CS6700: Reinforcement Learning

Q-learning Report

- Shaunak Mujumdar (CH21B062)
- Vedant Bhume (ED21B017)

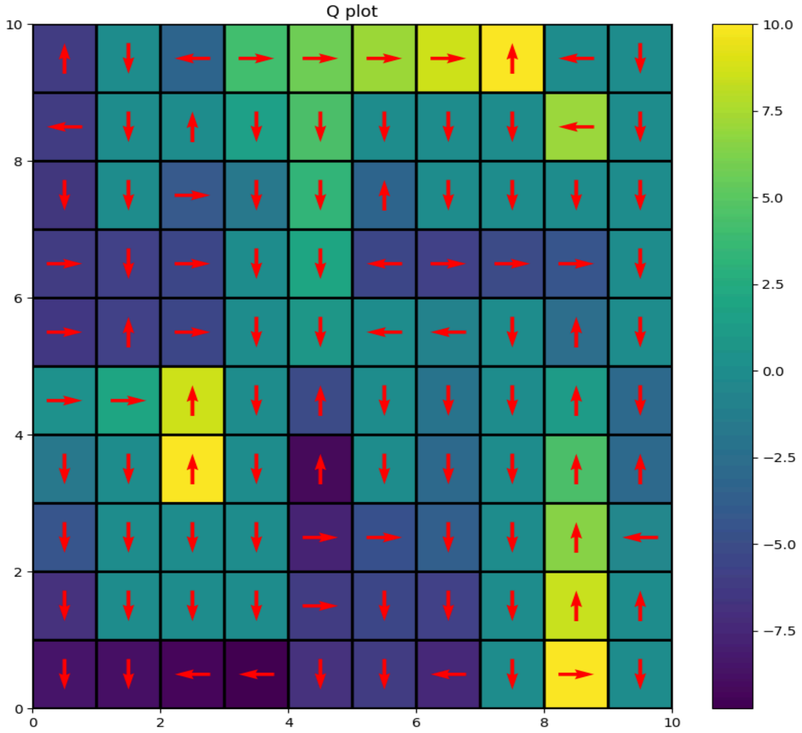
Both of the policies with specific hyper parameters were trained for 1000 episodes. The best hyper parameters were chosen based on high mean reward of the policy for the last 100 episodes of training. The policy with the best hyper parameters was trained for 1000 episodes on 5 different experiments and the final results are shown.

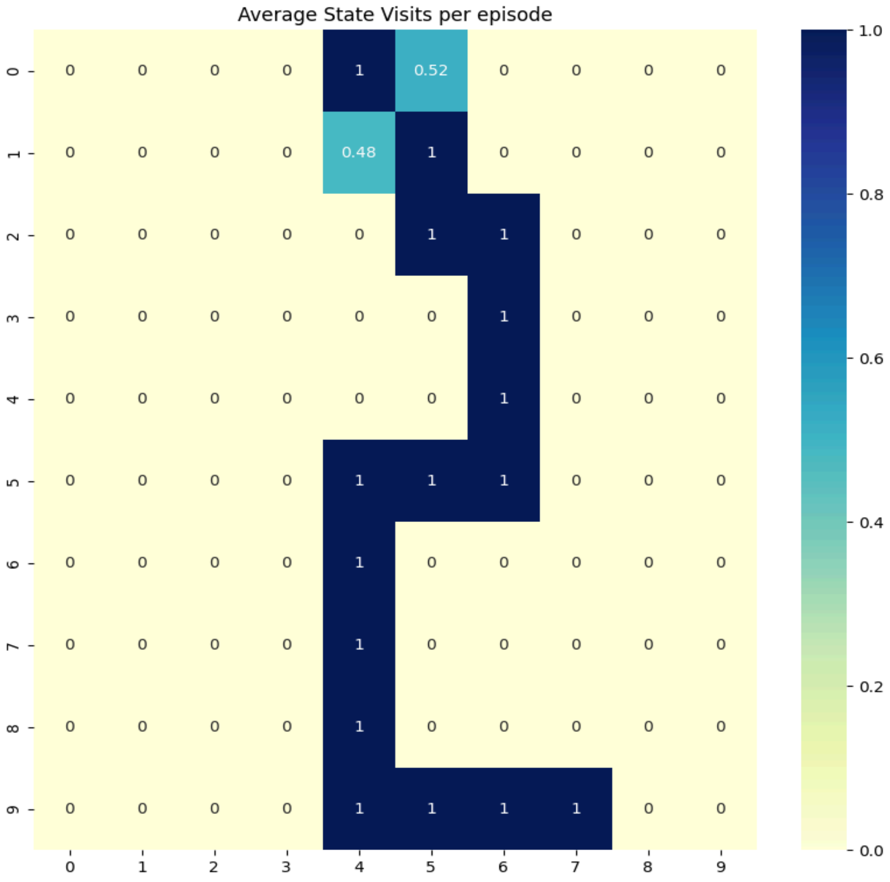
Because of the time constraints, it wasn't possible for us to explore a wide range of hyperparameters for all the given situations. We have only chosen a subset of hyperparameters and the algorithm will test out all possible combinations of these hyperparameters.

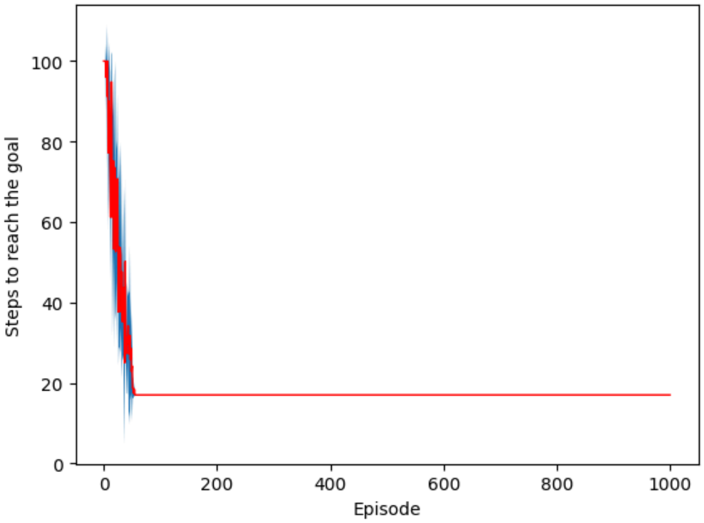
1. Alphas => [0.1,0.2,0.3,0.5,.7]
2. Gammas => [0.1,0.3,0.5,0.7,0.95]
3. Policy params => [0.01,0.05,0.3,0.5]

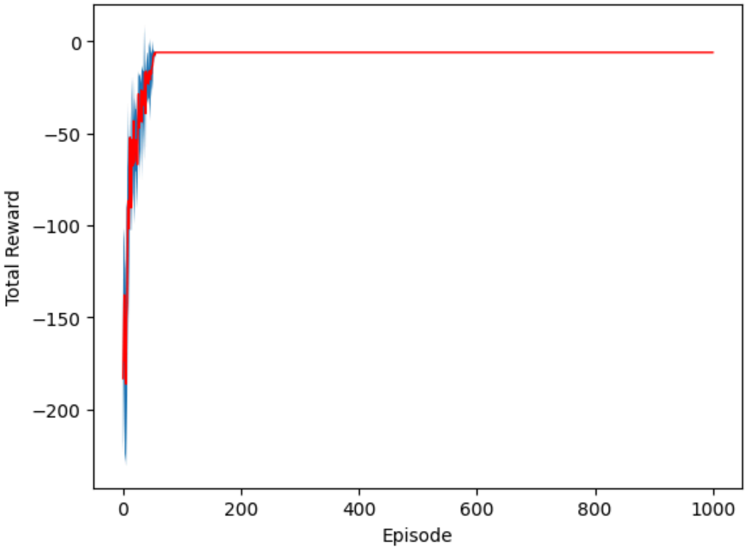
Start state	Wind	Transition Probability	Action Policy	Best Learning Rate (α)	Best Discount Factor (γ)	Best Policy Parameter
[0,4]	False	1	Softmax	0.7	0.95	0.05
[0,4]	False	1	E-greedy	0.7	0.95	0.01
[3,6]	False	1	Softmax	0.7	0.7	0.01
[3,6]	False	1	E-greedy	0.7	0.5	0.01
[0,4]	False	0.7	Softmax	0.5	0.95	0.01
[0,4]	False	0.7	E-greedy	0.3	0.95	0.01
[3,6]	False	0.7	Softmax	0.5	0.95	0.05
[3,6]	False	0.7	E-greedy	0.3	0.95	0.01
[0,4]	True	1	Softmax	0.7	0.95	0.01
[0,4]	True	1	E-greedy	0.7	0.95	0.01
[3,6]	True	1	Softmax	0.5	0.7	0.01
[3,6]	True	1	E-greedy	0.7	0.95	0.01

1. **Start state => [0,4] ; Wind => False ; P_transition = 1; Policy = Softmax:**



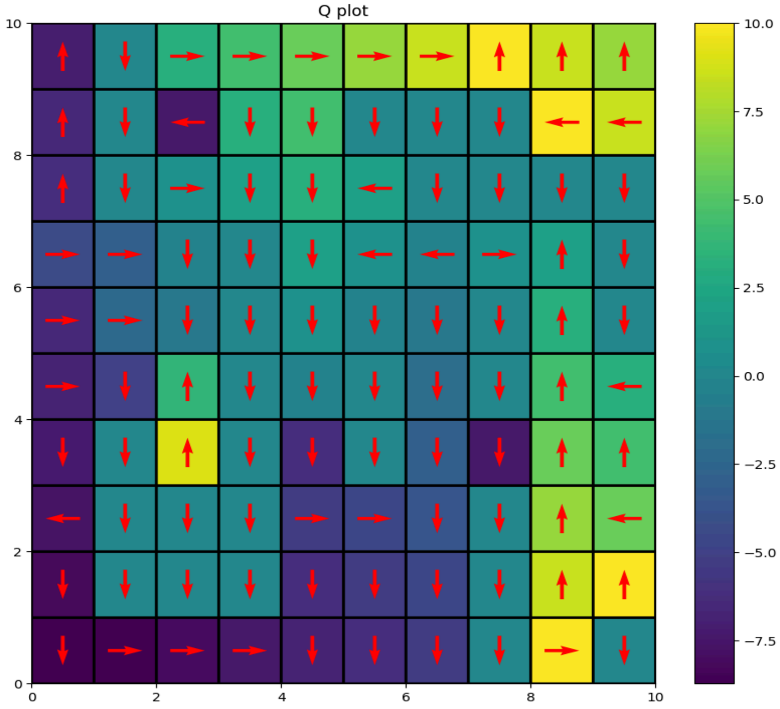


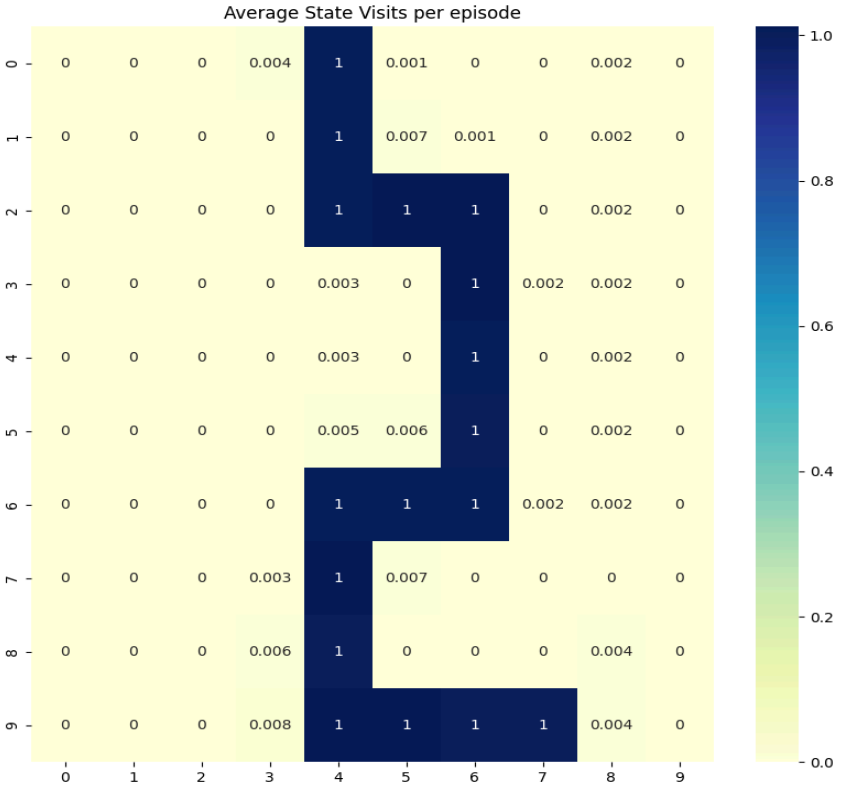


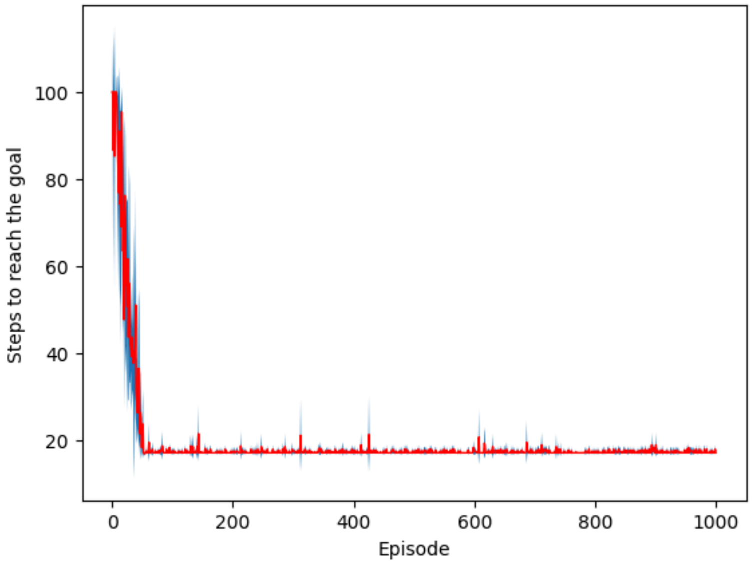


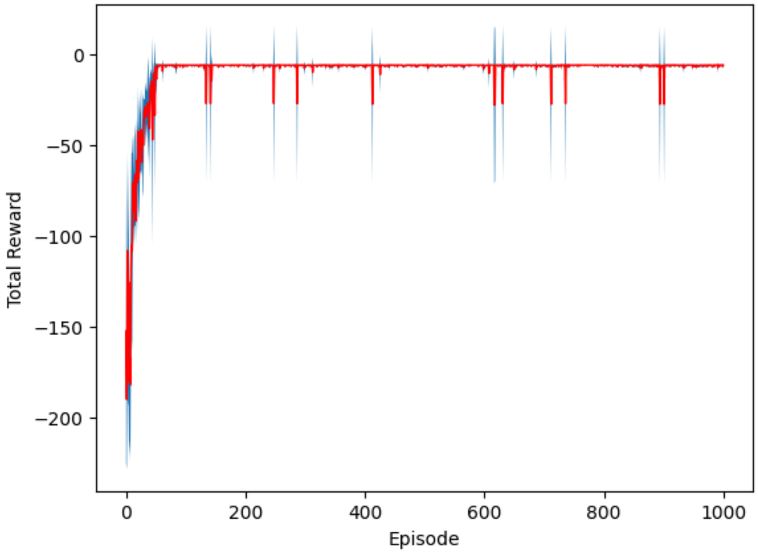
By making p=1, we're removing all elements of stochasticity from the environment.The agent immediately learns a route to a goal state and keeps following this path for all following episodes. From the state visit heat map , you can see that policy never ends up accidentally exploring the environment and possibly finding the other goal states.
For all the experiments total reward and steps to reach the goal become constant after around 50 episodes and there is no standard deviation in the total reward and steps value.

2. Start state => [0,4] ; Wind => False ; P_transition = 1; Policy = E-greedy:





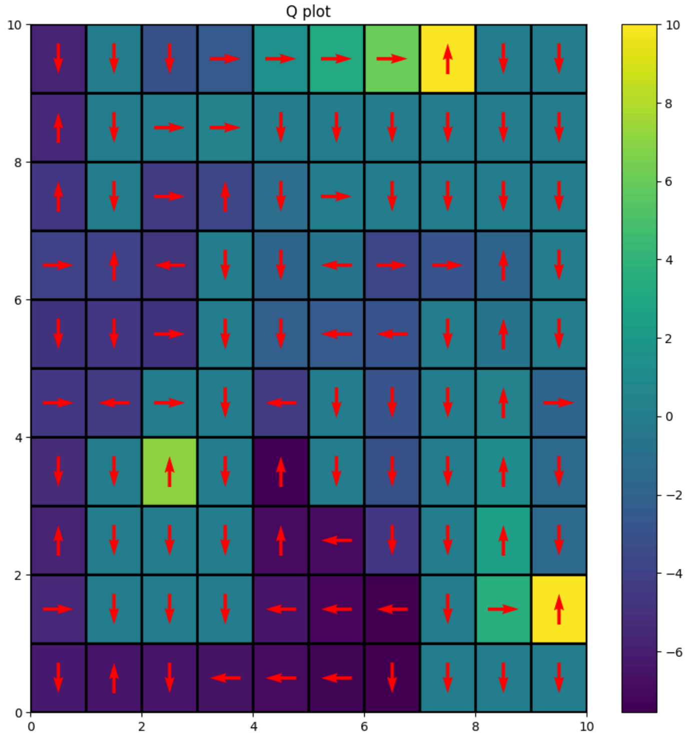


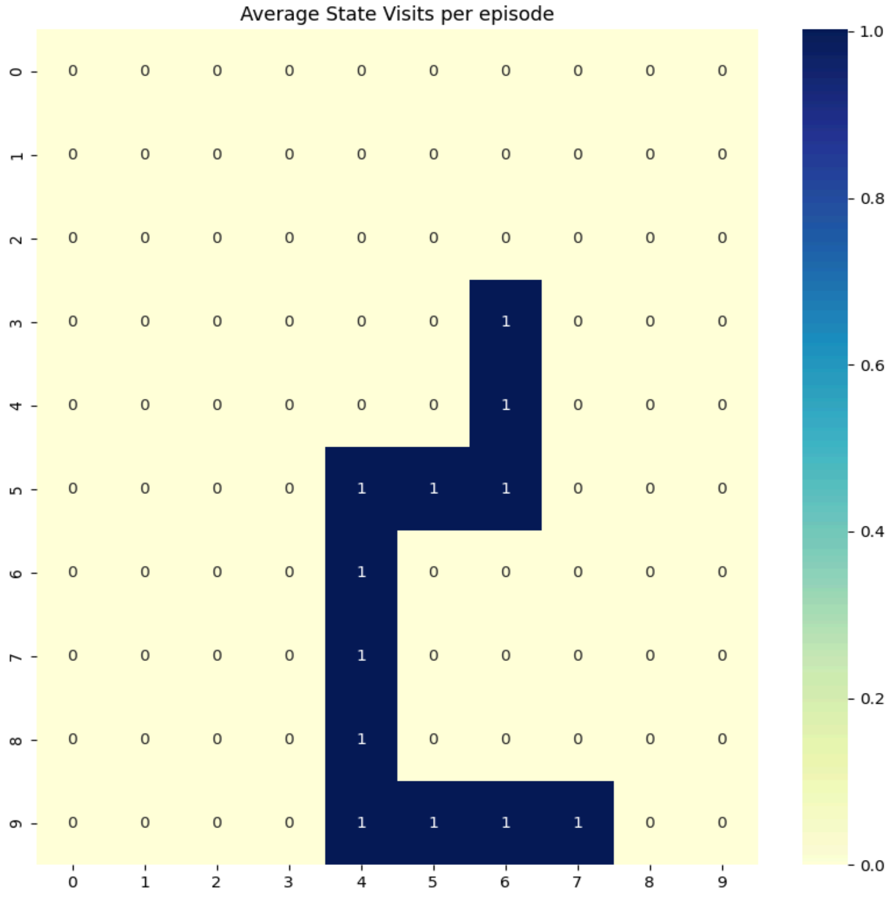


From the state visited heatmap and reward/step vs episodes graphs, we can see that E-greedy approach shows far more random exploration than its softmax counterpart that was shown before this.

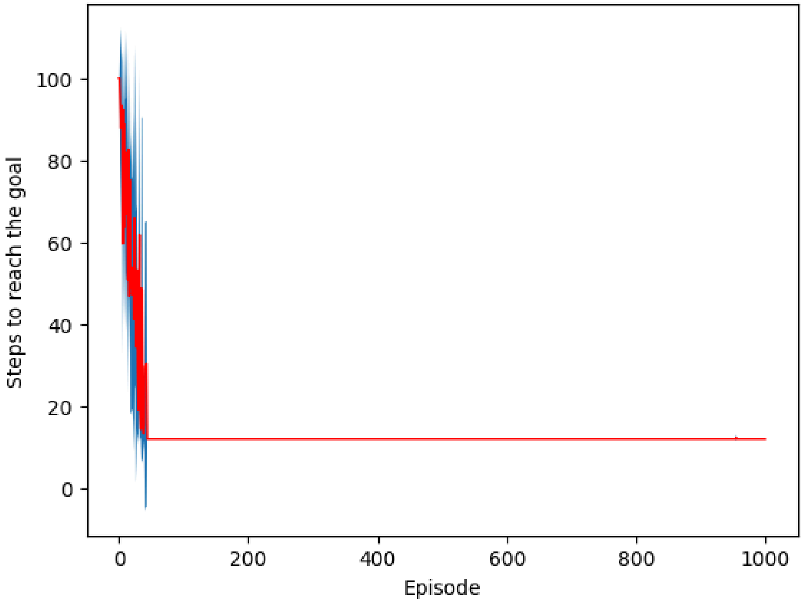
The mean number of steps is 17 and the mean reward is -4 (exactly like softmax).

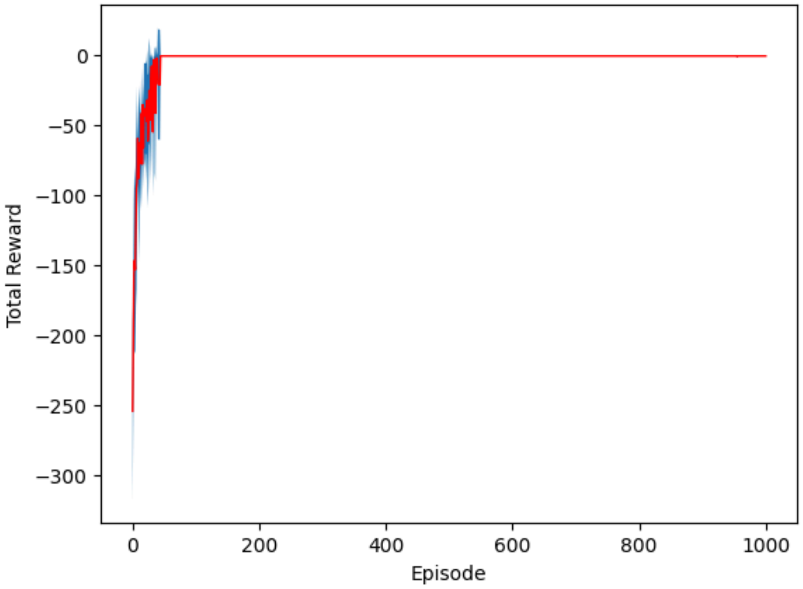
3. Start state => [3,6] ; Wind => False ; P_transition = 1; Policy = Softmax :





	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	1	0	0	0
4	0	0	0	0	0	0	1	0	0	0
5	0	0	0	0	1	1	1	0	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	0	1	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	1	1	1	1	0	0

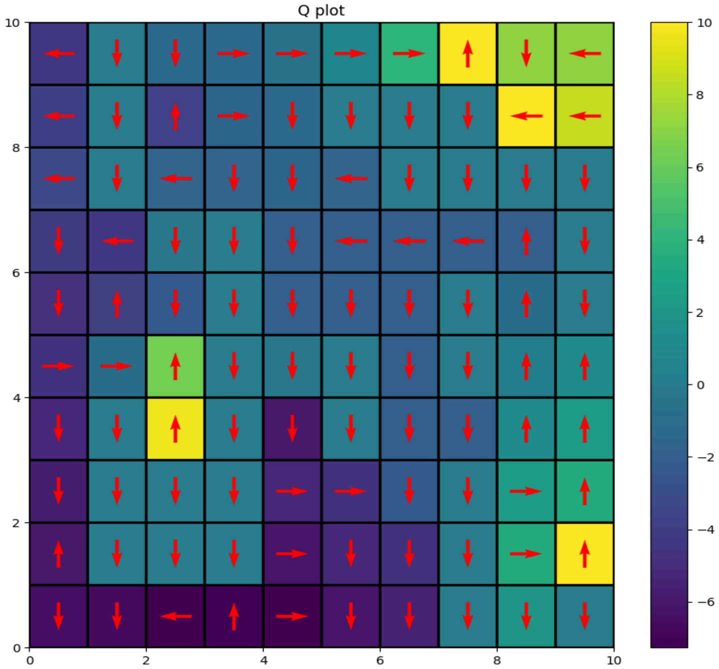


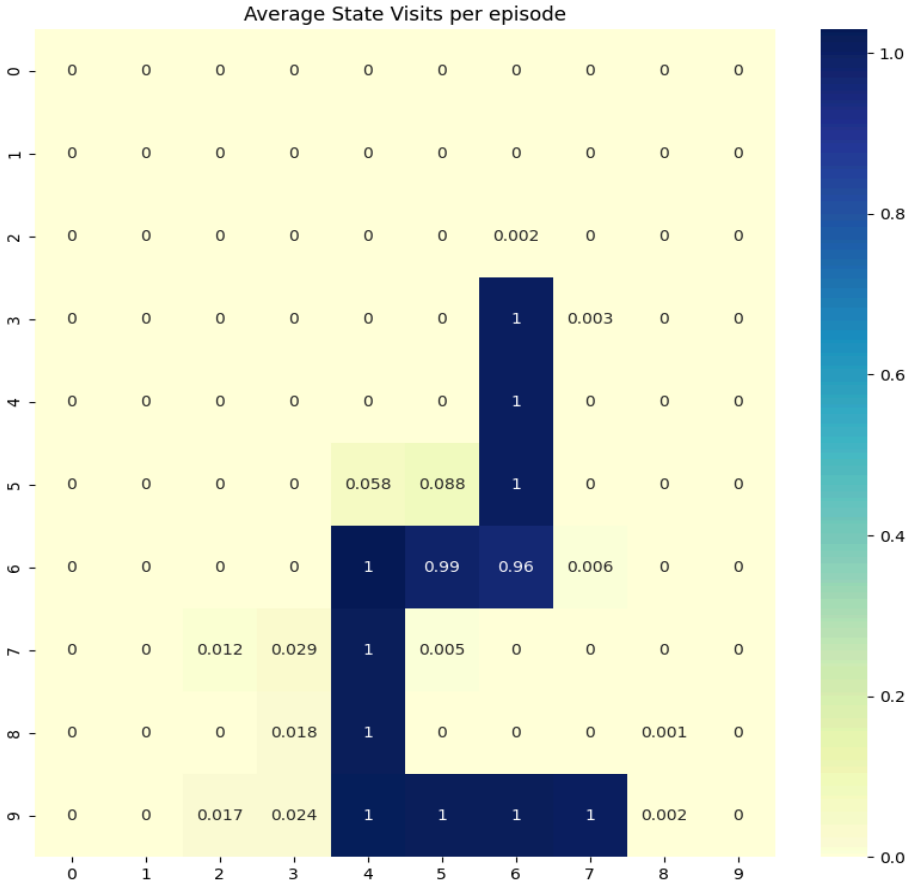


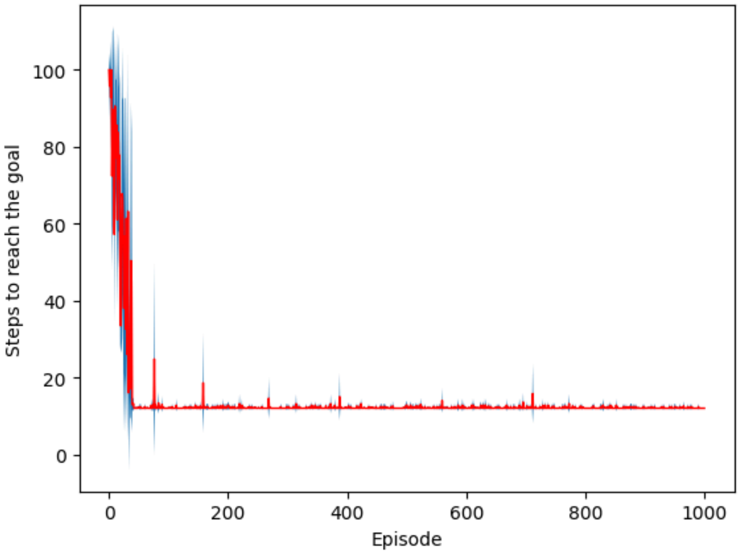
The agent immediately learns a route to a goal state and keeps following this path for all following episodes. From the state visit heat map , you can see that policy never ends up accidentally exploring the environment and possibly finding the other goal states.

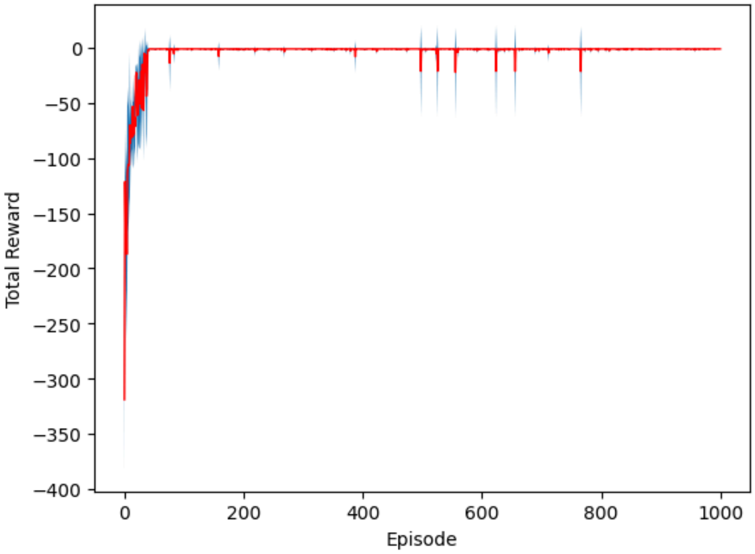
For all the experiments total reward and steps to reach the goal become constant after around 50 episodes and there is no standard deviation in the total reward and steps value. The mean number of steps is 10 and the mean reward is 0.

4. Start state => [3,6] ; Wind => False ; P_transition = 1; Policy = E-greedy:



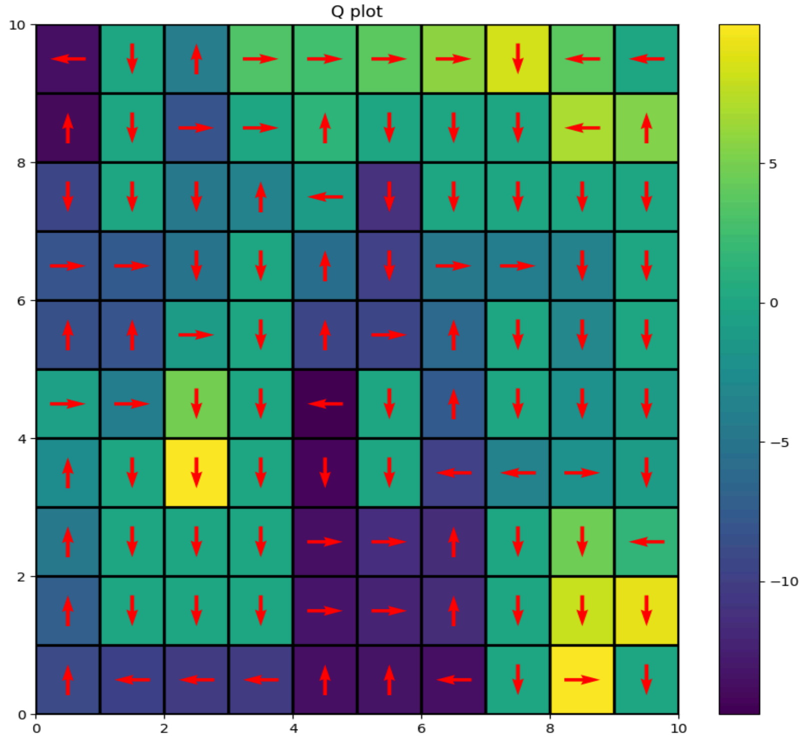


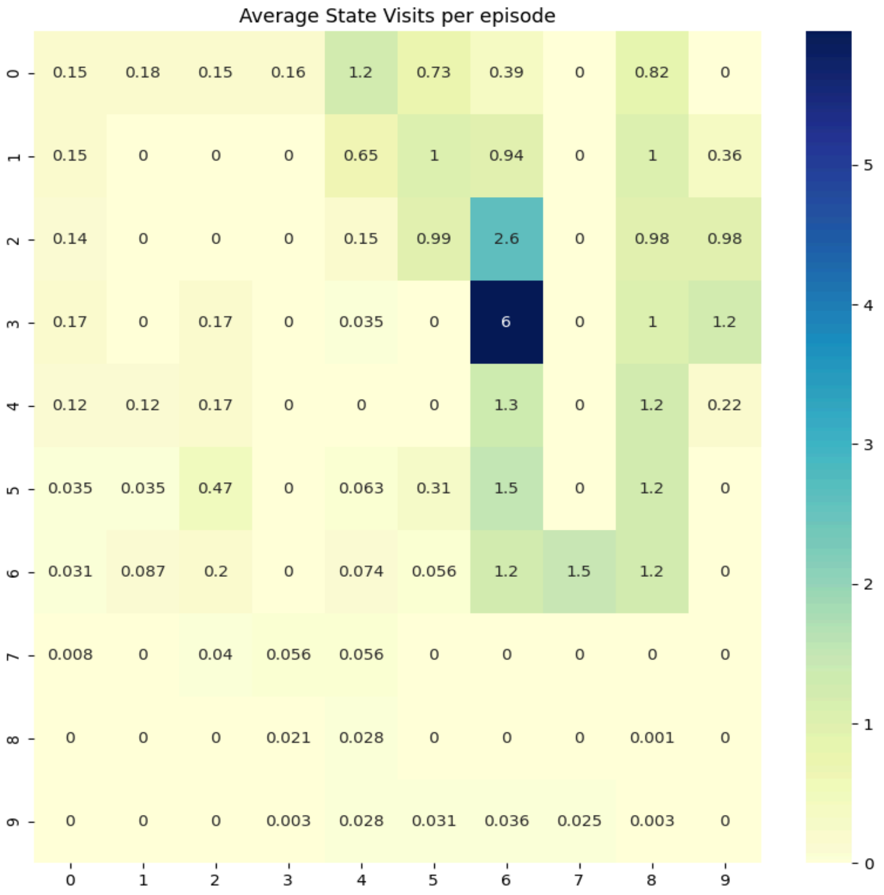


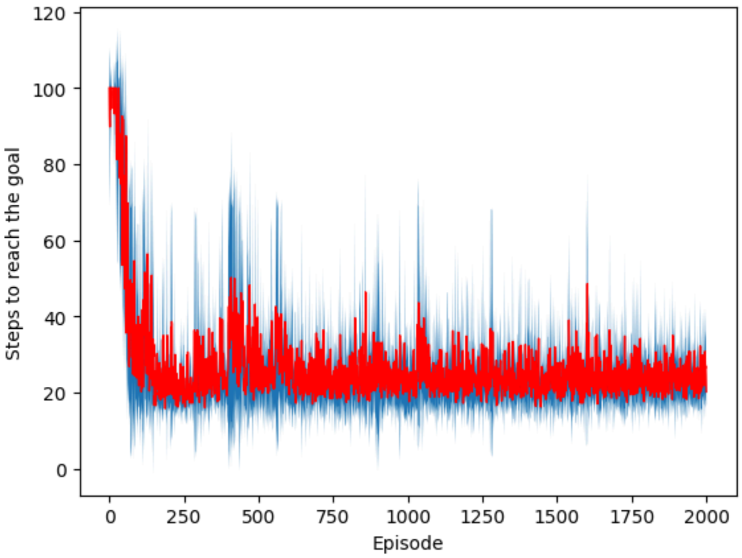


Again just as in case with start case => [0,4],E-greedy approach shows far more random exploration than its softmax counterpart which is explicitly evident in the plots above. Instead of continuously exploiting the route that converges constantly, the E-greedy approach has the agent randomly explore the grid sometimes which accounts for the **random bumps present in the steps and rewards plots** and some **random non zero values in state visit heatmap.**

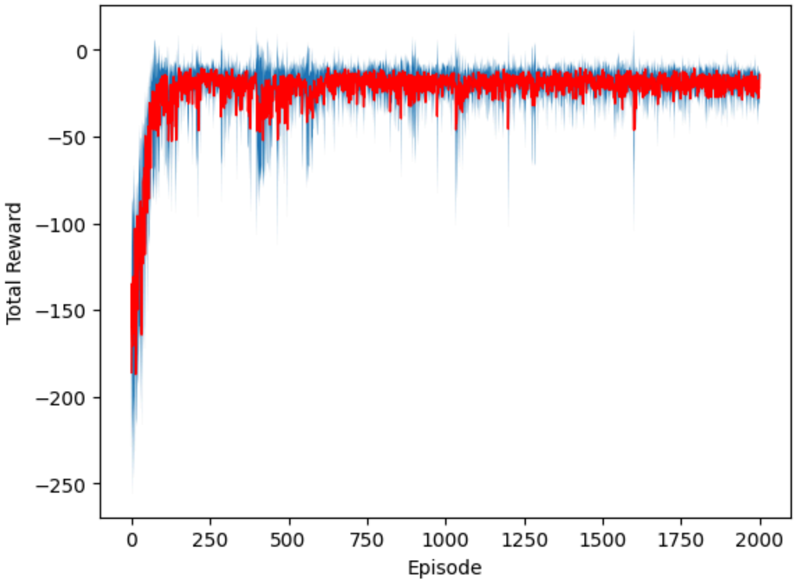
5. Start state => [0,4] ; Wind => False ; P_transition = 0.7; Policy = Softmax :





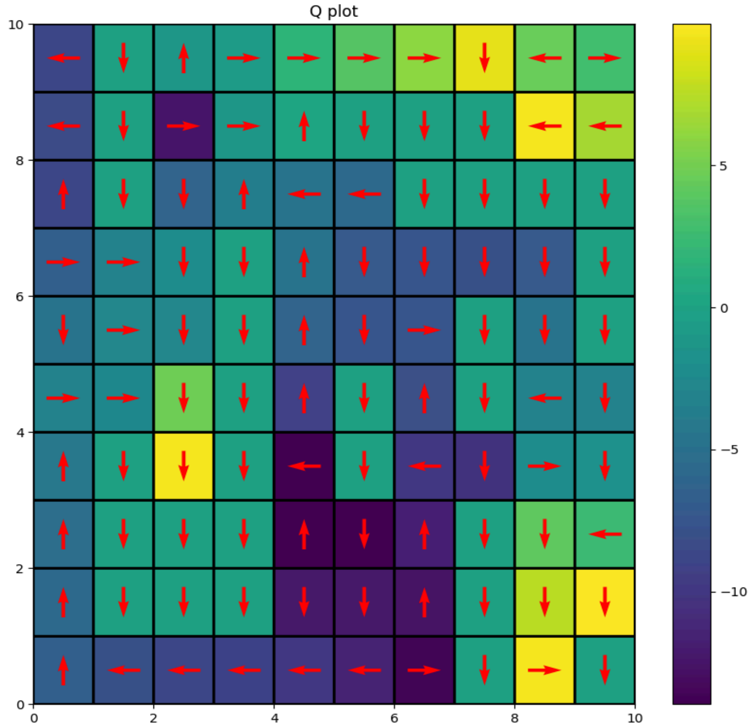


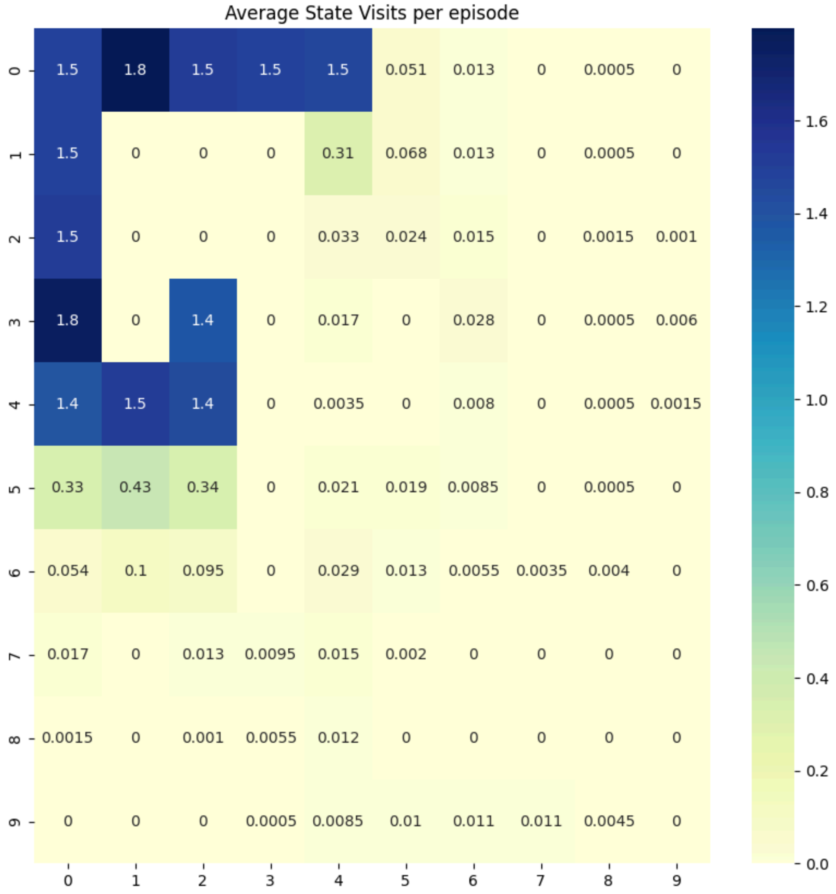
Steps to reach the goal
120
100
80
60
40
20
0
0
250
500
750
1000
1250
1500
1750
2000
Episode



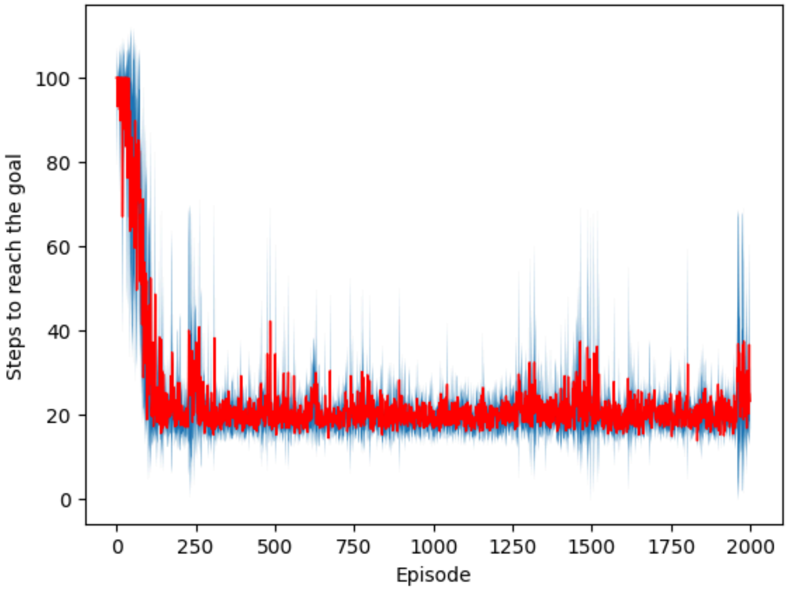
Total Reward
0
−50
−100
−150
−200
−250
0
250
500
750
1000
1250
1500
1750
2000
Episode

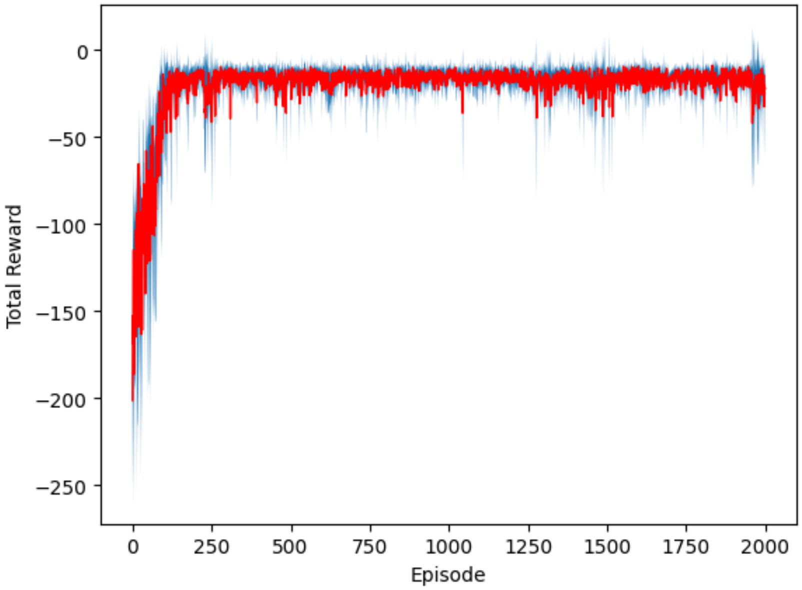
6. Start state => [0,4] ; Wind => False ; P_transition = 0.7; Policy = E-greedy:





	0	1	2	3	4	5	6	7	8	9
0	1.5	1.8	1.5	1.5	1.5	0.051	0.013	0	0.0005	0
1	1.5	0	0	0	0.31	0.068	0.013	0	0.0005	0
2	1.5	0	0	0	0.033	0.024	0.015	0	0.0015	0.001
3	1.8	0	1.4	0	0.017	0	0.028	0	0.0005	0.006
4	1.4	1.5	1.4	0	0.0035	0	0.008	0	0.0005	0.0015
5	0.33	0.43	0.34	0	0.021	0.019	0.0085	0	0.0005	0
6	0.054	0.1	0.095	0	0.029	0.013	0.0055	0.0035	0.004	0
7	0.017	0	0.013	0.0095	0.015	0.002	0	0	0	0
8	0.0015	0	0.001	0.0055	0.012	0	0	0	0	0
9	0	0	0	0.0005	0.0085	0.01	0.011	0.011	0.0045	0

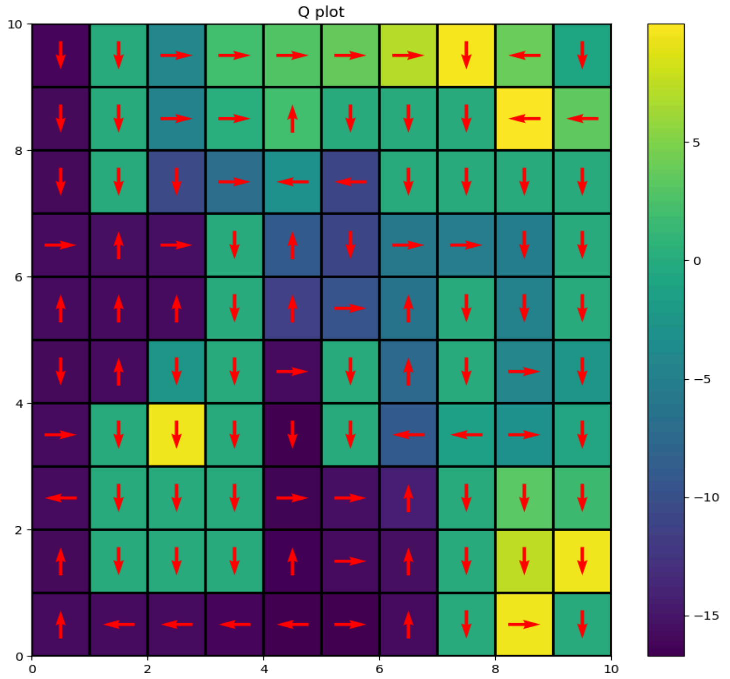


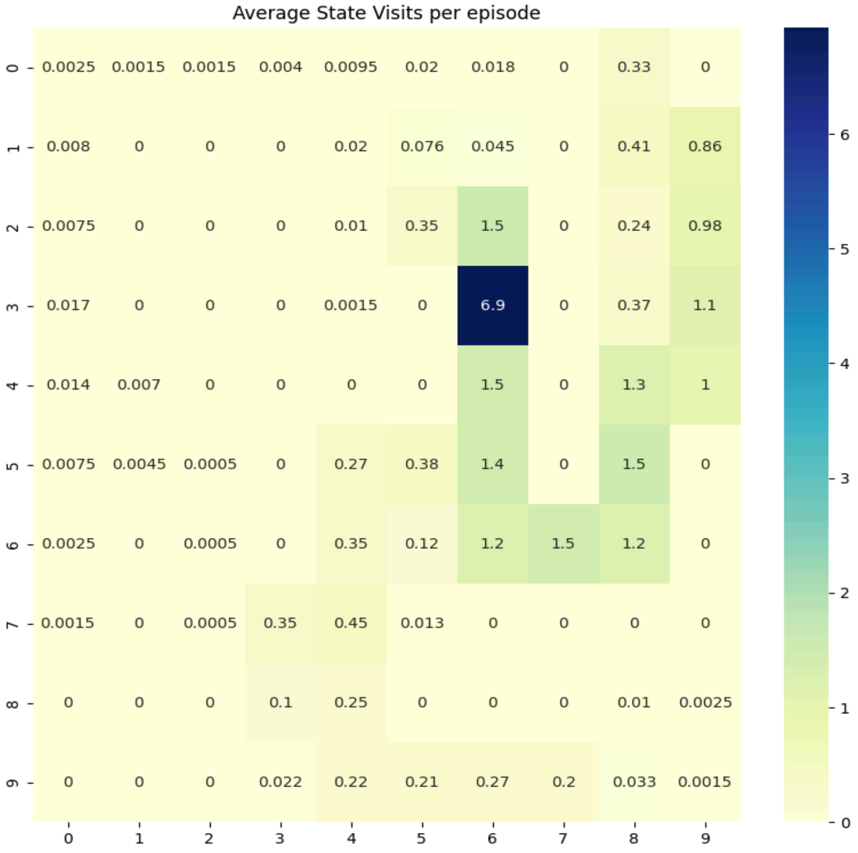


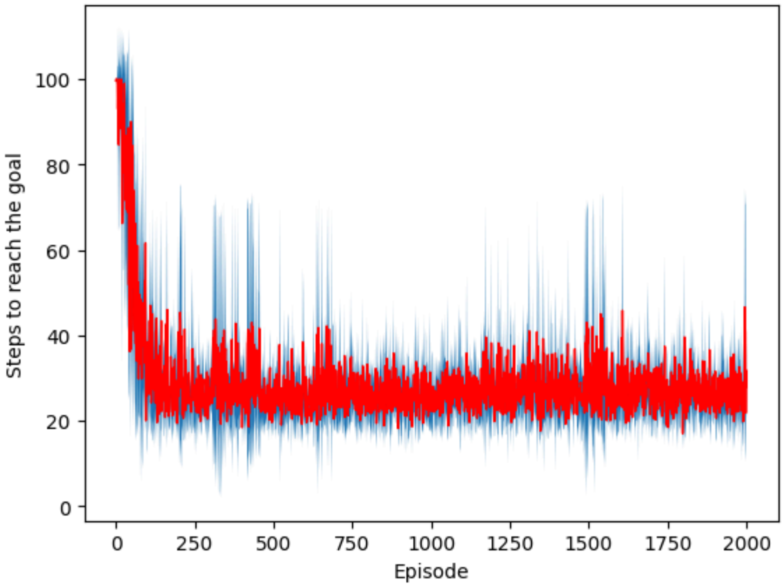
From the state visit heat map, the agent learns the path to the nearest goal state. The path to the second nearest goal is also traced albeit very few times. This is in contrast to the SARSA algorithm where this second path is traced multiple more times. This indicates that the Q learning approach exploits the confirmed route to a goal state. The alternative route is taken primarily when the stochastic nature of the environment comes into play.

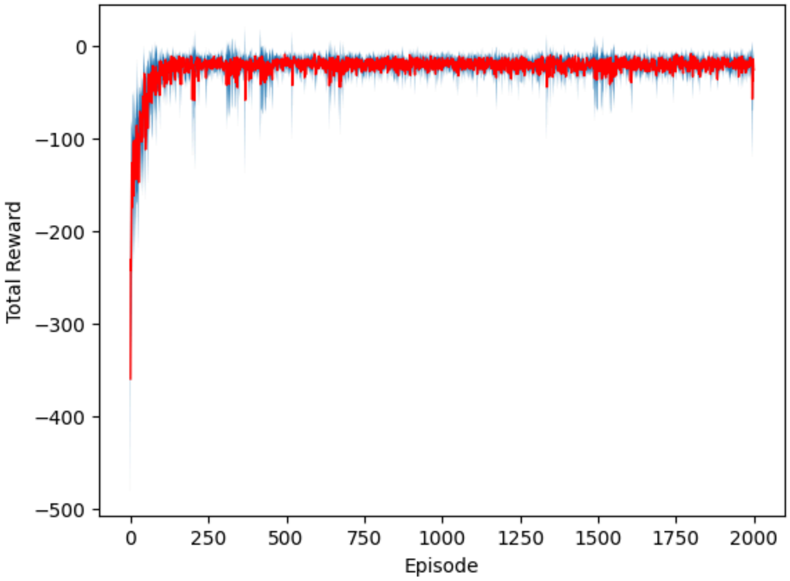
As the episodes increase, average reward converges to around -6 and average number of steps converges around 20.

7. Start state => [3,6] ; Wind => False ; P_transition = 0.7; Policy = Softmax :







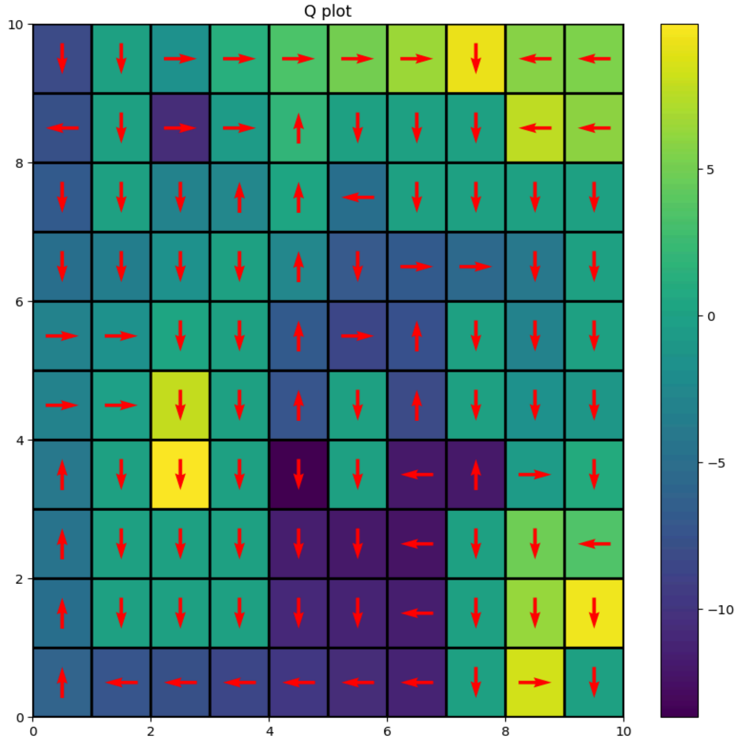


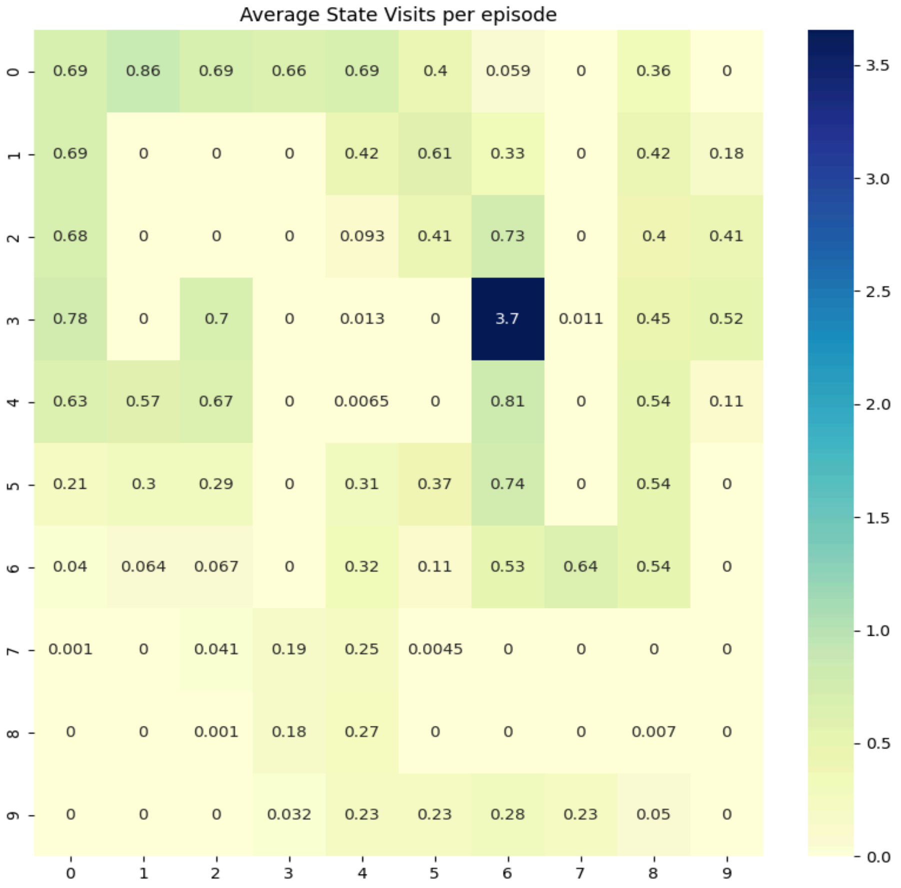
This case is exactly similar to the first case, except the start state is located more in the middle of the environment.

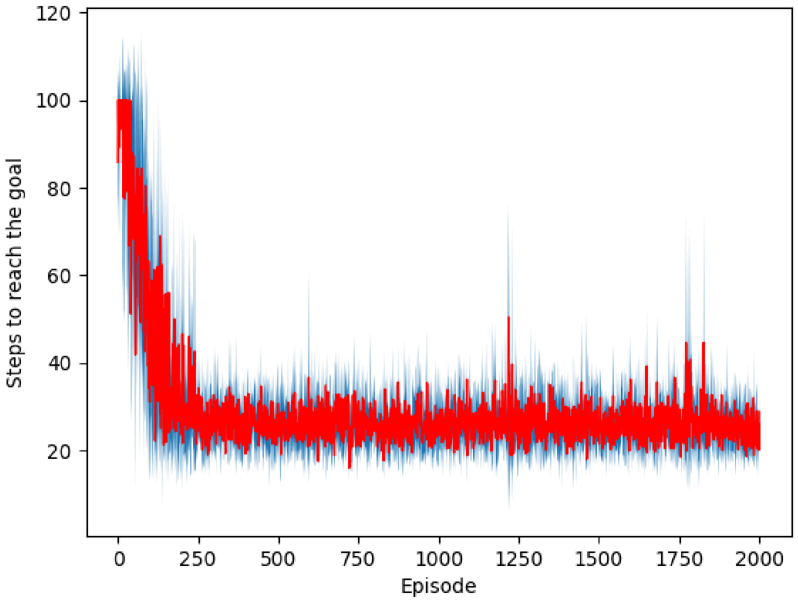
As a result, the policy seems to equally prioritize between the two closest goal states unlike the first case.

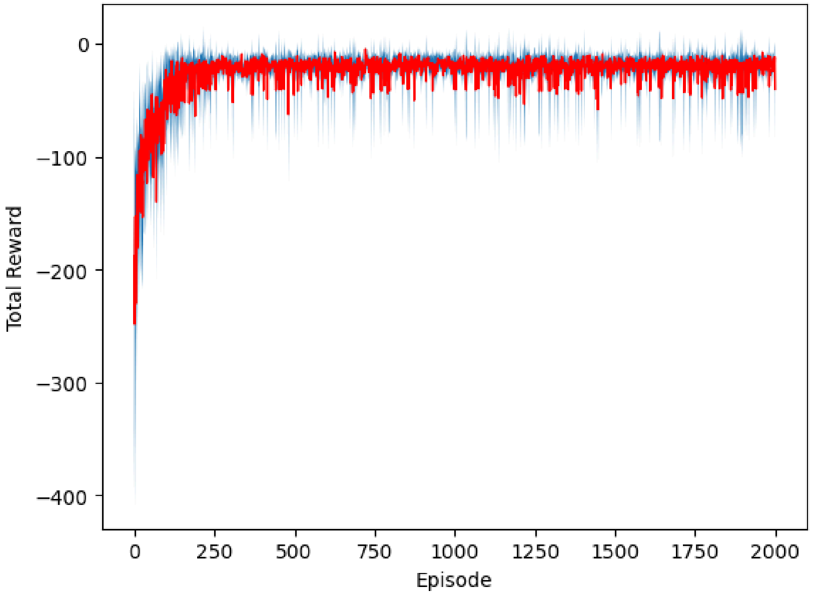
The algorithm converges to 20 steps and -10 reward.

8. Start state => [3,6] ; Wind => False ; P_transition = 0.7; Policy = E-greedy :



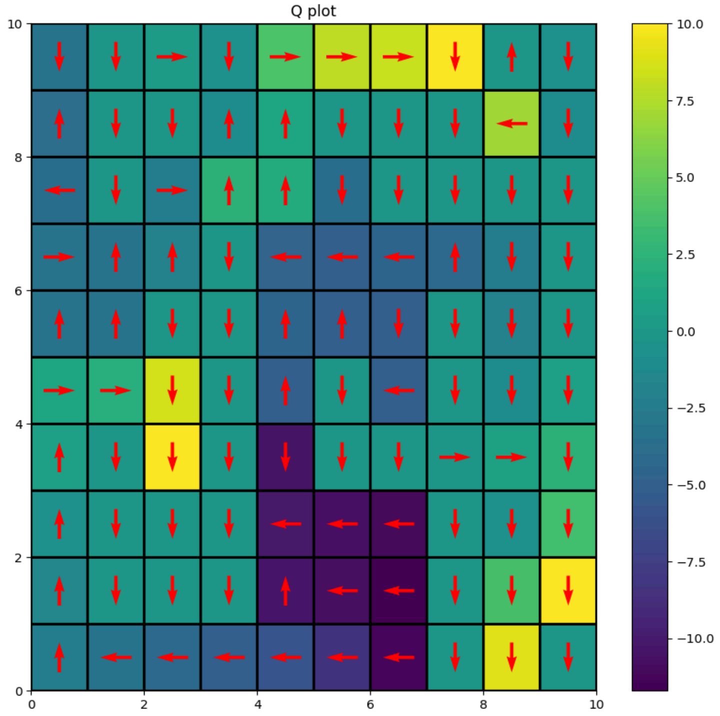


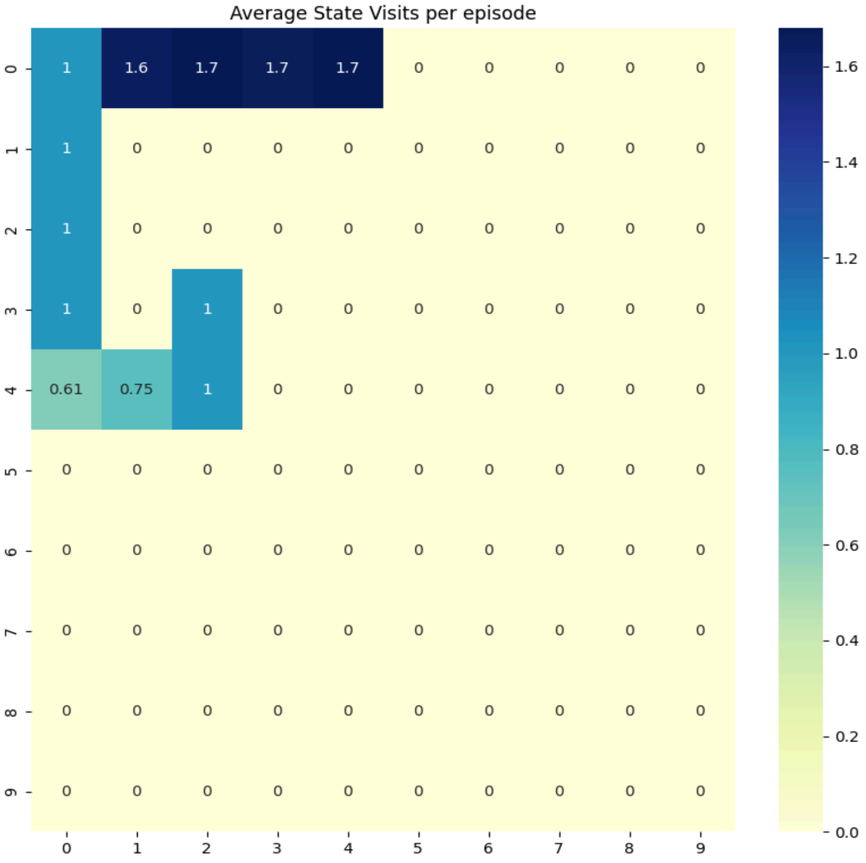


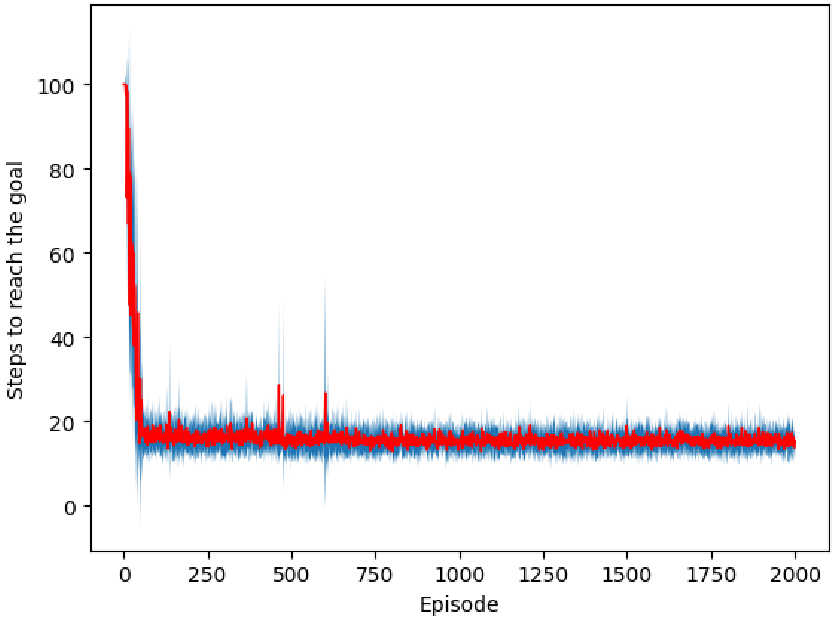


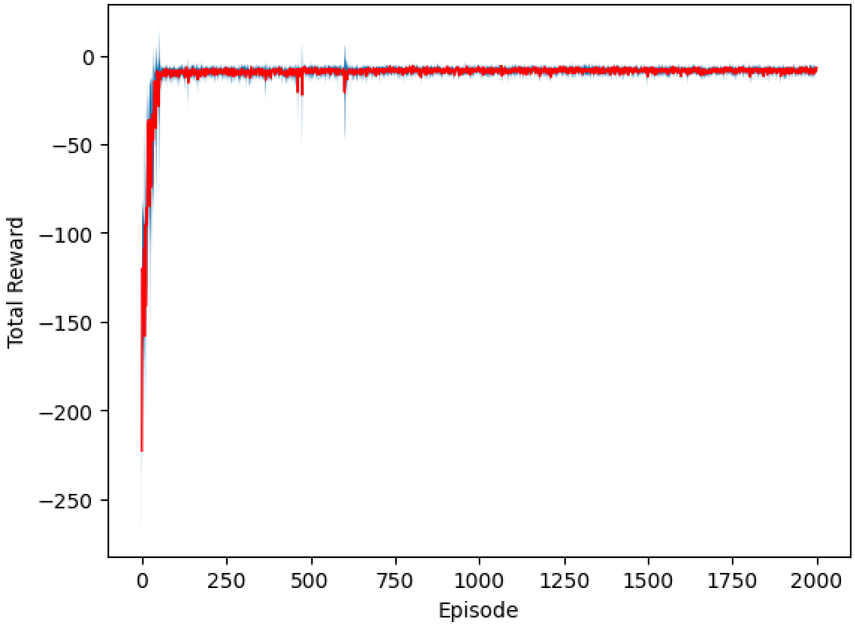
Large noise in the mean total reward and steps to reach the goal even with increasing episode number is because of the random exploration by E-greedy policy.

9. Start state => [0,4] ; Wind => True ; P_transition = 1 ; Policy = Softmax :





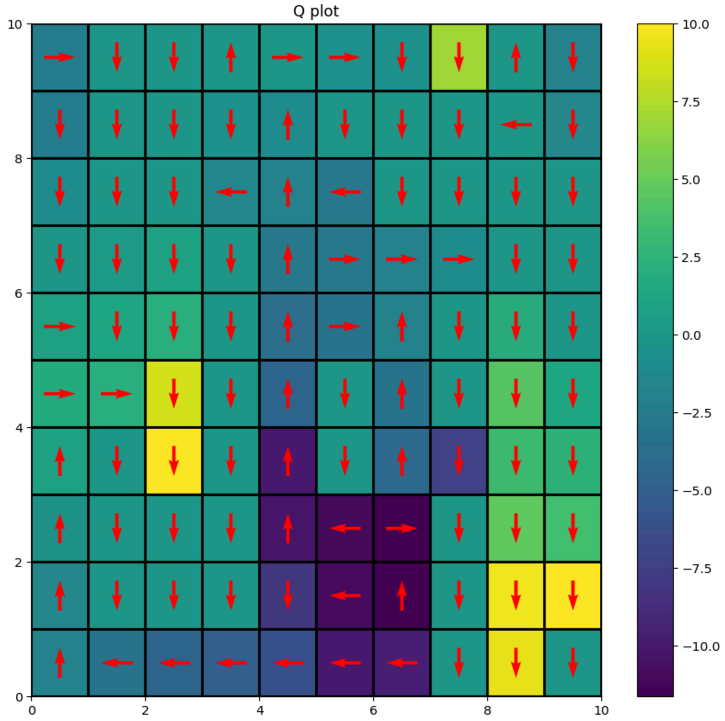


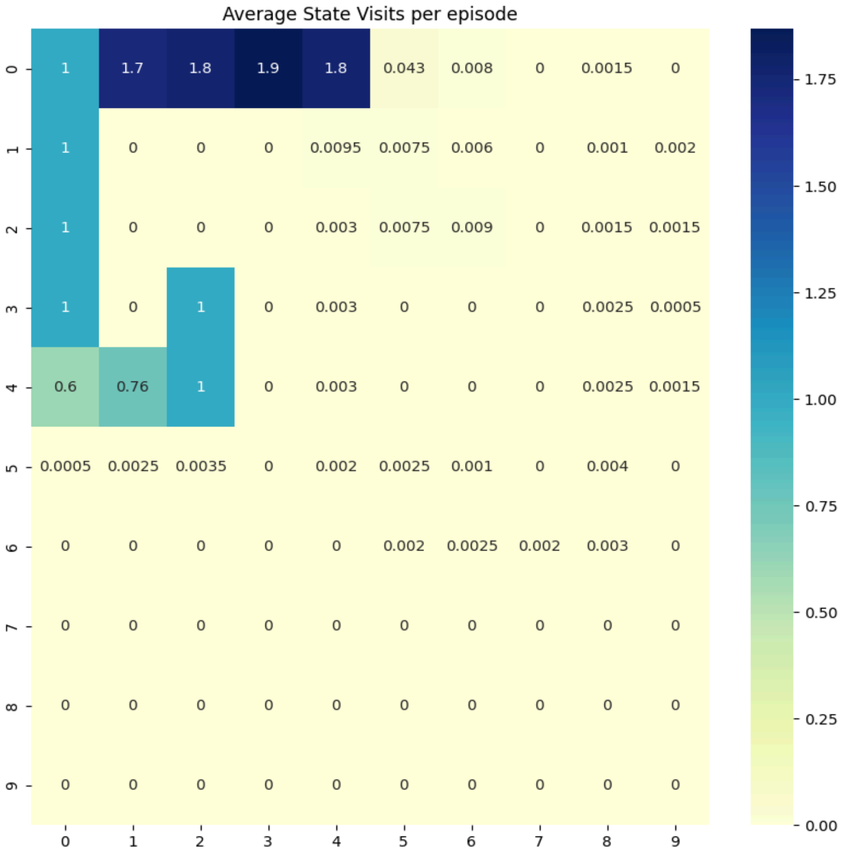


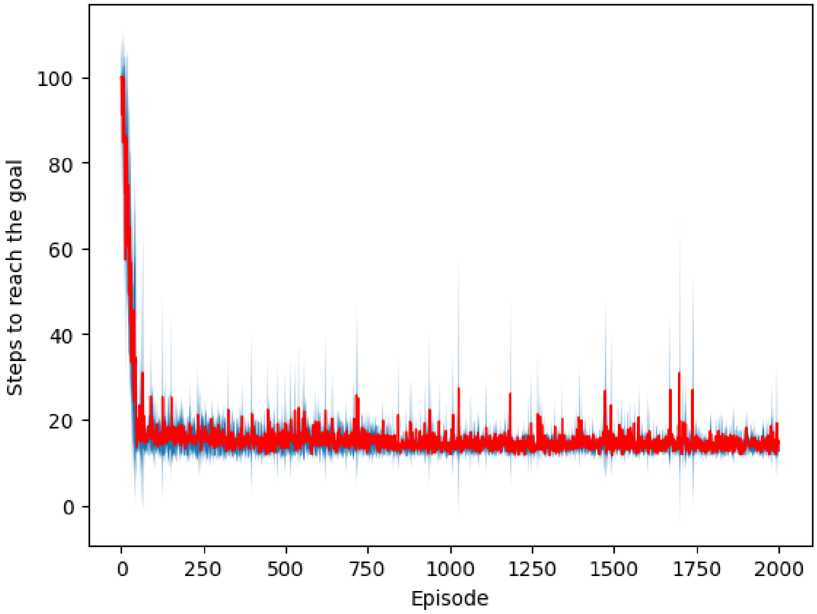
Because of wind, the agent converges to the goal state closer to the starting point. It does seem like it explores the environment a lot.

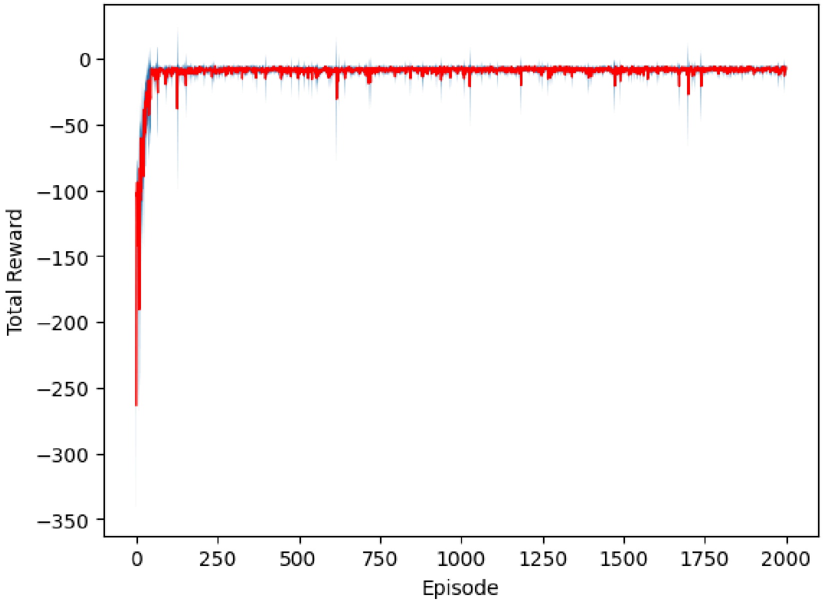
Algorithm converges to around mean total reward -5 and steps at 17.

10. Start state => [0,4] ; Wind => True; P_transition = 1 ; Policy = E-greedy:

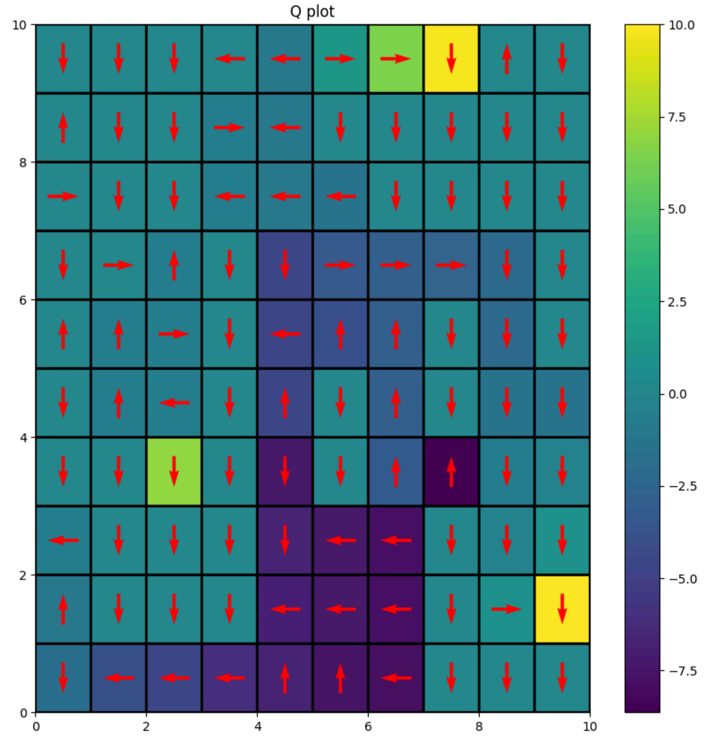


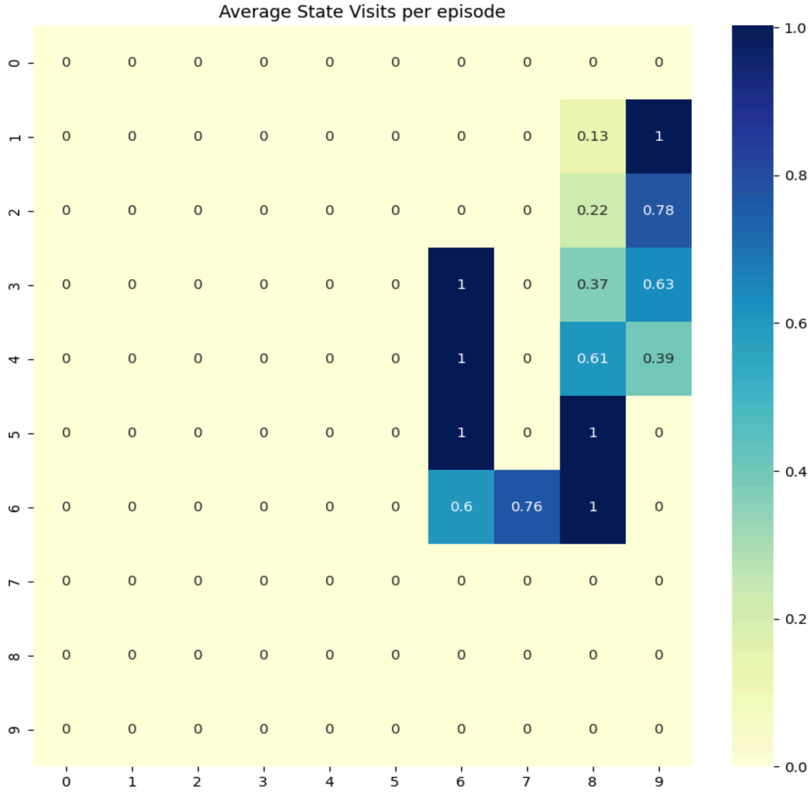



Steps to reach the goal
100
80
60
40
20
0
0
250
500
750
1000
1250
1500
1750
2000
Episode

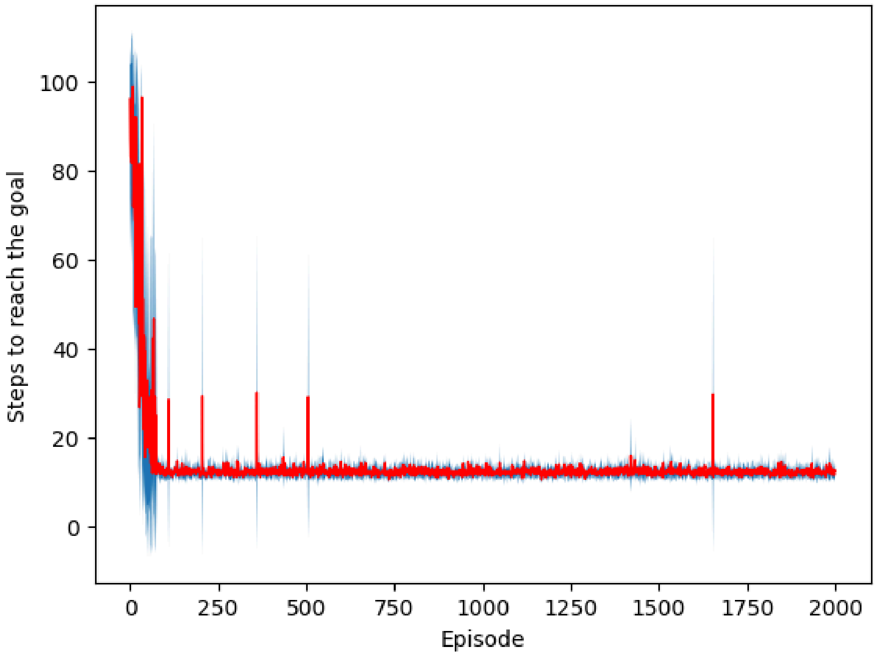

Total Reward
0
−50
−100
−150
−200
−250
−300
−350
0
250
500
750
1000
1250
1500
1750
2000
Episode

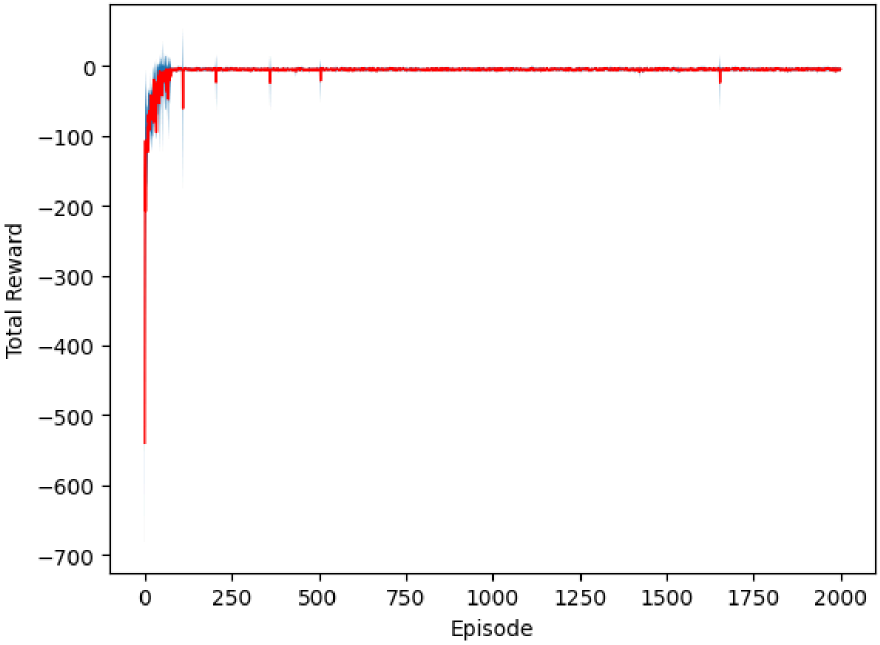
11. Start state => [3,6] ; Wind => True ; P_transition = 1 ; Policy = Softmax :



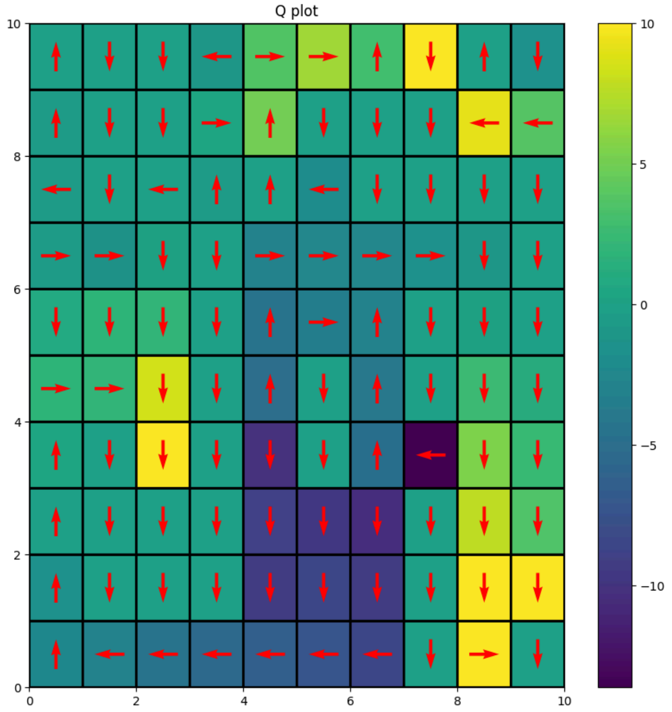


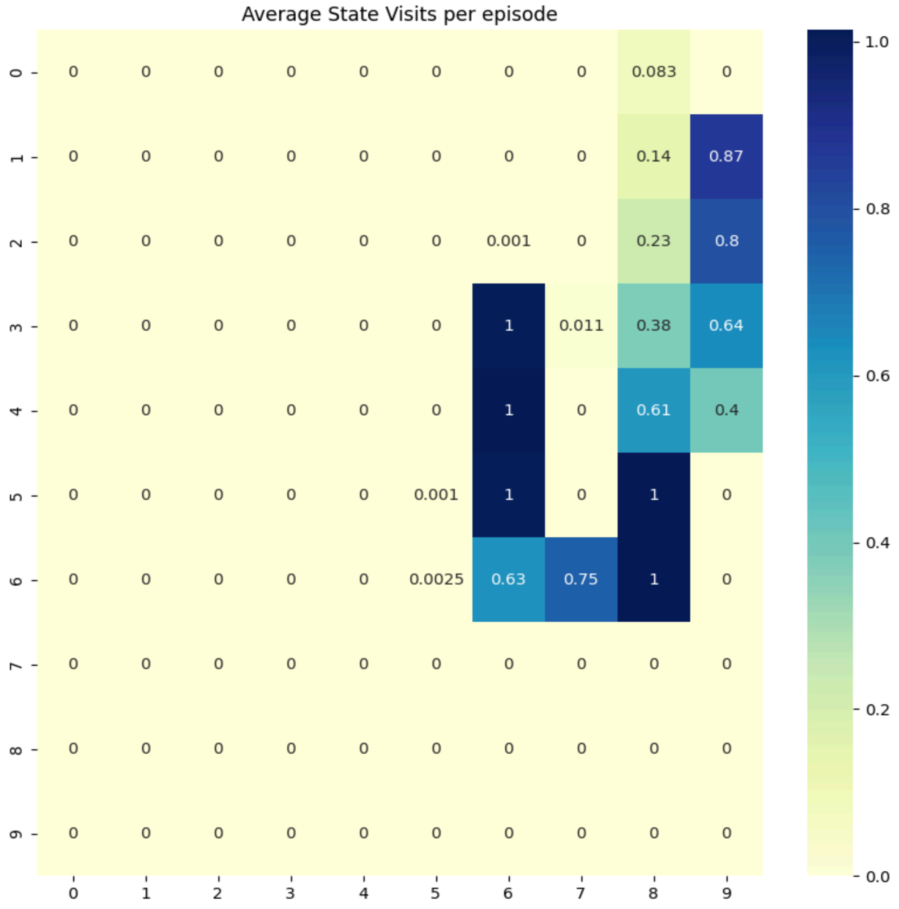
	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0.13	1
2	0	0	0	0	0	0	0	0	0.22	0.78
3	0	0	0	0	0	0	1	0	0.37	0.63
4	0	0	0	0	0	0	1	0	0.61	0.39
5	0	0	0	0	0	0	1	0	1	0
6	0	0	0	0	0	0	0.6	0.76	1	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0


Steps to reach the goal
100
80
60
40
20
0
0
250
500
750
1000
1250
1500
1750
2000
Episode

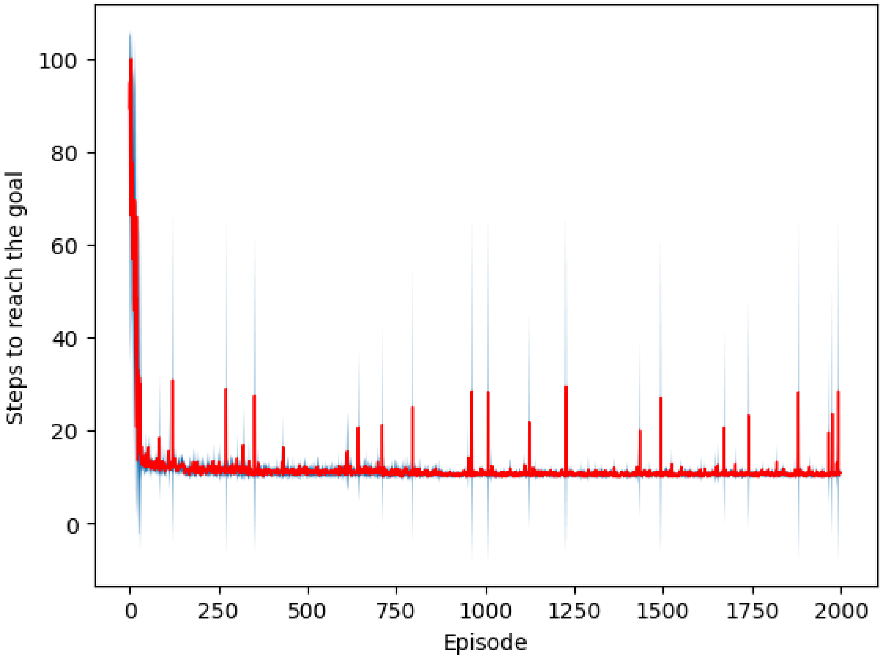

Total Reward
0
−100
−200
−300
−400
−500
−600
−700
0
250
500
750
1000
1250
1500
1750
2000
Episode

12. Start state => [3,6] ; Wind => True ; P_transition = 1 ; Policy = E-greedy:

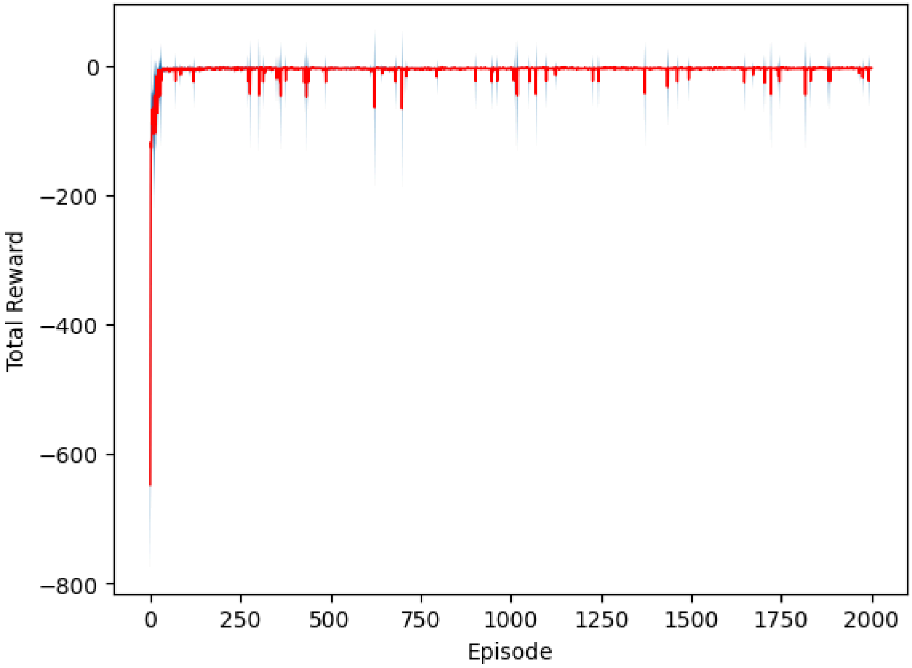




	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0.083	0
1	0	0	0	0	0	0	0	0	0.14	0.87
2	0	0	0	0	0	0	0.001	0	0.23	0.8
3	0	0	0	0	0	0	1	0.011	0.38	0.64
4	0	0	0	0	0	0	1	0	0.61	0.4
5	0	0	0	0	0	0.001	1	0	1	0
6	0	0	0	0	0	0.0025	0.63	0.75	1	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0



Steps to reach the goal
100
80
60
40
20
0
0
250
500
750
1000
1250
1500
1750
2000
Episode



Total Reward
0
−200
−400
−600
−800
0
250
500
750
1000
1250
1500
1750
2000
Episode

Comparison SARSA vs Q-learning :

1. It seems like both SARSA and Q-learning learn similar or same paths to the same goal state.
2. But it seems like, in some cases, Q-learning is able to find the best possible path to the best goal state.
3. However in case of stochasticity in the environment, SARSA opts to find a safer path that avoids restart states.