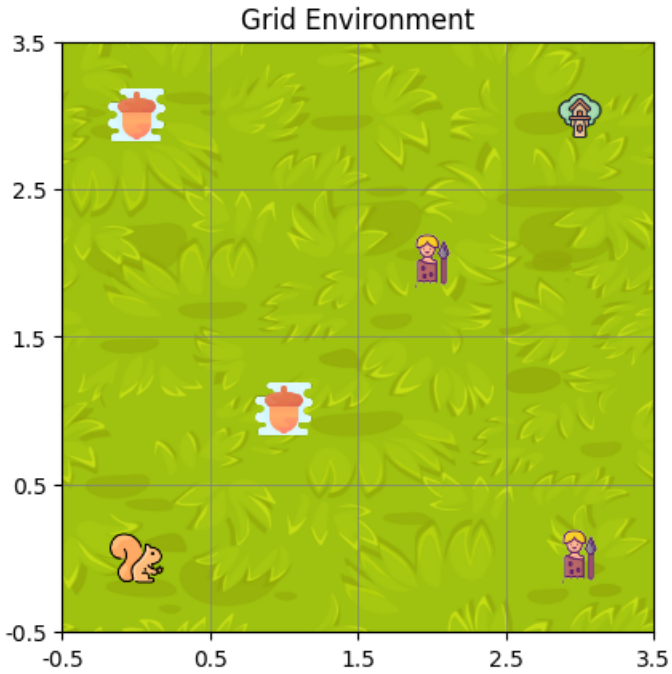
As a part of the assignment we are defining two environments one is Deterministic and the other is the stochastic focusing on a squirrel navigating through a grid to achieve goals while avoiding hunters. Both environments have the same state space, action space, and objective as shown in the figure. Here's a comparison between the deterministic and stochastic environments described:



Set of States:

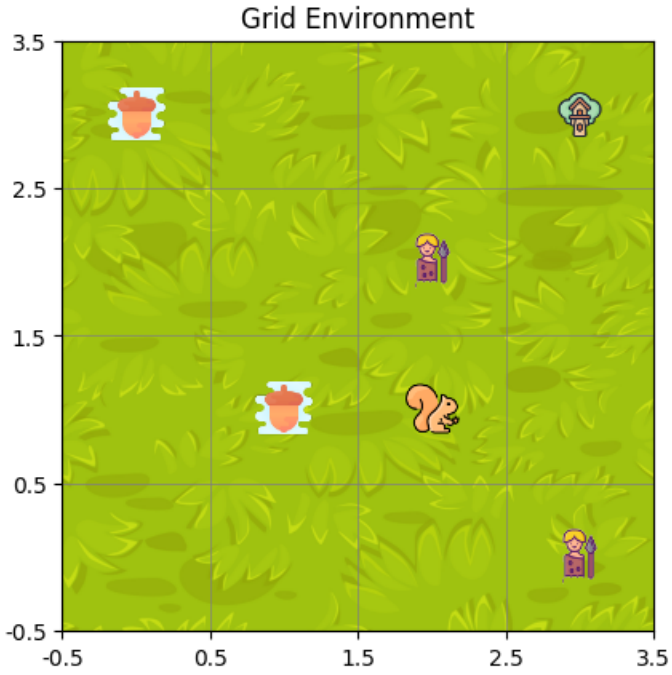
Here are the various states for the agent squirrel, described in bullet points:

- **Initial State:** The starting position of the squirrel.



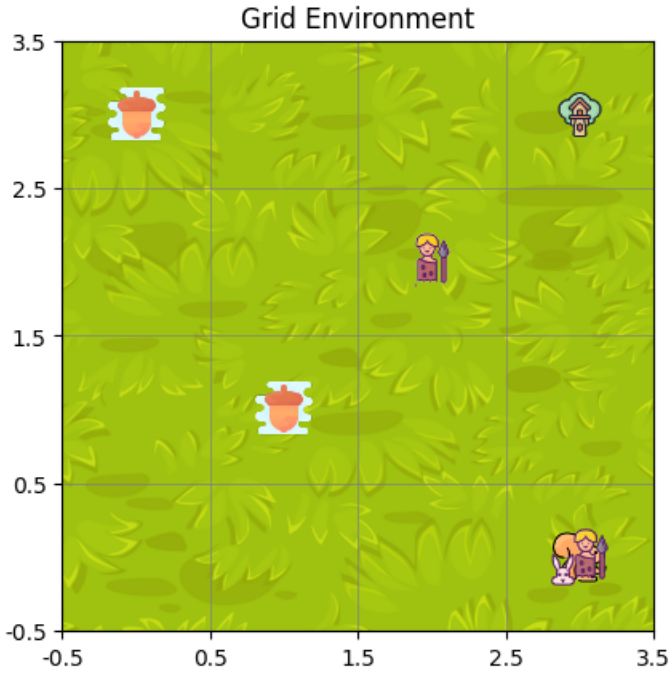
- **Going Out of the Grid State:** The squirrel tries to leave the environment, indicating an attempt to move beyond the grid boundaries.

- **Normal State:** The squirrel wanders around without receiving any reward, nor is it caught by the hunter or reaching home.

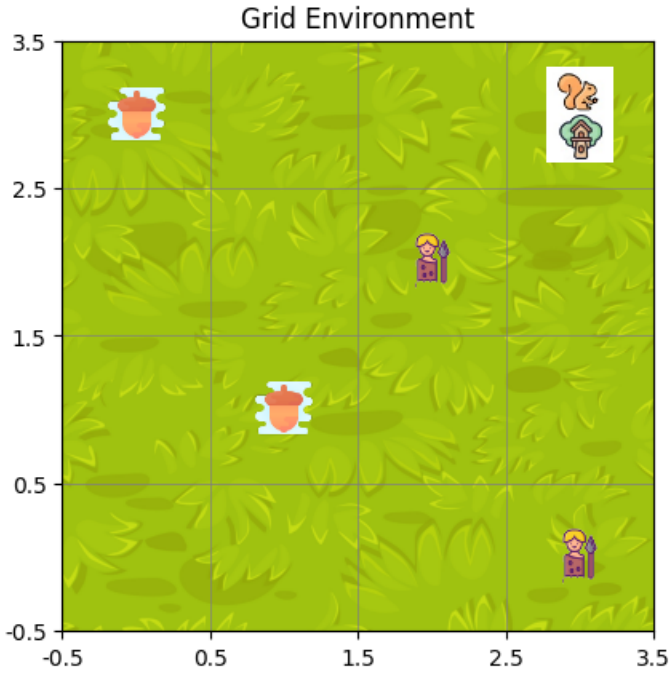


- **Acorn Pickup State:** The squirrel finds an acorn and consumes it, earning a reward for this action.

- **Encounter with Hunter State:** The squirrel is caught by the hunter, representing a negative outcome.



- **Goal State:** The final state where the squirrel successfully reaches its home, achieving its primary objective.



1.1 Deterministic Environment: `class SquirrelPet(gym.Env)`

Main Characteristics:

- **State Space:** A 4×4 grid with discrete states.
- **Action Space:** Four actions (left, right, down, up).
- **Rewards:** Collecting acorns grants rewards of 50 and 80, while encountering hunters incurs penalties of -20 and -50. Reaching the goal location grants a reward of 10. Each step towards the goal could potentially earn a reward of 10. However, the exact reward depends on whether each step objectively decreases the distance to the goal. [-10] for moving away to goal.
- **Objective:** Maximize reward by collecting acorns, avoiding hunters, and reaching the goal location.
- **Behavior:** Agent's movements are deterministic, meaning the outcome of an action is predictable and consistent.

Key Features:

- The environment utilizes fixed probabilities for action outcomes (100% chance of the action leading to the intended result).
- The agent has a set number of timesteps to achieve its objective, with the environment resetting after reaching the maximum timesteps or the goal.

1.2 Stochastic Environment: `class SquirrelPet_stoch(gym.Env)`

Main Characteristics:

- Similar state space, action space, and objective as the deterministic version.
- Introduces randomness in action outcomes, making it a stochastic environment.

Key Features:

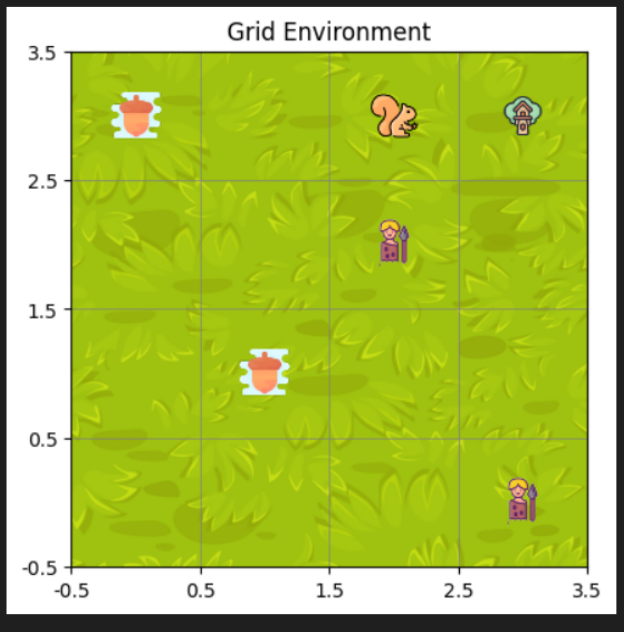
- **Action Probabilities:** The actual action taken by the agent is determined by a probability distribution, where each intended action has a 70% chance of being executed and a 10% chance of any other action occurring instead. This simulates real-world uncertainty where actions might not always lead to expected outcomes.
- **Randomness:** Incorporates uncertainty and unpredictability into the agent's navigation, requiring strategies that account for varied outcomes.

The Difference:

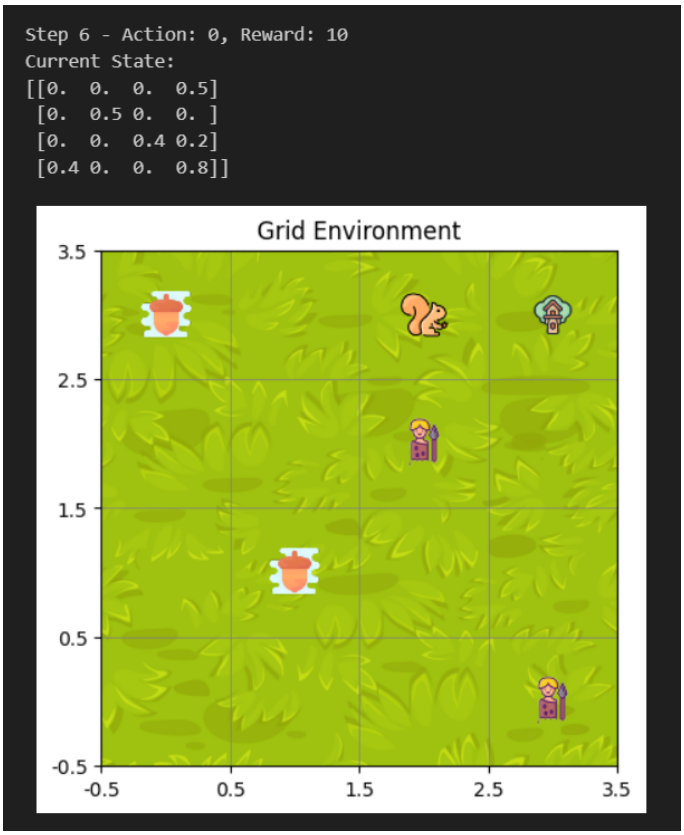
Let's delve into the distinction between deterministic and stochastic environments in our setup, highlighting the outcomes of specific actions. First, it's important to clarify the action labels used in our environment: '0' corresponds to moving right, '1' to moving left, '2' to moving up, and '3' to moving down.

In a **deterministic environment**, the outcome of any action taken by the agent is predictable and certain. For example, consider the action labeled '0', which instructs the agent to move right. As illustrated in the figure below, when the agent chooses action '0', it moves right, aligning perfectly with the expected outcome. If moving right brings the agent closer to its goal (home), it receives a reward of 10, acknowledging its progress towards the objective.

```
Step 6 - Action: 0, Reward: 10
Current State:
[[0.  0.  0.  0.5]
 [0.  0.5 0.  0. ]
 [0.  0.  0.4 0.2]
 [0.4 0.  0.  0.8]]
```



Contrastingly, the s**tochastic environment** introduces an element of unpredictability into the agent's actions. Although the agent may intend to perform a specific action, such as '0' to move right, the actual outcome is subject to randomness due to the environment's inherent uncertainty. In this scenario, each intended action has a 70% chance of being executed as planned and a 10% chance for any other action to occur instead.



For instance, as shown in the below figure, when the agent selects action '0' with the intention to move right, the stochastic nature of the environment might lead to a different outcome. Due to the introduced randomness, the agent might end up performing action '2', which results in it moving up instead of right. This deviation from the expected outcome illustrates the key characteristic of stochastic environments – the uncertainty and unpredictability in the actions' results.

Safety in AI

The `SquirrelPet` environment implements several measures to ensure safety and maintain defined operational boundaries within its state-space:

1. **Discrete Action and Observation Spaces:** The environment defines its observation space (`obs_space`) and action space (`action_space`) using discrete spaces, limiting the agent to a finite set of actions and states. This constraint ensures the agent's actions are predictable and manageable within the defined state-space.
2. **Boundary Checki:** The agent's movement is constrained within the grid boundaries by clipping its position (`np.clip(self.myagent, 0, 3)`). This prevents the agent from moving outside the defined grid, ensuring it navigates within the allowed state-space.
3. **Penalty and Reward System:** The environment has a system of rewards and penalties based on the agent's interactions with objects within the grid (e.g., acorns, hunters, goal location).

4. **State and Action Validation:** The environment checks if the agent remains stationary or attempts to revisit a previously visited state, prompting a change in action if necessary. This mechanism prevents the agent from engaging in futile or potentially unsafe actions.
5. **Termination and Truncation Conditions:** The environment implements conditions for termination (`terminated`) and truncation (`truncated`), ensuring the agent's episode ends either upon reaching the goal, encountering a penalty condition, or exceeding the maximum number of timesteps. This prevents the agent from continuing in potentially unsafe or undefined states.

These measures ensure that the agent operates safely within the environment, adhering to defined rules and boundaries, and making decisions that lead to safe outcomes.

PART 2 - Applying Tabular Methods

Q-Learning - Deterministic

Initial Setup and Parameters

- **Exploration Rate (Epsilon):** Begins at 1.0, allowing for full exploration, and decays to a minimum of 0.01 to encourage exploitation of learned behaviors.
- **Learning Rate (Alpha):** Set at 0.15, dictating the rate at which new information overrides old information.
- **Discount Factor (Gamma):** At 0.95, it influences the importance of future rewards.
- **Decay Rate:** Adjusted by 0.995 after each episode, gradually reducing epsilon to shift from exploration to exploitation.
- **Total Episodes:** The learning process spans 1000 episodes, with a maximum of 10 timesteps each.

Q-table after 1000 episodes:

```
Episode: 1000
Q-table:
[[702.78546141 735.38148183 759.33519153 734.90640895]
 [803.62897    728.31917969 670.51499993 694.96044495]
 [394.17456029 417.95738749 765.48491079 529.10554141]
 [  0.           0.           0.           0.        ]
 [360.19811203 431.26045707 800.34327107 348.24060199]
 [  0.           0.           0.           0.        ]
 [698.64932936 701.67548831 715.90739337 801.9729527 ]
 [ 93.24628619 759.4693669  305.49917348 307.36816022]
 [129.2931932  592.96311953 254.48080279 183.16562562]
 [361.80741793 798.11936106 537.19198502 309.55012467]
 [  0.           0.           0.           0.        ]
```

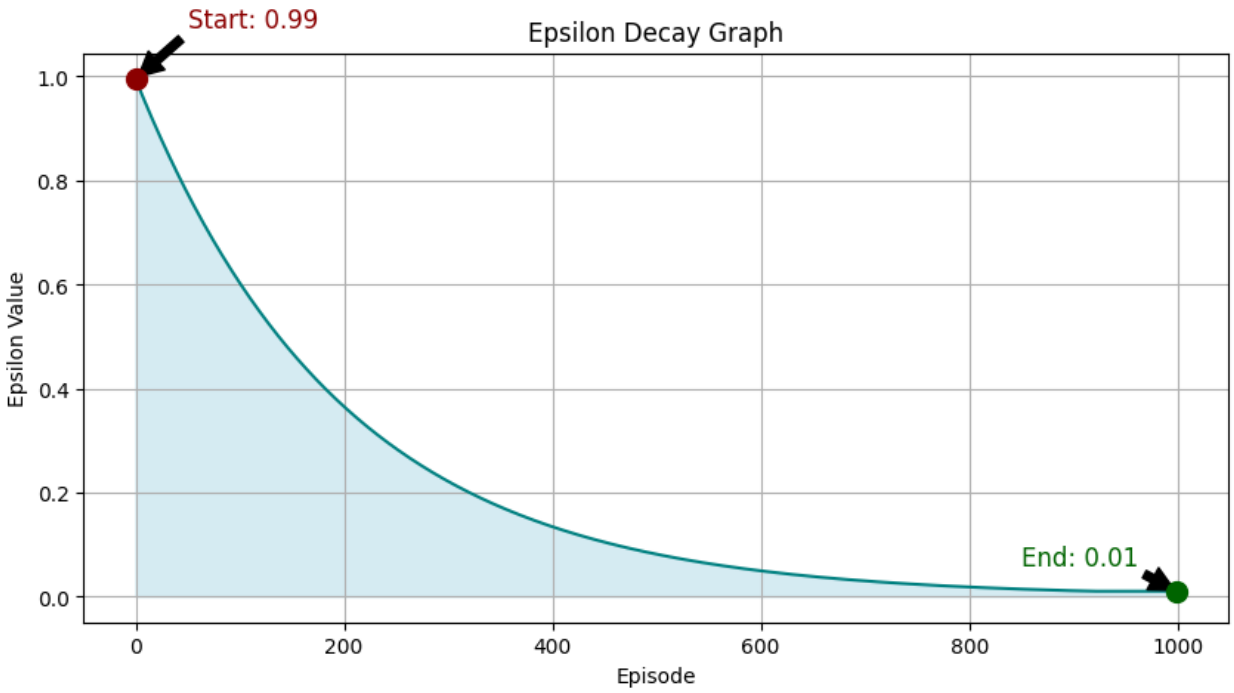
```
 [709.40974314 157.05852912 108.23966958  50.43468215]
 [  0.           0.           0.           0.        ]
 [  1.28848026 526.52970472  14.7211199   53.4404678 ]
 [  1.47124737 408.01027657  23.33481605  19.34677054]
 [  0.           0.           0.           0.        ]]
Average Penalties in Last 100 Episodes: 0.02
Episode: 1000, Average Steps: 9.98
```

Rewards/Episode Graph:

The graph "Total Reward The Agent Receive Per Episode" shows the performance of an agent over 1000 episodes. Initially, the agent's rewards vary greatly but show a clear trend of improvement as episodes continue. By the end of the training, the rewards level out at higher values, suggesting that the agent has learned an effective strategy. Occasional dips in rewards hint at some remaining challenges, possibly due to exploration or environmental randomness.

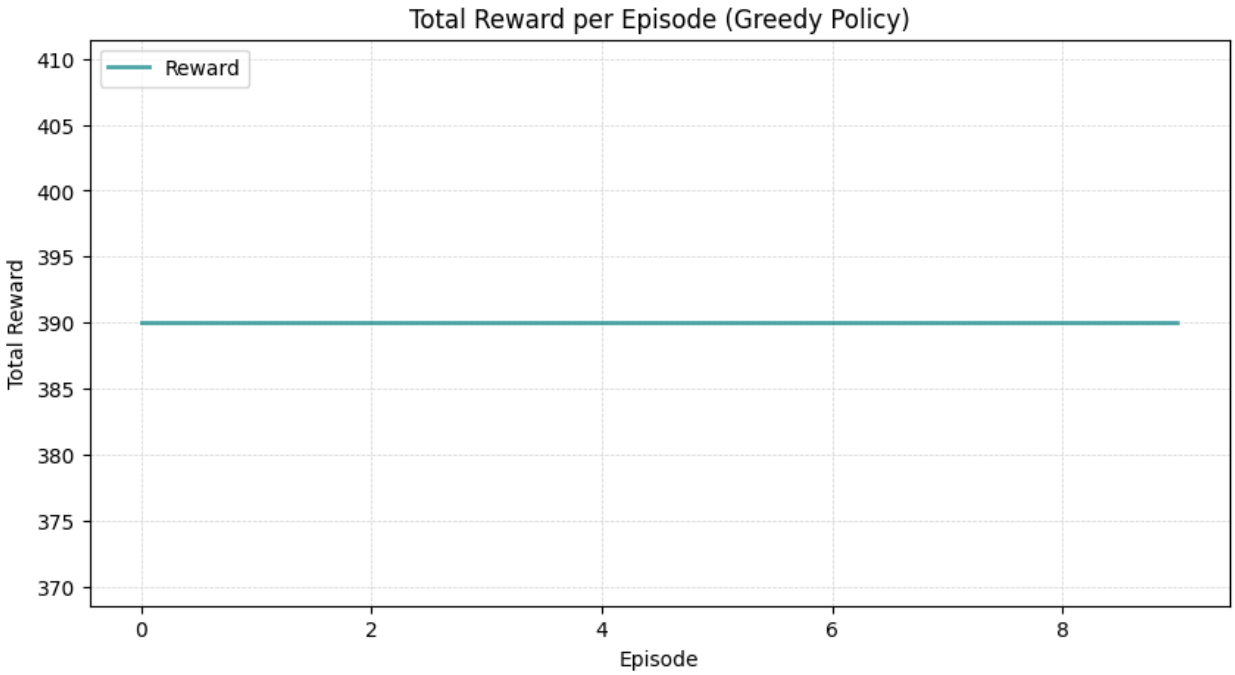
Epsilon Decay Graph:

Here the episilon decay graph It starts at a high value of 0.99, signifying a strong preference for exploration at the beginning of the learning process. As the number of episodes increases, epsilon gradually decays following a predefined rate, reaching a final low value of 0.01.



Total Rewards per episode graph

Here we ran the agent in environment for at 10 episodes, where the agent chooses only greedy actions from the learned policy. For an agent operating under a greedy policy, where it always chooses the action that it believes has the highest value based on its learned Q-table.



Experimenting with various hyperparameters:

Hyperparameter Variations:

- `max_timestamp_values`: Tested with 12, 15, and 20 to understand how longer episodes affect learning.
- `decay_val`: Examined decay values of 0.500, 0.750, and 0.995 to see how a slower or faster reduction in exploration impacts the agent's performance.

Outcomes:

- **Max Timestamp and Decay Rate at 12, 0.5:** The agent showed a relatively conservative learning curve, as indicated by the Q-table values. The rewards per episode were moderate, suggesting that the learning process may benefit from more exploration or longer episodes to acquire sufficient knowledge of the environment.
- **Max Timestamp and Decay Rate at 12, 0.75 and 12, 0.995:** Both scenarios produced higher rewards per episode, demonstrating that the agent was able to learn more effectively with a slower decay rate, allowing more exploration before settling into an exploitation strategy.
- **Max Timestamp at 15, Decay Rate at 0.5 and 0.75:** With more time per episode, the agent had additional opportunities to explore and learn from the environment. The resulting Q-tables and rewards per episode indicate improved learning outcomes.
- **Max Timestamp at 20, Decay Rate at 0.5, 0.75, and 0.995:** Extending the maximum timestamp further facilitated the agent's ability to learn, as shown by the increased rewards. Interestingly, the decay rate did not drastically alter the rewards, suggesting that the increase in maximum timesteps provided a substantial benefit by itself.

Key Insights:

- The increase in maximum timesteps generally led to better learning outcomes, as the agent had more opportunities within each episode to explore the state-action space.
- A slower decay rate of epsilon allowed for a more gradual shift from exploration to exploitation, which proved beneficial in environments where the agent initially had limited knowledge.
- Consistently high rewards across different decay rates, with an increased maximum timestamp, highlight the importance of allowing the agent sufficient time to interact with the environment for effective learning.

So, importance of tuning hyperparameters to balance exploration and exploitation in reinforcement learning and shows how extending the time allowed for each episode can significantly enhance the agent's ability to learn.

Base Model Evaluation

- **Trained Q-Table:** Shows significant learning with substantial values.
- **Reward:** Consistent and increasing, indicating effective learning.

Hyperparameter #1: Max Timestamp Values

- **12 Timesteps:**
  - **Reward:** Lower at 115, suggesting limited learning opportunities.

- **15 Timesteps:**
  - **Reward:** Increase to 465, indicating improved learning with more opportunities to act. Better than 12 but still room for improvement.
- **20 Timesteps:**
  - **Reward:** Further increased to 765, suggesting ample time for the agent to learn and collect rewards. The most efficient in this category, providing a balance between time and reward.

Hyperparameter #2: Decay Rate

- **0.5 Decay Rate:**
  - **Reward:** A consistent 115 across all max timestamp settings, which may indicate rapid loss of exploration and premature convergence.
- **0.75 Decay Rate:**
  - **Reward:** Consistent 465 across 12 and 15 max timestamps, with an increase to 670 at 20 max timestamps, indicating a better balance between exploration and exploitation. More effective than 0.5, allowing for more exploration.
- **0.995 Decay Rate:**
  - **Reward:** Consistently high at 765 across all max timestamp settings, indicating sustained exploration leading to a robust policy. The most effective decay rate, maintaining a balance between exploration and exploitation throughout learning.

Overall Efficient values

The most efficient hyperparameter values for this problem setup seem to be 20 max timestamps with a 0.995 decay rate. This combination allows the agent ample opportunity to explore and exploit the environment, leading to a higher and more stable reward. However, it's essential to consider the trade-offs between learning speed and robustness of the learned policy. Further fine-tuning and validation in different environment settings may be needed to confirm these suggestions.

Q-Learning - Stochastic

The analysis of Q-learning applied to the `SquirrelPet_stoch` environment with stochastic elements is as follows:

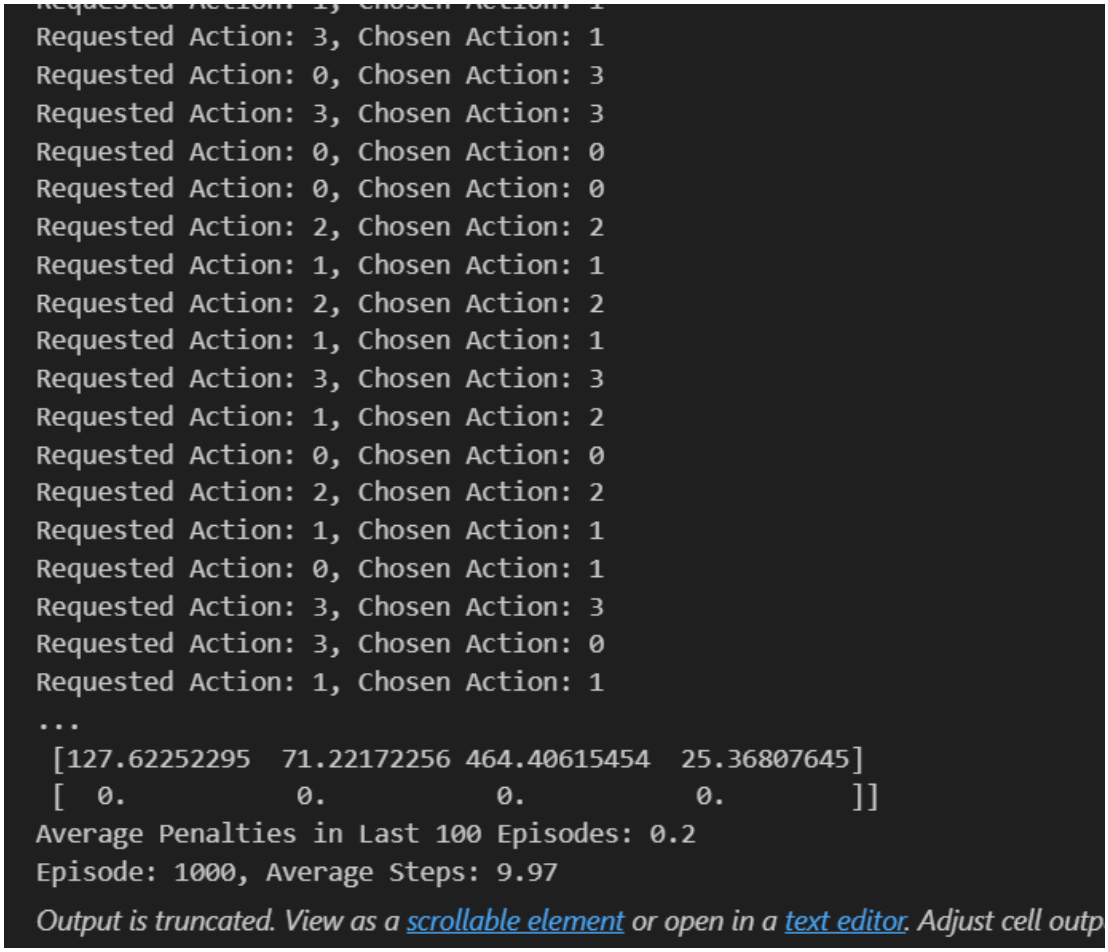
Initial Setup and Parameters

- The initial exploration rate (epsilon) starts at 1.0, allowing the agent to explore the environment fully initially.
- The minimum exploration rate is set at 0.01, ensuring that even at the end of learning, there is still a small chance of random actions for exploration.
- The discount factor (gamma) is 0.95, which emphasizes the importance of future rewards but not too far into the future.
- The learning rate (alpha) is 0.15, balancing between the new knowledge and old knowledge.

- The epsilon decay rate is set at 0.995, indicating a slow reduction in exploration as learning progresses.

Learning Process:

- Over 1000 episodes, the agent's ability to accumulate rewards and minimize penalties improves, as shown by the Q-table values and the average penalties.
- The agent's actions, when compared to the requested actions, show a degree of variability, which is expected in a stochastic environment. This implies that even with the correct policy, the outcome may not always be predictable, simulating real-world uncertainty.



Q-table after 1000 episodes:

```
Trained Q-table:
[[481.12347176 482.44245657 509.55412894 481.18577312]
 [583.52366303 489.9892792  488.2512628  442.35127078]
 [387.57068532 254.66866404 556.10805635 319.50275825]
 [  0.           0.           0.           0.        ]
 [349.92415802 407.40990848 563.03685045 430.65058569]
 [  0.           0.           0.           0.        ]
```

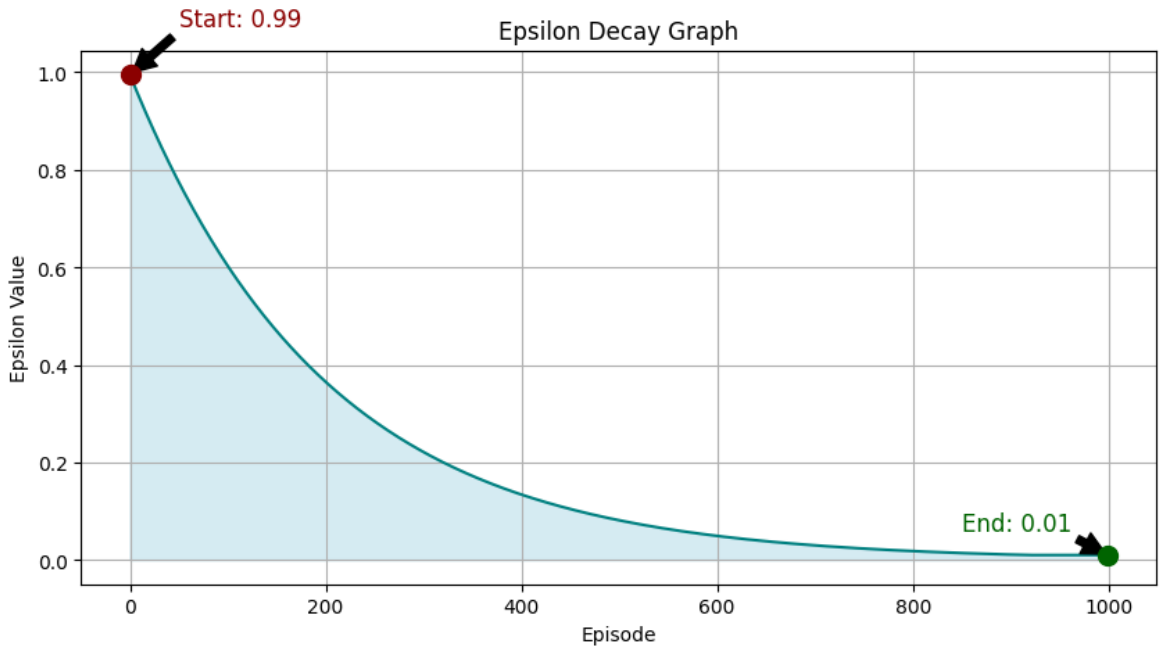
```
[461.86224667 461.20016042 468.76609861 548.4724644 ]
[119.13866406 541.29739567 278.6033686  301.15111247]
[163.42667306 503.65299535 169.41944791 228.55866233]
[212.95691342 519.77710979 260.96415249 327.99890918]
[  0.           0.           0.           0.        ]
[125.50918784 485.56316898  23.97898748 175.16160409]
[  0.           0.           0.           0.        ]
[ 26.01136036 153.12657016 418.13021717  60.84741138]
[127.62252295  71.22172256 464.40615454  25.36807645]
[  0.           0.           0.           0.        ]]
```

Rewards per Episode (Greedy Policy) Graph:

- This graph plots the total rewards received per episode when using a greedy policy.
- From the graph, it appears there is significant variability in rewards between episodes, indicating that the performance of the agent can fluctuate greatly from one episode to the next due to the introduction of stochastic environment.

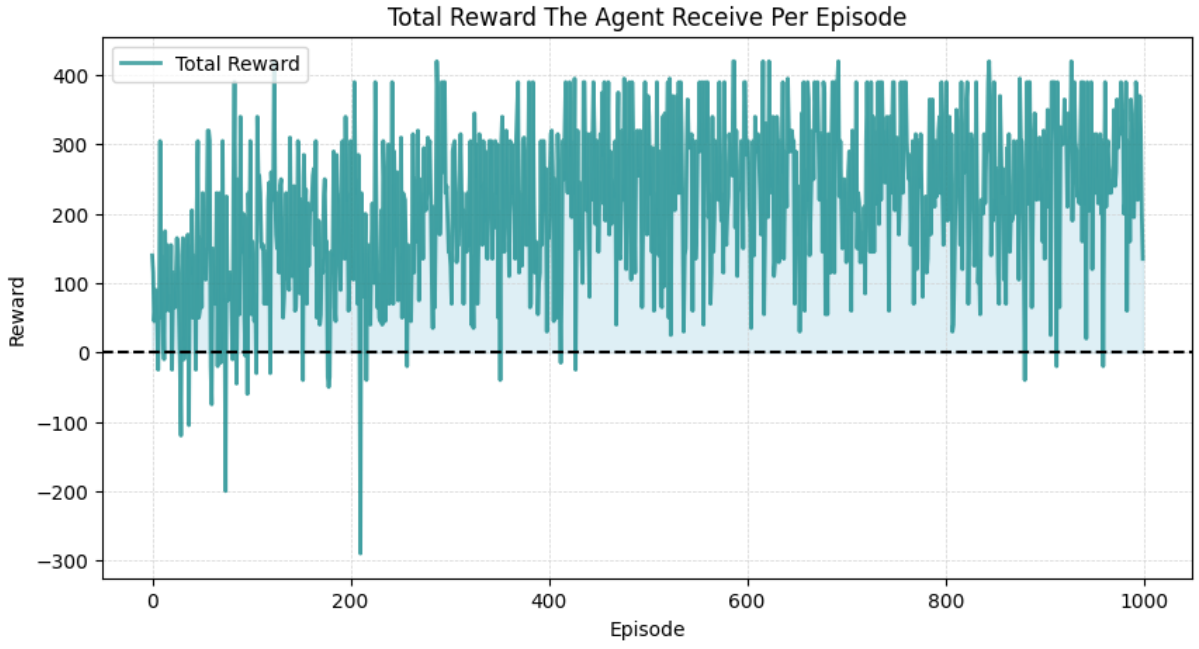
Epsilon Decay Graph:

- It starts at 0.99 (almost always choosing a random action) and decays to 0.01 (almost always choosing the best-known action), following an exponential decay curve. This suggests that as the agent learns more about the environment, it relies less on random exploration and more on its learned experiences.



Reward Episode Graph:

- This graph shows the total rewards received by the agent for each of 1000 episodes.
- It exhibits a pattern where initially the rewards vary greatly, with some dips below zero, indicating the agent might be exploring the environment and occasionally choosing suboptimal actions.
- Over time, the variability in rewards seems to decrease, and the overall trend appears to be stabilizing. However, there are still fluctuations, suggesting the agent continues to explore to some extent, which is typical behavior in a learning phase to avoid local maxima.



Tuning Hyperparameters

To evaluate the results of the Q-learning agent's performance with different hyperparameter values, we will consider the following aspects:

1st Variation:

- **12, 0.5**: The Q-table shows moderate learned values, and the reward is 300. A faster decay (0.5) may have led the agent to exploit early without sufficient exploration.
- **12, 0.75**: Slightly higher Q-values are learned, with a reward of 335. A slower decay rate has potentially allowed for better exploration before exploitation.
- **12, 0.995**: This shows high Q-values with a reward of 465. A very slow decay rate likely allowed for extensive exploration, leading to a better-informed policy.

2nd Variation:

- **15, 0.5**: The Q-values are lower with a corresponding reward of 35, indicating that the agent might be converging too quickly to a suboptimal policy.
- **15, 0.75**: The Q-values and reward (455) are higher, suggesting a balance between exploration and exploitation.
- **15, 0.995**: The highest Q-values and reward (535) suggest that the agent had ample opportunity to explore and has likely learned a good policy.

3rd Variation:

- **20, 0.5**: Lower Q-values and a negative reward of -40 suggest that the agent is not learning effectively, potentially due to too rapid convergence to a suboptimal policy.
- **20, 0.75**: The highest reward of 575 with considerable Q-values suggests an effective balance between exploration and exploitation.
- **20, 0.995**: Similar to the 15, 0.995 scenario, high Q-values and a reward of 465 indicate a well-explored policy, but not as high as the 20, 0.75 scenario.

Overall Efficient values

- The combination of **max_timestamp = 20** and **decay_rate = 0.75** produced the highest reward, which suggests that it's the most efficient set of hyperparameters among those tested. This setup likely provided a good balance between exploration and exploitation, allowing the agent to learn effectively over time.

Double Q-Learning - Deterministic

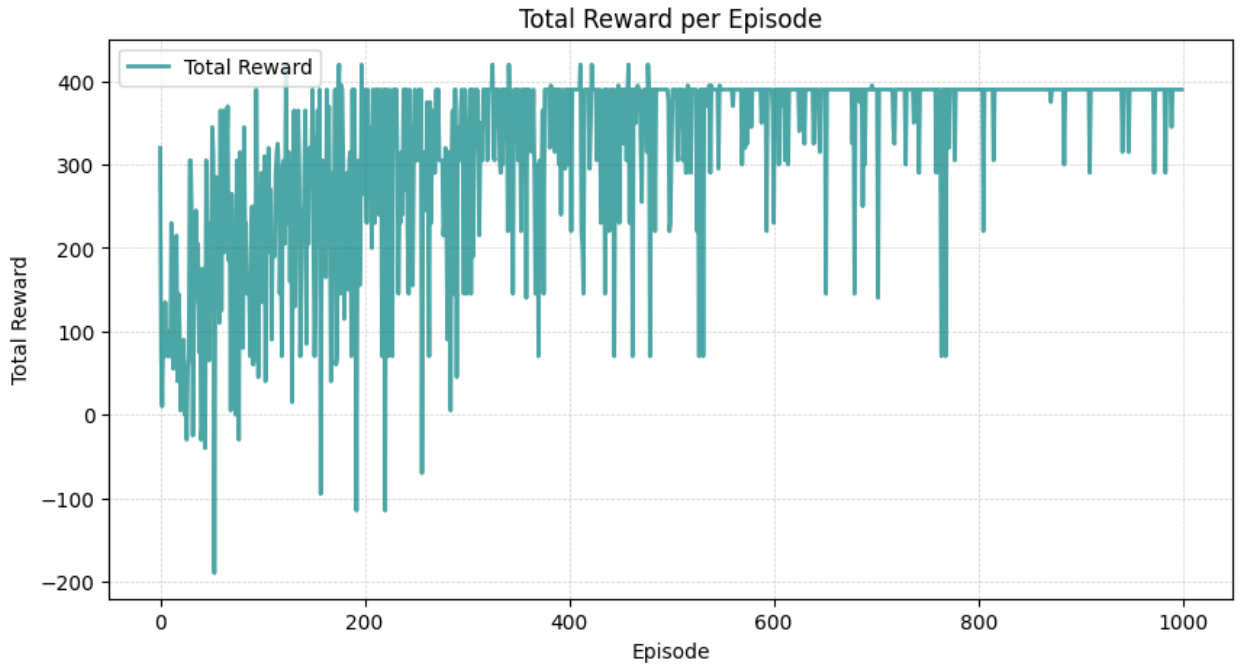
Q-table after 1000 episodes:

```
Trained Q-table 1:
[[ 5.94336730e+02  7.27681493e+02  7.58497860e+02  6.64276585e+02]
 [ 8.02266463e+02  5.58418720e+02  5.31337980e+02  5.84087011e+02]
 [ 3.10388603e+02  1.62379631e+02  7.31896063e+02  3.45962596e+02]
```

```
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 1.12625657e+02  1.48406094e+02  7.82928450e+02  1.50767497e+02]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 6.59685721e+02  6.45859724e+02  5.96202795e+02  8.01351973e+02]
 [ 4.34489536e+01  7.06281368e+02  1.39643088e+02  2.91717288e+02]
 [ 4.73655065e+01  2.26568498e+02  1.73220842e+01 -7.50000000e-01]
 [ 4.63453563e+01  7.16061535e+02  2.91715636e+01  2.04009261e+01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 6.53936197e+02  1.16912304e+00  4.44430199e+01  1.08211994e+02]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 2.37077580e+01  5.64836356e+00  1.50000000e+00  1.89343906e+02]
 [ 0.00000000e+00  0.00000000e+00  1.36216309e+02 -2.09890078e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]
Trained Q-table 2:
[[ 6.35187353e+02  7.27714244e+02  7.58443551e+02  6.92557469e+02]
 [ 8.01060638e+02  4.50721182e+02  4.82377152e+02  5.74043226e+02]
 [ 1.20237807e+02  1.60862122e+02  7.45560871e+02  2.56634644e+02]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 1.00485930e+02  2.71481777e+02  7.48671018e+02  2.26507741e+02]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 6.80930775e+02  6.42264658e+02  5.33232515e+02  8.00662262e+02]
 [ 3.85875000e+00  6.97711690e+02  1.02378410e+02  1.00492947e+02]
 [ 0.00000000e+00  2.50212386e+02  2.07437365e+01  0.00000000e+00]
 [ 2.40766222e+01  7.39878948e+02  2.47482021e+02  5.81229647e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 6.20114016e+02  6.31017210e+01 -1.38750000e+00  1.71190275e+01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-5.36250000e-01  3.29374132e+01  1.01564626e+01  1.76107074e+02]
 [ 0.00000000e+00  7.54617818e+01  1.86264262e+02 -5.16750937e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]
```

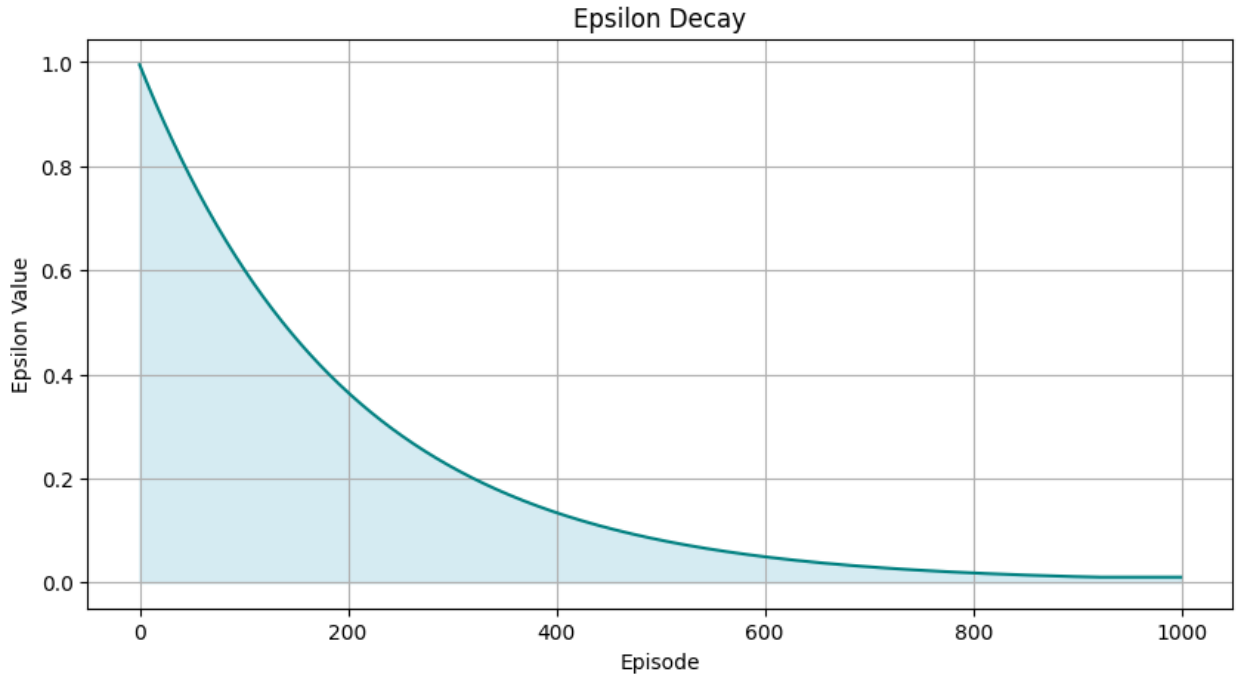
Rewards/Episode Graph:

Initially, the rewards exhibit significant variability, with some instances dropping to zero. However, as the number of episodes increases, the rewards become more consistent, predominantly stabilizing within the 200 to 400 range. This is in contrast to the early stages, where rewards could fluctuate dramatically from -150 in one episode to 400 in the subsequent one.



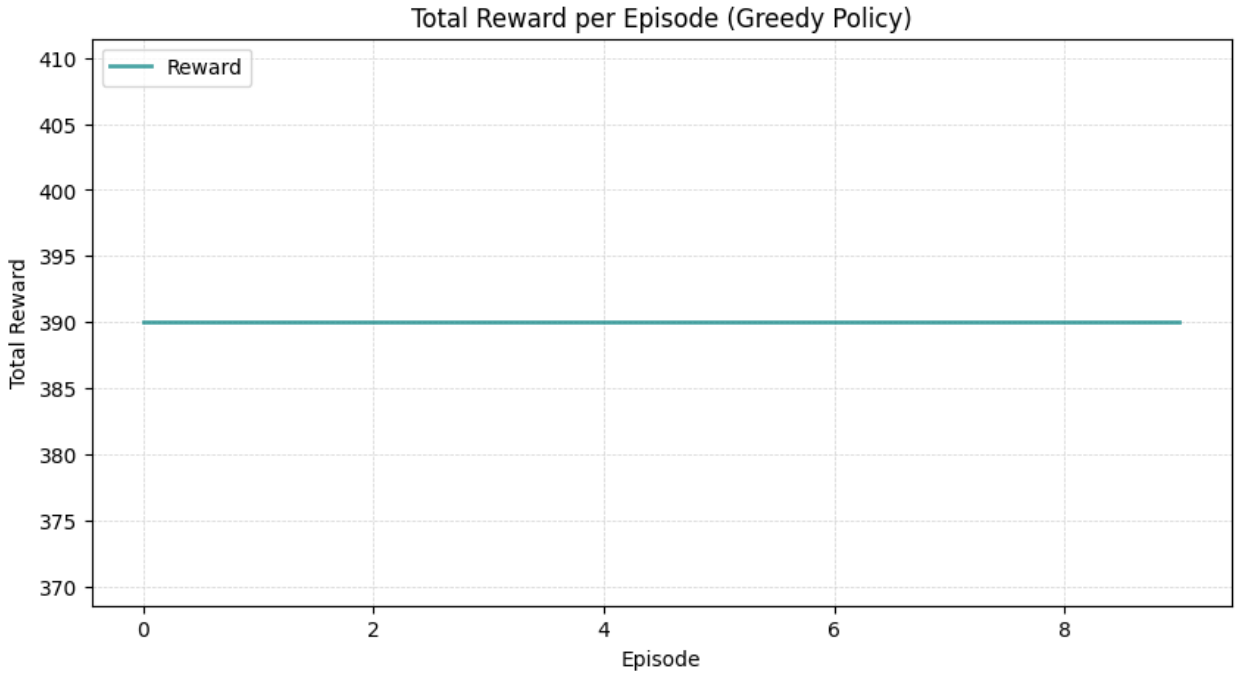
Epsilon Decay Graph:

The rate at which epsilon decreases is steep at the beginning and levels off as it approaches closer to 0. This suggests that most exploration and learning happen in the early episodes, and as the agent learns more about the environment, it relies more on its learned policy.



Total Rewards per episode graph

The graph depicts a consistent reward pattern per episode under a greedy policy, indicating minimal fluctuation and a steady performance.



Double Q-Learning - Stochastic

Q-table after 1000 episodes:

```
Trained Q-table 1:
[[386.81303978 413.63159117 506.30399958 413.69329661]
 [549.62601773 240.81986472 344.2807137  295.4872256 ]
 [181.93389696 176.44934643 513.28820835 218.23467822]
 [  0.           0.           0.           0.        ]
 [214.77483376 208.54035155 546.06640012 172.98449213]
 [  0.           0.           0.           0.        ]
 [325.94221185 318.77305363 278.51686092 524.68636901]
 [ 15.88615047 517.13224896  53.26131381 123.86299758]
 [ 11.6563695  416.28394928  69.04156304  12.36451457]
 [ 20.01020168 506.85432024 106.32767553 111.86277466]
 [  0.           0.           0.           0.        ]
 [459.66809846   9.07706275 119.49608833 115.85767172]
 [  0.           0.           0.           0.        ]
 [  1.50517766  -2.58824843 212.10755783   2.80892206]
 [ -0.87335579   2.52009498 332.89193094  -1.80311086]
 [  0.           0.           0.           0.        ]]
Trained Q-table 2:
[[391.56068937 423.32567063 500.65156598 386.64436762]
```

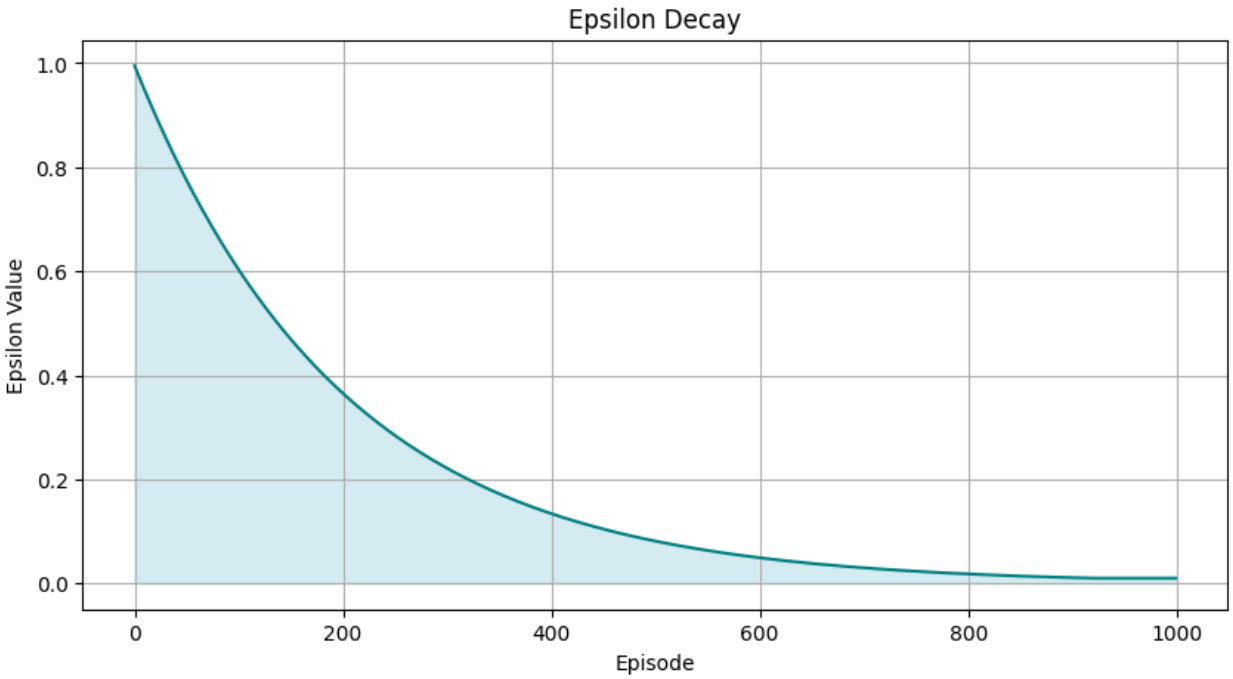
```
[537.06808633 335.86948922 305.86701732 304.32574171]
[230.52513987 140.40827785 511.29318985 226.860051  ]
[  0.           0.           0.           0.        ]
[190.50539143 221.40900727 519.14047819 223.23793782]
[  0.           0.           0.           0.        ]
[341.86081939 293.46749465 253.83421042 541.23331081]
[ 13.95877347 502.92830275  79.27037245 130.23565051]
[-10.61275157 442.93660167  65.40849836   1.61334367]
[ 73.10859124 458.30997904  99.96329608  43.49579389]
[  0.           0.           0.           0.        ]
[462.59406392  32.68791365   5.98935534  60.33448491]
[  0.           0.           0.           0.        ]
[ -2.12709375  27.42950038 240.60078728   4.52675395]
[  4.67521499   0.         376.20848268   0.        ]
[  0.           0.           0.           0.        ]]
```

Rewards/Episode Graph:

The graph presented shows the total reward per episode over a series of 1000 episodes. Initially, the rewards appear to be highly volatile, with values ranging from -300 to above 400, indicating that the agent's performance varied significantly from episode to episode. Over time, while there's still considerable fluctuation in the reward, there's a noticeable consistency in the range of values, mostly staying within 100 to 300. This pattern suggests that as the number of episodes increases, the agent might be learning and improving its strategy, leading to a more stabilized performance with fewer instances of extremely low rewards.

Epsilon Decay Graph:

The graph represents the decay of the epsilon value over 1000 episodes Initially, epsilon is high, close to 1.0, allowing the agent to explore a wide range of actions without strictly following a predetermined policy. As the number of episodes increases, the epsilon value decays exponentially towards 0, suggesting that the agent is shifting from exploration to exploitation, relying more on the learned policy to make decisions.



Total Rewards per episode graph

The graph indicates significant variability in the rewards received per episode, with peaks suggesting episodes where the agent's chosen actions led to high rewards, while valleys represent episodes with much lower rewards. The peaks at episodes 2 and 6 indicate successful outcomes, possibly where the agent made optimal decisions based on its policy. Conversely, the sharp drop after episode 6 suggests an episode where the chosen actions did not result in a high reward, indicating a potential limitation or miscalculation in the policy's effectiveness for that particular episode.

Summary

Tabular methods, including Q-learning and Double Q-learning, are reinforcement learning techniques used for solving problems where the environment is modeled as a Markov Decision Process (MDP). In both methods, an agent interacts with an environment, receiving rewards for its actions and updating its knowledge (represented by a Q-table) accordingly.

Q-learning:

- Update Function: The Q-value for a state-action pair is updated using the equation: `Q(s, a) = Q(s, a) + α * (reward + γ * max(Q(s', a')) - Q(s, a))`, where Q(s, a) represents the Q-value for state-action pair (s, a), α is the learning rate, reward is the immediate reward received, γ is the discount factor, max(Q(s', a')) represents the maximum Q-value for the next state s', and s' is the next state.

Double Q-learning:

- Update Function: Double Q-learning employs two Q-tables, denoted as Q1 and Q2. At each step, one Q-table is used to select the action, while the other Q-table is used to evaluate the chosen action. The Q-values are updated using the equation similar to Q-learning, but with alternating updates between Q1 and Q2 to reduce overestimation bias.

Criteria for a good reward function:
A good reward function should incentivize the agent to achieve the desired behavior or goals in the environment while avoiding unintended side effects. In our environment theare are some implementations that enourage the agent to take good actions by providing the rewards.

- The reward function should assigns rewards to actions that move the agent closer to achieving its objectives.

- There are multiple Rewards, so it can help the agent's exploration and exploitation, encouraging it to discover valuable states and actions efficiently.

Overview

Here's a neatly organized overview of the results obtained from various tabular methods for solving the squirrel maze environment:

Algorithm	Environment	Model Variation	Max Reward (Episode)	Episode 1000 Reward	Epsilon Decay Trend
Q-learning	Deterministic	Base Model	400+	390	Slow Decline
Q-learning	Deterministic	Hyperparameter Tuning (Max Timestamp, Decay Rate: 20, 0.75)	800+	765	Slow Decline
Q-learning	Deterministic	Hyperparameter Tuning (Max Timestamp, Decay Rate: 20, 0.995)	800+	765	Slow Decline
Q-learning	Stochastic	Base Model	400+	305	Slow Decline
Q-learning	Stochastic	Hyperparameter Tuning (Max Timestamp, Decay Rate: 20, 0.995)	800+	660	Slow Decline
Double Q-learning	Deterministic	Base Model	400	-	-
Double Q-learning	Stochastic	Base Model	400+	-	-

References

- CSE 574 D Introduction to ML course, Fall 2023, titled bhanucha_charviku_Assignment_3.
- *Double Q-learning*. (n.d.). Neurips.Cc. Retrieved February 8, 2024, from https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf
- Hasselt, H. V., Guez, A., & Silver, D. (2015). Deep reinforcement learning with Double Q-learning. *AAAI Conference on Artificial Intelligence*, 2094–2100. https://doi.org/10.1609/aaai.v30i1.10295
- *SARSA vs Q - learning*. (n.d.). Github.Io. Retrieved February 8, 2024, from https://tcnguyen.github.io/reinforcement_learning/sarsa_vs_q_learning.html

- *What is the difference between Q-learning and SARSA?* (n.d.). Stack Overflow. Retrieved February 8, 2024, from https://stackoverflow.com/questions/6848828/what-is-the-difference-between-q-learning-and-sarsa