Assignment 1 - Defining & Solving RL Environments

Apurva Banka
50610491

Part I

1. The set of actions defined for deterministic environment are -
    a. UP
    b. DOWN
    c. LEFT
    d. RIGHT
    e. PICKUP
    f. DROP

For the stochastic environment, the action RIGHT has a probability of 90%. So, there is a 90% chance of the RIGHT action taking place and a 10% chance of action not taking place.

The state for the environment is a 6x6 grid, containing a robot, a source package, a target and some (three) shelves. The robot is supposed to pick up the package from the source and drop it to the target avoiding the shelves which act as obstacles.

The rewards for actions for both deterministic and stochastic env are as follows -

a. +100 pts for successful delivery
b. +25 pts for picking up the object the first time
c. -25 pts for dropping the package in an incorrect position
d. -1 pts for each step taken
e. -20 pts for hitting a shelf
f. -10 pts for going outside the 6x6 grid
g. -20 for invalid pickup and dropoff

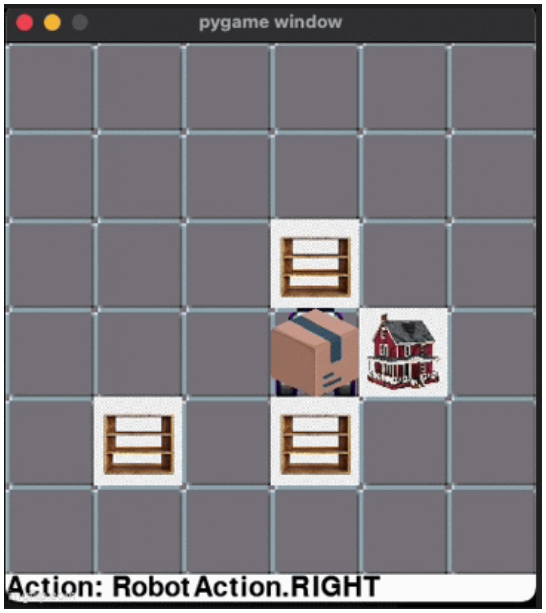
The main objective of the given environment is the robot is supposed to pick up the package from the source and drop it to the target avoiding the shelves which act as obstacles.

The states, rewards and main objectives are common for both deterministic and stochastic environments.

2. The visualization of the environment is as follows:

The robot starts from the (0,0) index. The robot doesn't have the package as of now. The actions taken by the robot are displayed at the bottom.

Once the robot reaches the package, the robot will pick up the package, if the robot position is equal to the package position.



As we can see, the robot is now carrying the package.

Once the robot reaches the target, if the robot has the package, it will drop the package and the environment will terminate.

3. The stochastic environment is defined by introducing uncertainty in the set of actions.

The probability of the robot taking the action has a 90% chance of success and a 10% chance of no action being taken.

This is implemented using the random function that results in the set of actions getting executed.

4. A deterministic environment in AI means that for any given state and action, there is only one possible outcome, with no randomness involved, while a stochastic environment introduces uncertainty, where the next state can vary even with the same action due to random factors, meaning the outcome is not guaranteed and can differ each time. With respect to the warehouse robot environment, the difference between deterministic and stochastic is as follows:

a. For the transitions, the deterministic environment is predictable with a success rate of 100% whereas the stochastic environment is probabilistic with a success rate of 90%.
b. For the deterministic environment, the agent relies on exact state-action mapping whereas in the stochastic environment the agent requires robustness to uncertainty.
c. The complexity of the deterministic environment is low and is simple to solve whereas the stochastic environment requires exploration and exploitation.

These are the key differences between a deterministic and stochastic environment.

5. In order to ensure the safety in the environment created, we have implemented robust constraints and validation mechanisms that prevent the agent from taking invalid action. We have explicitly defined the **action space** of a 6x6 grid outside which the robot cannot move. If the robot is in the leftmost block and the next action is to move further Left, then no action takes place. This is true for all directions. The **state space** is defined such that the robot can be prevented from taking invalid actions. For example not going over obstacles, dropping packages on the target etc. Also, episode

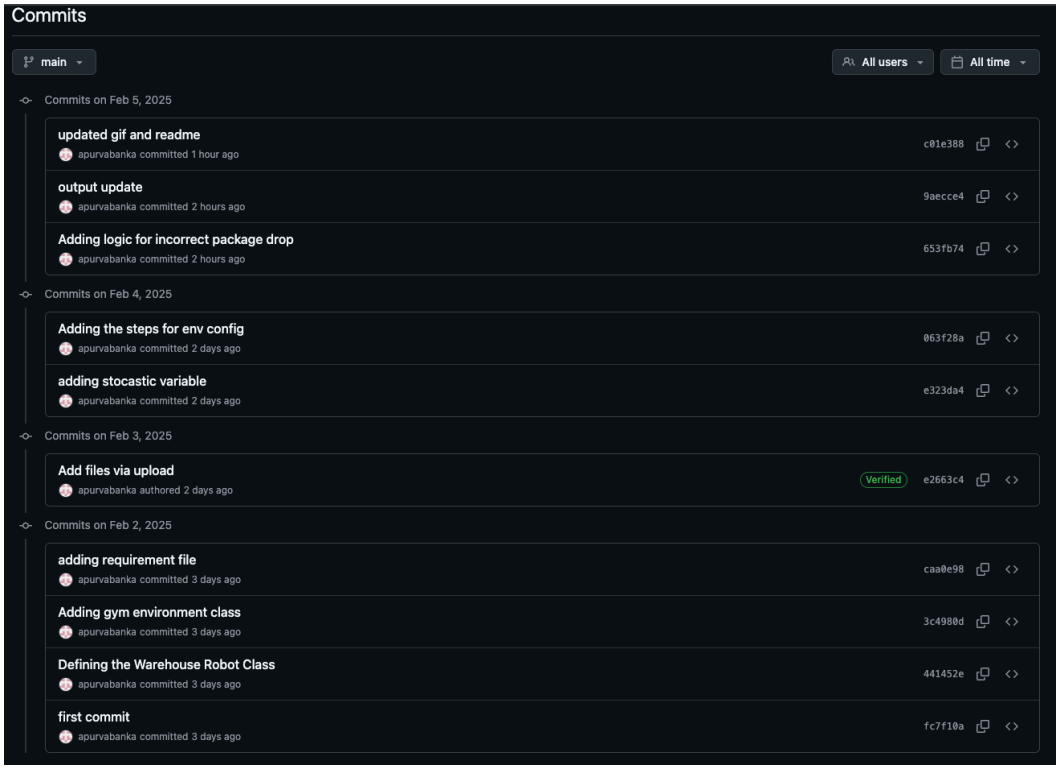
termination is used to halt unsafe trajectory and prevent the agent from continuing invalid state.

GitHub

A github project for part 1 is created. The link to the Github Repo is as follows.

https://github.com/apurvabanka/warehouse_robot_RL

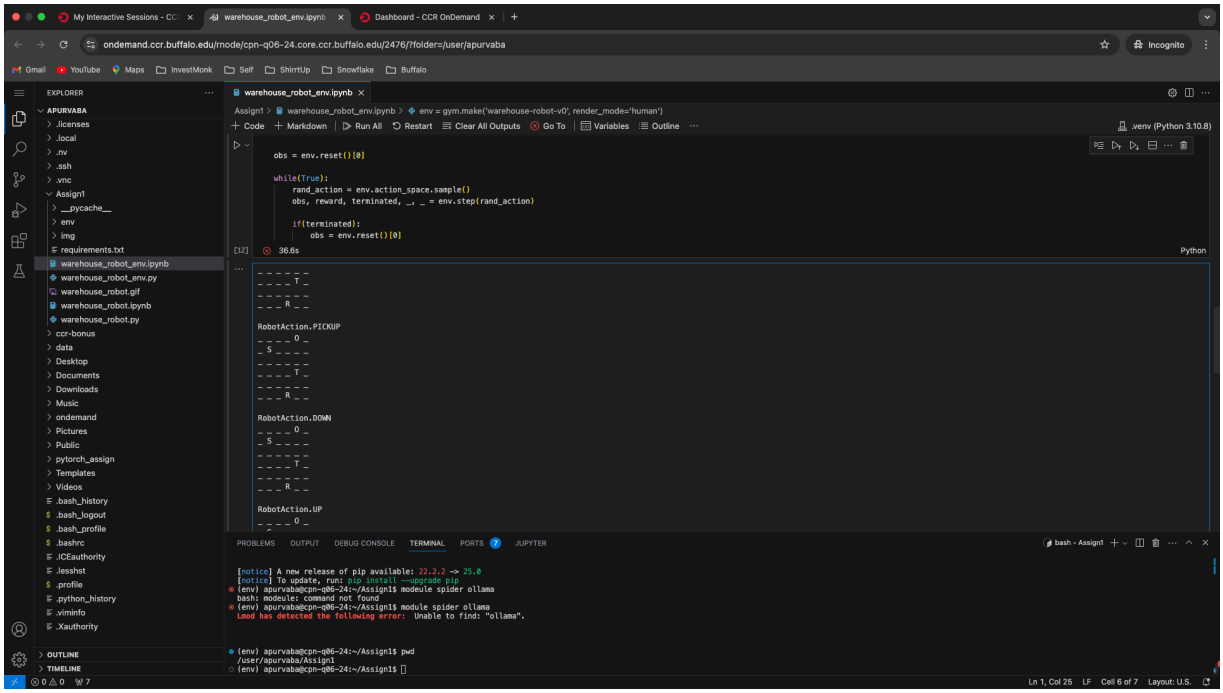
A screenshot of the commit history is as follows.



CCR Submission

The code was successfully executed on a CCR instance. A VSCode instance was requested. After configuring the virtual environment with the required libraries, the code was run using Jupyter Notebook on VS Code.

A screenshot of the same is attached below.



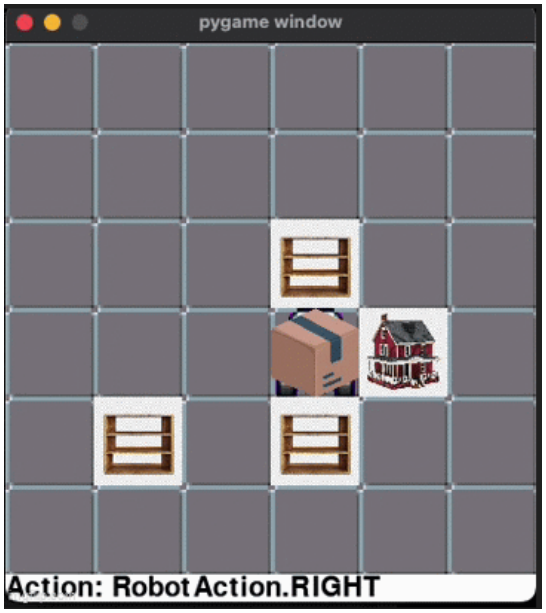
Grid-World Scenario Visualization

A visualisation of the environment is created using the ‘pygame’ python library. Below are the screenshots attached for the env.

This is an initialisation of the environment.

The package is the source, the shelf is the obstacle and the home is the target.

Here, the agent has picked up the package and the next task of the agent is to drop it to the target.

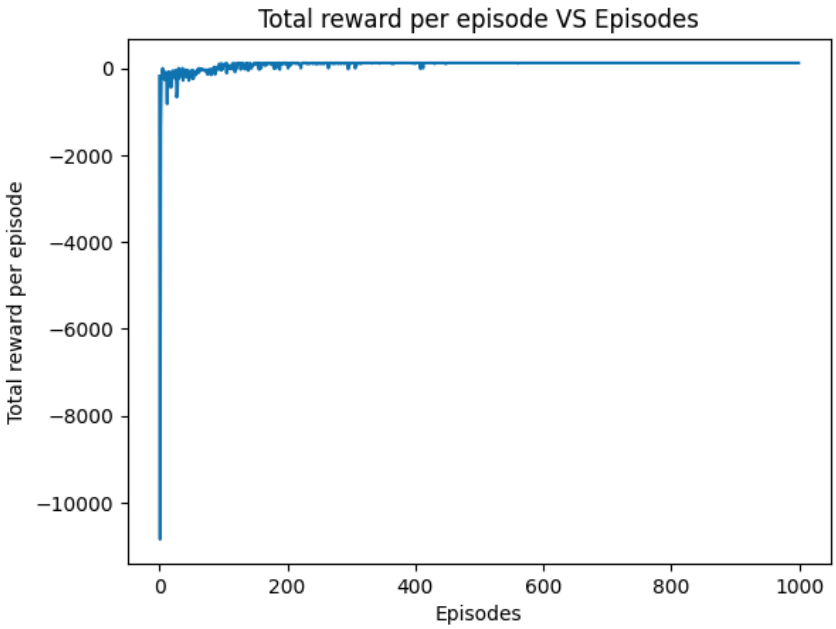


A GIF of the running scenario is attached below and included in the ZIP folder by the name “**warehouse_robot.gif**”.

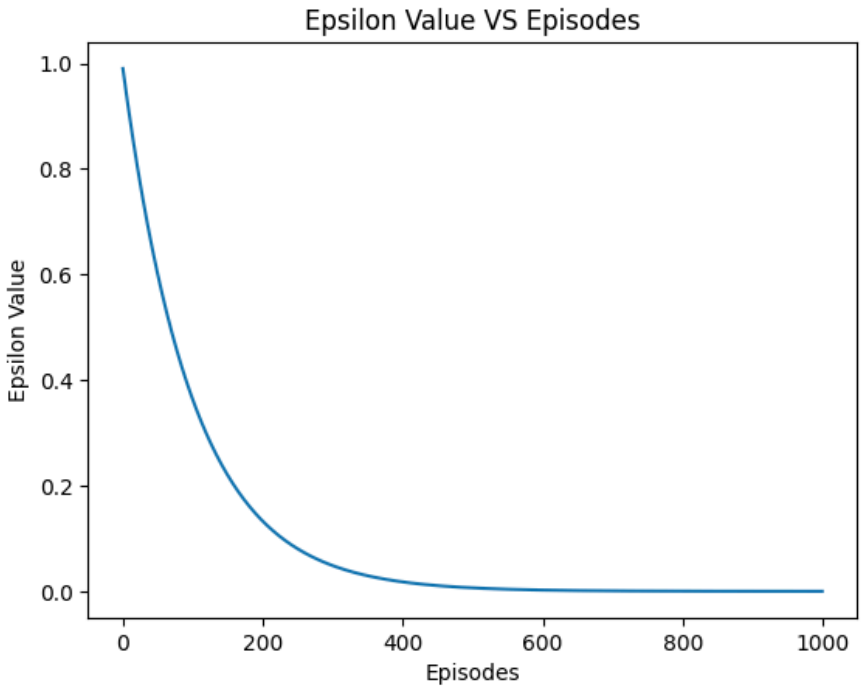
Part II

1. For Q-learning,

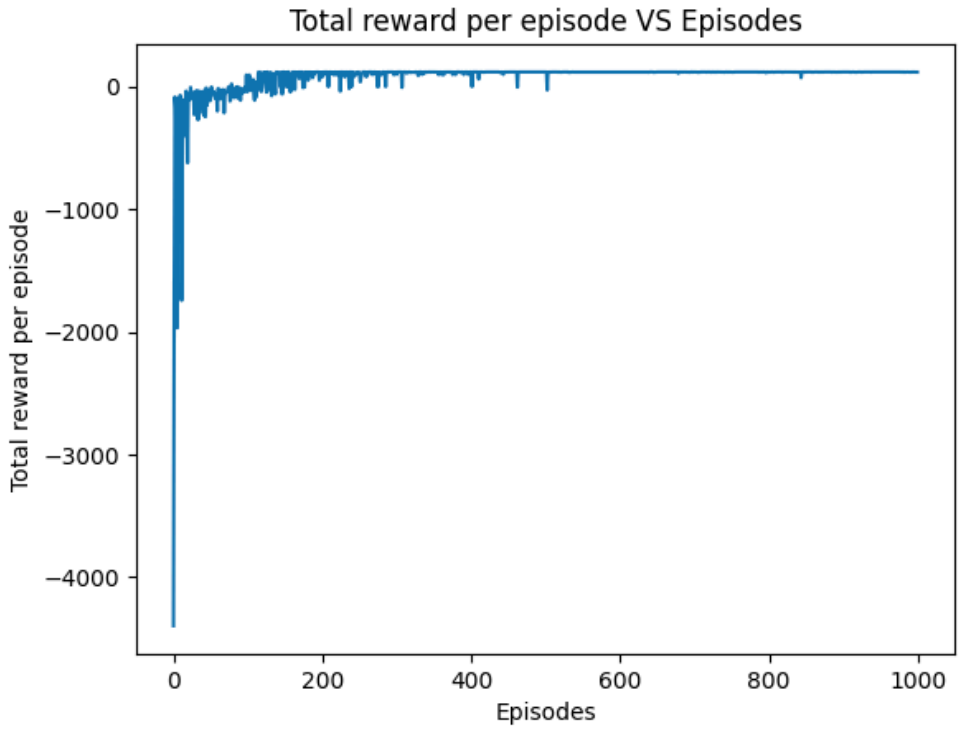
For the **deterministic environment**, the graph of Total reward per episode VS Episodes.



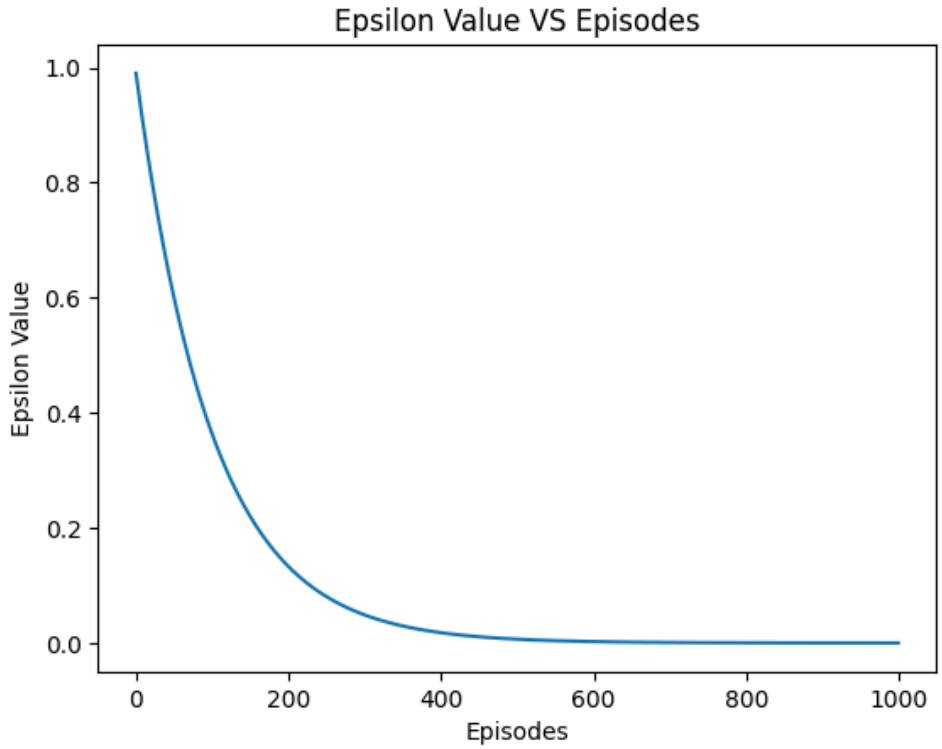
The graph for Epsilon decay.



For the **stochastic environment**, the graph of Total reward per episode VS Episodes.



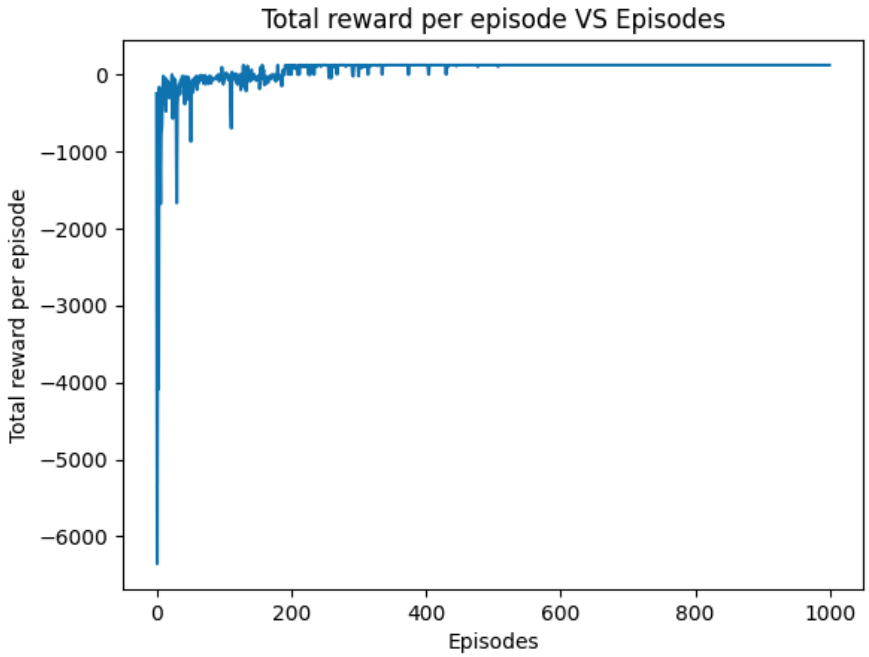
The graph for Epsilon decay.



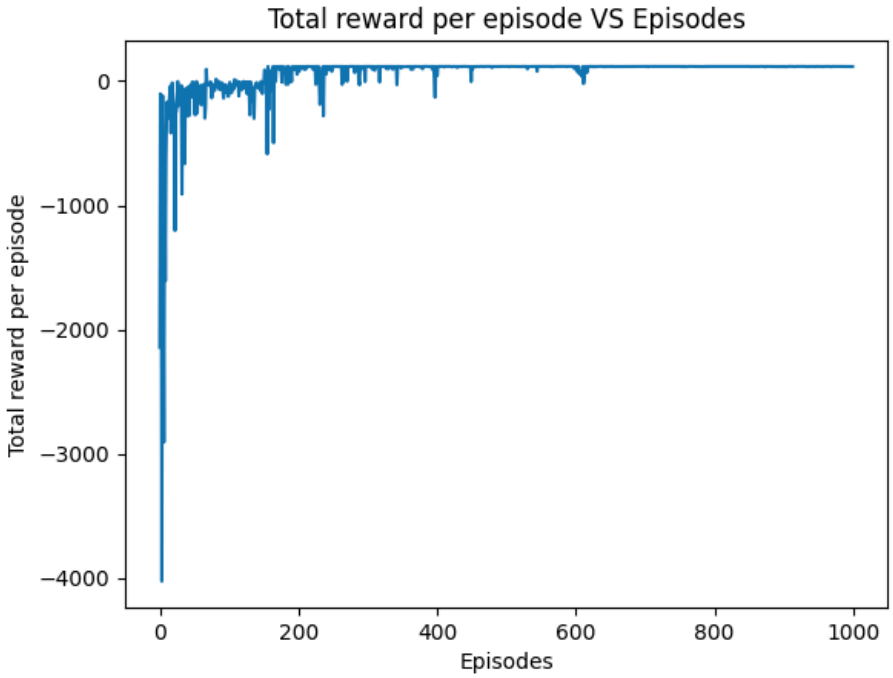
As we can see, there is a lot more noise in the Total Rewards per episode graph in the stochastic environment compared to the deterministic environment.

Using **Double Q-Learning**,

For a **deterministic environment**, the graph of Total reward per episode VS Episodes.

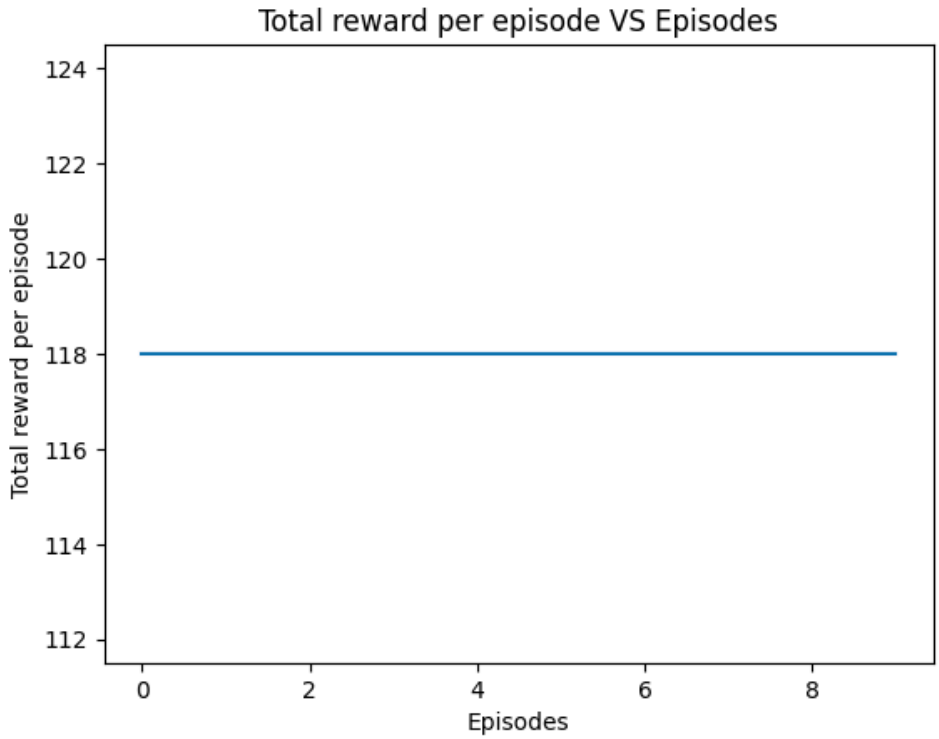


For a **stochastic environment**, the graph of Total reward per episode VS Episodes.

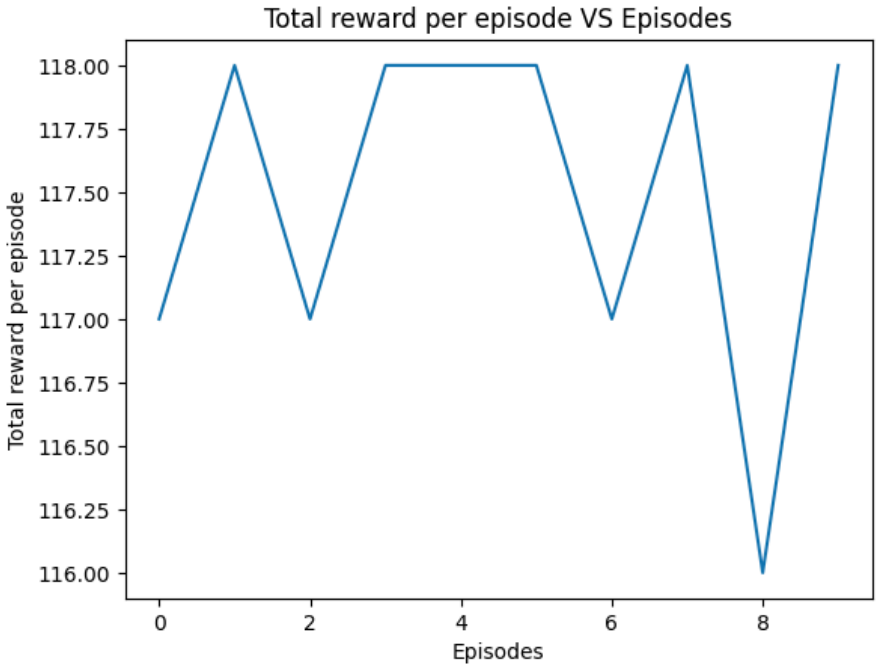


Here also, we can see there is a lot of noise in the stochastic environment compared to the deterministic environment in the Total rewards per episode graph.

Running the environments for 10 episodes, the outcome from the **Q-learning** for **deterministic environment** is as follows:



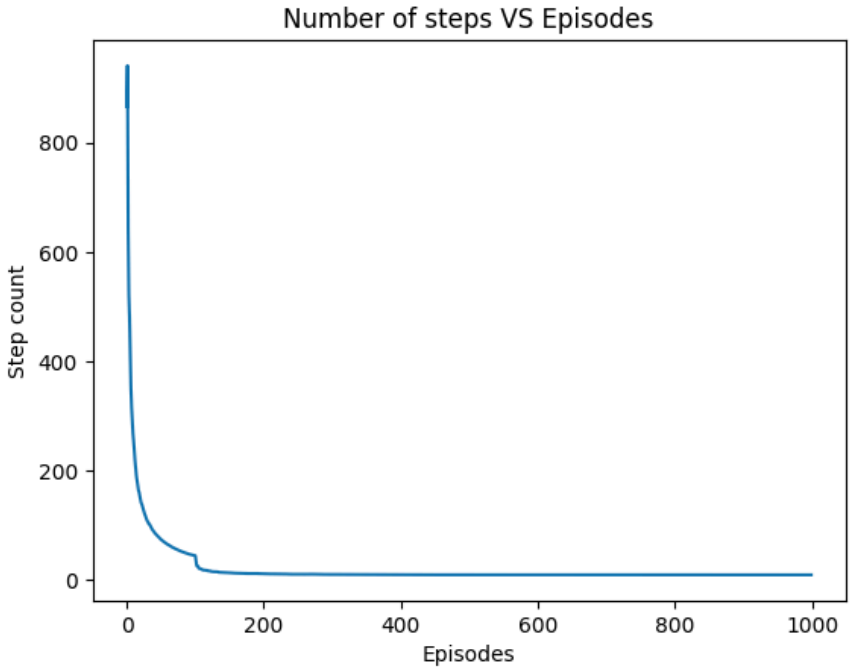
For **Q-learning**, the graph for the **stochastic environment** is as follows.



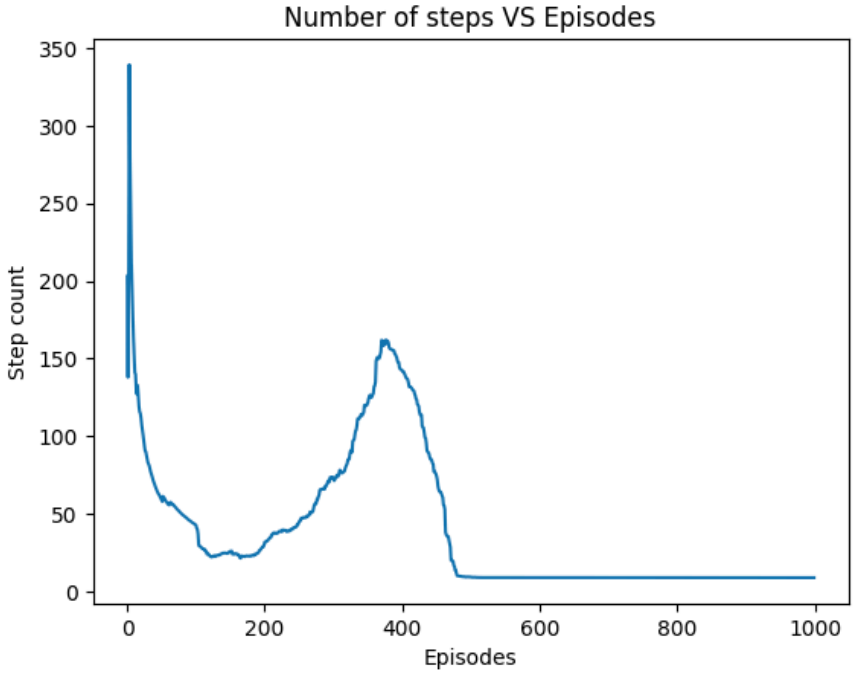
As we can see, for a deterministic environment, the reward is always the same throughout the episodes as the steps taken by the robot are fixed. For the stochastic environment, the reward varies as the steps taken by the robot are not fixed.

2. Comparing the performance of a **deterministic environment** for **Q-learning** and **Double Q-learning**.

We see the number of steps taken per episode by **Q-Learning** algorithm.



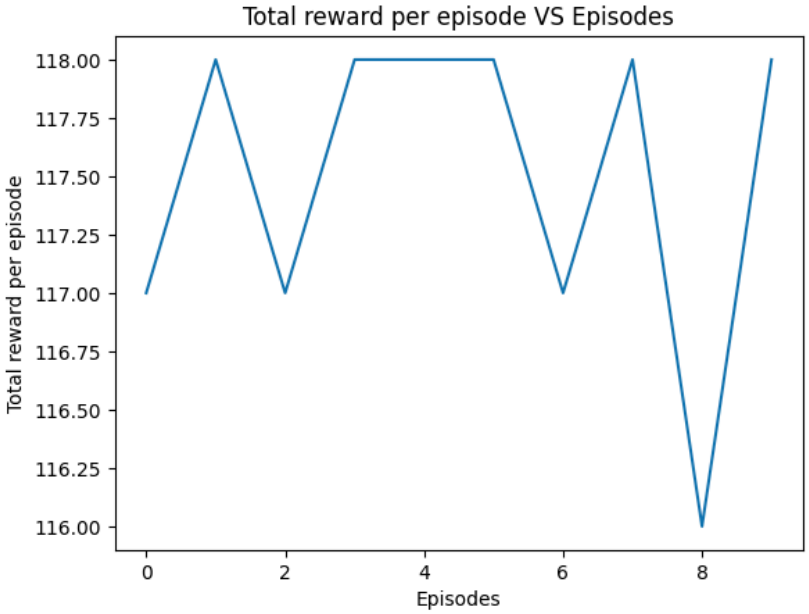
Now, we see the number of steps taken per episode by the **Double Q-Learning** algorithm.



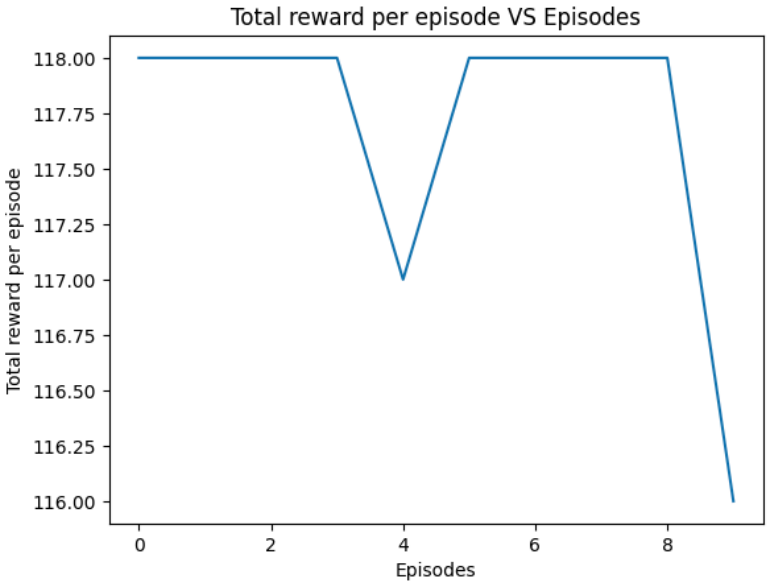
Here, we can see that Q-learning converges faster and is able to find an optimal path faster compared to Double Q-learning. So, we can conclude that **for a deterministic environment, Q-learning performs better compared to the Double Q-learning algorithm**.

3. Comparing the performance of a **stochastic environment** for **Q-learning** and **Double Q-learning**.

We see the Total rewards per episode by **Q-Learning** algorithm.



Now, we see the Total rewards per episode by **Double Q-Learning** algorithm.



As we can see, the fluctuation in the rewards for Q-learning is more compared to the Double Q-Learning algorithm. This shows that Double Q-learning is more stable compared to Q-learning. Thus, we can conclude that **Double Q-Learning performs better in Stochastic environment compared to Q-Learning**.

4. The tabular methods used to solve the problem are as follows:

a) Q-Learning

Q-learning is a value-based reinforcement learning algorithm that updates Q-values by considering the maximum possible future reward, making it an off-policy method.

Update Function:

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

Key Features:

- **Off-policy:** Learn using a greedy policy, even if exploration is used.
- Fast convergence in deterministic environments.
- Overestimates Q-values, which can lead to suboptimal paths in stochastic environments.
- Suitable for small grids where learning speed is more important than overestimation errors.

b) Double Q-Learning

Double Q-learning addresses Q-learning's overestimation bias by maintaining two Q-tables, using one to select the action and the other to evaluate it.

Update Function:

For table A,

$$Q_A(s,a) \leftarrow Q_A(s,a) + \alpha[r + \gamma QB(s', \text{argmax } Q_A(s',a')) - Q_A(s,a)]$$

For table B,

$$Q_B(s,a) \leftarrow Q_B(s,a) + \alpha[r + \gamma Q_A(s', \text{argmax } Q_B(s',a')) - Q_B(s,a)]$$

Key Features:

- Reduces overestimation bias, leading to more accurate Q-values.
- More stable in stochastic environments, where rewards or transitions are uncertain.
- Converges more reliably in large state spaces.
- Slightly slower than Q-learning due to maintaining two Q-tables.

5. A good reward function

A well-designed reward function is **crucial** for reinforcement learning because it directly influences the agent's behavior. Below are the key criteria for an effective reward function:

a. **Goal-Oriented:**
- ❖ The reward should align well with the objective of the task.
- ❖ Example: In the warehouse robot scenario, the goal is to pick up and drop off items efficiently, so rewards should encourage shortest paths and successful deliveries.

b. **Sparse vs. Dense Rewards:**
- ❖ Sparse rewards can make learning slow.
- ❖ Dense rewards help guide the agent more smoothly.

c. **Avoid Reward Hacking:**
- ❖ Ensure the agent does not exploit loopholes in the reward structure.
- ❖ Example: If the robot gets +1 per move, it might take longer paths to collect more rewards instead of being efficient.

d. **Balance Between Positive and Negative Rewards:**
- ❖ Encourage desired actions (positive rewards) and discourage bad ones (negative rewards).

e. **Smooth Learning Curve:**
- ❖ The agent should receive consistent feedback to learn effectively.
- ❖ Extremely high or low reward magnitudes can cause instability in learning.

For our model, we were using a reward function mentioned in this report. In order to enhance the performance of the model we made minor changes to the existing reward function. Some of the changes are -

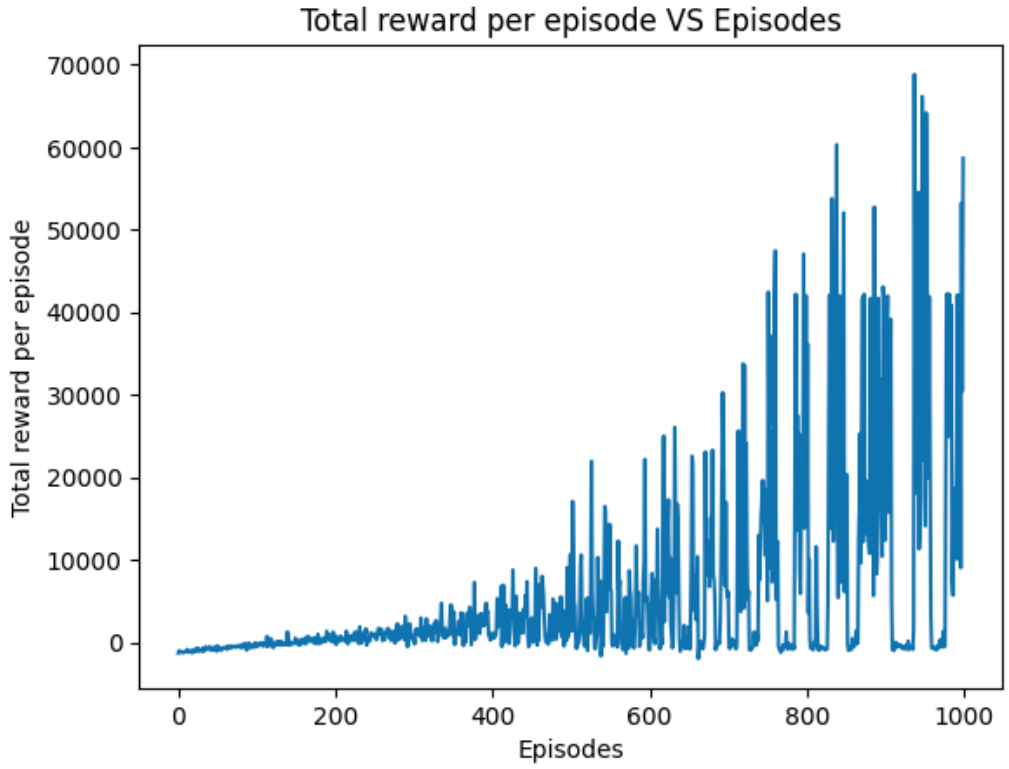
a. Returning -10 reward for when the robot attempts to move out of the specified grid of 6x6. This helped the robot to reduce the number of invalid steps taken by the robot.

b. Returning -25 reward for invalid PICKUP and DROPOFF. This helped the robot to learn not to have PICKUP and DROPOFF actions at invalid positions. The PICKUP must happen at the package location and the DROPOFF must happen at the target.

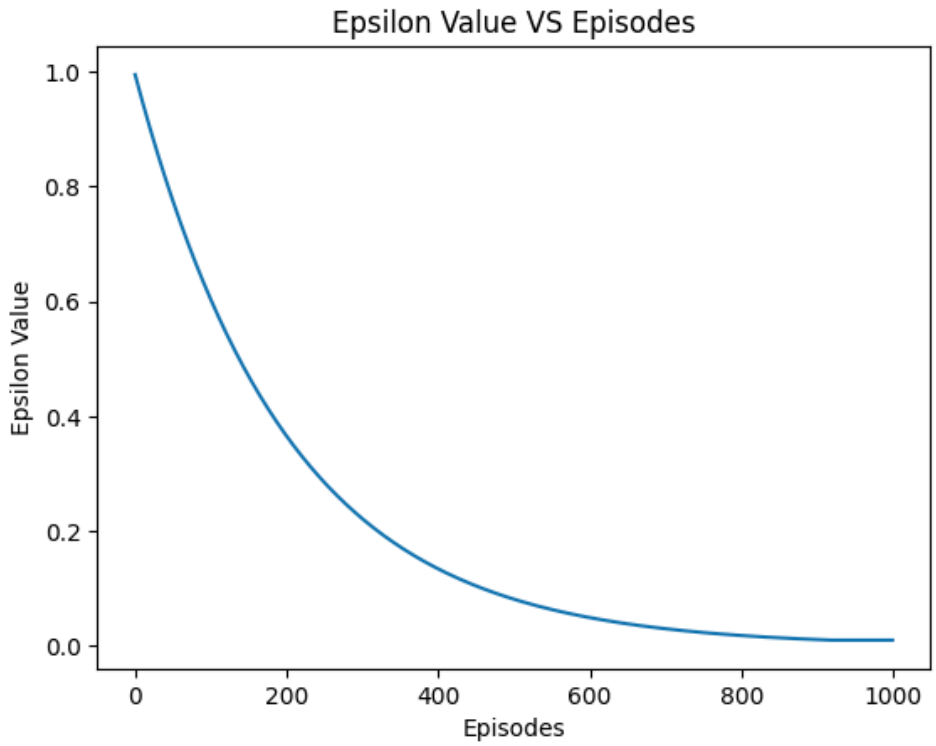
Using the above enhancements, the robot was able to attain a stable policy faster. This improved the efficiency of the training process.

Part III

1. After applying Q-Learning the plots for Epsilon Decay and Total rewards per episode is as follows:

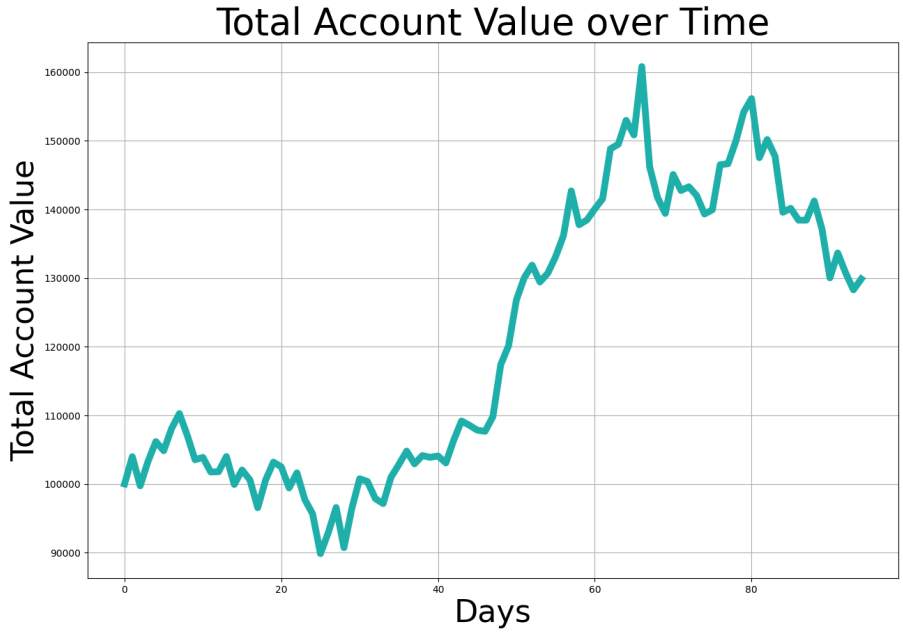


As we can see, the rewards per episode is increasing as the number of episodes increases. This shows that the profit for the agent will increase with time.



This shows a constant decay in the epsilon exponentially. Comparing the two graphs, we see a gradual increase in the Total Rewards per episode as the epsilon decays. So, the rate of epsilon decay works well with the total rewards.

2. Here, we try to render the trained model for 100 days and check the value of the invested amount (i.e., $ 100,000)



As we can see here, the value of the invested amount has increased significantly over the years. The invested amount is about $130,000 after 100 days of investment, which is about 30% profit on the invested amount.

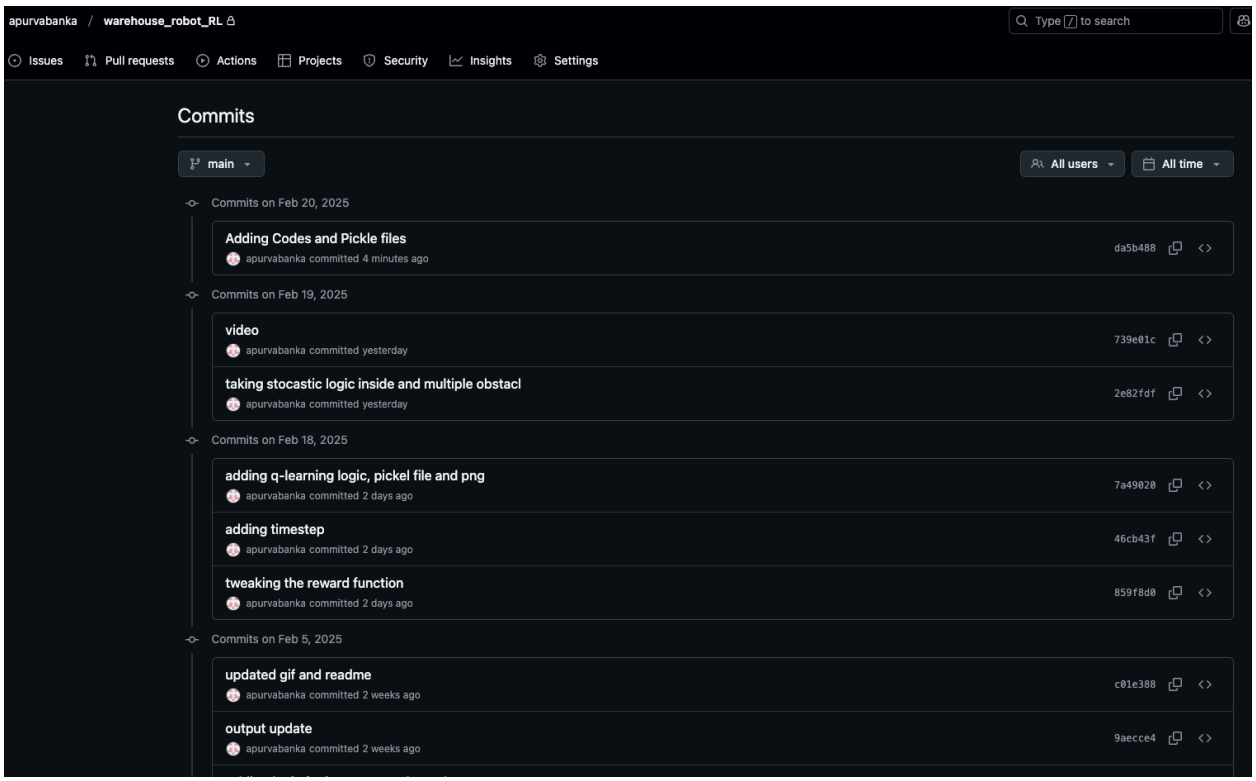
Thus, we can conclude the model is working as intended and we are able to generate profit from the invested amount.

Github

A github project for the complete assignment is created. The link to the Github Repo is as follows.

https://github.com/apurvabanka/warehouse_robot_RL

A screenshot of the commit history is as follows.



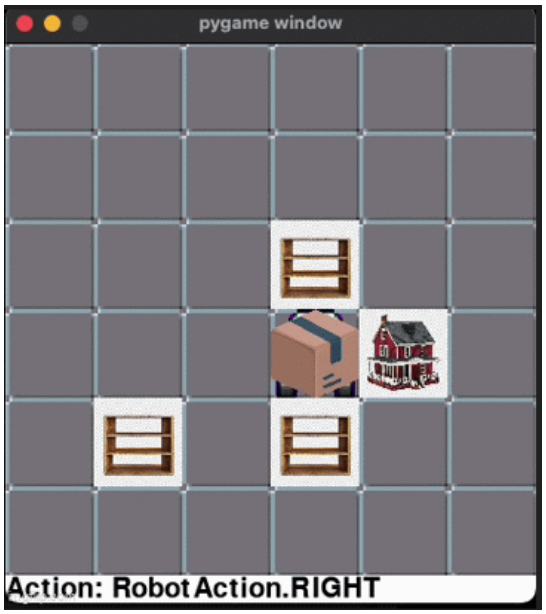
Grid-World Scenario Visualization

A visualisation of the environment is created using the ‘pygame’ python library. Below are the screenshots attached for the env.

This is an initialisation of the environment.

The package is the source, the shelf is the obstacle and the home is the target.

Here, the agent has picked up the package and the next task of the agent is to drop it to the target.



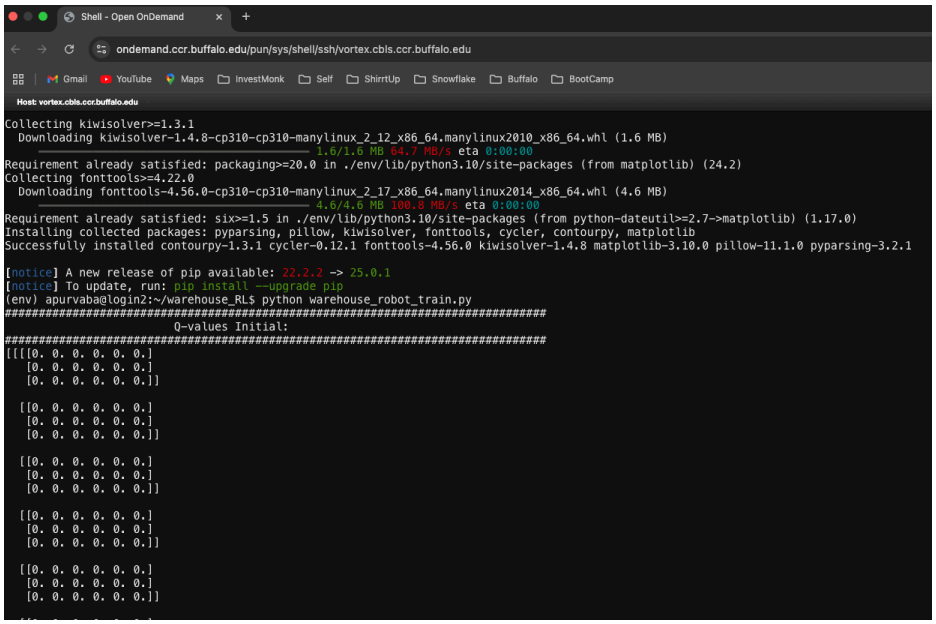
A GIF of the running scenario is attached below and included in the ZIP folder by the name

- “**warehouse_robot.gif**” for deterministic Q-Learning algorithm.
- “**warehouse_robot_double_q_learning.gif**” for deterministic Double Q-Learning algorithm.

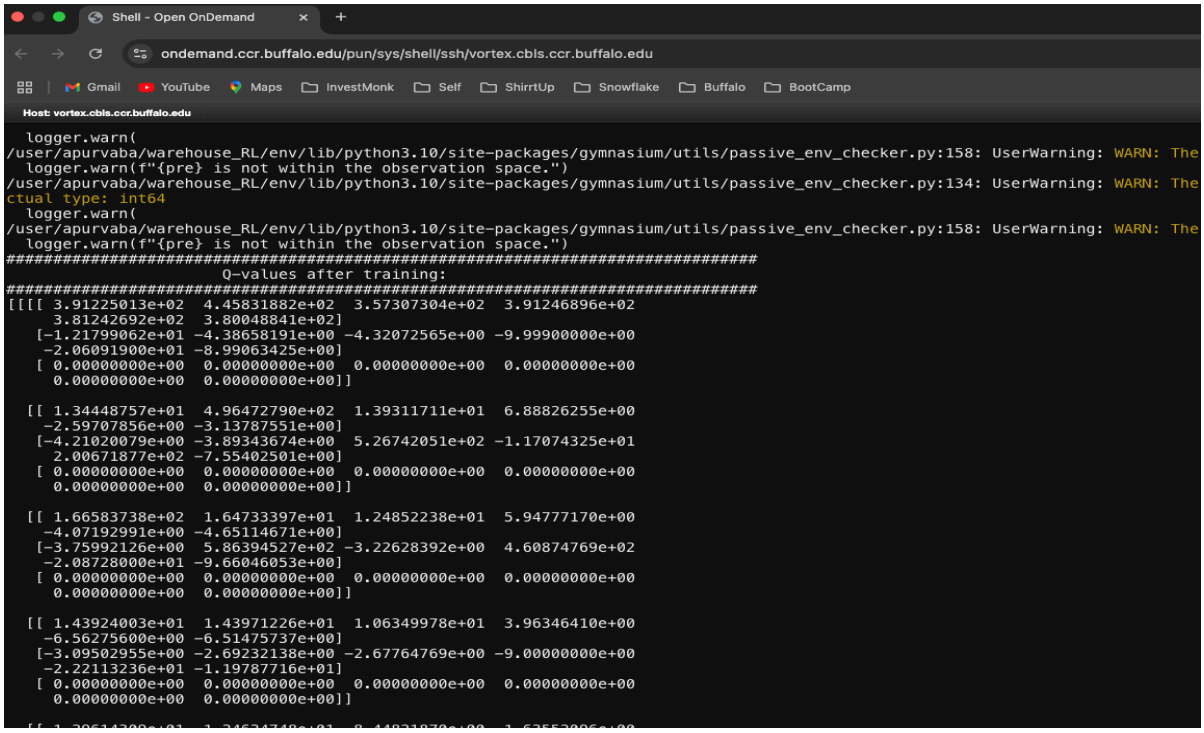
CCR Submission

The code was successfully executed on a CCR instance. The code is run on the personal instance. After creating a virtual environment, installing the required libraries, the model is trained for 1000 episodes. The model is rendered for 1 episode.

A screenshot of the same is attached below.



The initial and final values for Q-tables are displayed in the console.


Shell - Open OnDemand
ondemand.ccr.buffalo.edu/pun/sys/shell/ssh/vortex.cbls.ccr.buffalo.edu
Gmail YouTube Maps InvestMonk Self ShirrtUp Snowflake Buffalo BootCamp
Host: vortex.cbls.ccr.buffalo.edu
logger.warn(
/user/apurvaba/warehouse_RL/env/lib/python3.10/site-packages/gymnasium/utils/passive_env_checker.py:158: UserWarning: WARN: The
logger.warn(f"{pre} is not within the observation space.")
/user/apurvaba/warehouse_RL/env/lib/python3.10/site-packages/gymnasium/utils/passive_env_checker.py:134: UserWarning: WARN: The
ctual type: int64
logger.warn(
/user/apurvaba/warehouse_RL/env/lib/python3.10/site-packages/gymnasium/utils/passive_env_checker.py:158: UserWarning: WARN: The
logger.warn(f"{pre} is not within the observation space.")
Q-values after training:
[[[[3.91225013e+02 4.45831882e+02 3.57307304e+02 3.91246896e+02
3.81242692e+02 3.80048841e+02]
[-1.21799062e+01 -4.38658191e+00 -4.32072565e+00 -9.99900000e+00
-2.06091900e+01 -8.99063425e+00]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00]]
[[1.34448757e+01 4.96472790e+02 1.39311711e+01 6.88826255e+00
-2.59707856e+00 -3.13787551e+00]
[-4.21020079e+00 -3.89343674e+00 5.26742051e+02 -1.17074325e+01
2.00671877e+02 -7.55402501e+00]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00]]
[[1.66583738e+02 1.64733397e+01 1.24852238e+01 5.94777170e+00
-4.07192991e+00 -4.65114671e+00]
[-3.75992126e+00 5.86394527e+02 -3.22628392e+00 4.60874769e+02
-2.08728000e+01 -9.66046053e+00]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00]]
[[1.43924003e+01 1.43971226e+01 1.06349978e+01 3.96346410e+00
-6.56275600e+00 -6.51475737e+00]
[-3.09502955e+00 -2.69232138e+00 -2.67764769e+00 -9.00000000e+00
-2.22113236e+01 -1.19787716e+01]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00]]


Shell - Open OnDemand
ondemand.ccr.buffalo.edu/pun/sys/shell/ssh/vortex.cbls.ccr.buffalo.edu
Gmail YouTube Maps InvestMonk Self ShirrtUp Snowflake Buffalo
Host: vortex.cbls.ccr.buffalo.edu
RobotAction.RIGHT
RobotAction.DOWN
RobotAction.DOWN
RobotAction.RIGHT
RobotAction.RIGHT
RobotAction.DROPOFF

The model is rendered for 1 episode and the outcome is displayed in the console as we can see above. Here, S represents the package to be picked, R represents robot, O represents obstacles/shelves and T presents the target. The whole grid is denoted by "_" for each state.

Reference

- Sutton & Barto - Reinforcement Learning: An Introduction (2nd Edition)
    - Used definitions and pointers from this book.
- Github - https://github.com/johnnycode8/gym_custom_env
    - Used this a basis for designing the GYM environment
    - Used the implementation for PyGame to visualize the environment.