COL333 : ASSIGNMENT 3

Part – A

1. MDP Formulation

A) MDP state space has 650 states each one of them is 5-tuple described as – (A,B,C,D,E)

> A = X-coordinate of taxi position in grid
>
> B = Y- coordinate of taxi position in grid
>
> C = X-coordinate of passenger position in grid
>
> D = Y-coordinate of passenger position in grid
>
> E = Boolean value which represent if passenger is inside the taxi or not.
>
> The x,y values used in the states are according to the the picture shown in the problem. (i,e a = from bottom to up, b = from left to right)

B) Action space is represented as a list which contains all possible action in a state.
> Action = [Up , Right , Down , Left , Drop , Pick]
> Each action means similar to the convectional meaning of the word.

C) Transition Model is implemented by a function which takes three input – current_state, action, next_state and return corresponding probability. Probability assignment is done in accordance with the problem statement.
The pick and drop actions are deterministic. Whereas, the navigation actions are probabilistic. The correct navigation occurs with probability 0.85 and error navigation each with 0.05.

D) Reward Model is also implemented using a function which takes two input-current_state and action, return corresponding reward.
The reward model is also according to the problem statement.
- If the passenger and taxi are in the destination position and “drop” is performed, then reward is +20
- If passenger and taxi are at different position and “drop” or pick is performed, then reward is -10
- Reward is -1 in all other cases.

Note – Simulation is implemented in a way that it prints (state – optimal action – action taken)

> Pair in terminal.

2. Value Iteration Implementation

A) We have taken EPSILON = 0.00001, GAMMA = 0.9 and initial instance as –
Taxi Position = (4,1)
Pass Position = Y = (0,0)
Destination = G = (4,4)
In this case, no of iteration required for convergence is 47.

B) Since, rate of convergence is dependent both on initial instance and discount factor. So, to observe relation of convergence with discount factor, we need to fix initial instance.
In this case initial instance is same as previous.
No. of iterations for different value of discount factors are:

Discount Factor (Gamma)	**No. of iterations**
0.01	4
0.1	7
0.5	20
0.8	38
0.9	47
0.99	60

Observations:

Following observations can be drawn from the graphs –

1) For low values of discount factors, the values converge very quickly. for example, in case of γ = 0.01, only 4 iterations are required for convergence. but final policy is very inaccurate.
2) For high value of discount factors, the values converge very late. for example, when the discount factor is 0.99, the values converge after 60 iterations. however, the optimal policy is obtained much earlier in comparison to the convergence of values and are accurate.
3) As the discount factor increases, the number of iterations required to converge also increases. γ determines the weight of distant rewards on the current state.

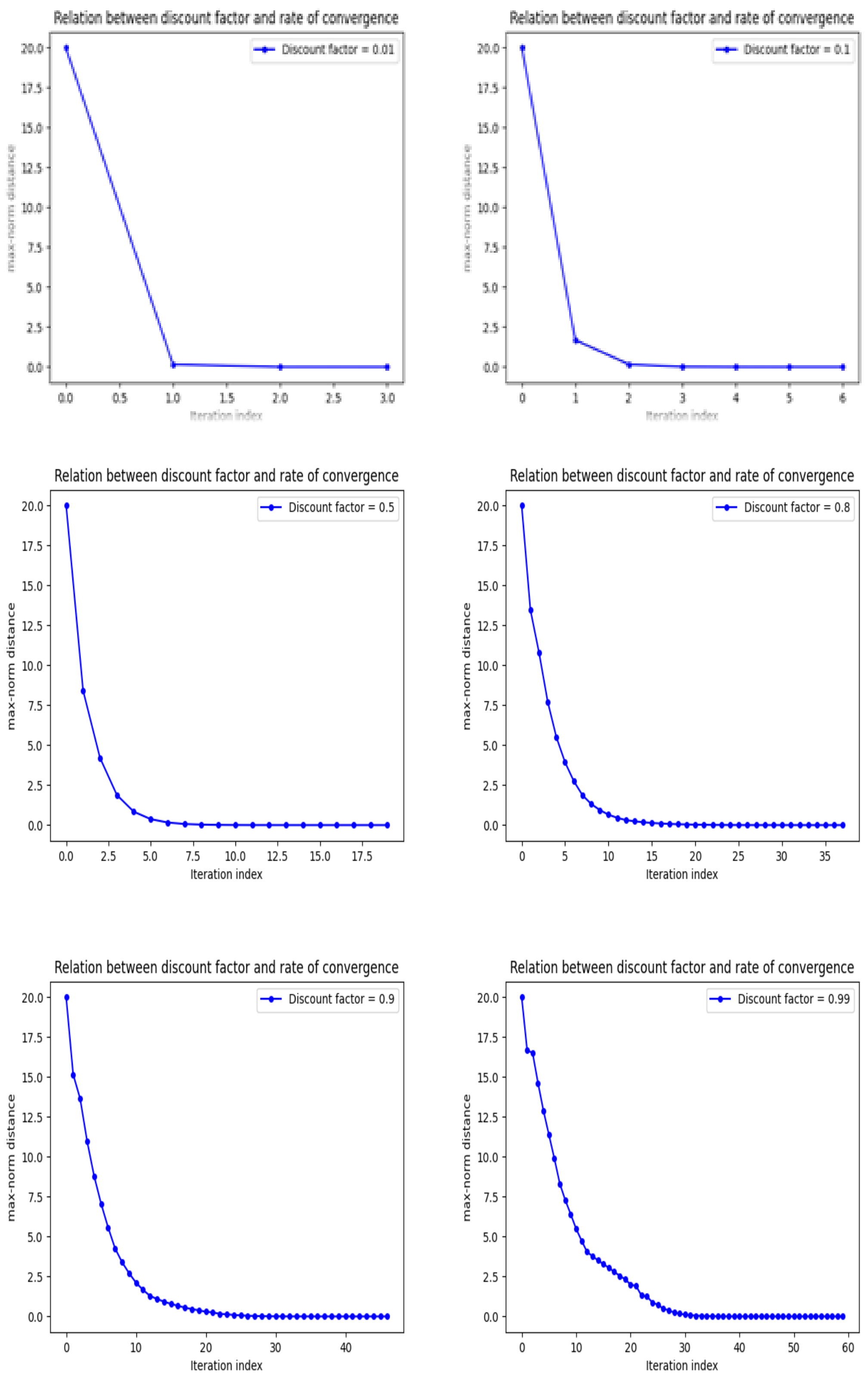


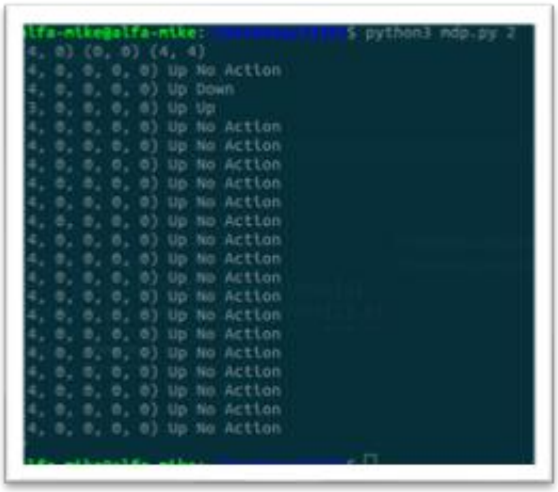
Figure – Rate of convergence for different discount factor

C) Simulation of first 20 states and corresponding action has been shown in the image for discount factor 0.1 and 0.99, taking different initial state for taxi and passenger.

From the outputs, we an observe that in case of discount factor = 0.1, agent is not reaching the goal in 20 steps because the policy is not optimal. Since no of iterations in case of γ = 0.01 is very small so, final utility will not be optimal which result in inaccurate final policy. So, the actions prescribed by the policy are not goal directed.
While, when discount factor = 0.99, the final policy is optimal, because number of iterations are large and within it final values will converge to optimal values. So, the actions prescribed by policy will be correct, which results agent to completes its goal of dropping passenger to destination.

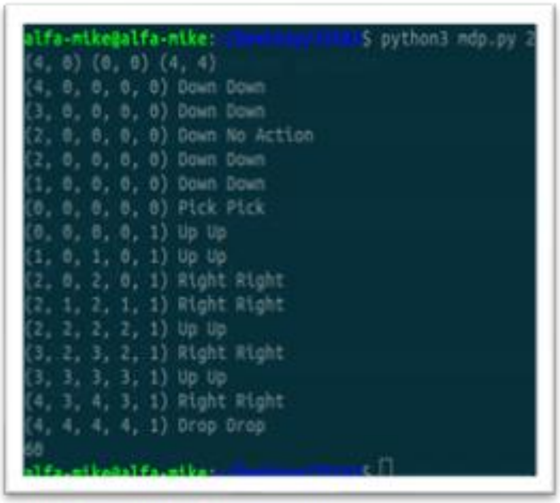
NOTE – output format: (state, action prescribed by policy, action taken by agent)

Discount factor = 0.01



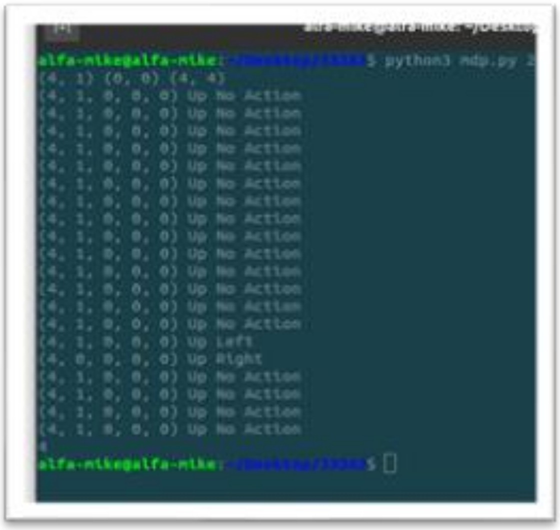
```
lfa-mike@alfa-mike:[illegible]$ python3 mdp.py 2
4, 0) (0, 0) (4, 4)
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up Down
3, 0, 0, 0, 0) Up Up
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
4, 0, 0, 0, 0) Up No Action
```

Discount factor = 0.99

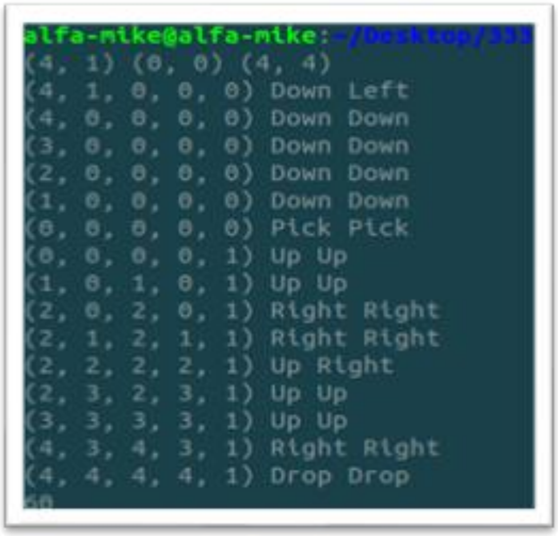


```
alfa-mike@alfa-mike:[illegible]$ python3 mdp.py 2
(4, 0) (0, 0) (4, 4)
(4, 0, 0, 0, 0) Down Down
(3, 0, 0, 0, 0) Down Down
(2, 0, 0, 0, 0) Down No Action
(2, 0, 0, 0, 0) Down Down
(1, 0, 0, 0, 0) Down Down
(0, 0, 0, 0, 0) Pick Pick
(0, 0, 0, 0, 1) Up Up
(1, 0, 1, 0, 1) Up Up
(2, 0, 2, 0, 1) Right Right
(2, 1, 2, 1, 1) Right Right
(2, 2, 2, 2, 1) Up Up
(3, 2, 3, 2, 1) Right Right
(3, 3, 3, 3, 1) Up Up
(4, 3, 4, 3, 1) Right Right
(4, 4, 4, 4, 1) Drop Drop
60
alfa-mike@alfa-mike:[illegible]$
```

Fig-1- TaxiPos = (4,0) PassPos = (0,0) destPos = (4,4)

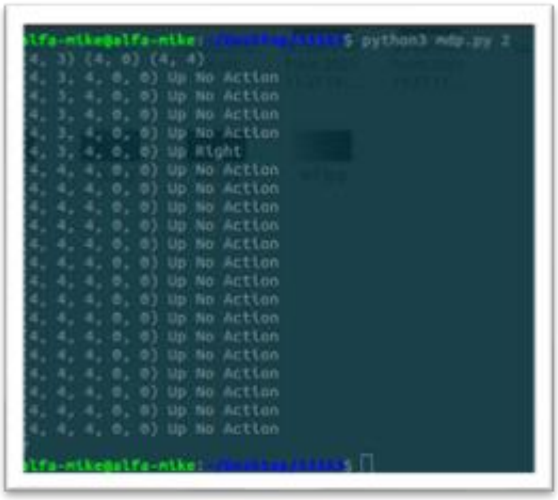


```
alfa-mike@alfa-mike:[illegible]$ python3 mdp.py 2
(4, 1) (0, 0) (4, 4)
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up Left
(4, 0, 0, 0, 0) Up Right
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
(4, 1, 0, 0, 0) Up No Action
4
alfa-mike@alfa-mike:[illegible]$
```



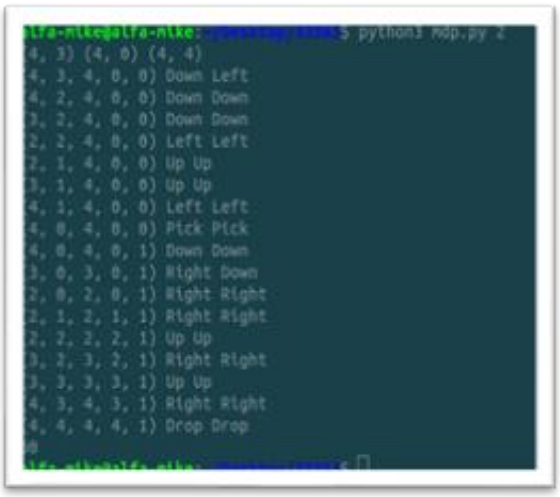
```
alfa-mike@alfa-mike:~/Desktop/333
(4, 1) (0, 0) (4, 4)
(4, 1, 0, 0, 0) Down Left
(4, 0, 0, 0, 0) Down Down
(3, 0, 0, 0, 0) Down Down
(2, 0, 0, 0, 0) Down Down
(1, 0, 0, 0, 0) Down Down
(0, 0, 0, 0, 0) Pick Pick
(0, 0, 0, 0, 1) Up Up
(1, 0, 1, 0, 1) Up Up
(2, 0, 2, 0, 1) Right Right
(2, 1, 2, 1, 1) Right Right
(2, 2, 2, 2, 1) Up Right
(2, 3, 2, 3, 1) Up Up
(3, 3, 3, 3, 1) Up Up
(4, 3, 4, 3, 1) Right Right
(4, 4, 4, 4, 1) Drop Drop
60
```

Fig-2- TaxiPos = (4,1) PassPos = (0,0) destPos = (4,4)



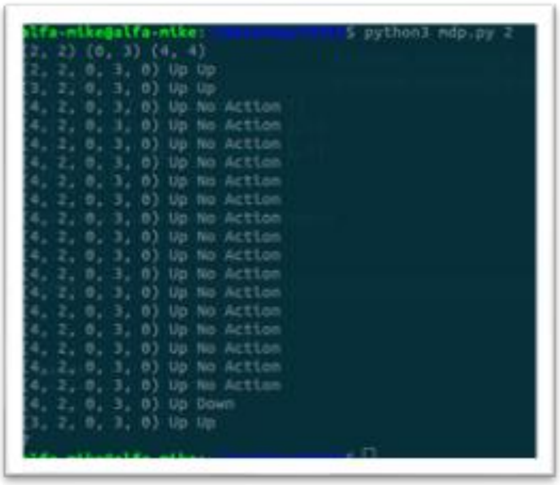
```
lfa-mike@alfa-mike:[illegible]$ python3 mdp.py 2
4, 3) (4, 0) (4, 4)
4, 3, 4, 0, 0) Up No Action
4, 3, 4, 0, 0) Up No Action
4, 3, 4, 0, 0) Up No Action
4, 3, 4, 0, 0) Up No Action
4, 3, 4, 0, 0) Up Right
4, 4, 4, 0, 0) Up No Action
4, 4, 4, 0, 0) Up No Action
4, 4, 4, 0, 0) Up No Action
4, 4, 4, 0, 0) Up No Action
4, 4, 4, 0, 0) Up No Action
4, 4, 4, 0, 0) Up No Action
4, 4, 4, 0, 0) Up No Action
4, 4, 4, 0, 0) Up No Action
4, 4, 4, 0, 0) Up No Action
4, 4, 4, 0, 0) Up No Action
4, 4, 4, 0, 0) Up No Action
4, 4, 4, 0, 0) Up No Action
4, 4, 4, 0, 0) Up No Action
4, 4, 4, 0, 0) Up No Action
4, 4, 4, 0, 0) Up No Action

lfa-mike@alfa-mike:[illegible]$
```



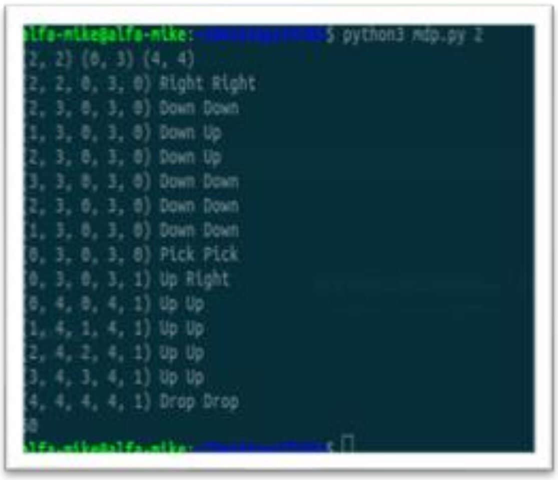
```
lfa-mike@alfa-mike:[illegible]$ python3 mdp.py 2
4, 3) (4, 0) (4, 4)
4, 3, 4, 0, 0) Down Left
4, 2, 4, 0, 0) Down Down
3, 2, 4, 0, 0) Down Down
2, 2, 4, 0, 0) Left Left
2, 1, 4, 0, 0) Up Up
3, 1, 4, 0, 0) Up Up
4, 1, 4, 0, 0) Left Left
4, 0, 4, 0, 0) Pick Pick
4, 0, 4, 0, 1) Down Down
3, 0, 3, 0, 1) Right Down
2, 0, 2, 0, 1) Right Right
2, 1, 2, 1, 1) Right Right
2, 2, 2, 2, 1) Up Up
3, 2, 3, 2, 1) Right Right
3, 3, 3, 3, 1) Up Up
4, 3, 4, 3, 1) Right Right
4, 4, 4, 4, 1) Drop Drop
0
lfa-mike@alfa-mike:[illegible]$
```

Fig-3- TaxiPos = (4,3) PassPos = (4,0,)   destPos = (4,4)



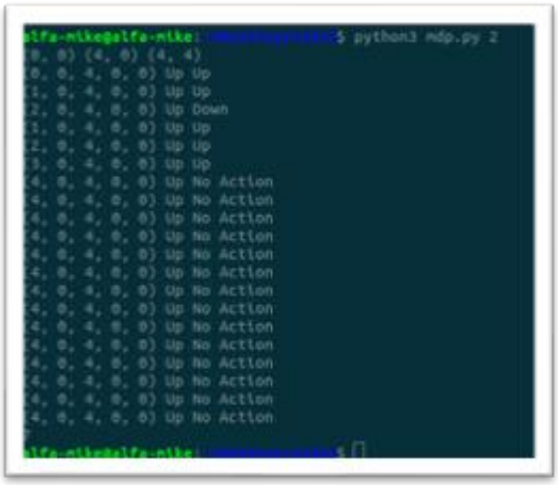
```
lfa-mike@alfa-mike:[illegible]$ python3 mdp.py 2
2, 2) (0, 3) (4, 4)
2, 2, 0, 3, 0) Up Up
3, 2, 0, 3, 0) Up Up
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up No Action
4, 2, 0, 3, 0) Up Down
3, 2, 0, 3, 0) Up Up

lfa-mike@alfa-mike:[illegible]$
```



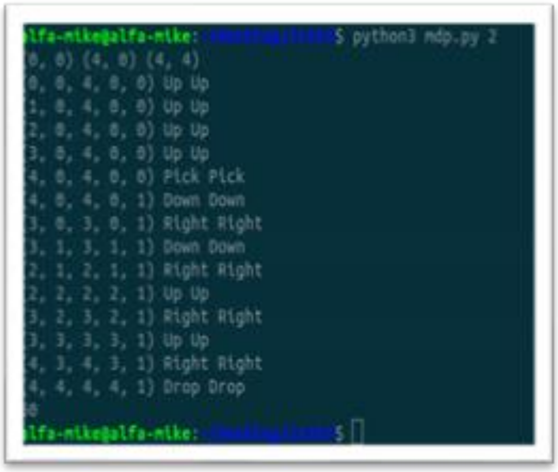
```
lfa-mike@alfa-mike:[illegible]$ python3 mdp.py 2
2, 2) (0, 3) (4, 4)
2, 2, 0, 3, 0) Right Right
2, 3, 0, 3, 0) Down Down
1, 3, 0, 3, 0) Down Up
2, 3, 0, 3, 0) Down Up
3, 3, 0, 3, 0) Down Down
2, 3, 0, 3, 0) Down Down
1, 3, 0, 3, 0) Down Down
0, 3, 0, 3, 0) Pick Pick
0, 3, 0, 3, 1) Up Right
0, 4, 0, 4, 1) Up Up
1, 4, 1, 4, 1) Up Up
2, 4, 2, 4, 1) Up Up
3, 4, 3, 4, 1) Up Up
4, 4, 4, 4, 1) Drop Drop
0
lfa-mike@alfa-mike:[illegible]$
```

Fig-1- TaxiPos = (2,2) PassPos = (0,3)   destPos = (4,4)



```
lfa-mike@alfa-mike:[illegible]$ python3 mdp.py 2
0, 0) (4, 0) (4, 4)
0, 0, 4, 0, 0) Up Up
1, 0, 4, 0, 0) Up Up
2, 0, 4, 0, 0) Up Down
1, 0, 4, 0, 0) Up Up
2, 0, 4, 0, 0) Up Up
3, 0, 4, 0, 0) Up Up
4, 0, 4, 0, 0) Up No Action
4, 0, 4, 0, 0) Up No Action
4, 0, 4, 0, 0) Up No Action
4, 0, 4, 0, 0) Up No Action
4, 0, 4, 0, 0) Up No Action
4, 0, 4, 0, 0) Up No Action
4, 0, 4, 0, 0) Up No Action
4, 0, 4, 0, 0) Up No Action
4, 0, 4, 0, 0) Up No Action
4, 0, 4, 0, 0) Up No Action
4, 0, 4, 0, 0) Up No Action
4, 0, 4, 0, 0) Up No Action
4, 0, 4, 0, 0) Up No Action
4, 0, 4, 0, 0) Up No Action

lfa-mike@alfa-mike:[illegible]$
```



```
lfa-mike@alfa-mike:[illegible]$ python3 mdp.py 2
0, 0) (4, 0) (4, 4)
0, 0, 4, 0, 0) Up Up
1, 0, 4, 0, 0) Up Up
2, 0, 4, 0, 0) Up Up
3, 0, 4, 0, 0) Up Up
4, 0, 4, 0, 0) Pick Pick
4, 0, 4, 0, 1) Down Down
3, 0, 3, 0, 1) Right Right
3, 1, 3, 1, 1) Down Down
2, 1, 2, 1, 1) Right Right
2, 2, 2, 2, 1) Up Up
3, 2, 3, 2, 1) Right Right
3, 3, 3, 3, 1) Up Up
4, 3, 4, 3, 1) Right Right
4, 4, 4, 4, 1) Drop Drop
0
lfa-mike@alfa-mike:[illegible]$
```

Fig-1- TaxiPos = (0,0) PassPos = (4,0)   destPos = (4,4)

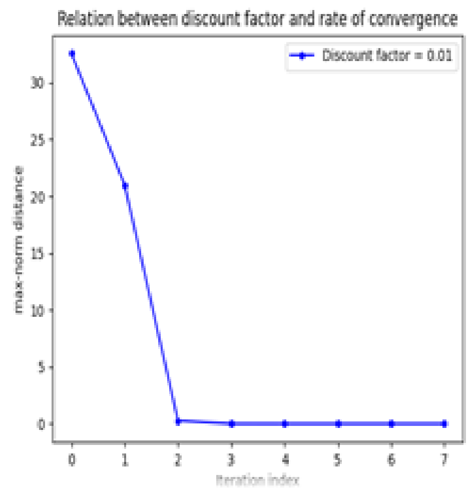
3. policy iteration

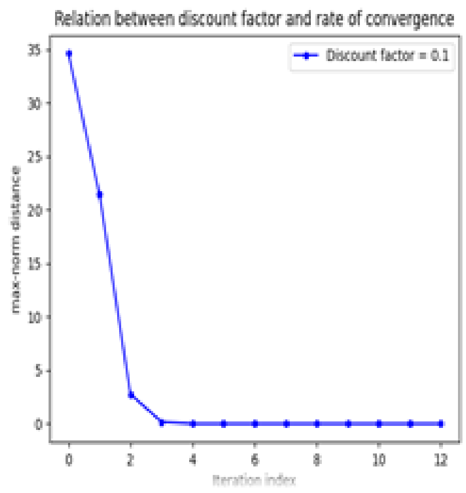
Plots for policy loss for different values of discount factor.

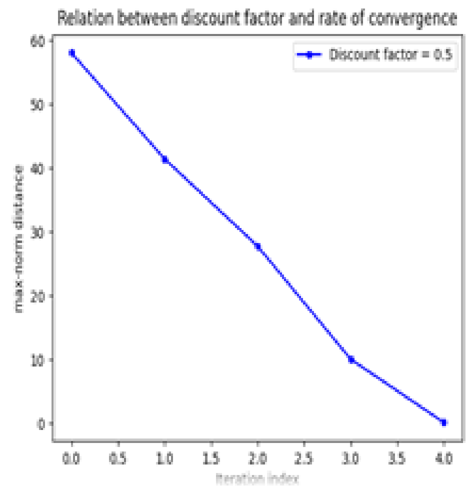
We can observe from the graphs below that:

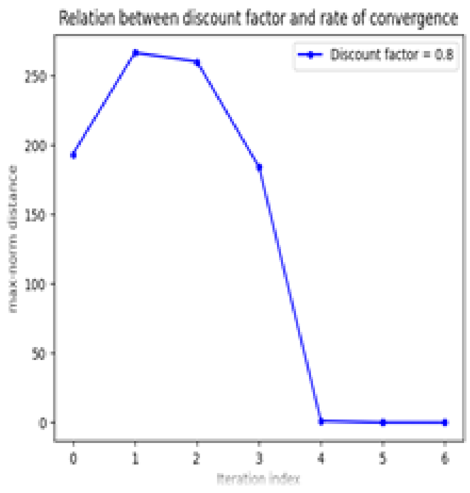
1) The policies converge faster in case of lower discount rates because lower discount rate policies tend to favour the rewards obtained in closer states.
2) As the discount rate increases, the graphs become less drastic and more gradual, thus increasing the time of convergence. this is because, higher discount rates tend to wait for better rewards in more distant states. So, on the longer run, higher discount rates always return better policies.
3) From last three graph, policy loss is first increasing and the decreasing. This is happening because policy loss for first iteration is calculated using initial utility, which is zero for all states, so policy loss for first iteration is increasing with discount factor.

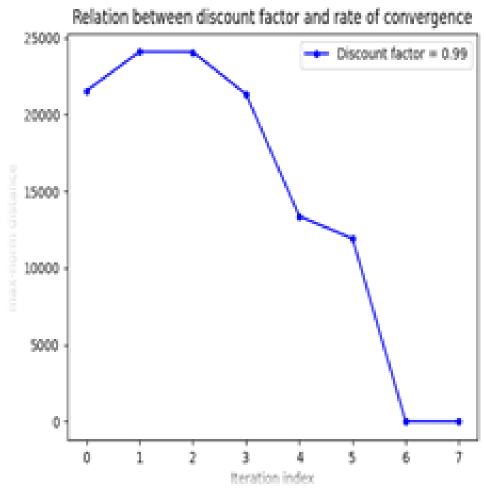
Discount Factor (Gamma)	**No. of iterations**
0.01	8
0.1	13
0.5	5
0.8	7
0.9	6
0.99	8

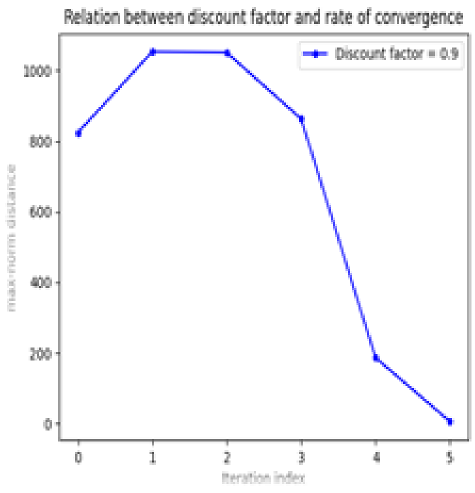

Relation between discount factor and rate of convergence
Discount factor = 0.01
max-norm distance
Iteration index
0
5
10
15
20
25
30
0
1
2
3
4
5
6
7


Relation between discount factor and rate of convergence
Discount factor = 0.1
max-norm distance
Iteration index
0
5
10
15
20
25
30
35
0
2
4
6
8
10
12


Relation between discount factor and rate of convergence
Discount factor = 0.5
max-norm distance
Iteration index
0
10
20
30
40
50
60
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Relation between discount factor and rate of convergence
Discount factor = 0.8
max-norm distance
Iteration index
0
50
100
150
200
250
0
1
2
3
4
5
6


Relation between discount factor and rate of convergence
Discount factor = 0.99
max-norm distance
Iteration index
0
5000
10000
15000
20000
25000
0
1
2
3
4
5
6
7


Relation between discount factor and rate of convergence
Discount factor = 0.9
max-norm distance
Iteration index
0
200
400
600
800
1000
0
1
2
3
4
5

Part B : Learning algorithms and incorporating learning

1. Model implementation of the learning algorithms

a. Q-Learning

The Q-learning algorithm, uses the transition model and the reward model from the environment and updates the Q-values of the state-action pairs.

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

Since the learning is model-free, we sample an iteration and update Q-values using learning factor "alpha".

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

In Q-learning, the key factor is where we choose the action.

- Here we choose a random action action with probability 'epsilon
- with probability '1-epsilon', we choose the action that returns the highest Q-value.

b. Q-Learning with decaying epsilon

Here the only difference that comes with respect to Q-learning is that the epsilon is decreased through out the learning process.

The new epsilon is inversely proportional to the learning updates done so far.

c. SARSA Learning with constant epsilon

the SARSA learning algorithm is similar to the Q-learning algorithm. Only difference is that, it selects both its next states and next actions during each iteration. It backs the Q-values calculated for its next (state,action) pair.

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma\, Q(s',a')\right]$$

d. **SARSA with decreased epsilon**

here only the epsilon value is decreased.

2. Simulation and comparison

Total no. of episodes taken = 4000

Total no. of iterations per episode = 500

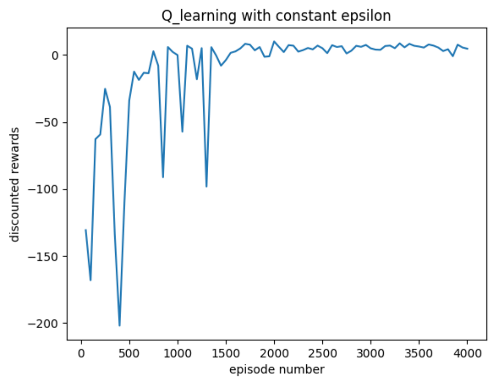
Alpha = 0.25

Epsilon = 0.1

Gamma = 0.99

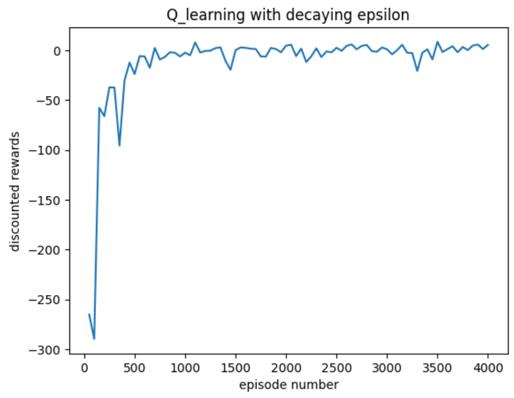
1. q-learning

We can see that the converged reward is approximately 5.23 . since the epsilon is constant, the probability of exploring new states is constant. As a result, it reaches the optimal policy after a certain number of episodes.



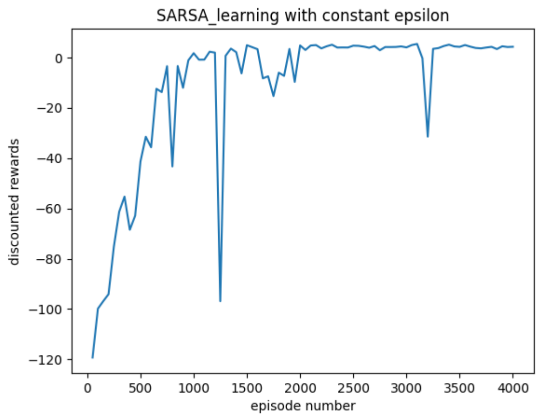
2. q_learning with decreasing epsilon

We can see that the converged reward is approximately 3.2. since the epsilon is decreasing, the probability of exploring new states decreases. So, we see less fluctuations towards the end part of the graph.



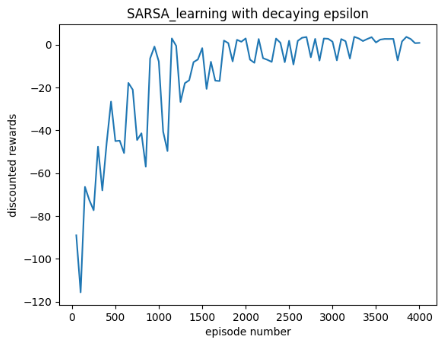
3. sarsa with constant epsilon

We can see that the discounted reward converges to 4.1 .



4. sarsa with decaying epsilon

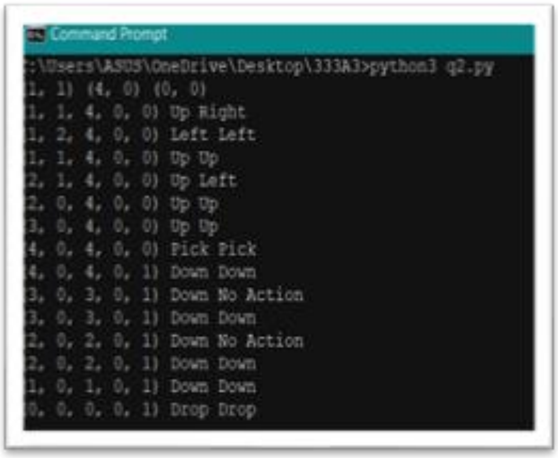
We can see that the discounted reward stabilizes at 2.7. also we can notice that it takes more time to converge as the exploration rate decreases. So it takes more time to reach optimal policy.



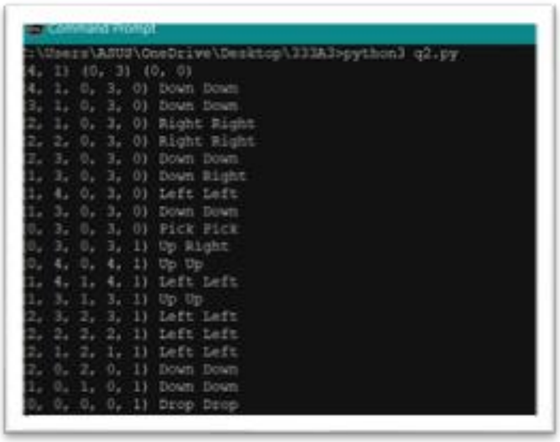
Analysis:

It is seen that the Q-learning with constant epsilon is the algorithm that converges to maximum discounted reward. Also, the optimal policy is reached in less no. of episodes. So it can be inferred as the best learning algorithm and is used in next questions.

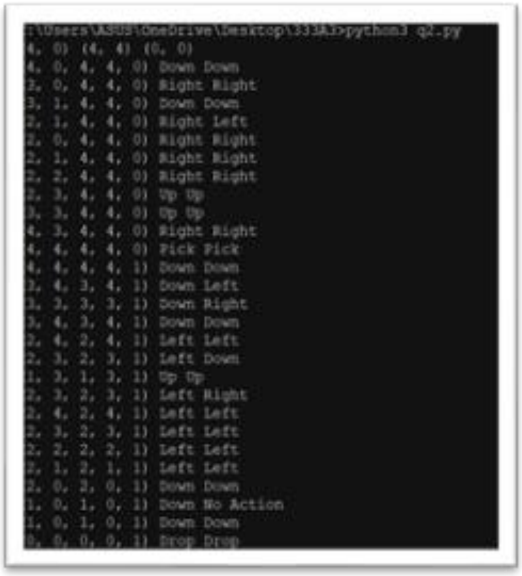
3. here Q-learning with constant epsilon is used as the best model.



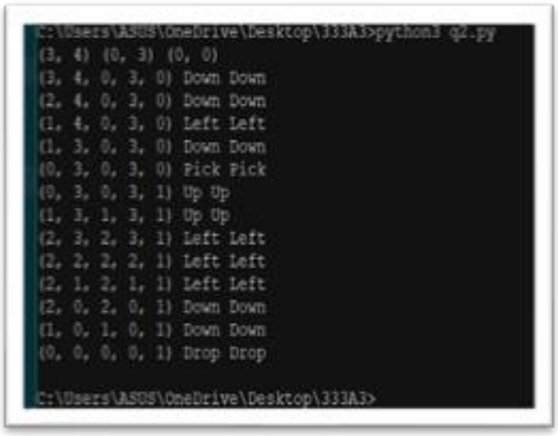
```
:\Users\ASUS\OneDrive\Desktop\333A3>python3 q2.py
1, 1) (4, 0) (0, 0)
1, 1, 4, 0, 0) Up Right
1, 2, 4, 0, 0) Left Left
1, 1, 4, 0, 0) Up Up
2, 1, 4, 0, 0) Up Left
2, 0, 4, 0, 0) Up Up
3, 0, 4, 0, 0) Up Up
4, 0, 4, 0, 0) Pick Pick
4, 0, 4, 0, 1) Down Down
3, 0, 3, 0, 1) Down No Action
3, 0, 3, 0, 1) Down Down
2, 0, 2, 0, 1) Down No Action
2, 0, 2, 0, 1) Down Down
1, 0, 1, 0, 1) Down Down
0, 0, 0, 0, 1) Drop Drop
```



```
:\Users\ASUS\OneDrive\Desktop\333A3>python3 q2.py
4, 1) (0, 3) (0, 0)
4, 1, 0, 3, 0) Down Down
3, 1, 0, 3, 0) Down Down
2, 1, 0, 3, 0) Right Right
2, 2, 0, 3, 0) Right Right
2, 3, 0, 3, 0) Down Down
1, 3, 0, 3, 0) Down Right
1, 4, 0, 3, 0) Left Left
1, 3, 0, 3, 0) Down Down
0, 3, 0, 3, 0) Pick Pick
0, 3, 0, 3, 1) Up Right
0, 4, 0, 4, 1) Up Up
1, 4, 1, 4, 1) Left Left
1, 3, 1, 3, 1) Up Up
2, 3, 2, 3, 1) Left Left
2, 2, 2, 2, 1) Left Left
2, 1, 2, 1, 1) Left Left
2, 0, 2, 0, 1) Down Down
1, 0, 1, 0, 1) Down Down
0, 0, 0, 0, 1) Drop Drop
```

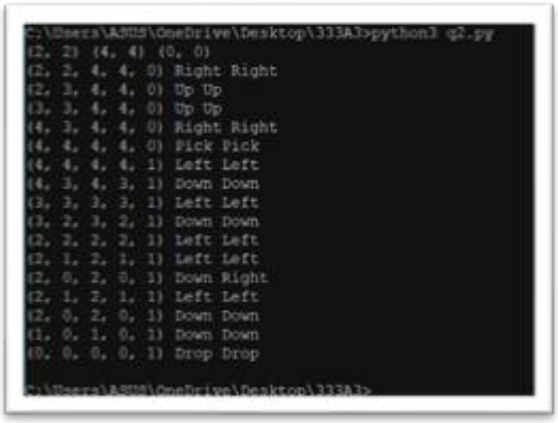


```
:\Users\ASUS\OneDrive\Desktop\333A3>python3 q2.py
4, 0) (4, 4) (0, 0)
4, 0, 4, 4, 0) Down Down
3, 0, 4, 4, 0) Right Right
3, 1, 4, 4, 0) Down Down
2, 1, 4, 4, 0) Right Left
2, 0, 4, 4, 0) Right Right
2, 1, 4, 4, 0) Right Right
2, 2, 4, 4, 0) Right Right
2, 3, 4, 4, 0) Up Up
3, 3, 4, 4, 0) Up Up
4, 3, 4, 4, 0) Right Right
4, 4, 4, 4, 0) Pick Pick
4, 4, 4, 4, 1) Down Down
3, 4, 3, 4, 1) Down Left
3, 3, 3, 3, 1) Down Right
3, 4, 3, 4, 1) Down Down
2, 4, 2, 4, 1) Left Left
2, 3, 2, 3, 1) Left Down
1, 3, 1, 3, 1) Up Up
2, 3, 2, 3, 1) Left Right
2, 4, 2, 4, 1) Left Left
2, 3, 2, 3, 1) Left Left
2, 2, 2, 2, 1) Left Left
2, 1, 2, 1, 1) Left Left
2, 0, 2, 0, 1) Down Down
1, 0, 1, 0, 1) Down No Action
1, 0, 1, 0, 1) Down Down
0, 0, 0, 0, 1) Drop Drop
```



```
C:\Users\ASUS\OneDrive\Desktop\333A3>python3 q2.py
(3, 4) (0, 3) (0, 0)
(3, 4, 0, 3, 0) Down Down
(2, 4, 0, 3, 0) Down Down
(1, 4, 0, 3, 0) Left Left
(1, 3, 0, 3, 0) Down Down
(0, 3, 0, 3, 0) Pick Pick
(0, 3, 0, 3, 1) Up Up
(1, 3, 1, 3, 1) Up Up
(2, 3, 2, 3, 1) Left Left
(2, 2, 2, 2, 1) Left Left
(2, 1, 2, 1, 1) Left Left
(2, 0, 2, 0, 1) Down Down
(1, 0, 1, 0, 1) Down Down
(0, 0, 0, 0, 1) Drop Drop

C:\Users\ASUS\OneDrive\Desktop\333A3>
```



```
C:\Users\ASUS\OneDrive\Desktop\333A3>python3 q2.py
(2, 2) (4, 4) (0, 0)
(2, 2, 4, 4, 0) Right Right
(2, 3, 4, 4, 0) Up Up
(3, 3, 4, 4, 0) Up Up
(4, 3, 4, 4, 0) Right Right
(4, 4, 4, 4, 0) Pick Pick
(4, 4, 4, 4, 1) Left Left
(4, 3, 4, 3, 1) Down Down
(3, 3, 3, 3, 1) Left Left
(3, 2, 3, 2, 1) Down Down
(2, 2, 2, 2, 1) Left Left
(2, 1, 2, 1, 1) Left Left
(2, 0, 2, 0, 1) Down Right
(2, 1, 2, 1, 1) Left Left
(2, 0, 2, 0, 1) Down Down
(1, 0, 1, 0, 1) Down Down
(0, 0, 0, 0, 1) Drop Drop

C:\Users\ASUS\OneDrive\Desktop\333A3>
```

Observation

2. The model takes minimal actions to reach the destination in most of the cases. This is because due to enough explration, the optimal policy is reached earlier.
3. Also, we may see some cases where the action taken is not according to the policy, this is because of the probabilistic nature of action chosen by the agent.
4. Also, since exploration is done even after reaching optimal policy, so the actions are not always accrding to the policy.

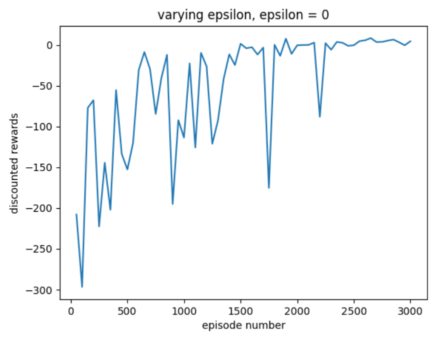
4. variations of rewards

a. varying epsilon and keeping alpha constant

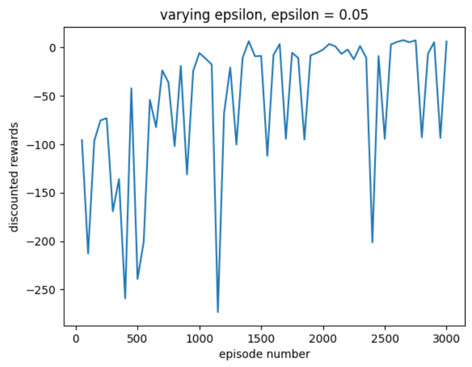
alpha = 0.1

plots:

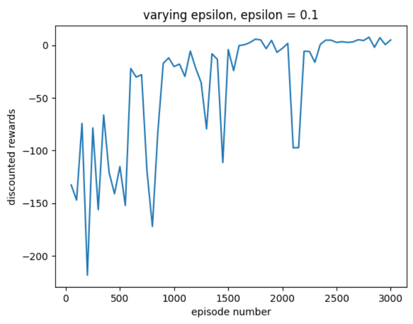
epsilon = 0



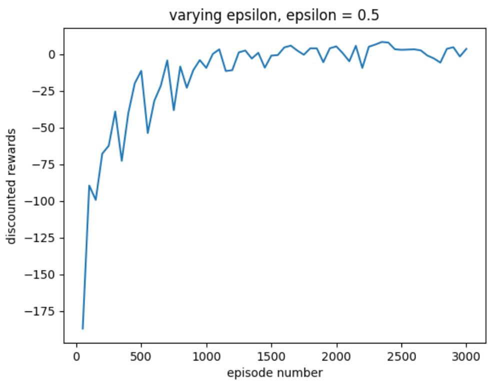
Epsilon = 0.05



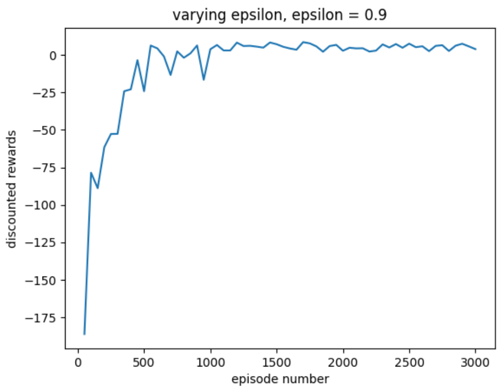
Epsilon = 0.1



Epsilon = 0.5



Epsilon = 0.9



Analysis:

- we know ‘epsilon’ denotes the probability to explore new states. So by increasing ‘epsilon’ the likelihood of exploring new actions increases.
- For epsilon = 0, the agent chooses actions greedily by checking the Q(state, action) value. Initially, since all the Q values are zeroes. So it chooses random action. But as it explores more and more steps, the greedy actions are always chooses. As a result of which, we see no fluctutations towards the end when it had found a optimal policy.

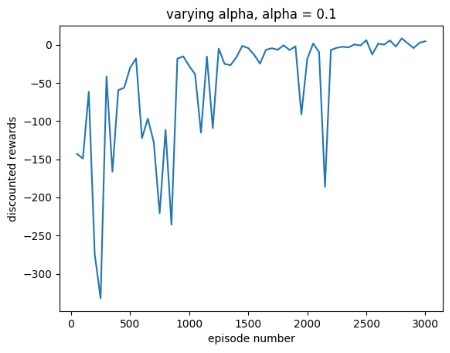
- As we increase the ‘epsilon’, it explores new states even when it had achieved an optimal state. So this in the reason why, even after stabilizing, there are small fluctuations in the graph. Also, since it explores more, the optimal policy is reached in less no. of episodes as compared to epsilon=0
- The average reward is higher for higher values of epsilon. This means exploration gives better policy.

b. varying alpha and keeping epsilon constant

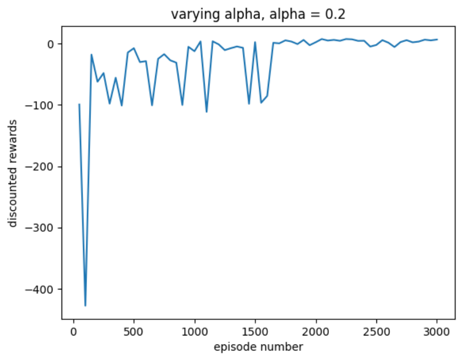
epsilon = 0.1

plots:

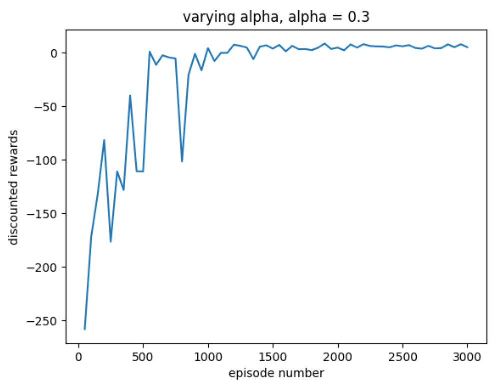
alpha = 0.1



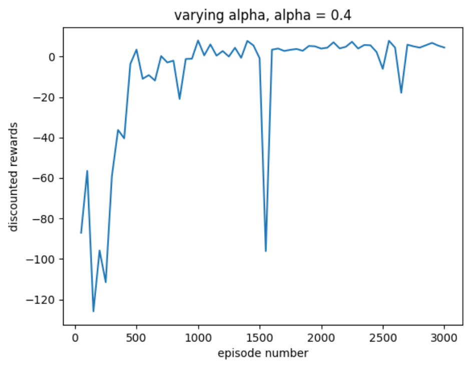
alpha = 0.2



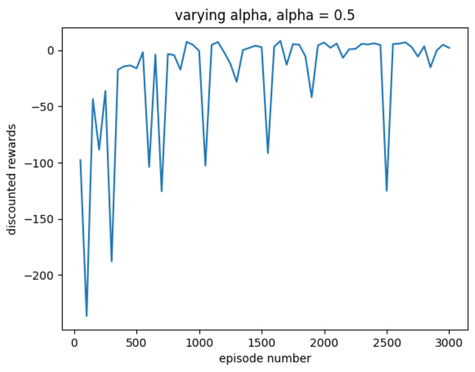
alpha = 0.3



alpha = 0.4



alpha = 0.5



Analysis:

- ‘alpha’ denotes the tendency to favour new samples over the already calculated Q-values.

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

- So, at low values of alpha, the agent favours those actions whose reward is already calculated. So, it undergoes only those transitions whose reward is maximum and as a result it reaches the optimal policy very late. As we see from graph in case of alpha=0.1

- By increasing 'alpha', the agents explores new samples, so it reaches the optimal policy faster. But for very higher values of alpha(alpha = 0.5), the agent gives equal priority to both explored samples and new samples. As a result, even after reaching optimal policy, it explores new samples so you can see fluctuations.
- So most favourable value of alpha, as can be inferred from the graph is 0.3.