

# Assignment 1 – Defining & Solving RL Environments

## Part 1: Defining RL Environments

### 1. Describing deterministic and stochastic environments.

#### Deterministic Environment:

**1. Definition:** The deterministic environment follows a strict rule-based approach where the drone's actions lead to predictable outcomes and there is no randomness which means every action results in its intended effect.

#### 2. States:

- The environment is represented as a 6×6 grid (default).
- The drone has a (x, y) position and a binary flag (0 or 1) indicating whether it has picked up the package.
- The grid contains no-fly zones, a warehouse (pickup location), and a customer destination (drop-off point).

#### 3. Actions:

The drone can take the following six actions:

1. UP – Move one cell up.
2. DOWN – Move one cell down.
3. LEFT – Move one cell left.
4. RIGHT – Move one cell right.
5. PICKUP – Collect the package at the warehouse.
6. DROPOFF – Deliver the package at the customer's location.

#### 4. Rewards:

- Moving: -1 (penalty for taking steps).
- Entering a no-fly zone: -20 (severe penalty).
- Picking up a package: +10.
- Successfully delivering the package: +20 (episode ends if not in multi-delivery mode).

#### 5. Main Objective:

The goal is to deliver the package efficiently while minimizing movement penalties and avoiding no-fly zones. Since the environment is fully predictable, the agent can plan its route perfectly.

#### Stochastic Environment:

**1. Definition:** The stochastic environment introduces random disturbances, meaning actions may not always produce the intended movement. There is a 10% probability that the drone's movement deviates to an adjacent cell in a random direction.

## 2. States:

- The grid structure and state definitions are identical to the deterministic environment.
- The drone's position may unexpectedly shift due to stochastic effects.

## 3. Actions:

- The available six actions (UP, DOWN, LEFT, RIGHT, PICKUP, DROPOFF) are the same as in the deterministic environment.
- However, when moving (UP, DOWN, LEFT, RIGHT), there is a 10% chance that the drone will move off-course in a random adjacent direction.

## 4. Rewards:

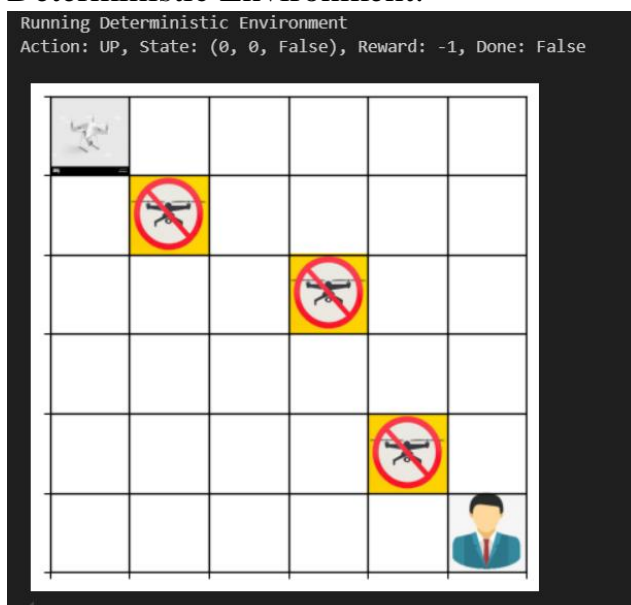
- The reward system remains the same:
  - -1 for each step,
  - -20 for entering no-fly zones,
  - +10 for picking up a package,
  - +20 for successful delivery.
- Due to randomness, additional penalties may be incurred if the drone is forced into a longer path or an obstacle.

## 5. Main Objective:

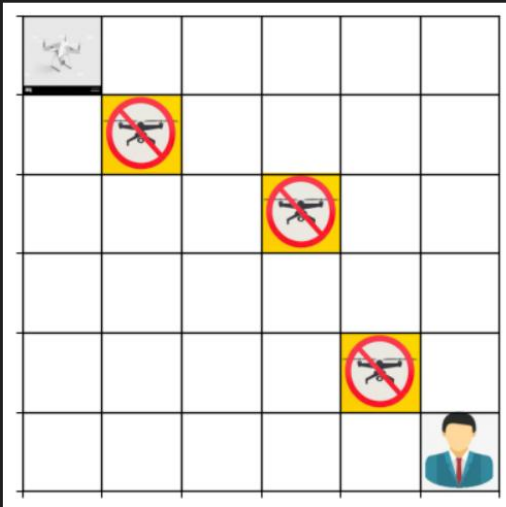
- The goal remains efficient package delivery, but the agent must now adapt to uncertainty.
- The drone must account for potential deviations and choose more robust paths to minimize the risk of failure.
- Strategic planning becomes crucial to compensate for random errors in movement.

## 2. Provide visualizations of your environments.

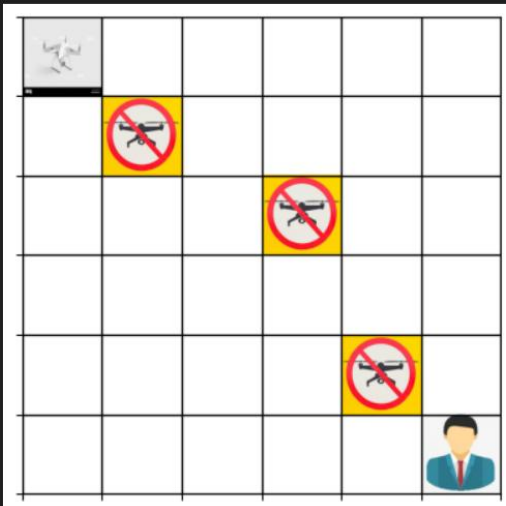
### Deterministic Environment:



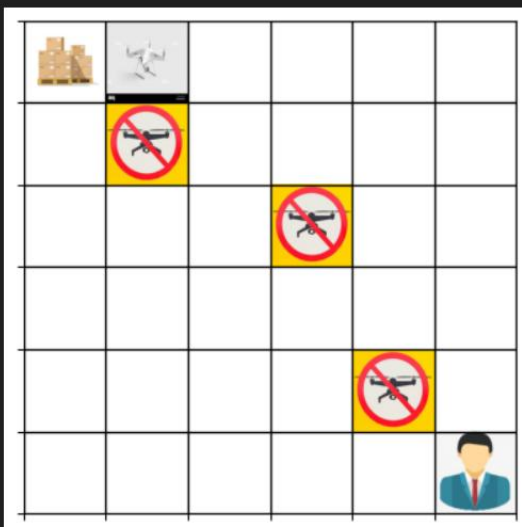
Action: LEFT, State: (0, 0, False), Reward: -1, Done: False



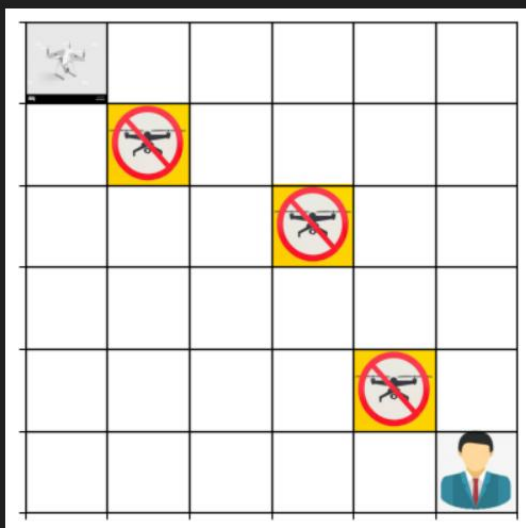
Action: LEFT, State: (0, 0, False), Reward: -1, Done: False



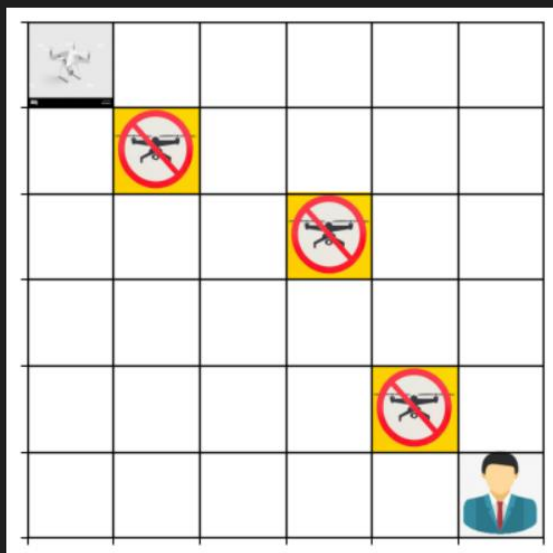
Action: RIGHT, State: (0, 1, False), Reward: -1, Done: False



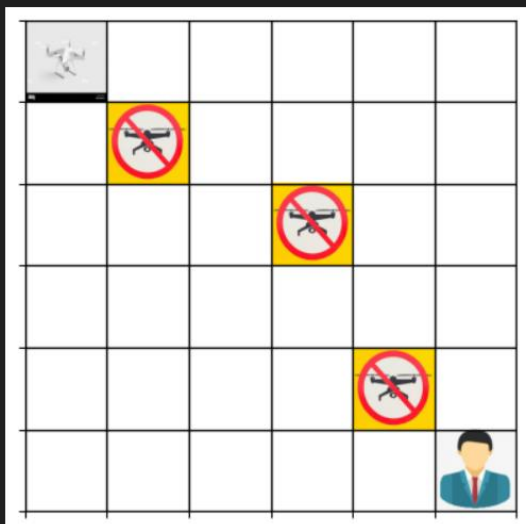
Action: LEFT, State: (0, 0, False), Reward: -1, Done: False



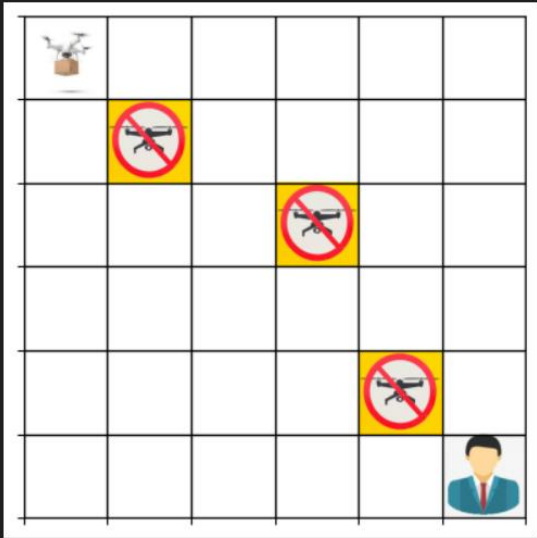
Action: UP, State: (0, 0, False), Reward: -1, Done: False



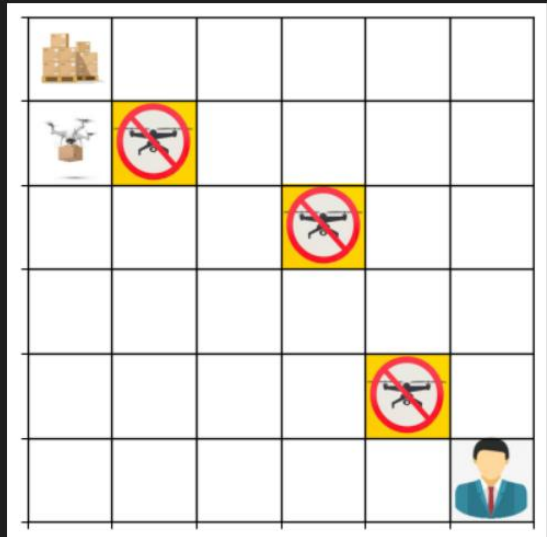
Action: UP, State: (0, 0, False), Reward: -1, Done: False



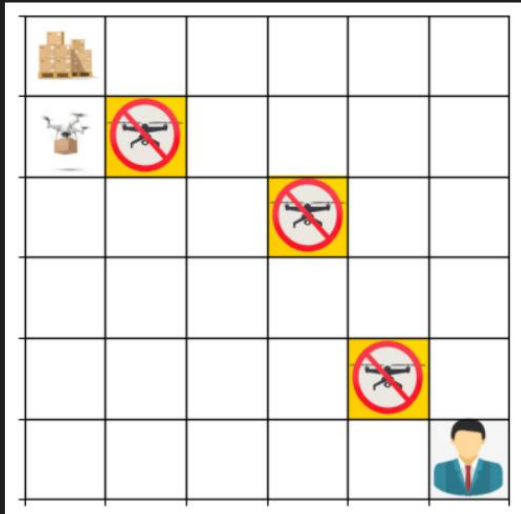
Action: PICKUP, State: (0, 0, True), Reward: 9, Done: False



Action: DOWN, State: (1, 0, True), Reward: -1, Done: False

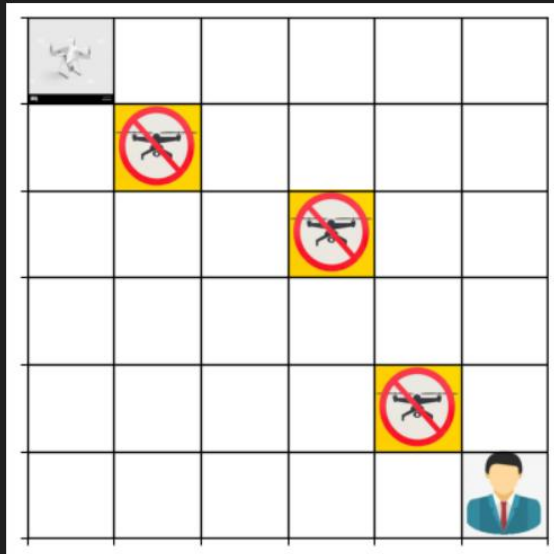


Action: DROPOFF, State: (1, 0, True), Reward: -1, Done: False

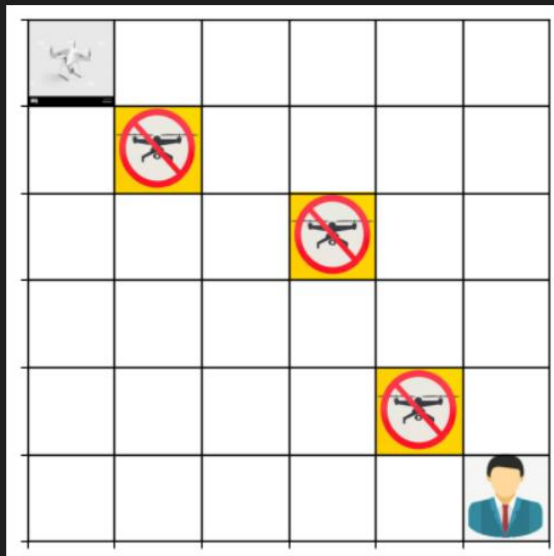


## Stochastic Environment:

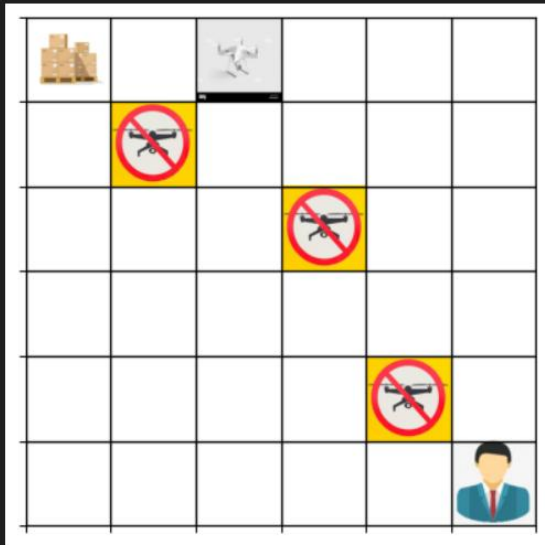
Running Stochastic Environment  
Action: LEFT, State: (0, 0, False), Reward: -1, Done: False



Action: LEFT, State: (0, 0, False), Reward: -1, Done: False

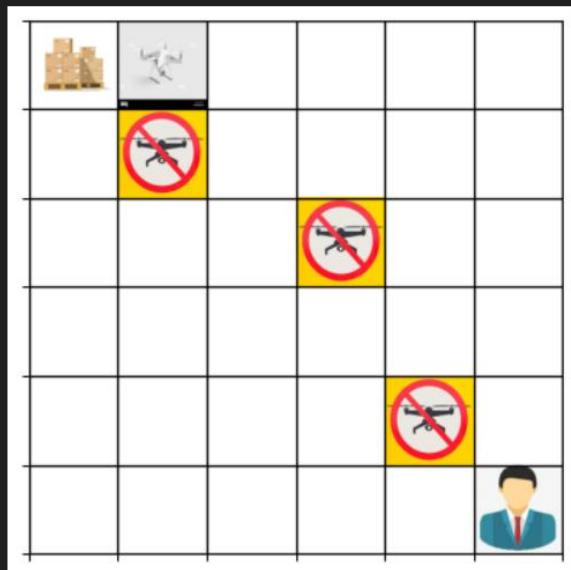


Action: RIGHT, State: (0, 2, False), Reward: -1, Done: False

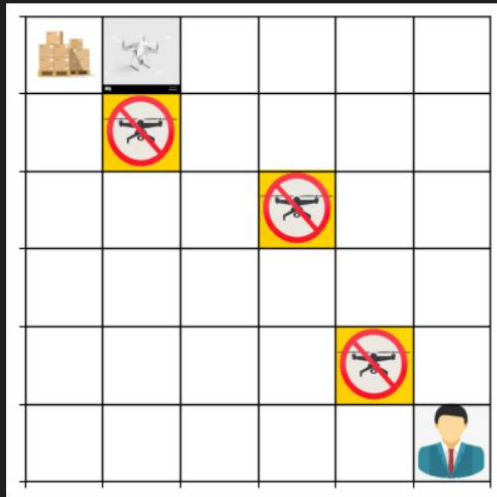


Here we can observe that the drone moved right by 2 boxes, this is because of the action RIGHT, the drone went right 90% but due to stochastic wind conditions of 10% it went right one more box. This proves the stochastic logic.

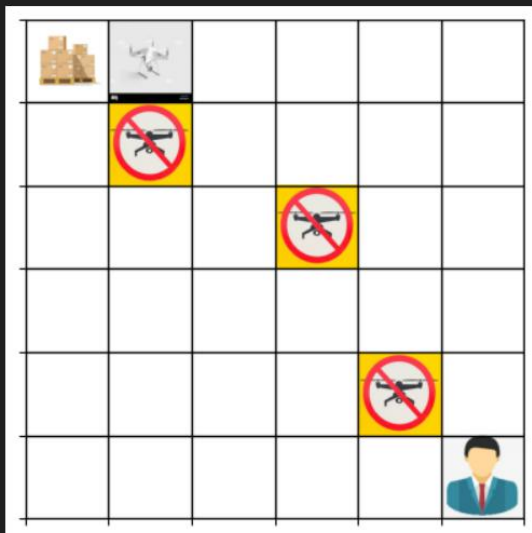
Action: LEFT, State: (0, 1, False), Reward: -1, Done: False



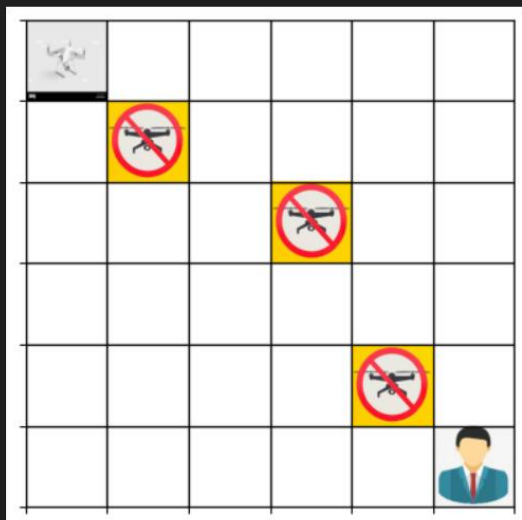
Action: DOWN, State: (0, 1, False), Reward: -20, Done: False



Action: UP, State: (0, 1, False), Reward: -1, Done: False

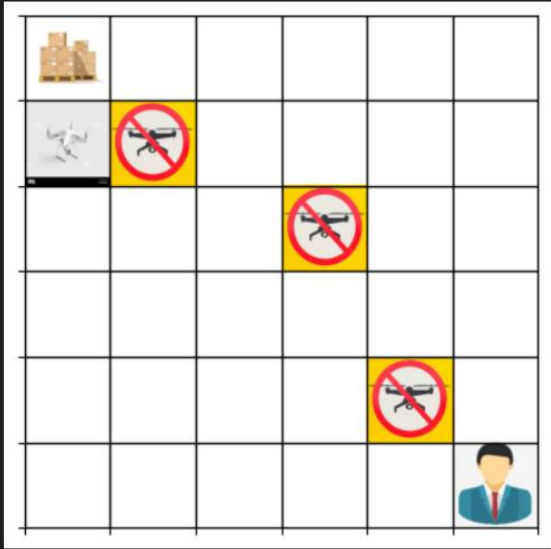


Action: LEFT, State: (0, 0, False), Reward: -1, Done: False

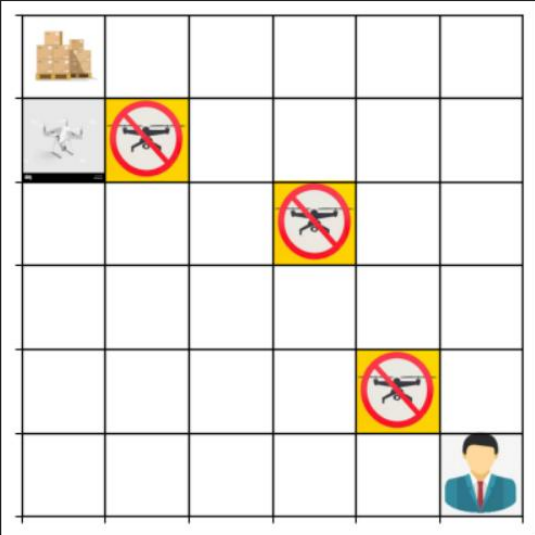




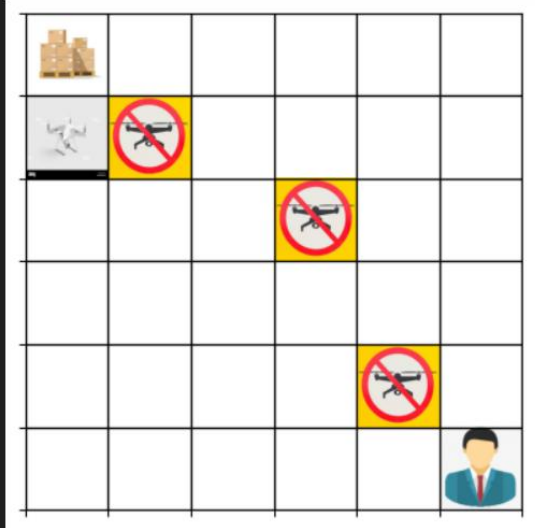
Action: DOWN, State: (1, 0, False), Reward: -1, Done: False



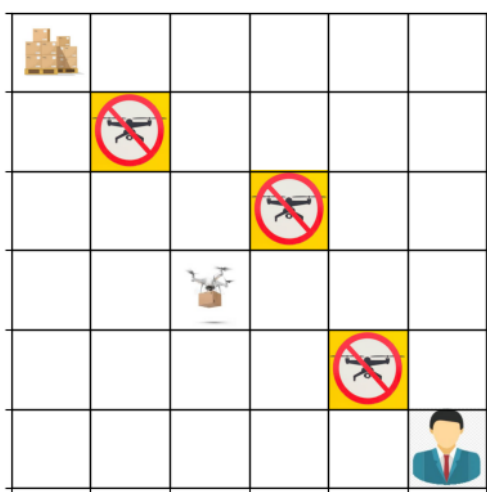
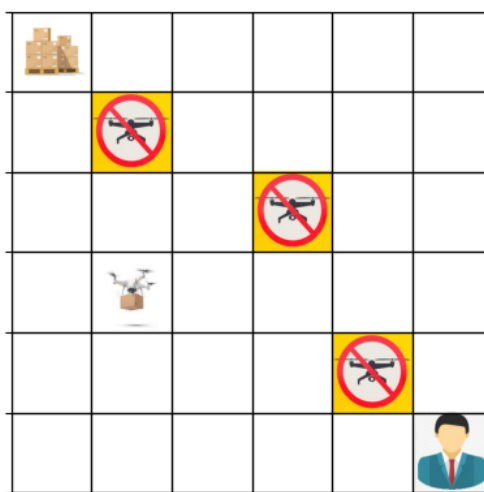
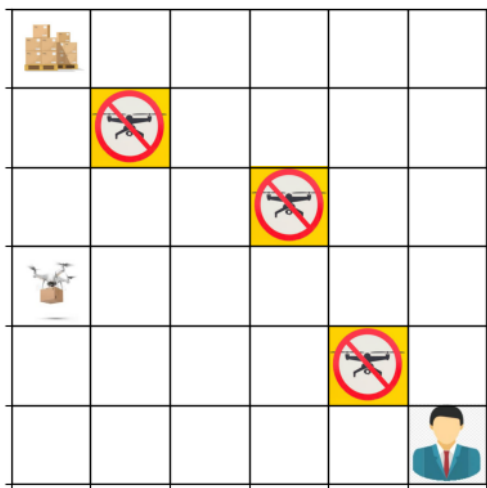
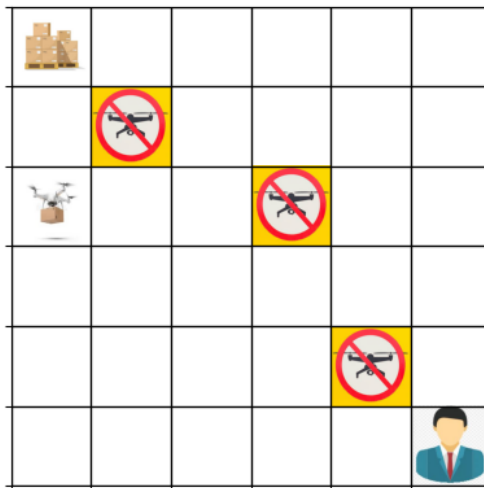
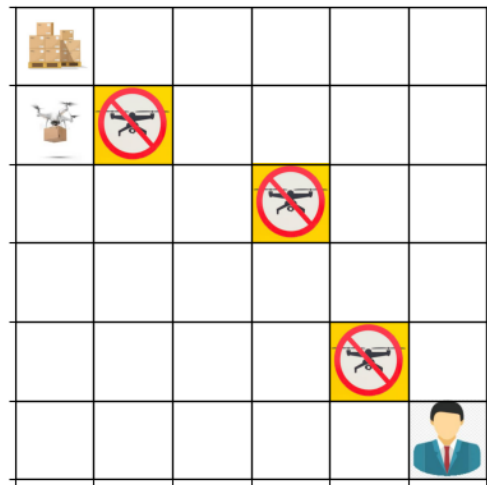
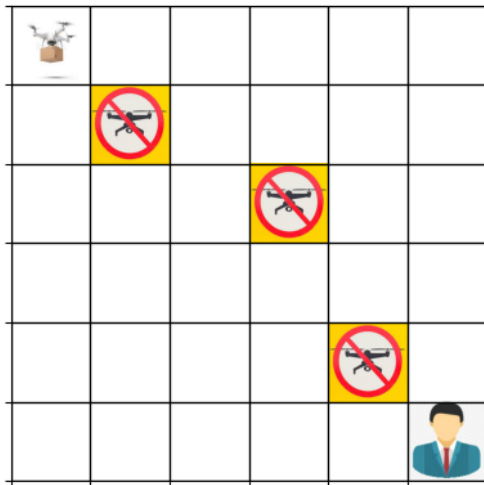
Action: DROPOFF, State: (1, 0, False), Reward: -1, Done: False

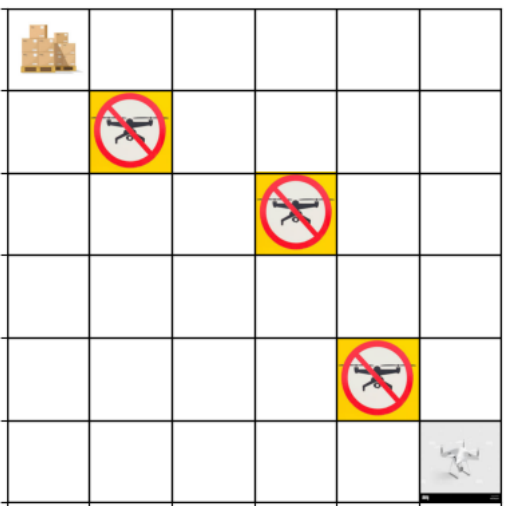
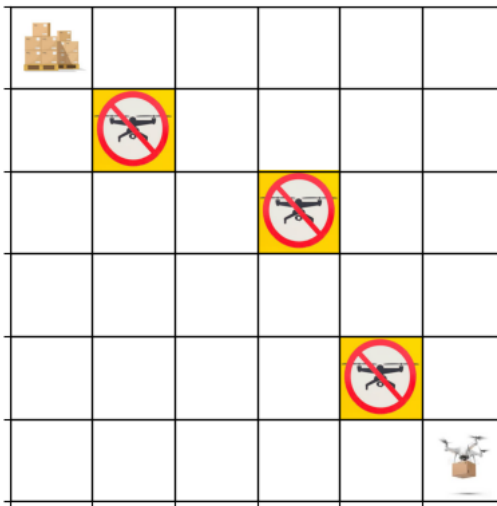
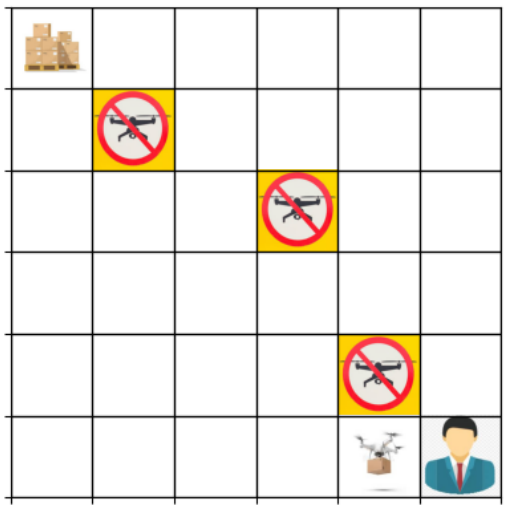
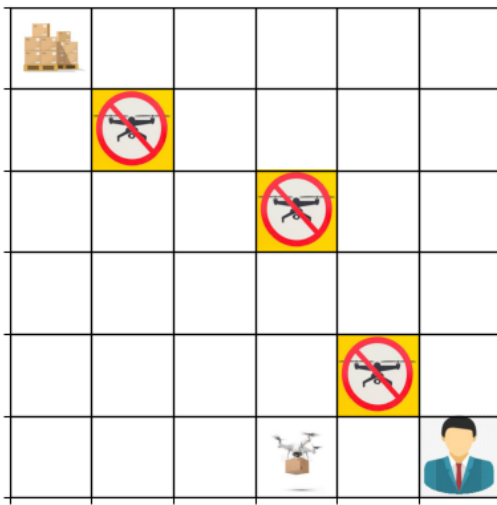
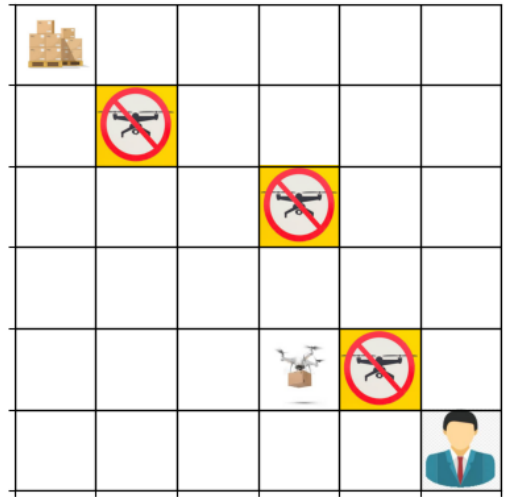
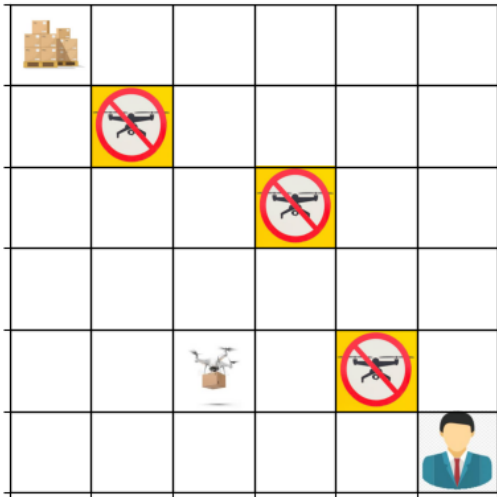


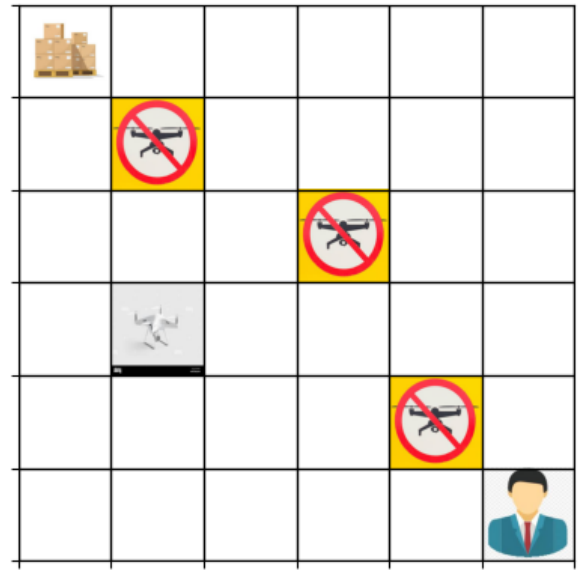
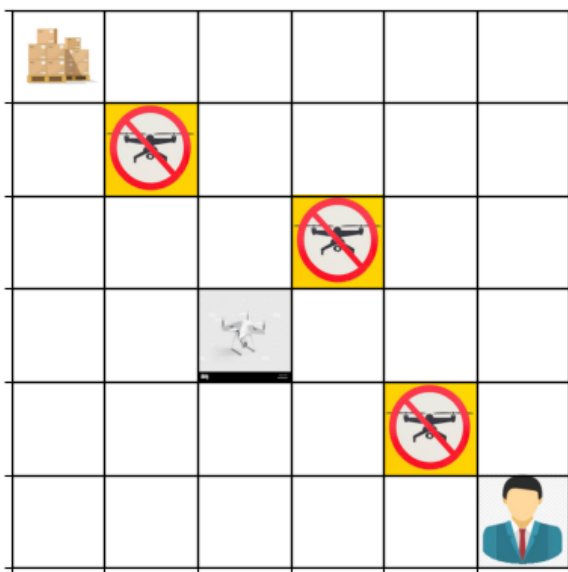
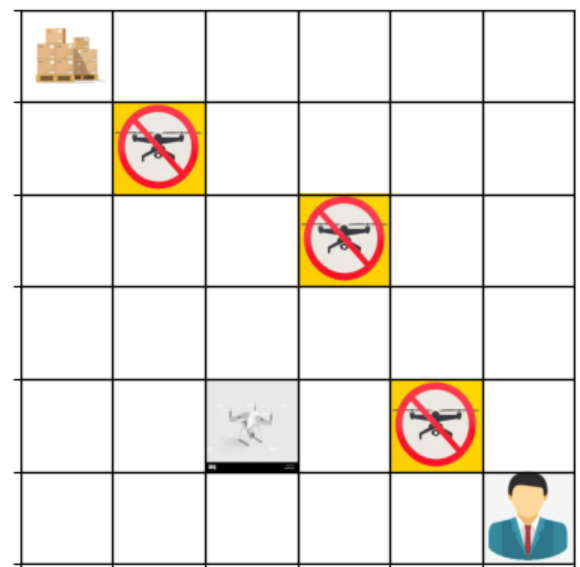
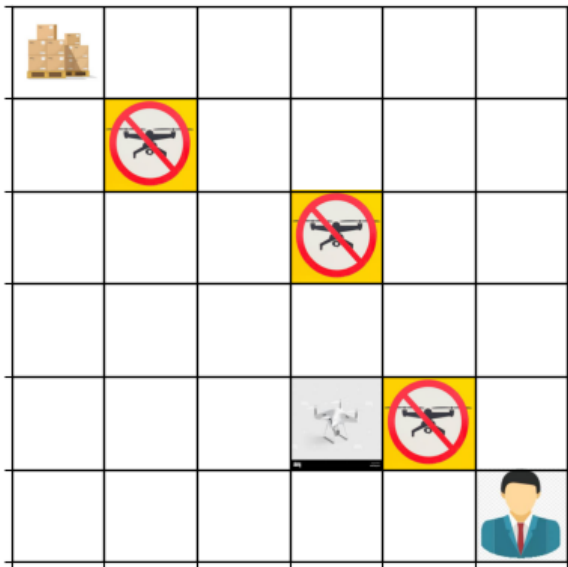
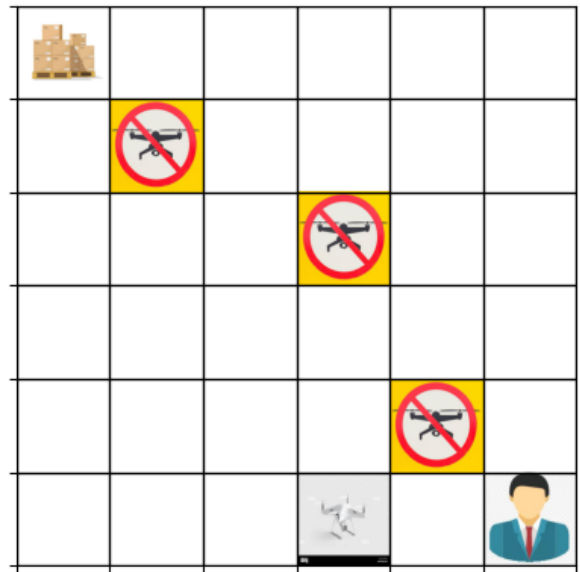
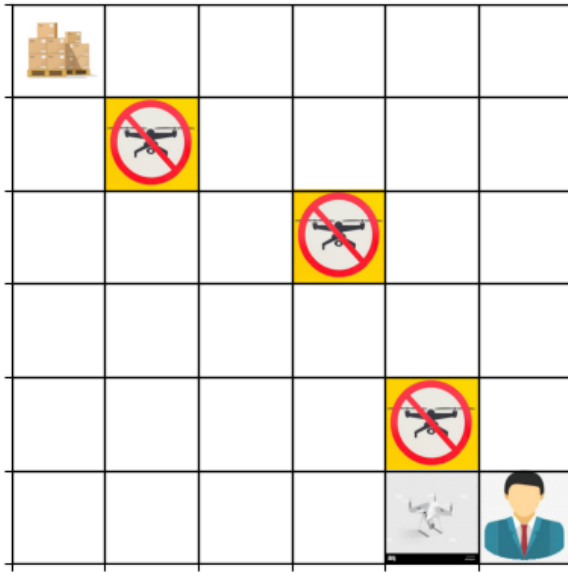
Action: RIGHT, State: (1, 0, False), Reward: -20, Done: False

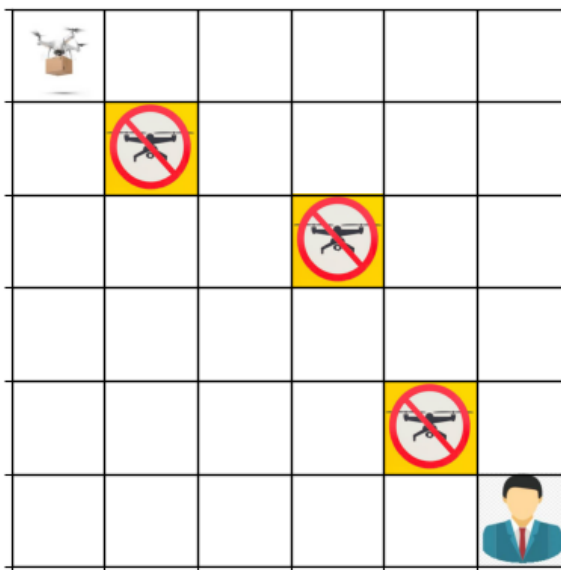
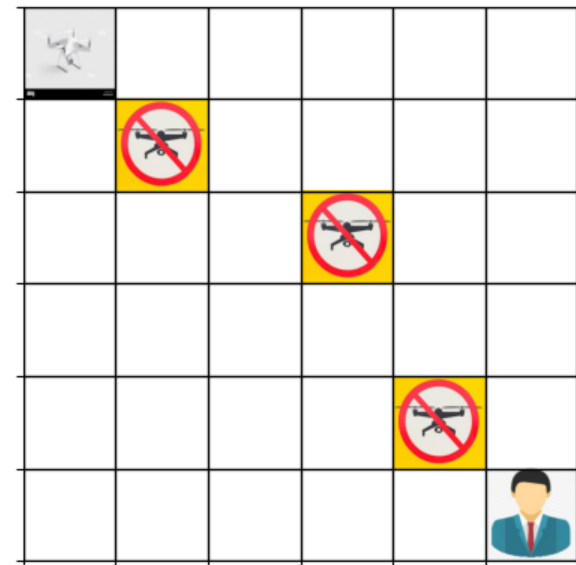
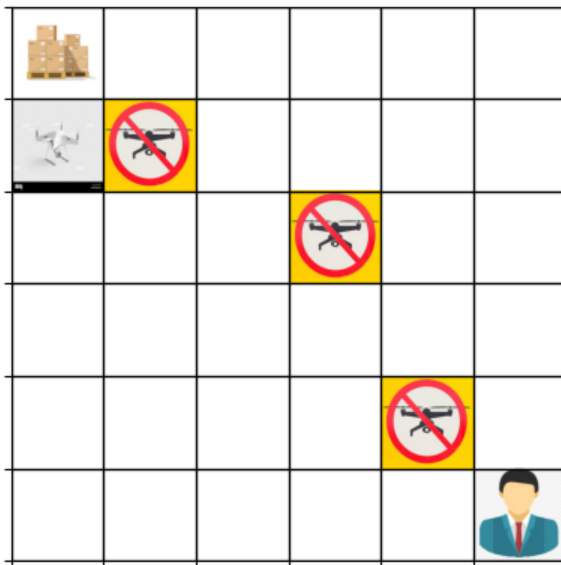
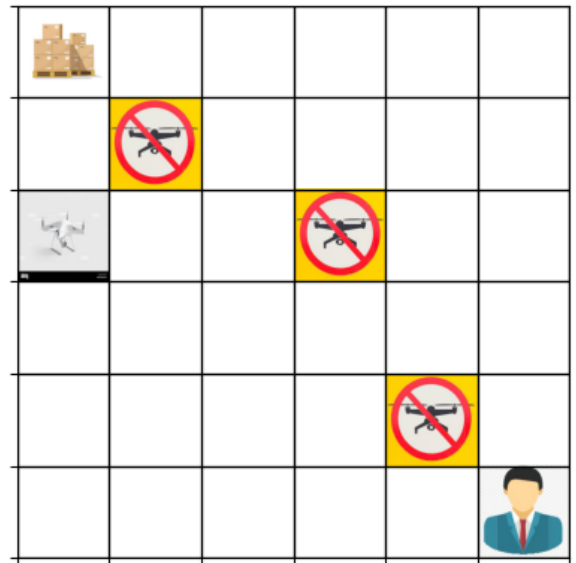
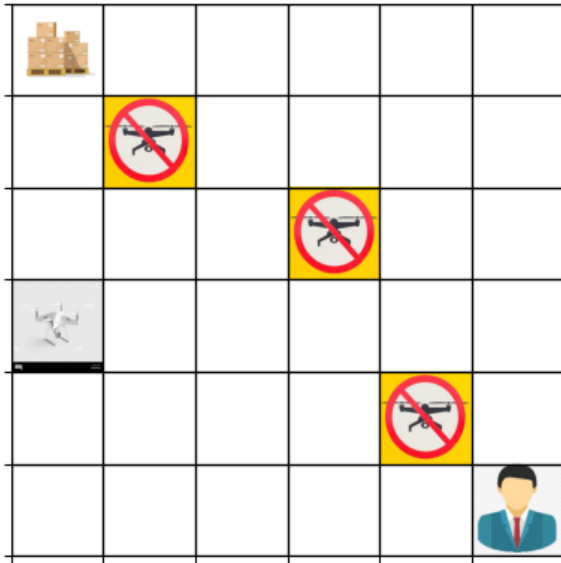


Multiple deliveries visualization logic (Bonus Part – After training the model with Q-learning):









The drone delivered first package and returned back to the pickup point to pickup another package and this continues until the end of first episode. The rest of the visualization of multiple deliveries in one episode is displayed in the jupyter notebook file.

### **3. How did you define the stochastic environment?**

- The stochastic environment was defined by introducing random deviations in the drone's movement. This was implemented in the `_applying_stochastic_conditions` method which affects movement with a 10% probability and also makes sure that new position remains within grid boundaries. When the drone attempts to move (UP, DOWN, LEFT, RIGHT), there is a 10% chance that instead of moving in the intended direction, it will move in a random adjacent direction.

The possible deviations are:

- (0, 1) → Move right
- (1, 0) → Move down
- (0, -1) → Move left
- (-1, 0) → Move up
- The drone does not always move exactly where it intends and if the drone wants to move RIGHT then there is a 10% chance that it may instead move UP, DOWN, RIGHT or LEFT. This increases uncertainty which forces the agent to adapt and plan paths more cautiously.

### **4. What is the difference between the deterministic and stochastic environments?**

#### **1. Deterministic Environment:**

- The drone moves exactly in the direction specified by the chosen action (UP, DOWN, LEFT, RIGHT).
- The environment is fully predictable, if the drone chooses to move right, it will always move right.
- The agent can plan optimal paths without worrying about unexpected deviations.
- No randomness is introduced in drone's movement.

#### **2. Stochastic Environment:**

- The drone has a 10% chance of deviating from the chosen direction.
- Actions do not always result in the intended movement, if the drone chooses to move RIGHT, there is a 10% chance it may instead move UP, DOWN, RIGHT or LEFT.
- The agent must adapt to occasional movement errors which makes it difficult to plan paths optimally.
- The `_applying_stochastic_conditions()` function randomly modifies the movement when triggered.

### **5. Safety in AI:**

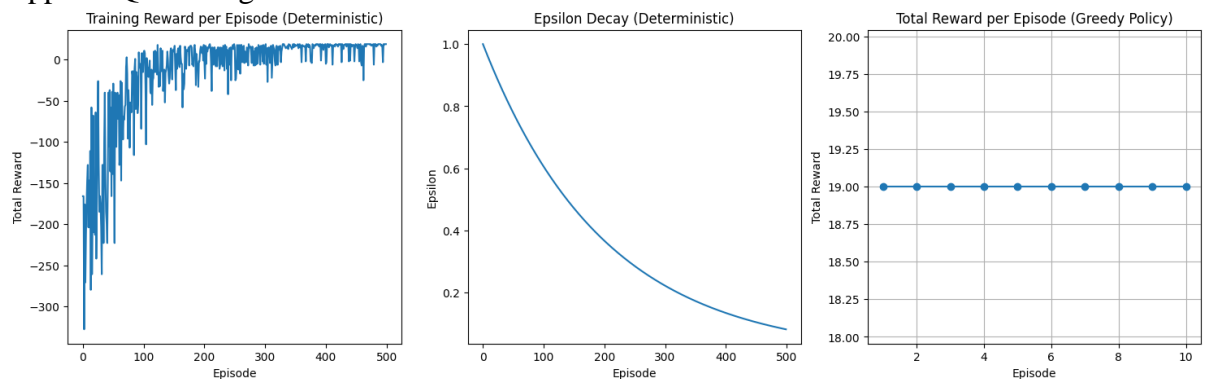
To keep the AI environment safe, we make sure that the drone only takes actions that are allowed (up, down, right, left, pickup, dropoff). It can't fly into no-fly zones or move outside the grid so it always stays within the defined space. The drone also follows clear rules like

only picking up packages at the warehouse and dropping them off at the correct location of the customer. Even in the stochastic environment where some randomness is added, we control it so the drone doesn't move unpredictably or break the game. These checks help the drone navigate safely while still making decisions on its own.

## Part 2: Applying Tabular Methods

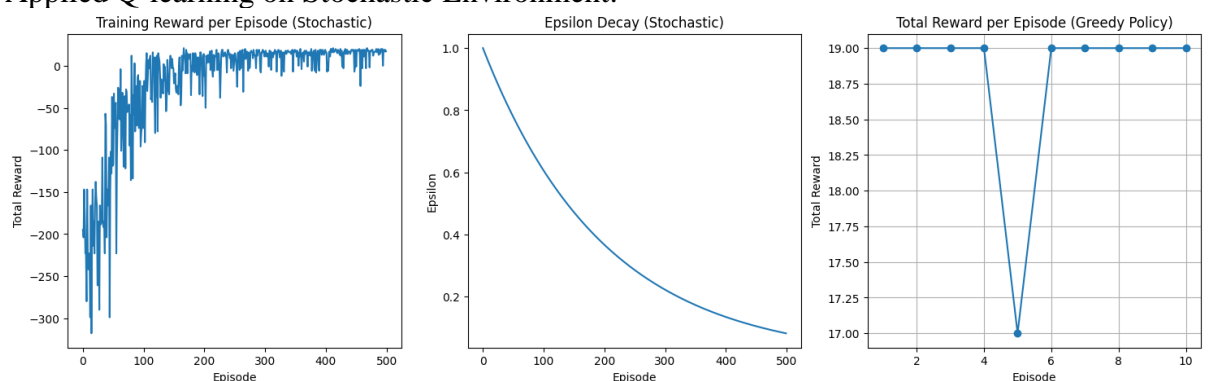
### 1. Show and discuss the results.

#### Applied Q-learning on Deterministic Environment:



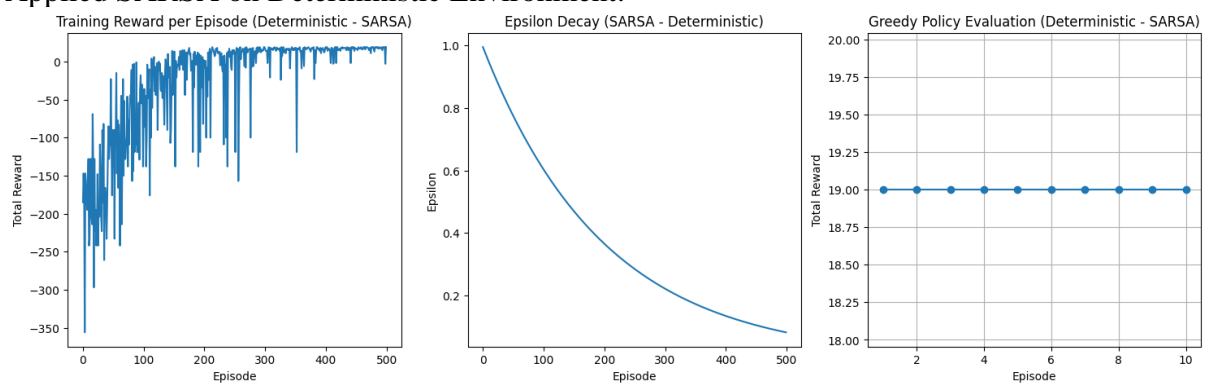
- Training Reward per Episode graph shows the total reward obtained in each episode during training under a deterministic policy. The reward increases steadily with some fluctuations and stabilizes at a high value. This indicates that the model is learning effectively over time with performance improving as the training progresses.
- Epsilon Decay graph shows how the epsilon (exploration rate) decreases over time. Epsilon starts high and decays gradually which aligns with the typical behaviour in reinforcement learning where exploration is reduced over time as the agent becomes more confident in its learned policy. The curve shows a smooth and gradual decrease in exploration.
- Total reward per episode (greedy policy) graph shows the reward achieved when the model follows a greedy policy where it always selects the best action based on its current knowledge. The rewards remain constant at 19 indicating that the model has converged to an optimal policy under the greedy approach and the performance is stable with minimal fluctuations.

#### Applied Q-learning on Stochastic Environment:



- Training reward per episode graph is similar to deterministic case where reward increases over time but there is a bit more variance and fluctuations in the rewards compared to deterministic version. This is expected in stochastic environment due to some randomness but the model still converges to high rewards as training progresses which shows successful learning despite being stochastic.
- The epsilon decay follows the same gradual decrease as in the deterministic case. The model starts with a high exploration rate and progressively shifts towards exploitation as training continues. A smooth decay is desirable because early exploration helps the model learn while later focusing on refining the optimal policy.
- In Greedy policy graph, the rewards are generally stable around 19, similar to the deterministic case but there is a noticeable dip in performance around episode 5 where the total reward falls to 17 before recovering. This could be due to stochastic variations in the environment or the agent encountering an unexpected scenario. Despite this dip, the policy appears to be robust, as the rewards quickly return to a stable high value.

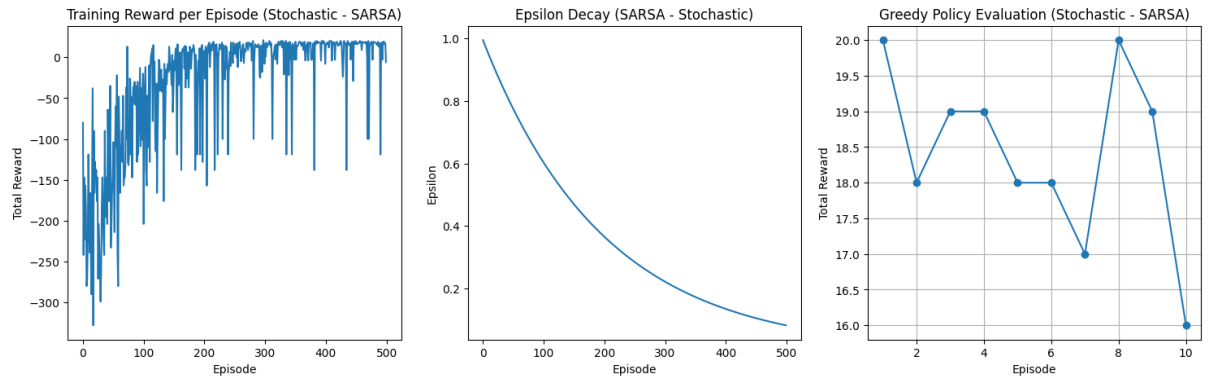
#### Applied SARSA on Deterministic Environment:



- Training reward per episode graph shows a clear improvement in rewards as episodes increases. Initially, there is a significant negative reward but the total reward gradually increases with fluctuations until it stabilizes close to zero or positive values. Compared to Q-learning, SARSA is an on-policy method which means it learns a policy while following it that could explain the slightly smoother progression.
- The epsilon decay follows the expected pattern, gradually decreasing over 500 episodes. This aligns with SARSA's exploration-exploitation trade-off, allowing the agent to explore at the beginning and exploit learned strategies later.
- Once training is complete, the greedy policy evaluation shows that the total reward is consistently at 19 in all episodes which shows that SARSA has learned an effective policy that performs consistently well in a deterministic setting.

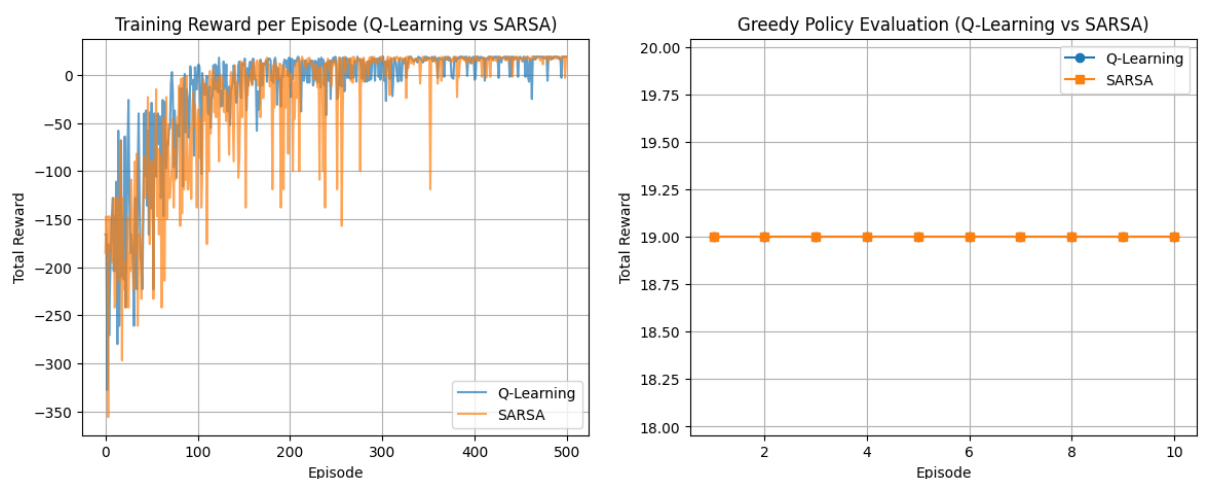


## Applied SARSA on Stochastic Environment:



- The Training reward per episode graph shows significant fluctuations early in training which is expected in a stochastic environment. Over time, rewards improve and the model appears to converge to a stable performance level, although some occasional drops remain. Compared to deterministic SARSA, the fluctuations persist longer due to randomness in the environment.
- The epsilon decay follows a standard pattern, decreasing gradually. This behaviour is typical in SARSA where it explores during the initial phases and begins to exploit learned knowledge as the agent becomes more confident in its actions.
- The greedy policy graph shows noticeable fluctuations unlike the deterministic case where rewards were stable. The total reward varies between 16 and 20 with no clear convergence. This variation suggests that SARSA's on-policy nature makes it more sensitive to stochasticity as it continually updates its policy based on the environment's randomness.

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



### 1. Training Reward per Episode (Q-Learning vs SARSA)

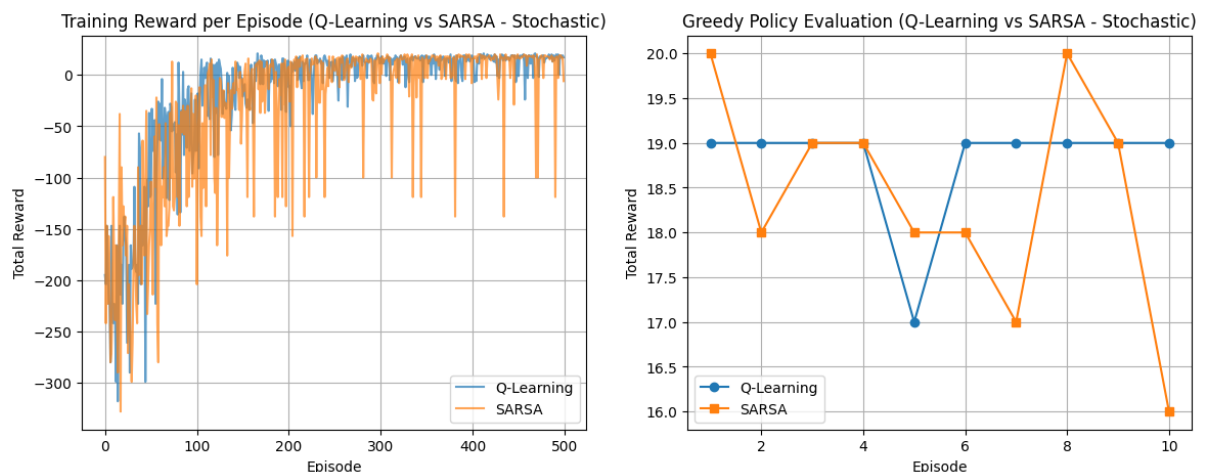
- Both algorithms show improving total rewards over episodes, confirming learning progress.

- Q-Learning (blue):
  - Shows less fluctuation during training, indicating more stable learning.
  - Converges slightly faster to optimal behaviour.
- SARSA (orange):
  - Exhibits more variability in rewards, likely due to its on-policy nature, where it follows its current policy while learning.
  - It tends to be more conservative since it updates using the action it actually takes, rather than the best possible action.
  - The larger fluctuations early on suggest that SARSA takes longer to stabilize compared to Q-Learning.

## 2. Greedy Policy Evaluation (Q-Learning vs SARSA)

- After training, both algorithms achieve a consistent total reward of 19 in the greedy evaluation phase.
  - This indicates that both have learned an effective policy for the deterministic environment.
  - Q-Learning and SARSA perform similarly in policy evaluation, suggesting that, in a deterministic setting, both algorithms can find optimal policies given sufficient training.
- 
- Q-Learning stabilizes faster making it a better choice in deterministic environments where an optimal policy is more straightforward.
  - SARSA is more conservative meaning it can take longer to reach an optimal policy but may be safer in stochastic environments, where unpredictable variations occur.
  - Final policy performance is similar but Q-Learning appears to be more efficient in deterministic cases.

## 3. Compare how both algorithms perform in the same stochastic environment (e.g. show one graph with two reward dynamics) and give your interpretation of the results.



## 1. Training Reward per Episode (Q-Learning vs SARSA - Stochastic):

- Both algorithms show an overall upward trend meaning they successfully learn the task.
- Q-Learning (blue):
  - Still maintains a relatively stable learning curve but shows slightly higher fluctuations compared to the deterministic case.
  - Since Q-Learning is off-policy, it focuses on maximizing long-term reward rather than adapting to randomness in the environment.
  - Achieves a more stable convergence towards optimal performance.
- SARSA (orange):
  - Displays greater fluctuations in the reward trajectory.
  - SARSA, being on-policy, is more sensitive to environment stochasticity since it learns from the actions it actually takes rather than the best possible action.
  - Experiences more instability, as seen in the sharp downward spikes, indicating difficulty in maintaining high rewards in a stochastic setting.

## 2. Greedy Policy Evaluation (Q-Learning vs SARSA - Stochastic):

- Q-Learning maintains a fairly consistent total reward (19) across episodes, showing that it learns a reliable policy even in a stochastic setting.
- SARSA exhibits more variation in the final policy's performance:
  - Some episodes reach optimal (20) reward while others drop to as low as 16.
  - This inconsistency suggests SARSA is adapting to randomness but at the cost of stability.
- Q-Learning is more stable in stochastic environments since it learns the optimal policy regardless of randomness.
- SARSA is more adaptive to environmental uncertainty, but at the cost of stability, making it more prone to fluctuations.
- Final policy performance is less reliable with SARSA, while Q-Learning consistently achieves near-optimal results.

## 4. Briefly explain the tabular methods, including Q-learning, that were used to solve the problems. Provide their update functions and key features.

**1. Q-Learning (Off-Policy):** Q-learning updates the Q-value using the maximum future reward irrespective of the policy being followed. It encourages aggressive exploration and is more effective in deterministic environments but can have high variance in stochastic ones.

Update Rule:

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

Key Features:

- Off-policy: Learns the optimal action-value function regardless of the policy used for exploration.
- Uses greedy action selection in the update rule by taking the maximum Q value.
- Converges faster to the optimal policy but may exhibit more variance in stochastic environments.

**2. SARSA (On-Policy):** SARSA updates the Q-value based on the actual next action taken, making it more stable in dynamic environments. It is better suited for stochastic environments since it follows the policy it is learning rather than assuming optimal future actions.

Update Rule:

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

Key Features:

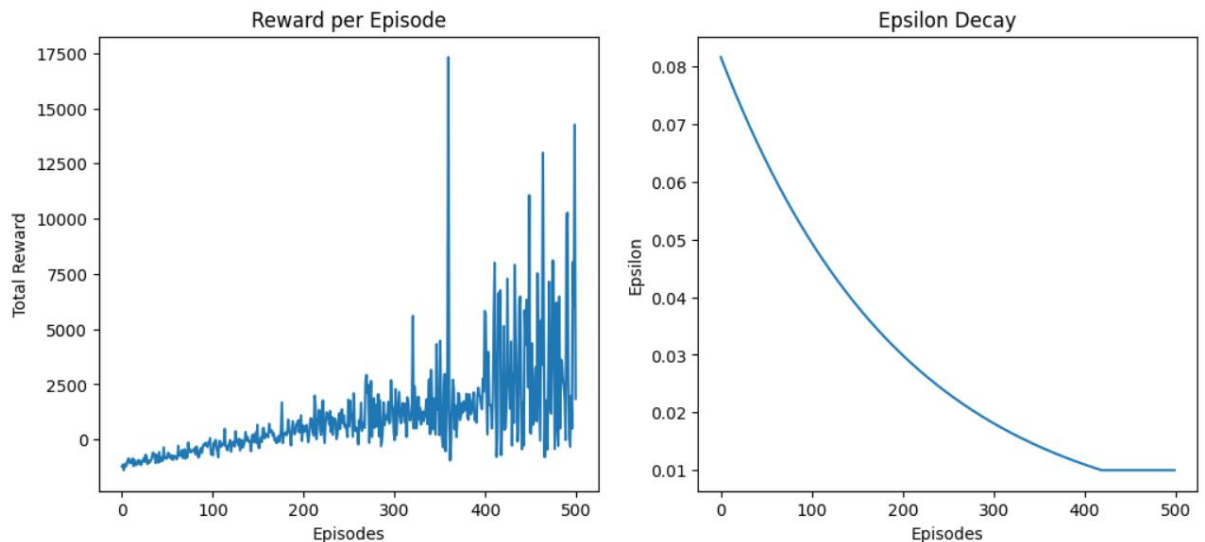
- On-policy: Learns the value of the policy it follows, making it more stable in stochastic environments.
- The update depends on the actual action taken rather than the best possible action.
- More adaptive but less aggressive compared to Q-learning.

**5. Briefly explain the criteria for a good reward function. If you tried multiple reward functions, give your interpretation of the results.**

- The reward should encourage the agent to actually solve the problem and not just hack the system or memorize things.
- The same actions in similar situations should lead to similar rewards so the agent doesn't get confused.
- If rewards only come at the very end (sparse), learning is slow. But if rewards come too often (dense), the agent might take shortcuts instead of learning the best way to do the task.
- The agent shouldn't find loopholes that maximize the reward but don't actually solve the problem.
- If some rewards are huge and others tiny, the agent might ignore important details. Keeping reward magnitudes in check helps smooth learning.

## Part 3: Applying Tabular Methods

1. **Show and discuss the results after applying the Q-learning algorithm to solve the stock trading problem. Plots should include epsilon decay and total reward per episode.**



### 1. Reward per Episode (Left Plot):

- At the start, the agent isn't great at trading. It earns small rewards and learns through trial and error.
- Over time, rewards generally increase which means the agent is figuring out better strategies.
- There are some wild spikes where it either makes a super profitable move or takes a big loss.

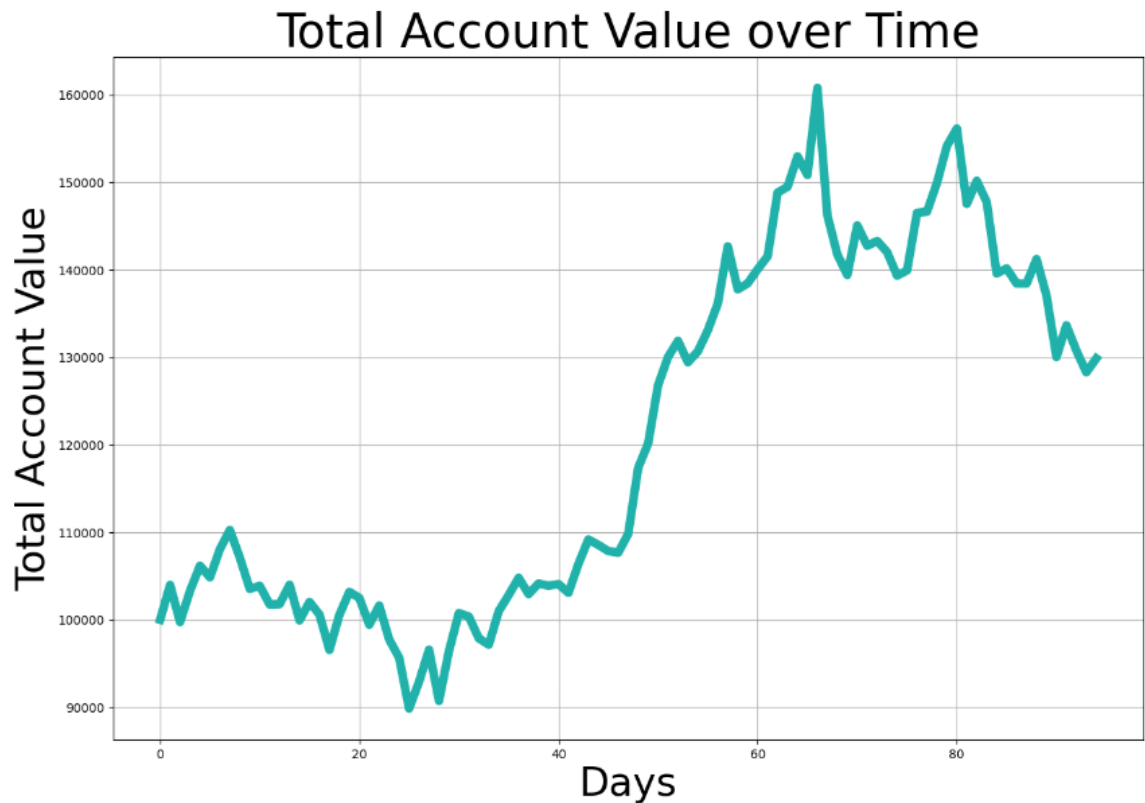
### 2. Epsilon Decay (Right Plot):

- Epsilon controls how much the agent explores vs follows what it already knows.
- At first, it explores a lot (random trades) but as training progresses, it starts making more informed decisions.
- By the end, epsilon is very low meaning the agent is mostly relying on what it has learned.

The agent is improving and making smarter trades as training goes on. The large ups and downs mean that while it's getting better its strategy still has some inconsistencies. Fine-tuning things like learning rate, reward function, or even using deep learning could make it more stable.

2. **Provide the evaluation results. Evaluate your trained agent's performance (you will have to set the train parameter set to False), by only choosing greedy actions from the learnt policy. Plot should**

include the agent's account value over time. Code for generating this plot is provided in the environment's render method. Just call `environment.render` after termination.



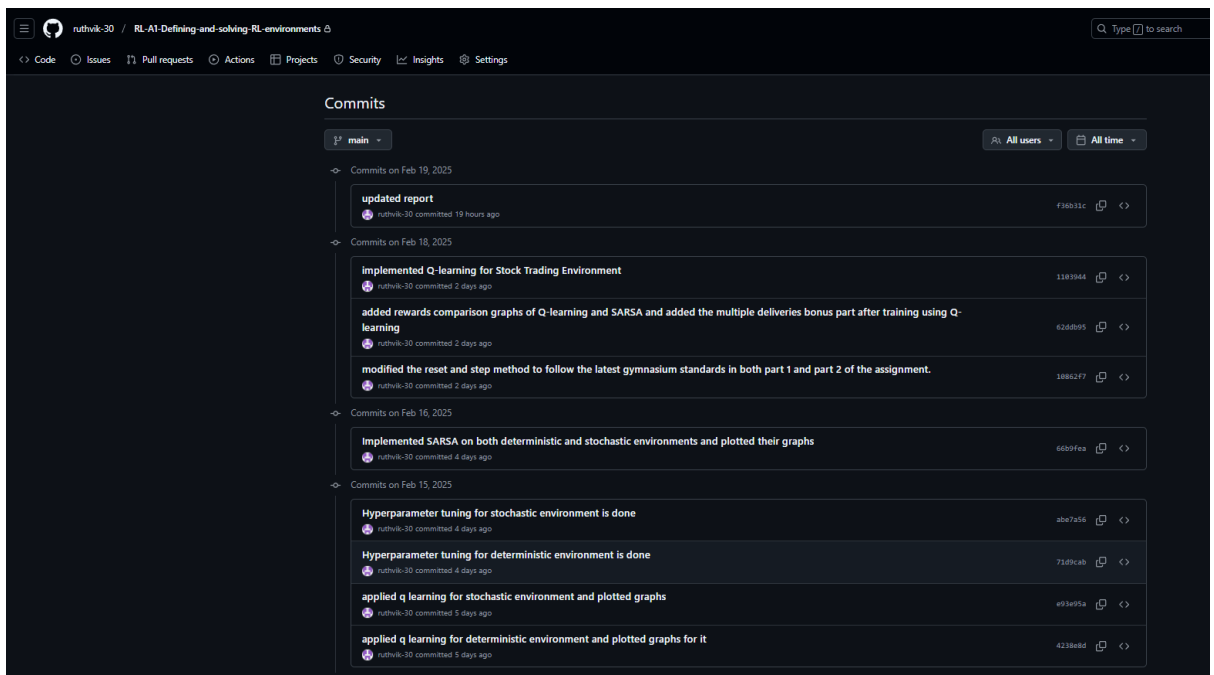
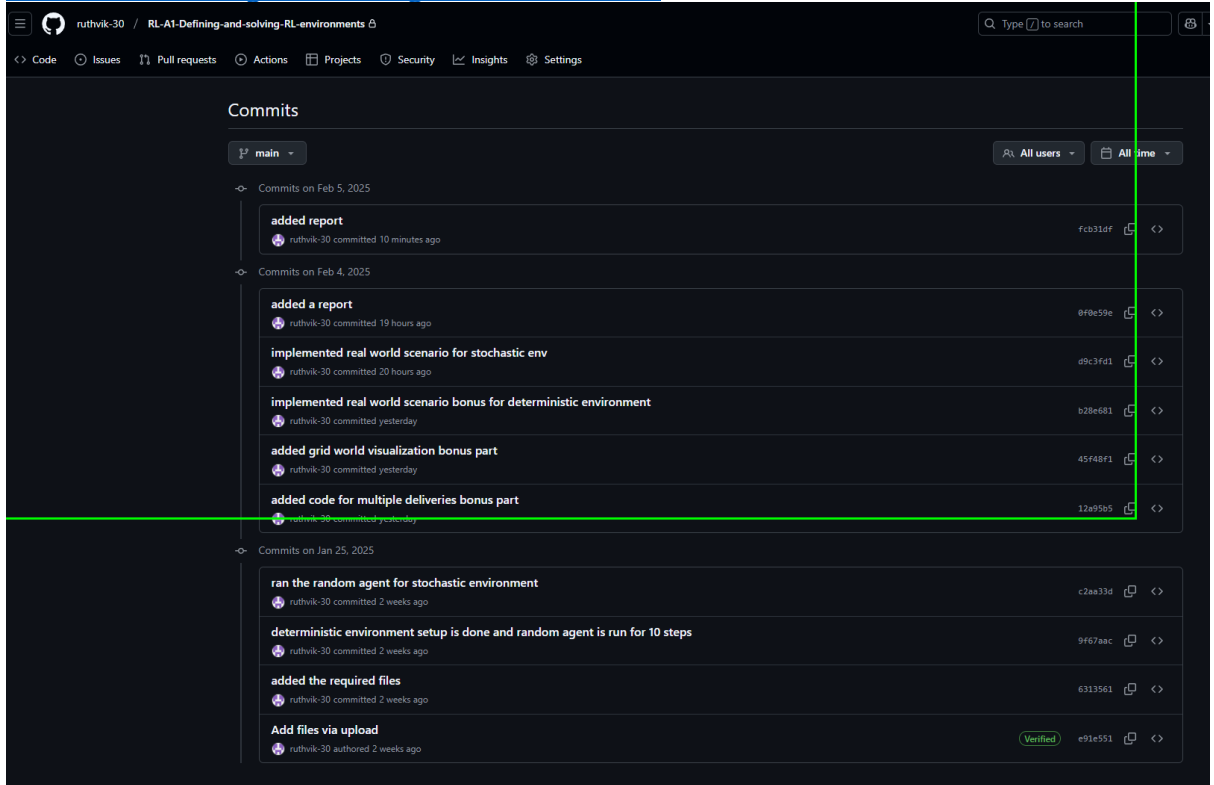
This graph shows how the agent's total account value changed over time when tested using only the best actions it learned during training. In the beginning, the agent struggles a bit but as it learns we see steady growth in account value. Around the middle, there's a sharp increase, meaning the agent made some really profitable trades. Toward the end, the account value drops a bit showing that the agent still has trouble dealing with market downturns.

The agent clearly learned some solid trading strategies, as shown by the overall upward trend. But it still needs improvement especially in handling losses and market dips. Tweaking the reward system or adding better risk management rules could help make it more stable.

## Bonus Tasks:

- **Git Expert:**

Link to my private GitHub project and commit history: <https://github.com/ruthvik-30/RL-A1-Defining-and-solving-RL-environments>



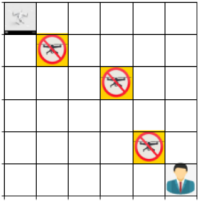
- **CCR Submission:**

ondemand.ccr.buffalo.edu/node/cpn-q07-12-02.core.ccr.buffalo.edu/52828/notebooks/part1.ipynb#

File Edit View Insert Cell Kernel Help Trusted Python 3 (ipykernel)

```
print("Running Deterministic Environment with multiple deliveries")
env_multiple_del.reset()
for _ in range(10):
    action = random.randint(0, 5)
    state, reward, done, truncated, _ = env_multiple_del.step(action)
    print(f"Action: {env_multiple_del.actions[action]}, State: {state}, Reward: {reward}, Done: {done}")
    env_multiple_del.render()
    if done:
        print("Delivery Complete!")
        break
```

Running Deterministic Environment with multiple deliveries  
Action: UP, State: (0, 0, False), Reward: -1, Done: False



Action: DROPOFF, State: (0, 0, False), Reward: -1, Done: False

In [13]: `pwd`

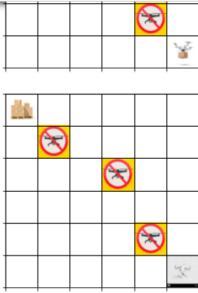
/user/ruthvikv

ondemand.ccr.buffalo.edu/node/cpn-q07-12-02.core.ccr.buffalo.edu/52828/notebooks/part2.ipynb#

File Edit View Insert Cell Kernel Help Trusted Python env new

```
state_index = next_state_index
total_reward += reward
env_multiple_del.render()
rewards_per_episode.append(total_reward)
return rewards_per_episode

mul_del_greedy_rewards_det = mul_del_evaluate_q_learning(env_multiple_del, mul_del_q_table_stoch, episodes=1)
```

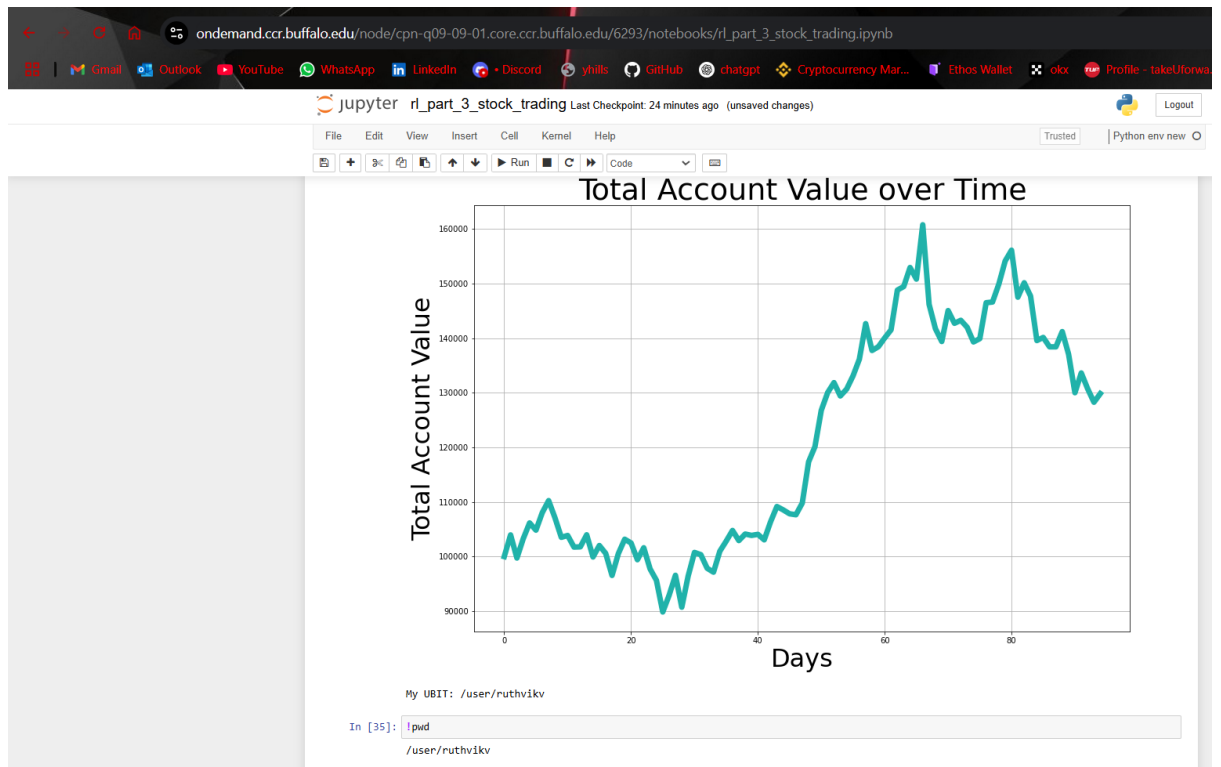


In [31]: `pwd`

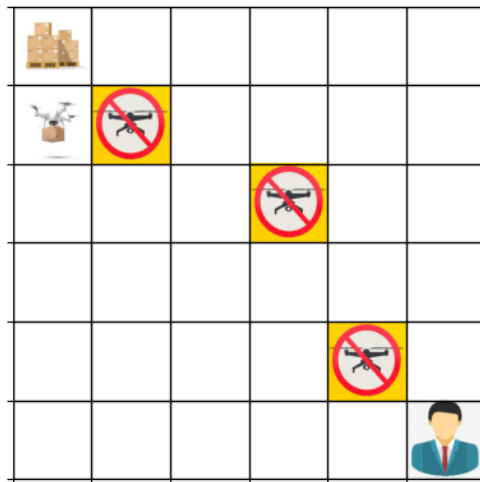
/user/ruthvikv

In [ ]:





- **Grid-World Scenario Visualization:**



- **Complex Environment Scenario:**

Refer to part 1 in this report under the heading “Multiple deliveries visualization logic (Bonus Part – After training the model with Q-learning)”

## References:

- Part I is based on my (Ruthvik Vasantha Kumar) Assignment 3 submission for CSE 574, Fall 2024.
- [https://gymnasium.farama.org/introduction/create\\_custom\\_env/](https://gymnasium.farama.org/introduction/create_custom_env/)
- [https://gymnasium.farama.org/introduction/train\\_agent/](https://gymnasium.farama.org/introduction/train_agent/)
- <https://www.geeksforgeeks.org/reinforcement-learning-in-python-implementing-sarsa-agent-in-taxi-environment/>

