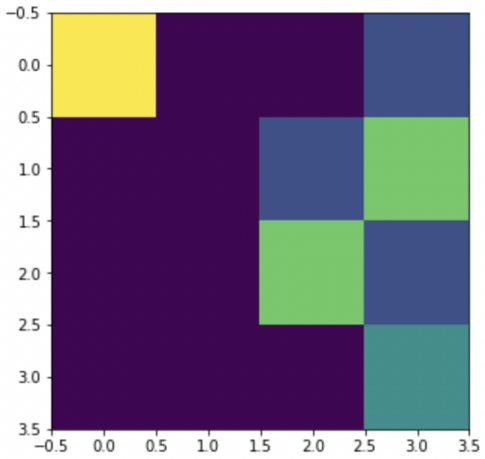
Github link for bonus points:
https://github.com/AnurimaVaishnavi/cse546assignment1

1. Describe the deterministic and stochastic environments, which were defined (set of actions/states/rewards, main objective, etc)

The grid environment class is initialized using the OpenAI gym Environment class. I have implemented the functions like render,step, init.

```
(array([1.  , 0.  , 0.  , 0.25, 0.  , 0.  , 0.25, 0.75, 0.  , 0.  , 0.75,
        0.25, 0.  , 0.  , 0.  , 0.5 ]), {})
```



The number of states : 16. I defined a 4x4 grid world.
We have an action space of 4 different actions - [UP,DOWN,RIGHT,LEFT]

1) Down, action = 0
2) Up, action = 1
3) Right, action = 2
4) Left, action = 3

The number of rewards:

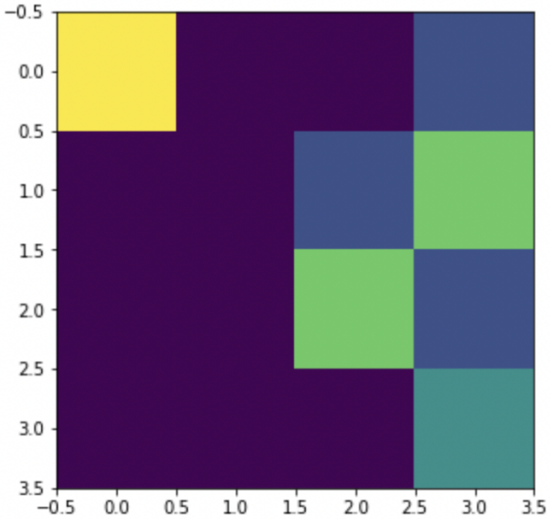
a) Based on the whether the agent is inside the grid or is going out of bounds it is awarded with one of the three values [0,-1,-2]
b) Based on proximity value, the agent is awarded with [-1,0,1]

c) If the agent reaches the goal position, he is rewarded with 10 points
d) If the agent lands up in the bonus2 position and is closer to the goal position than it was before, the agent is rewarded with +5 points. Additionally, if the environment is stochastic with a probability less than 0.5, its awarded with an additional +2 points
e) If the agent lands up in the bonus3 position and is closer to the goal position than it was before, the agent is rewarded with 5 points. Additionally, if the environment is stochastic with a probability less than 0.5, its awarded with an additional 2 points
f) If the agent lands up in the bonus1 position and is closer to the goal position than it was before, the agent is rewarded with 5 points. Additionally, if the environment is stochastic with a probability less than 0.5, its awarded with an additional +2 points
g) If the agent lands up in a pit position, it's rewarded with -5 points. Additionally, if the environment is stochastic with a probability less than 0.75, the agent is given an extra reward of -3

2. Provide visualizations of your environments.

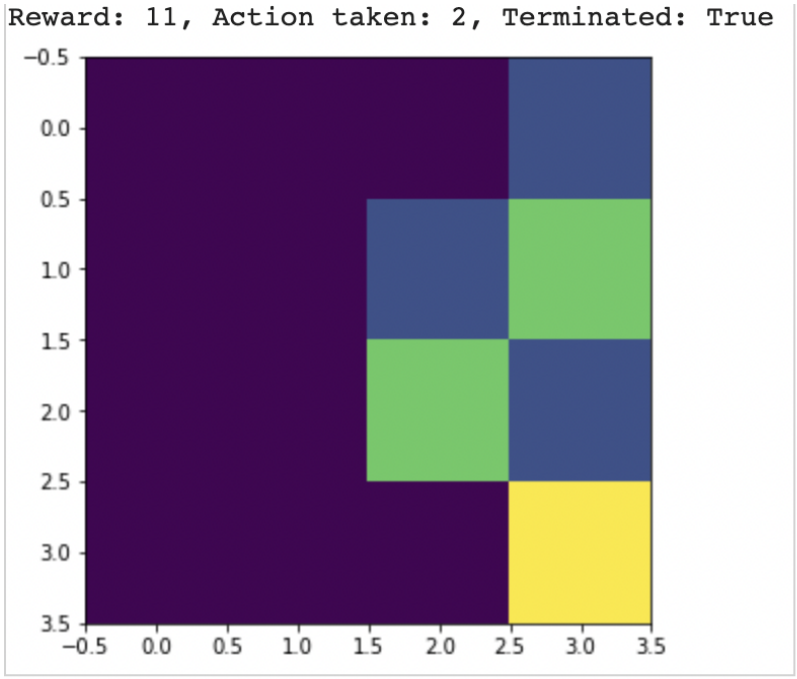
I have an observation space of 4x4

```
(array([1.  , 0.  , 0.  , 0.25, 0.  , 0.  , 0.25, 0.75, 0.  , 0.  , 0.75,
        0.25, 0.  , 0.  , 0.  , 0.5 ]), {})
```



For both environments, the agent's initial position is at the top-left corner [0,0] and the goal position is at the bottom right corner positioned at [3,3]. The environment has three positions and 2 pit/hole positions. Using the render function of the environment, we plot the environment. The yellow block represents the agent initial position

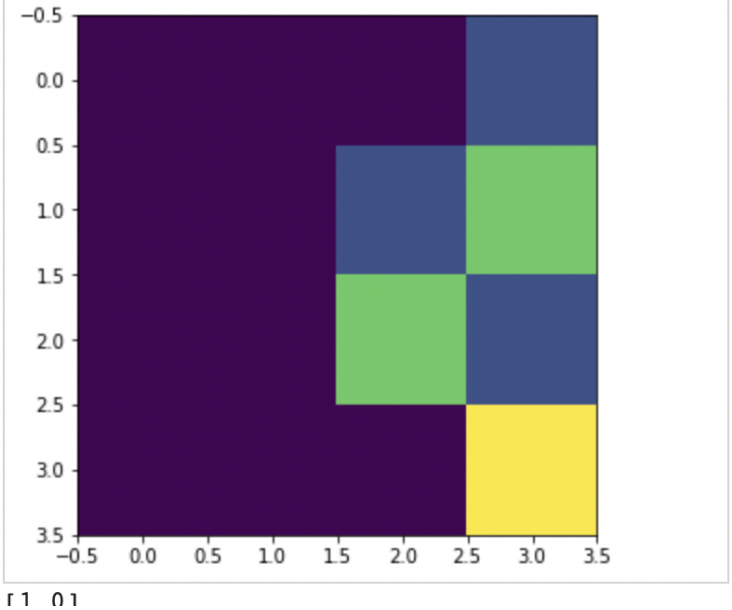
The below is the picture of the random agent in the deterministic environment after reaching the termination state :



[1 0]

The below is the picture of the random agent in the stochastic environment after reaching the termination state :

```
Reward: 1, Action taken: 3, Terminated: True
{'action_given': 3, 'action_taken': 2}
```



[1 0]

3. How did you define the stochastic environment?

The stochastic environment has a stochastic agent and receives stochastic rewards. It receives stochastic rewards at both the bonus and the pit positions based on the probability of it being stochastic.

The stochastic environment has the following properties:

1.State Space - 4x4 grid space
2. Action space - [right,left,up,down]
3. Reward Space ={ 10,+5,+2,-5,-3,-1,0,1}

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

The deterministic environment has the following properties:

1.State Space - 4x4 grid space

2. Action space - [right,left,up,down]
3. Reward Space ={ 10,+5,-5,-1,0,1}

The stochastic environment has a stochastic agent

1) The stochastic agent takes the action given with a probability greater than 0.75.
2) The agent takes a random action from the action space with a probability of 0.25.
3) The agent takes an additional reward +2 on landing a bonus position if the probability of the environment is less than 0.5
4) The agent takes an additional negative reward -3 on landing the pit positions if the probability of the environment is less than 0.75

5. Safety in AI: Write a brief review (~ 5 sentences) explaining how you ensure the safety of your environments. E.g. how do you ensure that agent choose only actions that are allowed, that agent is navigating within defined state-space, etc.

1) The agent selects actions randomly from the action space of {up,down,left,right}.

Even in the stochastic environment, with a probability 0.75 the agent does the action that is given and with a probability 0.25 the agent randomly selects an action from the action space defined by the environment ensuring safety.

2) To make sure the agent doesn't go out of the bounds of the grid, I clipped the coordinates of the agent to be in the range [0,3] and [0,3] for x-coordinates and y-coordinates respectively. This ensures the safety of the environment and the agent by restricting it to stay within the constraints.

Part 2

1) **Applying Q-learning to solve the deterministic environment defined in Part 1. Plots should include epsilon decay and total reward per episode**

The hyperparameters I chose to tune are

a) Discount factor (γ)
b) Number of episodes

The following are 6 graphs with different combinations of Discount factor and Number of episodes.

I used three different values for each of the hyperparameter:

1)

```
gamma value:
0.9577833503522386
episodes
2479
```

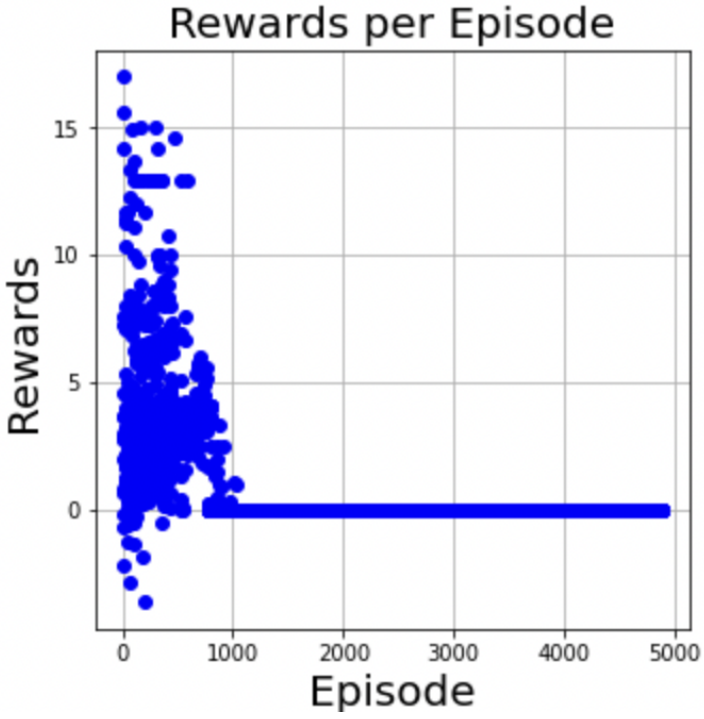
2)

```
gamma value:
0.9558072010833316
episodes
2923
```

3)

```
gamma value:
0.9345682990266965
episodes
2299
```

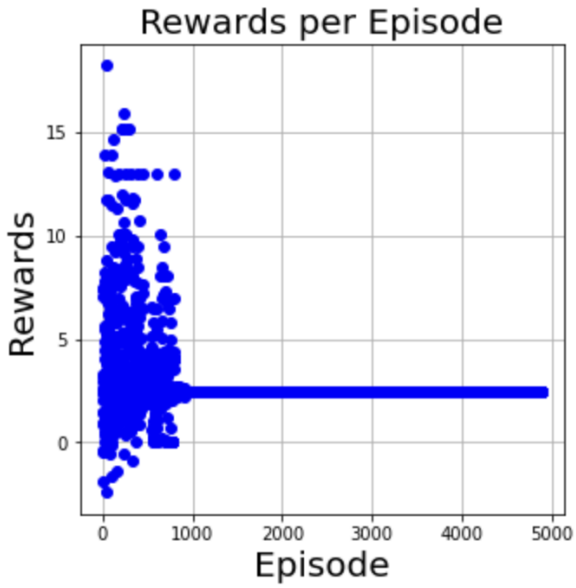
```
gamma value:
0.9016726342185128
episodes
4893
```




```
gamma value:
0.9414242888439699
episodes
2336
```

5)

```
gamma value:
0.9028347941299762
episodes
4891
```

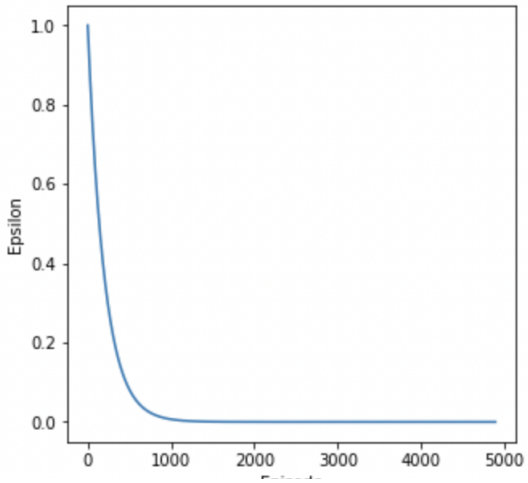


6)

Epsilon Decay for the graph:

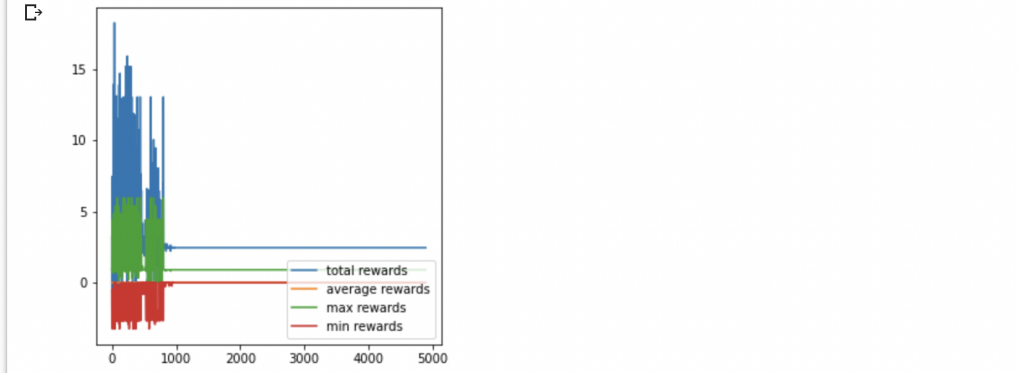
```
#Epsilon Decay Plot for Training
plt.rcParams['figure.figsize'] = (5, 5)
plt.plot(range(len(epsilon_array)), epsilon_array)
plt.xlabel('Episode')
plt.ylabel('Epsilon')
```

```
Text(0, 0.5, 'Epsilon')
```



The below graphs plot the maximum, minimum, total, and average rewards for 5000 episodes in the Deterministic Environment using Q-Learning tabular method:

```
#Graphs for Total Rewards/Average Rewards/Max Rewards/ Min rewards for training
plt.plot(dict['ep'], reward_array  ,label="total rewards")
plt.plot(dict['ep'], dict['avg'],label="average rewards")
plt.plot(dict['ep'], dict['max'],label="max rewards")
plt.plot(dict['ep'], dict['min'],label="min rewards")
plt.legend(loc=4)
plt.show()
```



2) **Applying Q-learning to solve the stochastic environment defined in Part 1. Plots should include epsilon decay and total reward per episode.**

The hyperparameters I chose to tune are

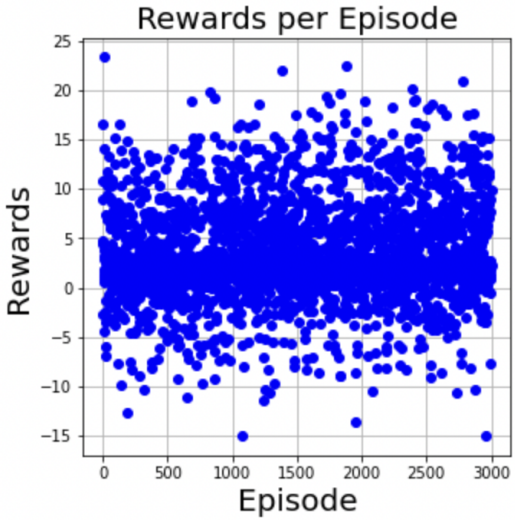
c) Discount factor (γ)
d) Number of episodes

The following are 6 graphs with different combinations of Discount factor and Number of episodes.

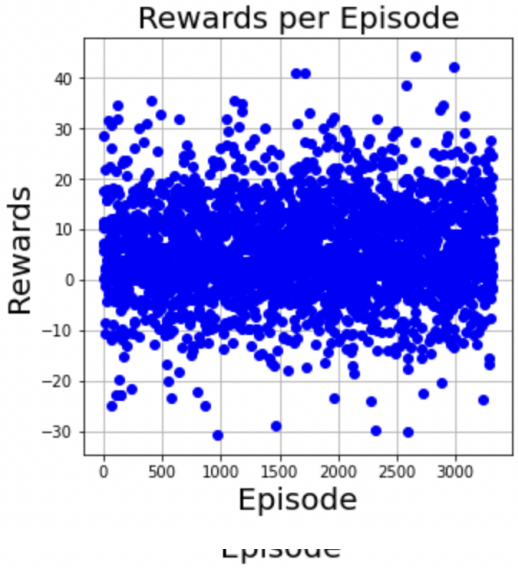
I used three different values for each of the hyperparameter:

```
gamma value:
0.9820769910058825
episodes
3866
```


```
gamma value:
0.914623146699755
episodes
3001
```



```
gamma value:
0.9727249733578349
episodes
3320
```




```
gamma value:
0.9037290334163204
episodes
4283
```


```
gamma value:
```

Episode

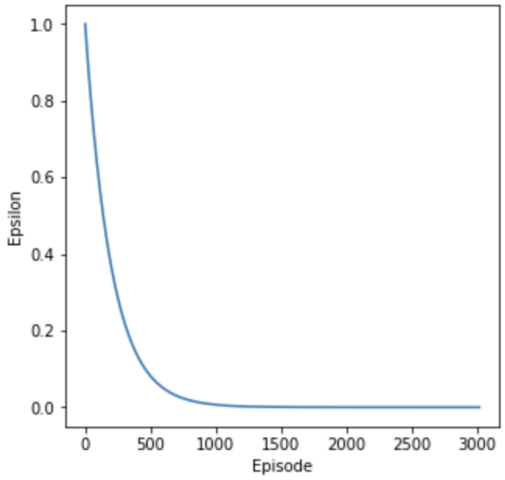
```
gamma value:
0.9201287845211024
episodes
4829
```


```
gamma value:
0.9216160014782118
episodes
3018
```

Epsilon Decay for the graph:

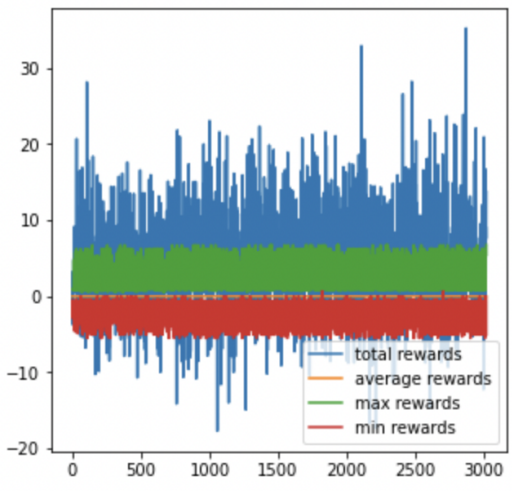
```
plt.rcParams['figure.figsize'] = (5, 5)
plt.plot(range(len(epsilon_array)), epsilon_array)
plt.xlabel('Episode')
plt.ylabel('Epsilon')
```

```
Text(0, 0.5, 'Epsilon')
```



The below graphs plot the maximum, minimum, total, and average rewards for 5000 episodes in the Deterministic Environment using Q-Learning tabular method:

```
#Max Min Avg graph for Stochastic Environment
plt.rcParams['figure.figsize'] = (5, 5)
plt.plot(dict['ep'], reward_array , label="total rewards")
plt.plot(dict['ep'], dict['avg'], label="average rewards")
plt.plot(dict['ep'], dict['max'], label="max rewards")
plt.plot(dict['ep'], dict['min'], label="min rewards")
plt.legend(loc=4)
plt.show()
```



Applying any other algorithm of your choice to solve the deterministic environment defined in Part 1. Plots should include total reward per episode.

I chose to go with **Double Q-Learning Algorithm** since it considers two Q_tables instead of one

```
gamma value:
0.9412209274180048
episodes
3666
```


```
gamma value:
0.9412209274180048
episodes
3195
```

gamma value:

```
gamma value:
0.9412209274180048
episodes
3790
```

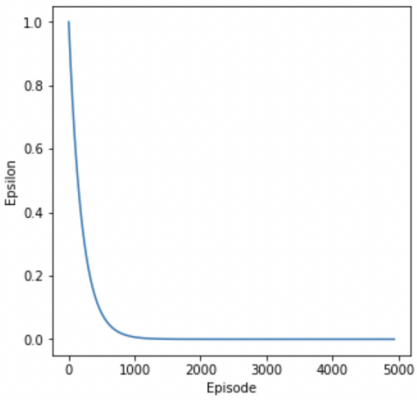

```
gamma value:
0.9412209274180048
episodes
3679
```

```
gamma value:
0.9412209274180048
episodes
4940
```

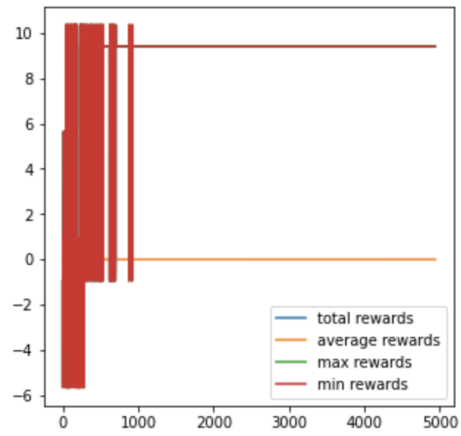
Epsilon Decay for the graph:

```
plt.rcParams['figure.figsize'] = (5, 5)
plt.plot(range(len(epsilon_array)), epsilon_array)
plt.xlabel('Episode')
plt.ylabel('Epsilon')
```

```
Text(0, 0.5, 'Epsilon')
```



```
#Plotting total/max/min/avg rewards for the Deterministic Environment
plt.rcParams['figure.figsize'] = (5, 5)
plt.plot(dict['ep'], reward_array , label="total rewards")
plt.plot(dict['ep'], dict['avg'], label="average rewards")
plt.plot(dict['ep'], dict['max'], label="max rewards")
plt.plot(dict['ep'], dict['min'], label="min rewards")
plt.legend(loc=4)
plt.show()
```



Applying Double Q Learning to solve the stochastic environment defined in Part 1. Plots should include total reward per episode.

```
gamma value:
0.9412209274180048
episodes
3865
```

gamma value:

```
gamma value:
0.9412209274180048
episodes
3605
```


```
gamma value:
0.9412209274180048
episodes
3299
```


```
gamma value:
0.9412209274180048
episodes
3996
```

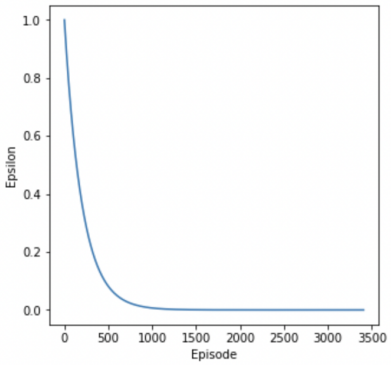
```
gamma value:
0.9412209274180048
episodes
3316
```


```
gamma value:
0.9412209274180048
episodes
3407
```

Epsilon decay graph is below:

```
plt.rcParams['figure.figsize'] = (5, 5)
plt.plot(range(len(epsilon_array)), epsilon_array)
plt.xlabel('Episode')
plt.ylabel('Epsilon')
```

```
Text(0, 0.5, 'Epsilon')
```

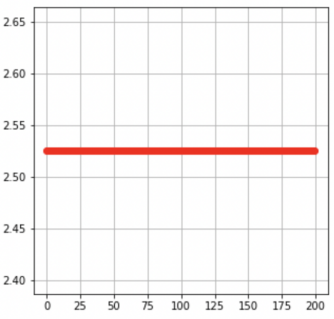


Provide the evaluation results. Run your environment for at least 10 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.

The evaluation graph for the Deterministic Environment using Q-learning:

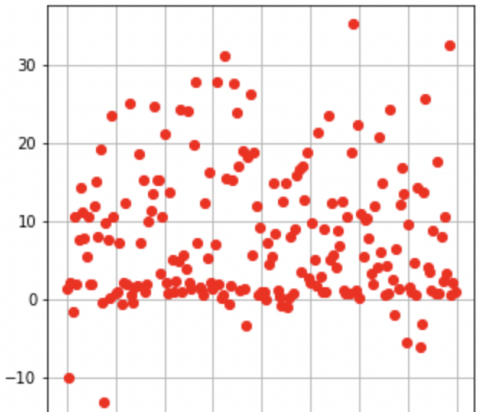
```
#Evaluation on my test environment
epsilon_array, reward_array , dict, alpha, gamma = model.fit(episodes=200, train=False)
print("gamma value:")
print(gamma)
print("episodes")
print(len(dict['ep']))
plt.plot(dict['ep'], reward_array, "ro")
plt.grid()
plt.show()
```

```
gamma value:
0.9163735951712031
episodes
200
```



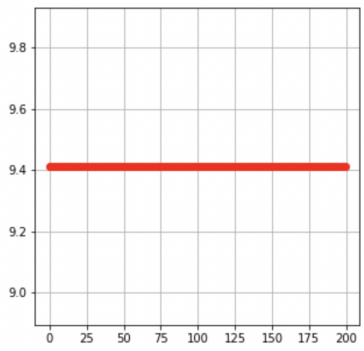
The evaluation graph for the Deterministic Environment using Q-learning:

```
[315] #Testing the learnt Q-Learning Algorithm on Stochastic Environment for the Grid World pro
      epsilon_array, reward_array , dict, alpha, gamma = model.fit(episodes=200, train=False,al
      plt.plot(dict['ep'], reward_array, "ro")
      plt.grid()
      plt.show()
```



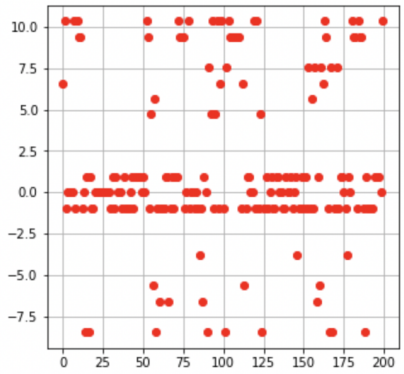
The evaluation graph for the Deterministic Environment using Double Q-learning:

```
#Testing the learnt Q-Learning Algorithm on Stochastic Environment for the Grid World problem
epsilon_array, reward_array , dict, alpha, gamma = model.fit(episodes=200, train=False)
plt.plot(dict['ep'], reward_array, "ro")
plt.grid()
plt.show()
```



The evaluation graph for the Stochastic Environment using Double Q-learning:

```
#Testing the learnt Q-Learning Algorithm on Stochastic Environment for the Grid World problem
epsilon_array, reward_array , dict, alpha, gamma = model.fit(episodes=200, train=False)
plt.plot(dict['ep'], reward_array, "ro")
plt.grid()
plt.show()
```



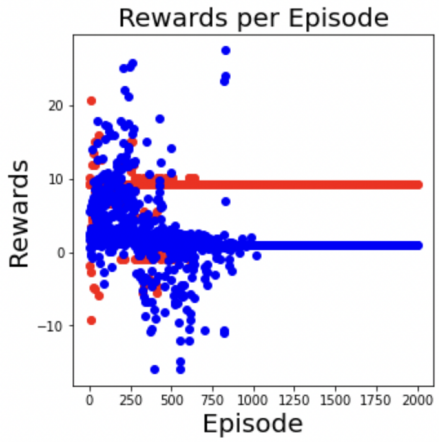
.

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.

For the deterministic environment, Double Q Learning is learning faster than Q-Learning. Double Q learning is also giving better rewards. The maximum rewards remain constant after a point of time for both the algorithms. If you remove the outliers, the maximum reward for Double Q Learning goes to a +10.

Deterministic Environment Comparison graph

```
#Comparison graph for deterministic environment - Q Learning vs Double Q Learning
env = GridEnvironment(stochastic = False, max_timesteps = 1000)
model1 = DoubleQLearning(env)
epsilon_array, reward_array , dict, alpha, gamma = model1.fit(episodes=2000,alpha = 0.15, gamma= 0.92)
rewards1, episodes1 = reward_array, dict['ep']
plt.plot(episodes1, rewards1,"ro", label="total rewards per episode")
model2 = QLearning(env)
epsilon_array, reward_array , dict, alpha, gamma = model2.fit(episodes=2000,alpha = 0.15, gamma= 0.92)
rewards1, episodes1 = reward_array, dict['ep']
plt.plot(episodes1, rewards1,"bo", label="total rewards per episode")
plt.xlabel('Episode',fontsize=20)
plt.ylabel('Rewards',fontsize=20)
plt.title("Rewards per Episode",fontsize=20)
plt.show()
```



Stochastic Environment Comparison graph:
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.

Double Q-Learning is collecting negative rewards initially. Rewards lie in the range of [-20,10] for Double Q Learning. Rewards lie in the range of [-15,30] for Q-Learning. If you do not consider the outliers, rewards lie between [-10,10] using both the algorithms

```
#Comparison graph for stochastic environment - Q Learning vs Double Q Learning
env = GridEnvironment(stochastic = True, max_timesteps = 1000)
model1 = DoubleQLearning(env)
epsilon_array, reward_array , dict, alpha, gamma = model1.fit(episodes=2000,alpha = 0.15, gamma= 0.92)
rewards1, episodes1 = reward_array, dict['ep']
plt.plot(episodes1, rewards1,"ro", label="total rewards per episode")
model2 = QLearning(env)
epsilon_array, reward_array , dict, alpha, gamma = model2.fit(episodes=2000,alpha = 0.15, gamma= 0.92)
rewards1, episodes1 = reward_array, dict['ep']
plt.plot(episodes1, rewards1,"bo", label="total rewards per episode")
plt.xlabel('Episode',fontsize=20)
plt.ylabel('Rewards',fontsize=20)
plt.title("Rewards per Episode",fontsize=20)
plt.show()
```

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

Q-Learning :

1) It is an offline policy algorithm
2) Q-Learning is model free
3) The update function of Q-Learning algorithm is as follows:
   Q(st,at) = Q(st,at) + alpha(r+gamma * argmax(st+1,at+1)- Q(st,at))
   st - current state
   at - current action
   gamma - discount factor
   alpha-learning rate
   st+1 -state you go from st taking the action at
   r-reward of taking (st,at)
4) You apply epsilon-greedy policy
5) Q-Learning helps us achieve long-term outcomes

Double Q-Learning :

1) It is also an offline policy algorithm
2) Double Q-Learning is model free
3) Double Q-Learning uses two Q tables
4) You apply epsilon greedy policy here as well
5) Double Q Learning performs better than Q-Learning. You achieve maximum rewards very easily.
6) The update function of Double Q Learning algorithm:
   Qa(S,A) = Qa(S,A) + alpha(r+Qb(Qa(argmax St+1), at+1) -Qa(S,A))
   Qb(S,A) = Qb(S,A) + alpha(r+Qa(Qb(argmax St+1), at+1) -Qb(S,A))

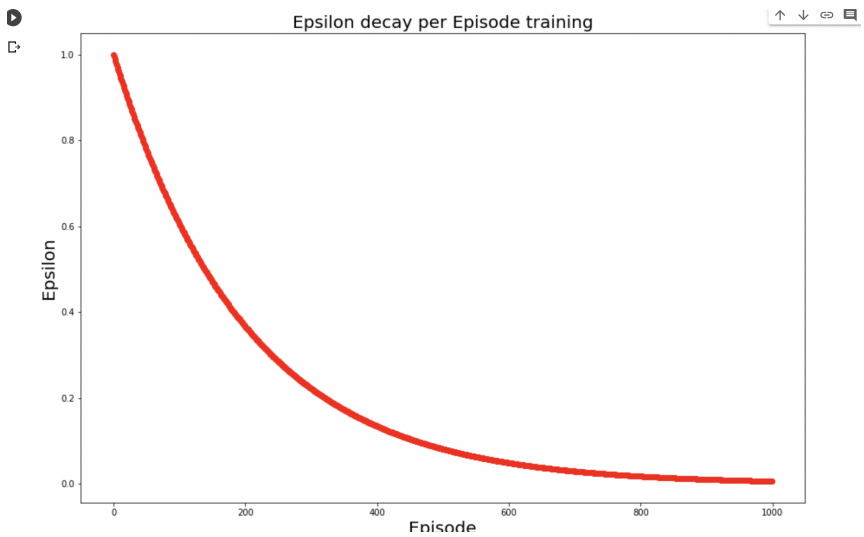
Briefly explain the criteria for a good reward function. If you tried multiple reward functions, give your interpretation of the results.

A good reward function makes the agent learn faster. Whenever we get a negative reward which means that the agent has fallen into one of the pit positions or is further from the goal than it was before, it understands that taking that action wasn't optimal. Hence, in the future the agent will rather take some other action when in the same position. The Q values of the (State, Action) will reduce if the agent is getting a negative reward.

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.

Epsilon Decay graph:

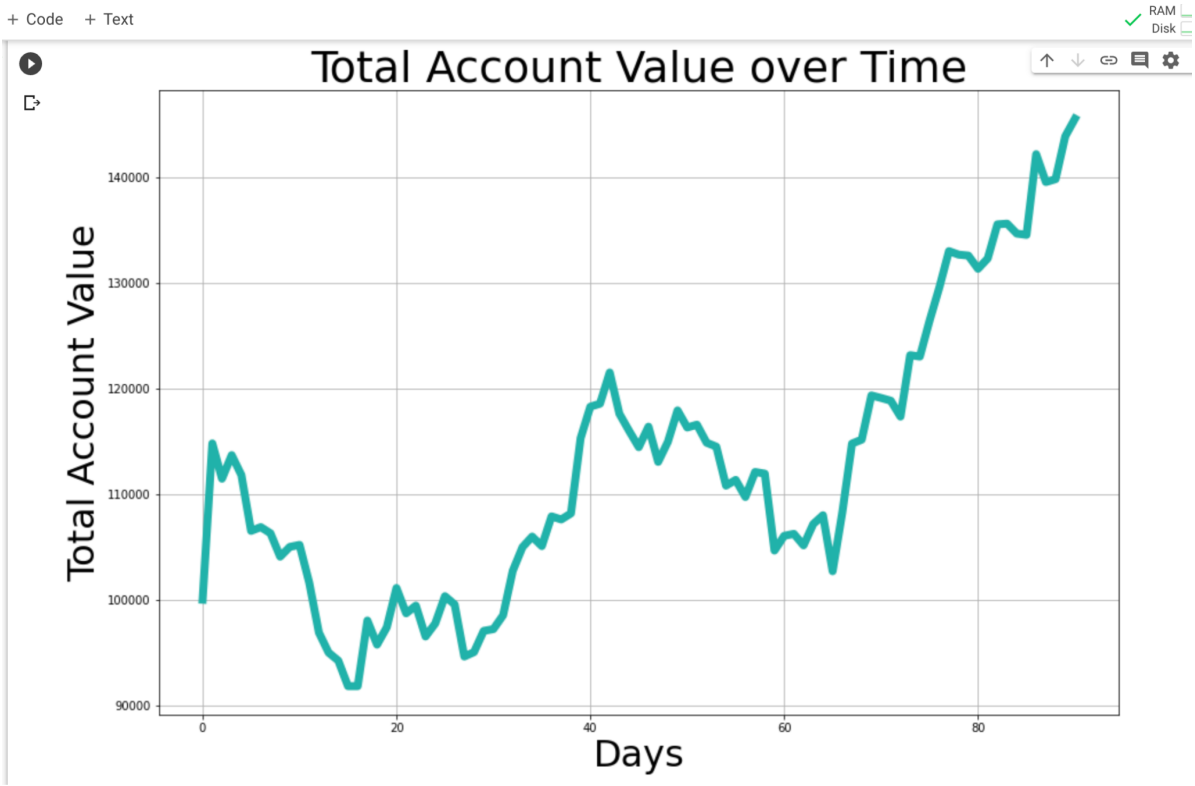
Epsilon graph is growing downwards. It's a logarithmic graph. At the end of every episode, you multiply epsilon with its decay factor.



Total Reward per Episode graph:

Rewards start at -40 initially i.e., for the first episode. Rewards graph is exponentially growing here indicating the Q-table is perfectly updating the values.

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.



References:

https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf