CS6700: Reinforcement Learning

SARSA Report

- Shaunak Mujumdar (CH21B062)

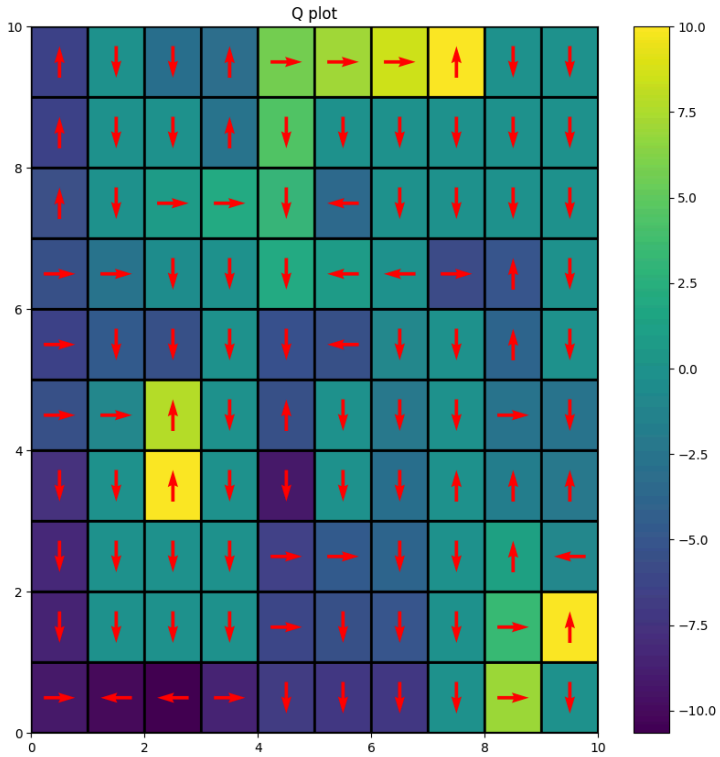
Both of the policies with specific hyper parameters were trained for 1000 episodes. The best hyper parameters were chosen based on high mean reward of the policy for the last 100 episodes of training. The policy with the best hyper parameters was trained for 1000 episodes on 5 different experiments and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

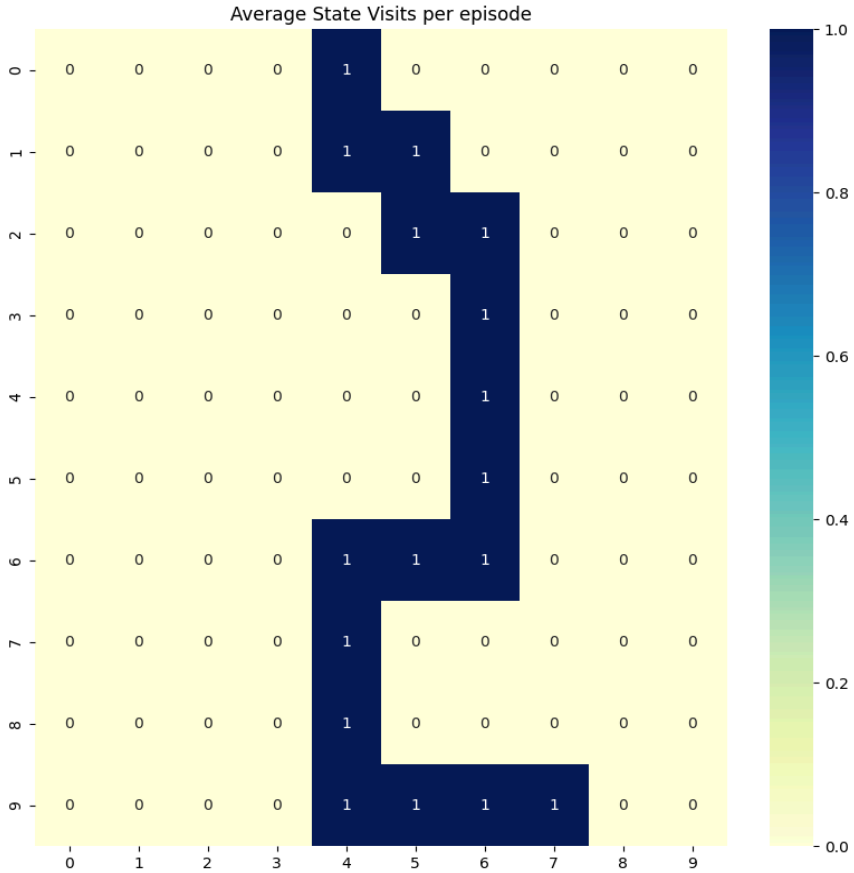
Because of the time constraints, it wasn't possible for us to explore a wide range of hyperparameters for all the given situations. We have only chosen a subset of hyperparameters and the algorithm will test out all possible combinations of these hyperparameters.

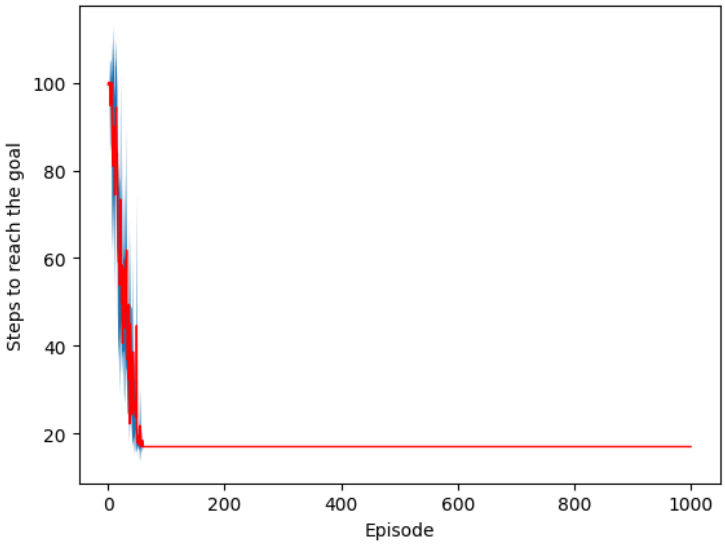
1. Alphas => [0.1,0.2,0.3,0.5,.7]
2. Gammas => [0.1,0.3,0.5,0.7,0.95]
3. Policy params => [0.01,0.05,0.3,0.5]

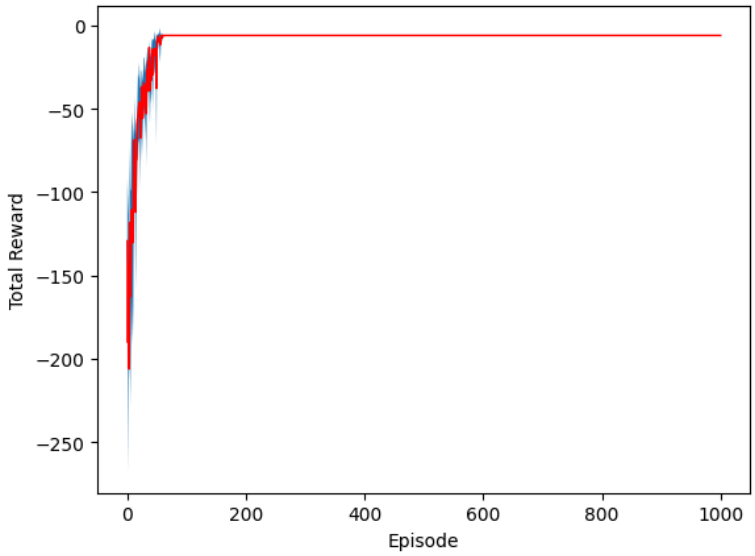
Start state	Wind	Transition Probability	Action Policy	Best Learning Rate (α)	Best Discount Factor (γ)	Best Policy Parameter
[0,4]	False	1	Softmax	0.7	0.95	0.2
[0,4]	False	1	E-greedy	0.7	0.95	0.01
[3,6]	False	1	Softmax	0.7	0.95	0.05
[3,6]	False	1	E-greedy	0.5	0.7	0.01
[0,4]	False	0.7	Softmax	0.5	0.95	0.2
[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	0.2
[0,4]	True	1	E-greedy	0.3	0.95	0.01
[3,6]	True	1	Softmax	0.5	0.7	0.01
[3,6]	True	1	E-greedy	0.2	0.95	0.01

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



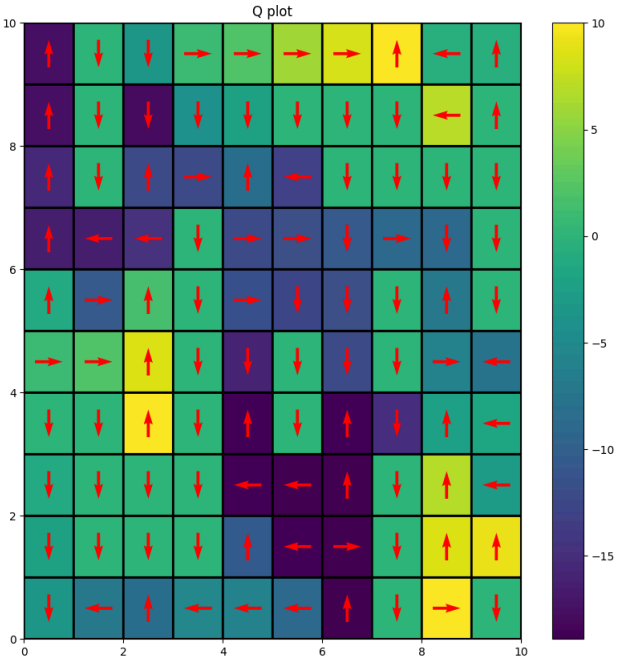


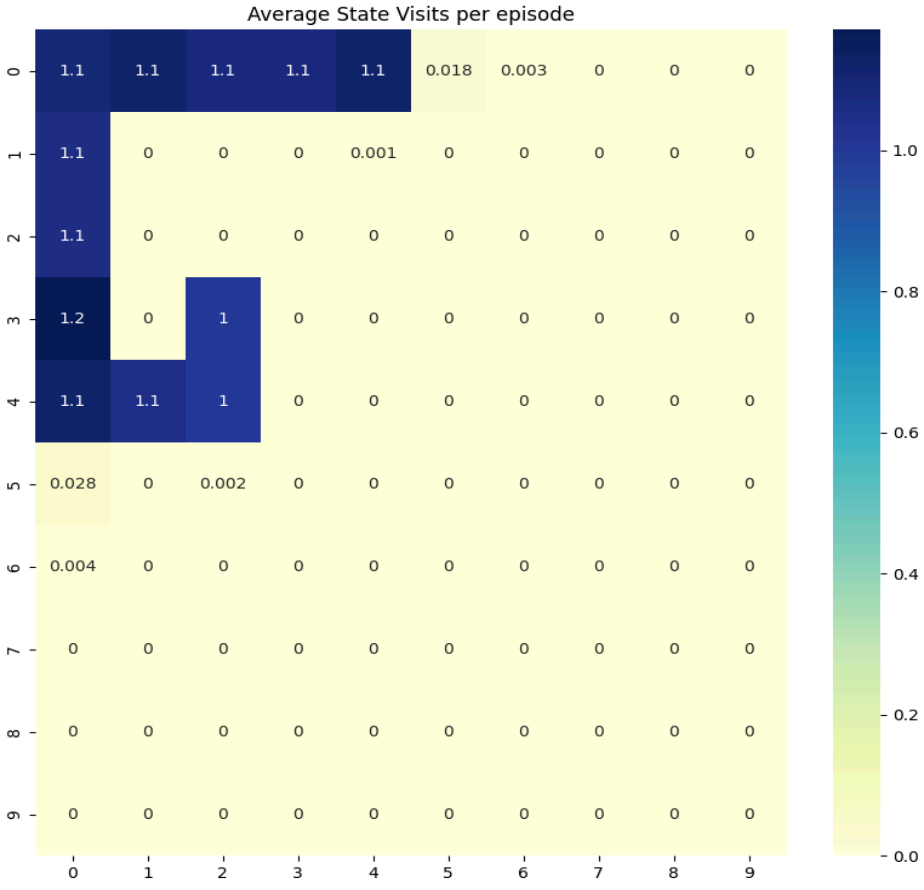


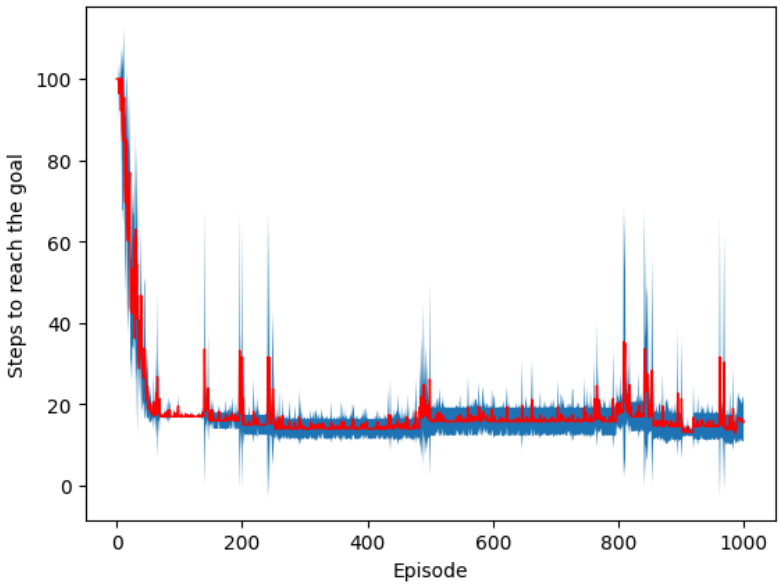


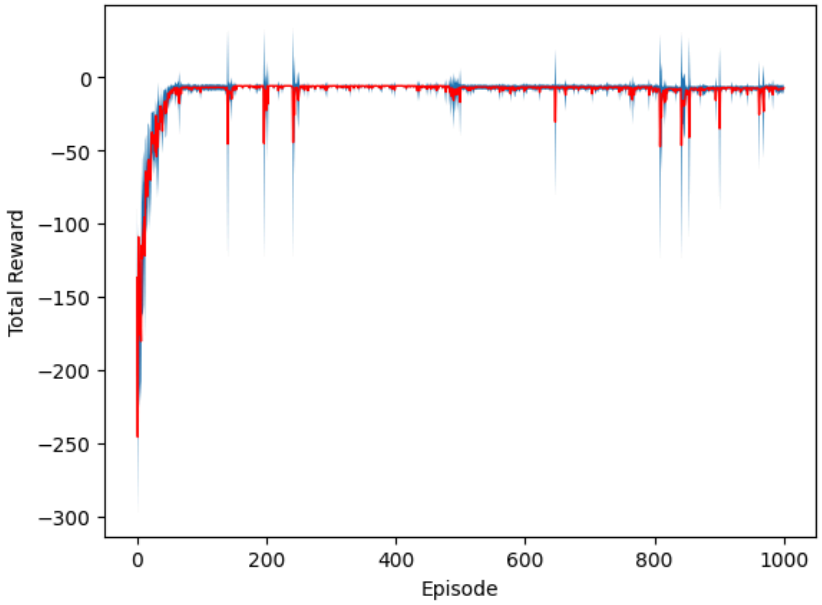
By making p=1, we're removing all elements of stochasticity from the environment.The agent immediately learns a route to a goal state and keeps following this path for all following episodes. From the state visit heat map , you can see that policy never ends up accidentally exploring the environment 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:



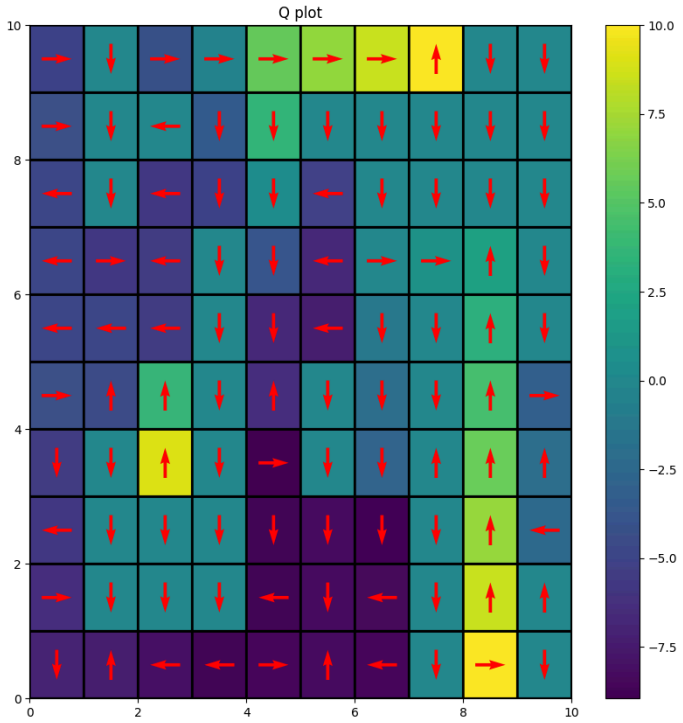


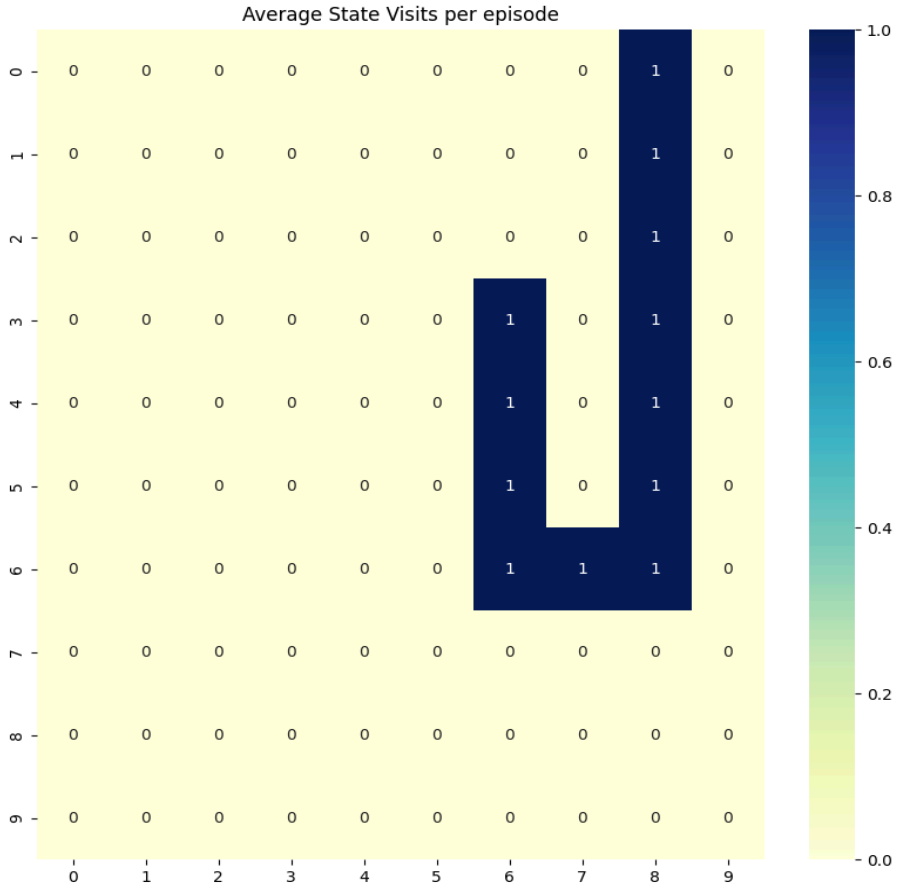


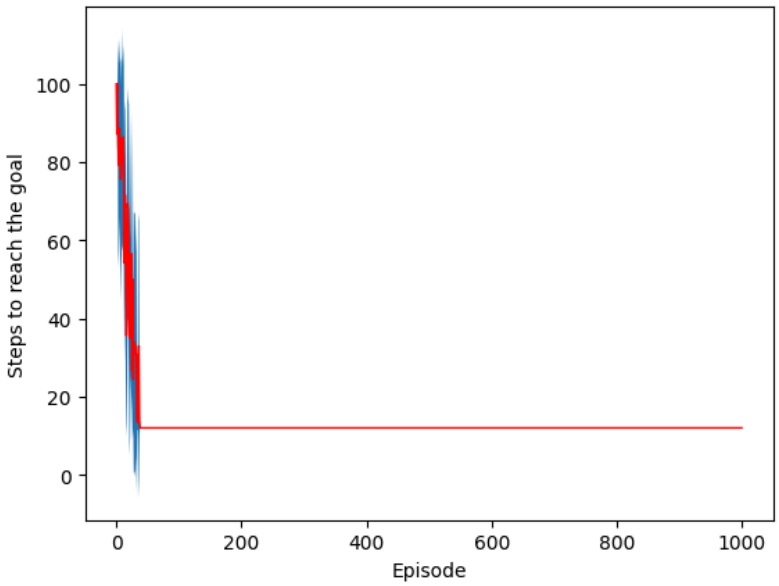


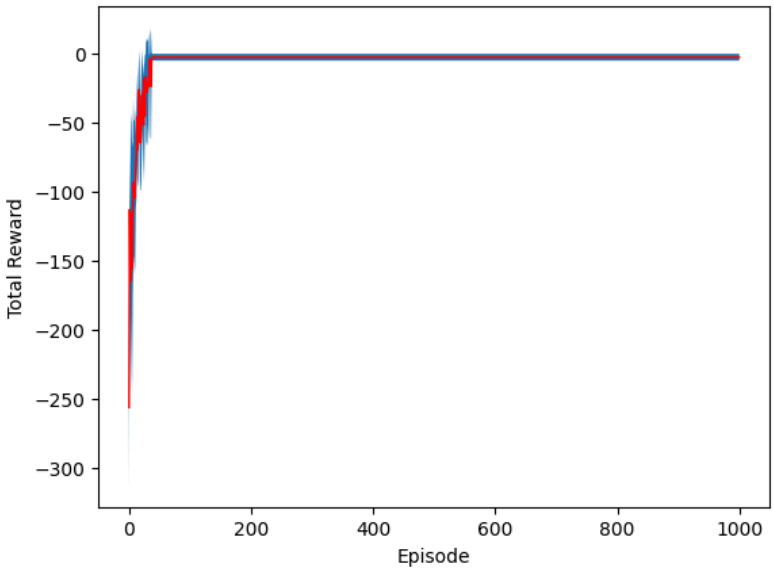
Here, the environment is completely deterministic, thus it learns the closest path with high reward and is unable to learn the best path due to less exploration. The algorithm converges to around 12 steps and -6 reward.
Some of the exploration you see in the reward, step plots and heat graphs is because of the epsilon - greedy action policy.

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



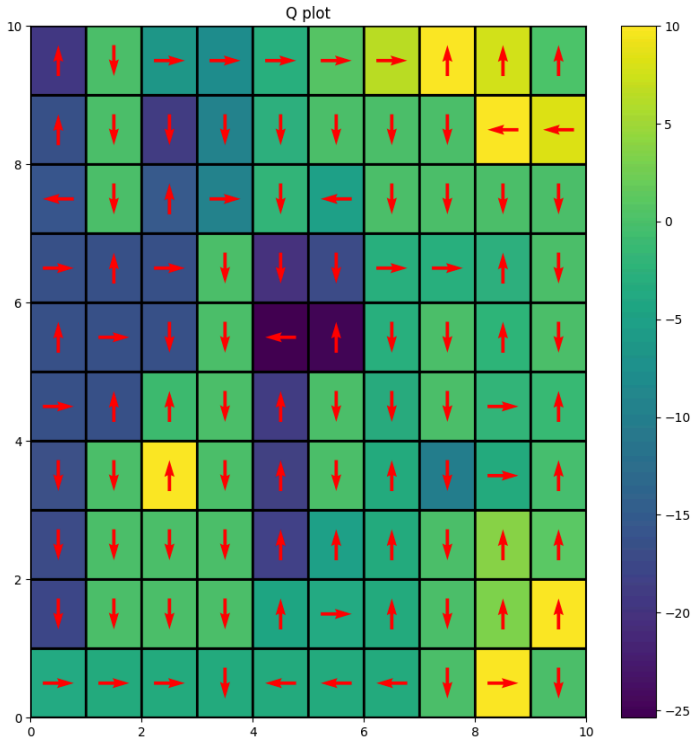


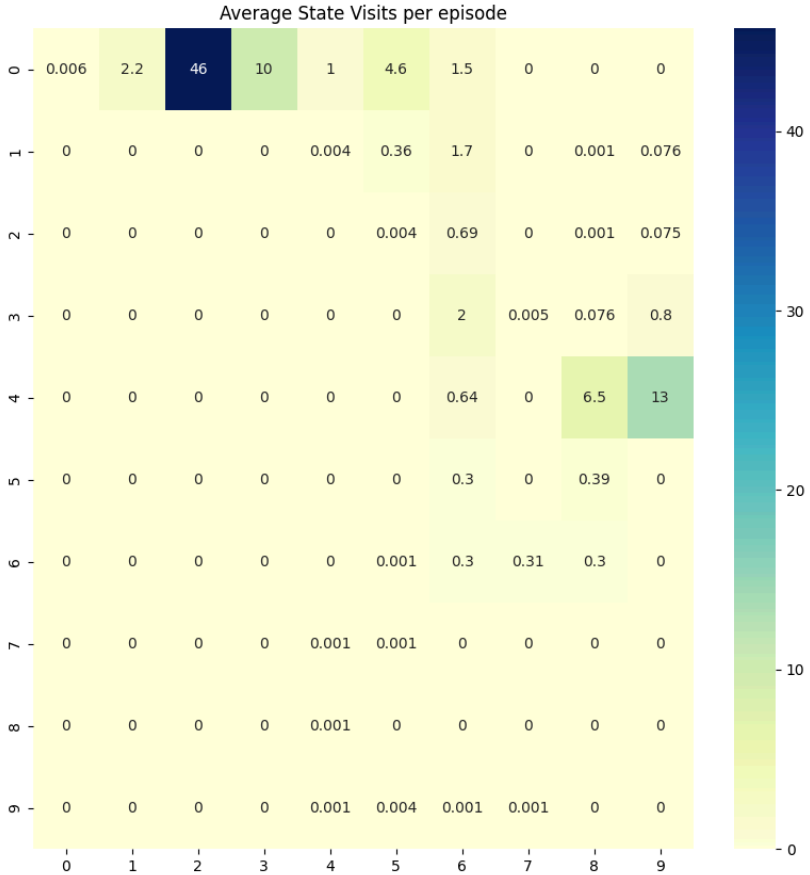


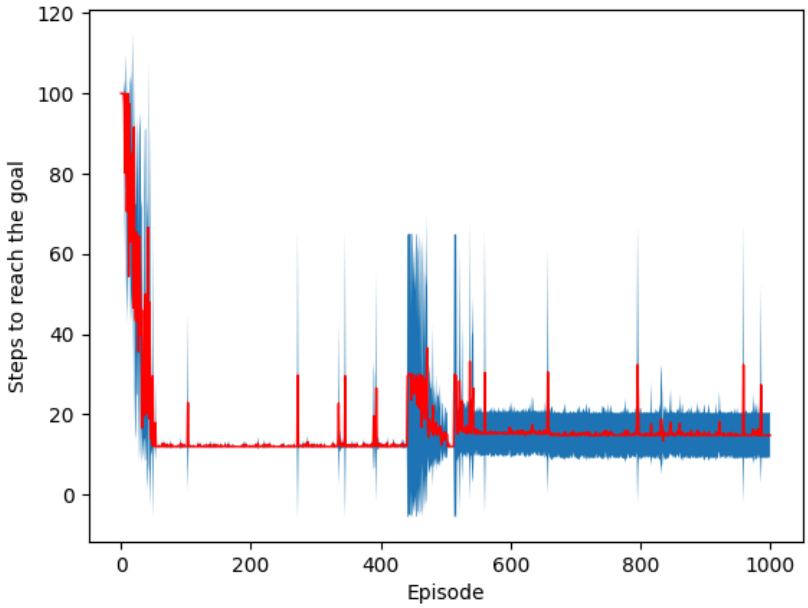


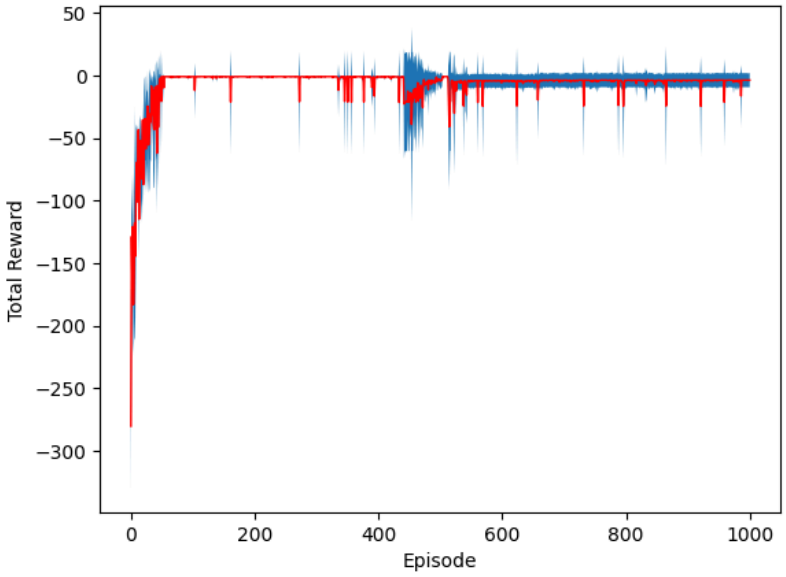
Just like the previous deterministic environment, here, as p = 1 and there is no wind effect, the stochasticity in the environment is completely removed. Thus it is a deterministic environment. The policy finds the best path to the second goal which avoids all restart states and other high negative rewards. Due to the deterministic nature, the policy doesn't discover any of the other goal states. Algorithm converges to 12 steps and a mean reward of -1 (as expected for that goal state).

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



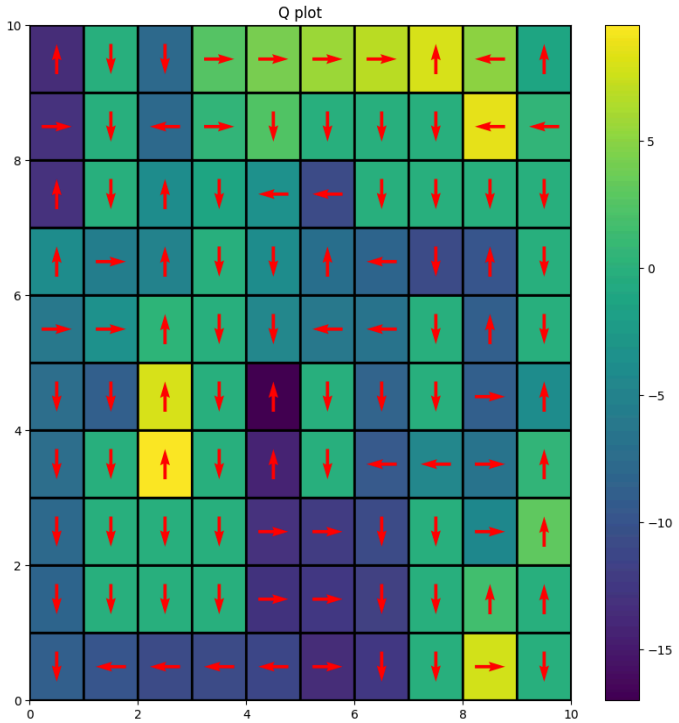


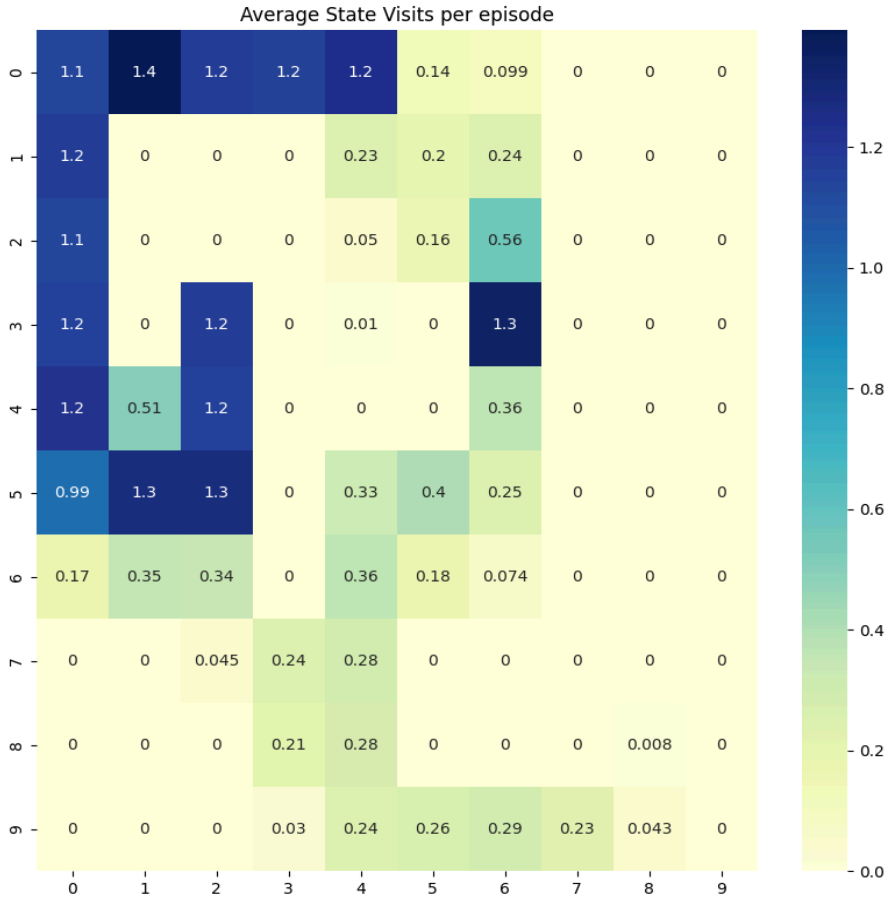


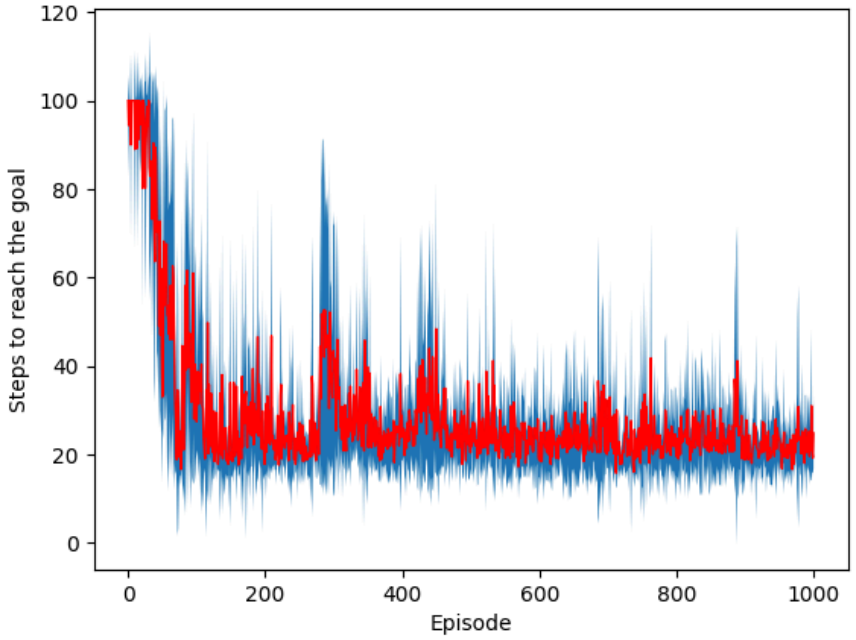


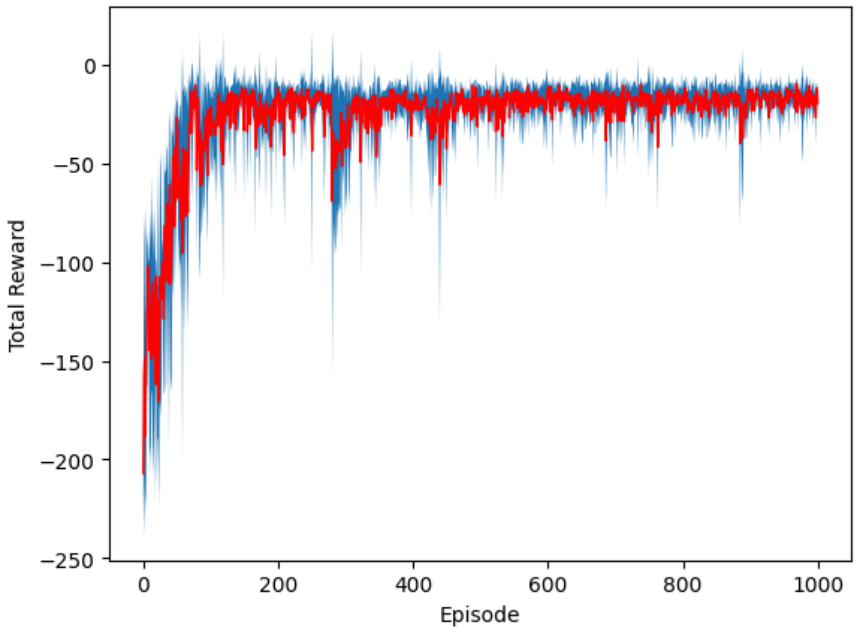
Algorithm converges to total reward 0 and the steps to reach the goal converges to 17 steps. Also it seems there are lots of random bumps in reward and step plots that is because of random exploration done by the E-greedy action policy.

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



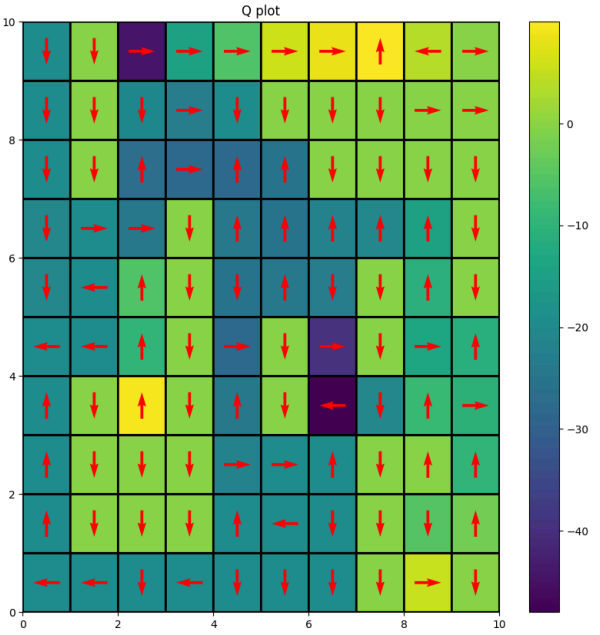


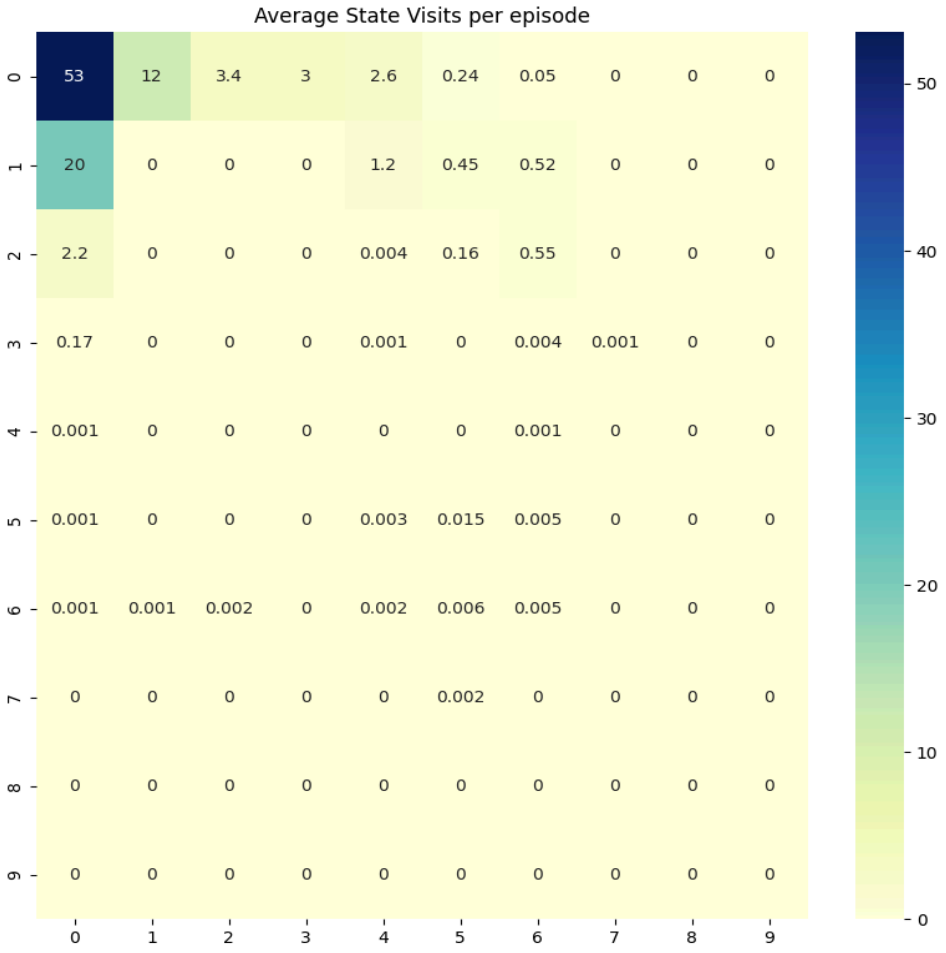


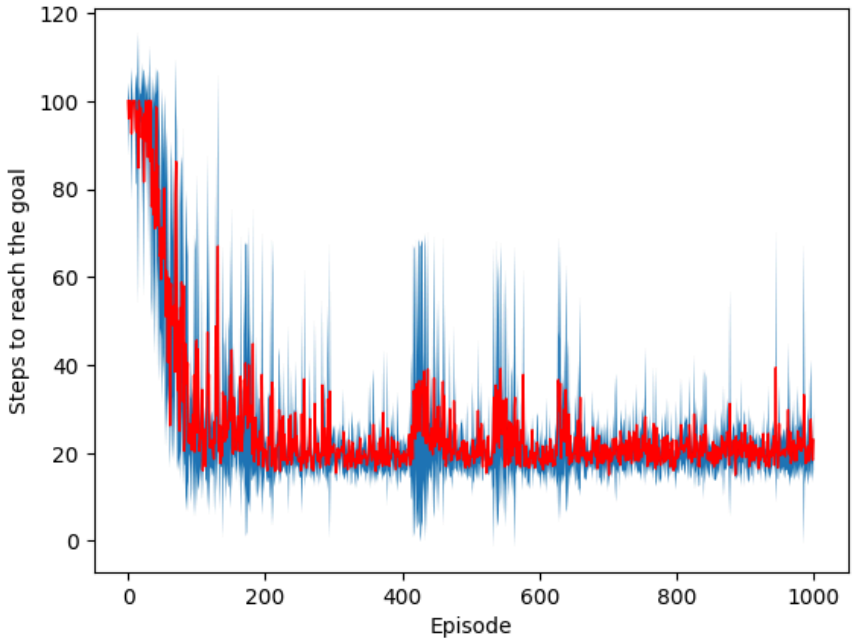


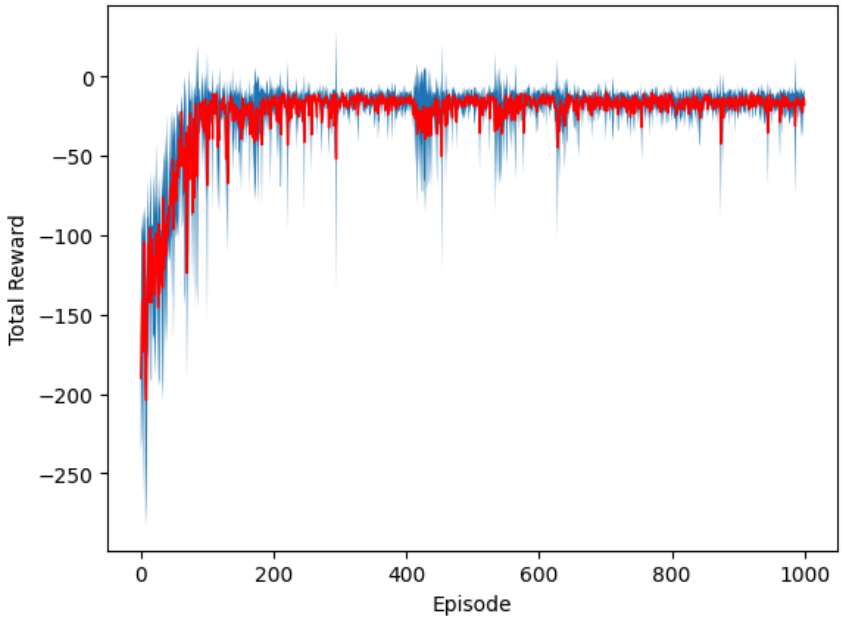
Algorithm learns the best path to the nearest goal (one with the highest reward) among the three. The path second goal state also receives a few hits as the reward difference between two goal states is very small.
Algorithm converges to around 20 steps and a mean reward of -15.

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



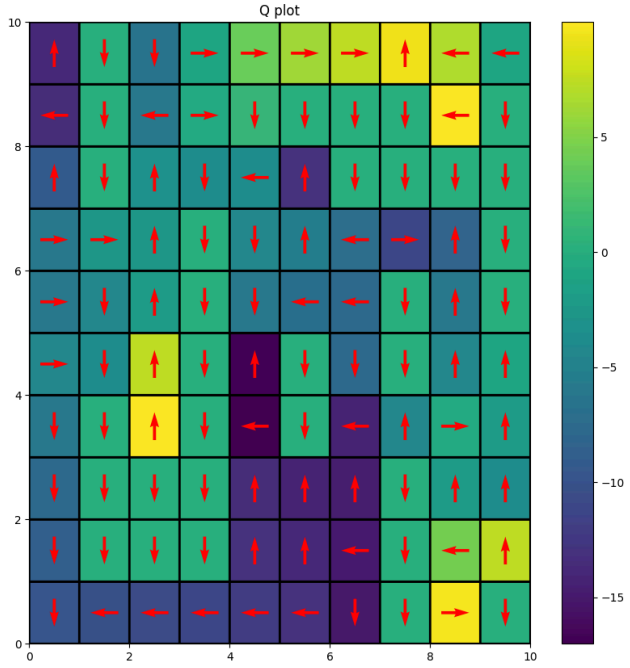


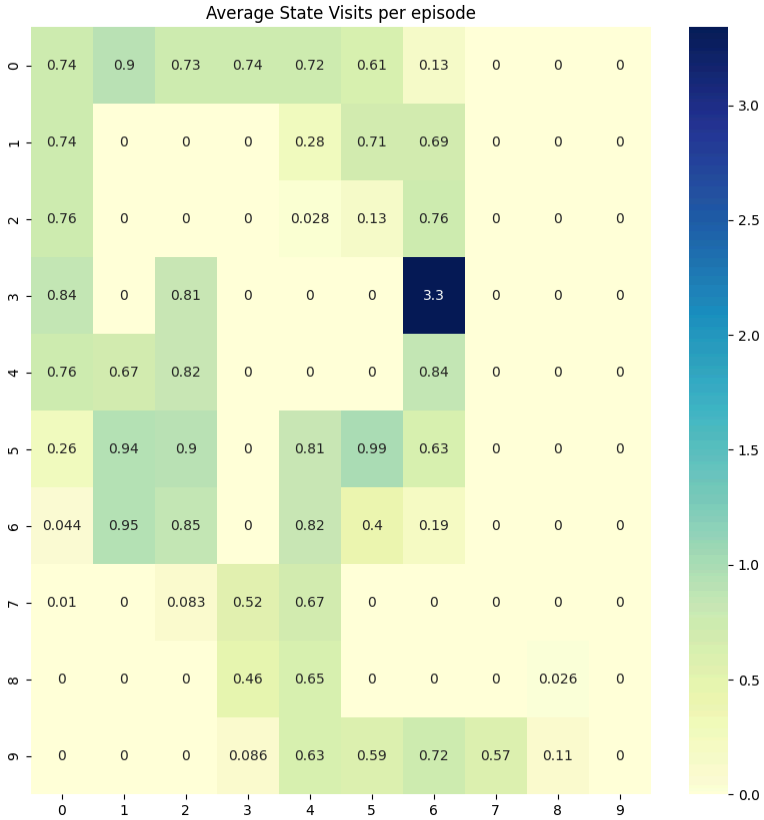


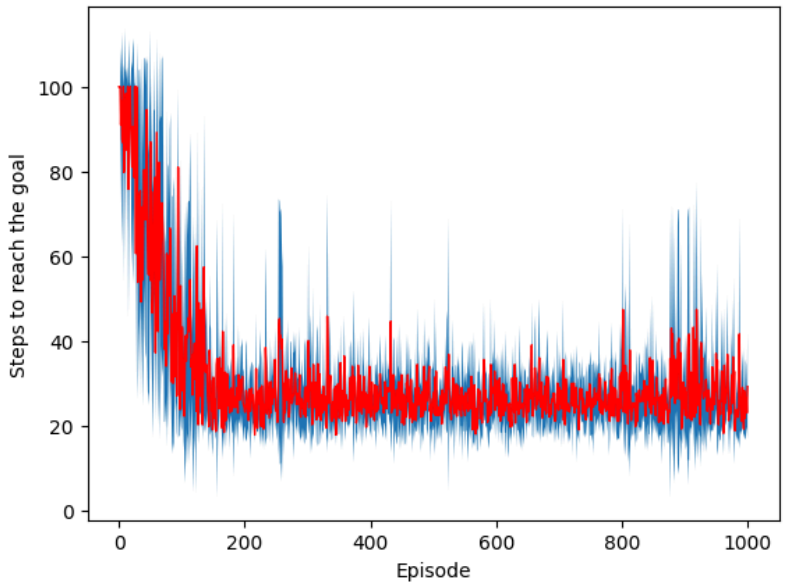


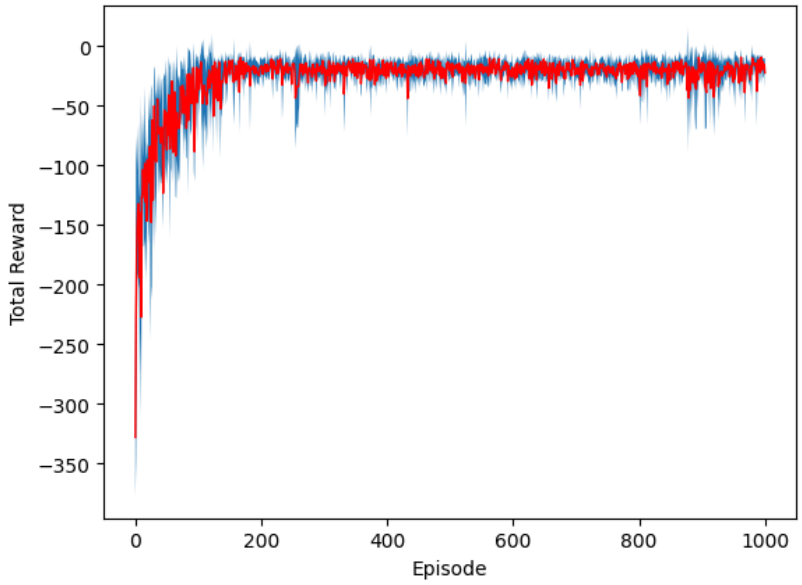
Algorithm converges to around 20 steps and a mean reward of -15.
Also as you can see from the state visit heatmap, there is a lot more exploration of other states than the previous softmax action policy.

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



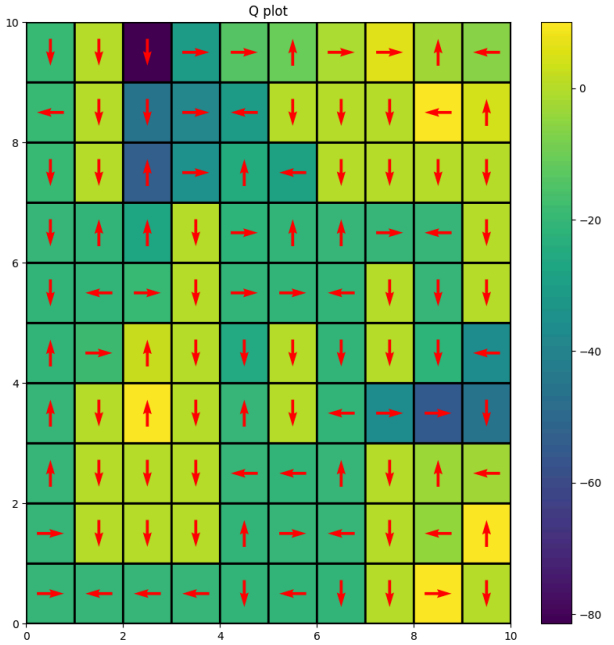


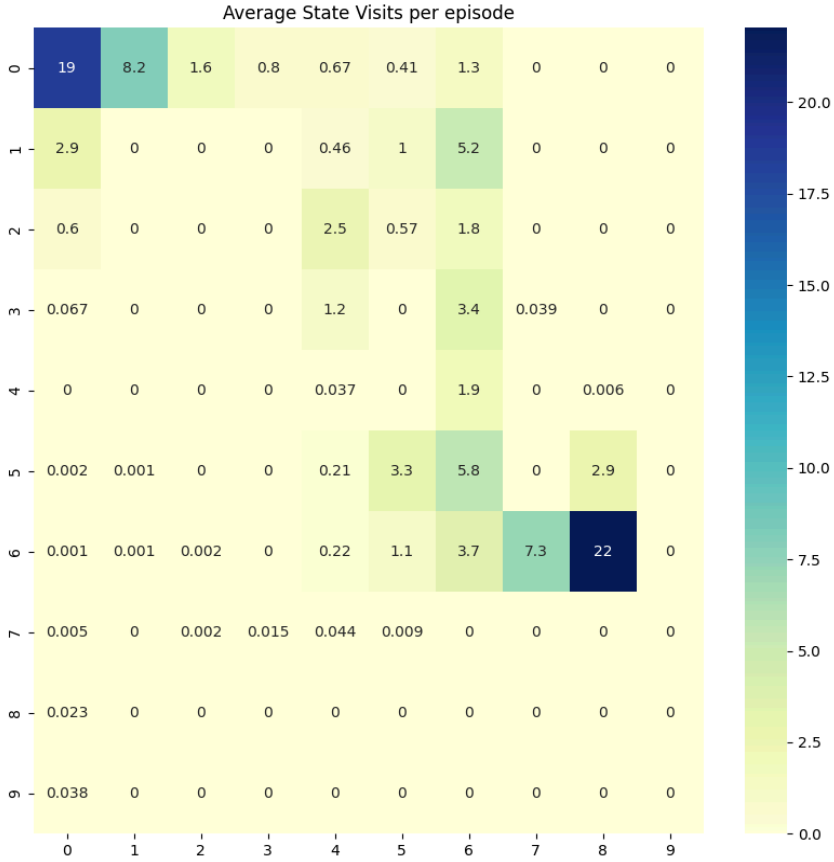


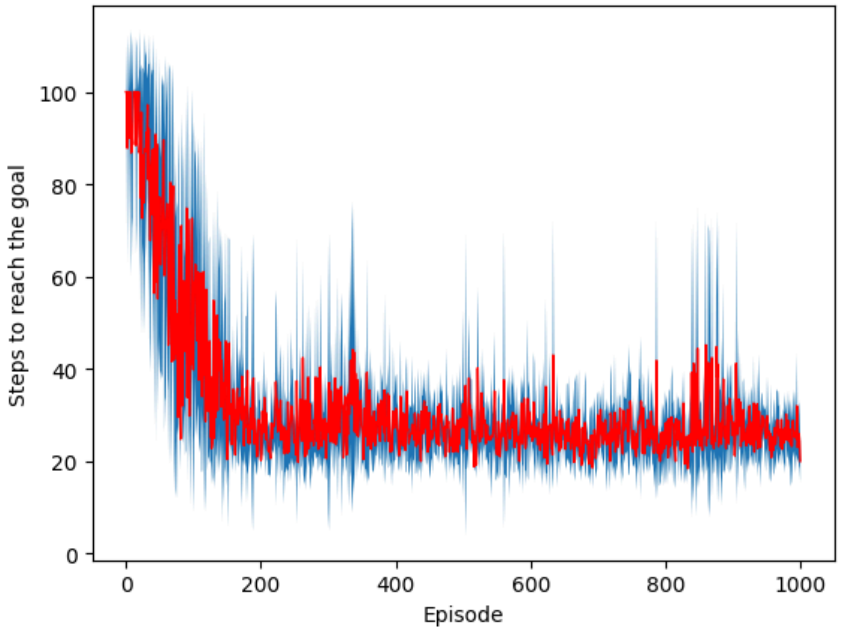


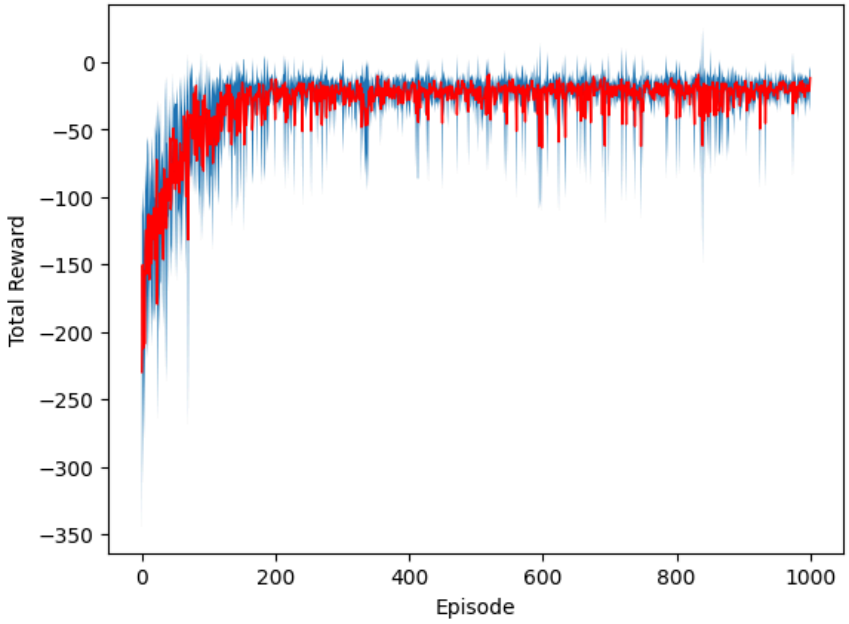
Algorithm converges to around 20 steps and a mean reward of -15.
The stochasticity comes from the transition probability. It ends up exploring all the goal states, stochasticity of the model makes it unable to detect the best route possible. It tries to find the safest route possible.

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



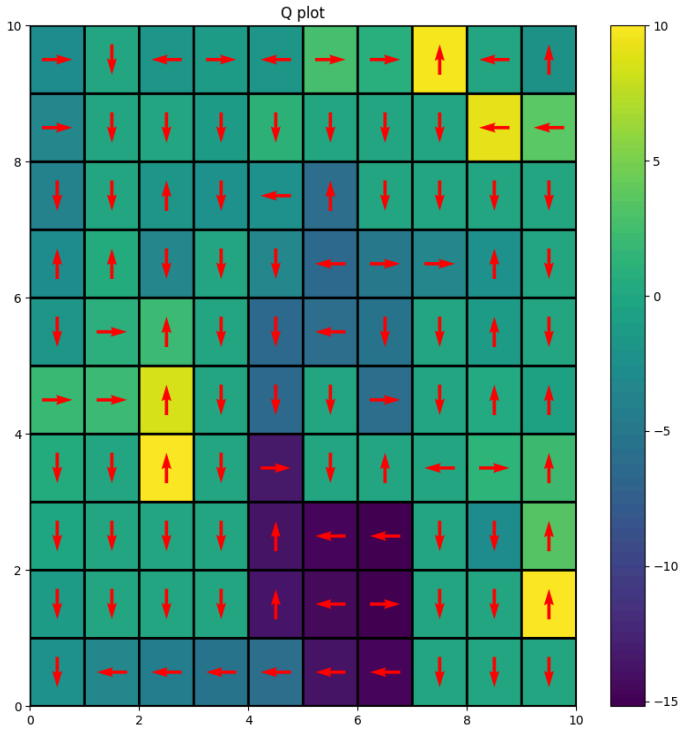


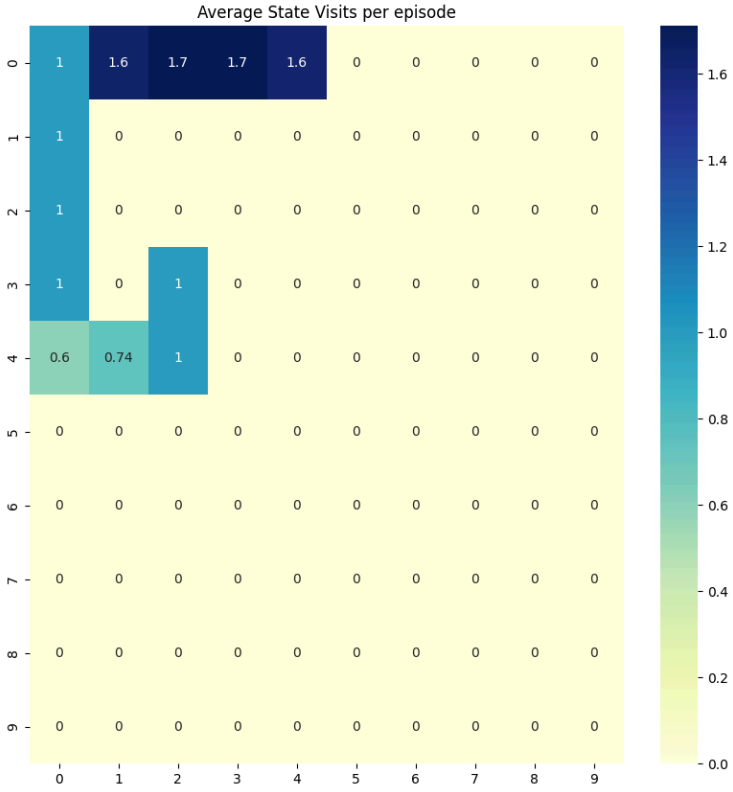


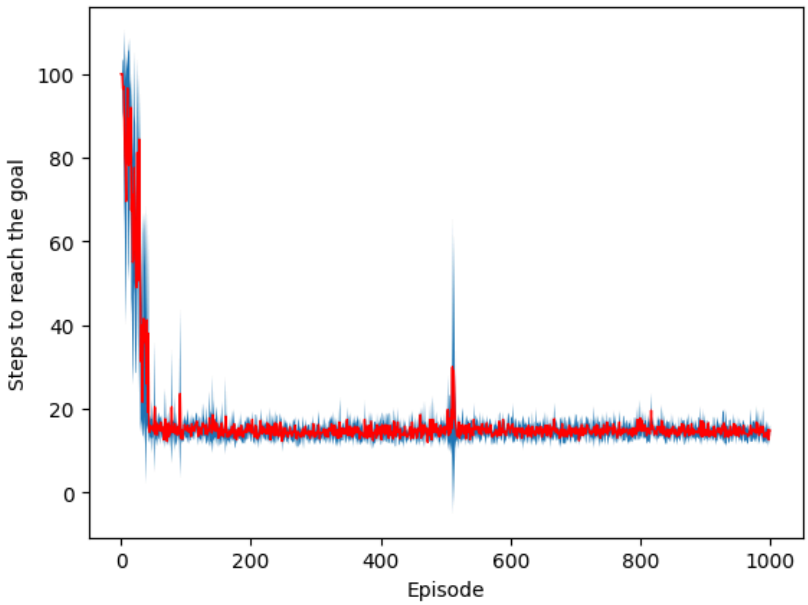


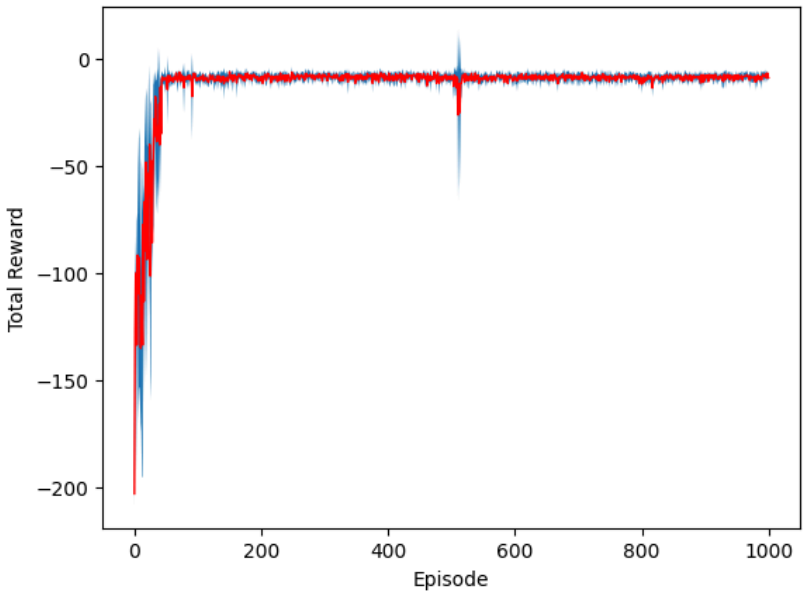
Again in this case, the transition probability of 0.7 introduces stochasticity in the model. Here also it tries to find the safe route instead of best possible route.

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



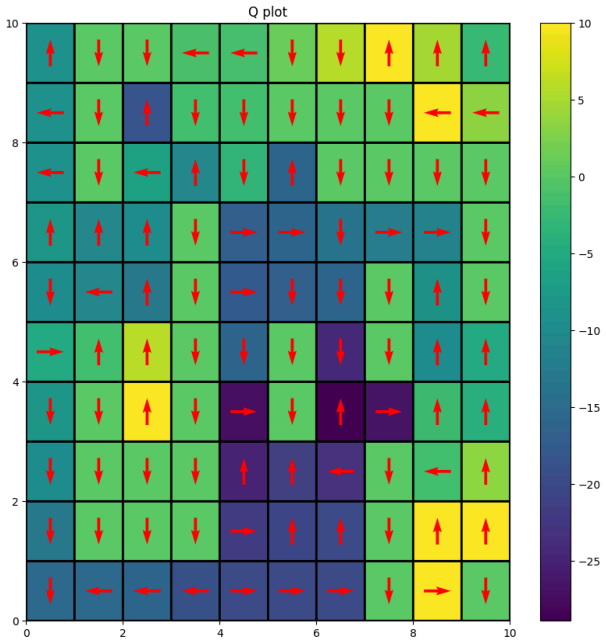


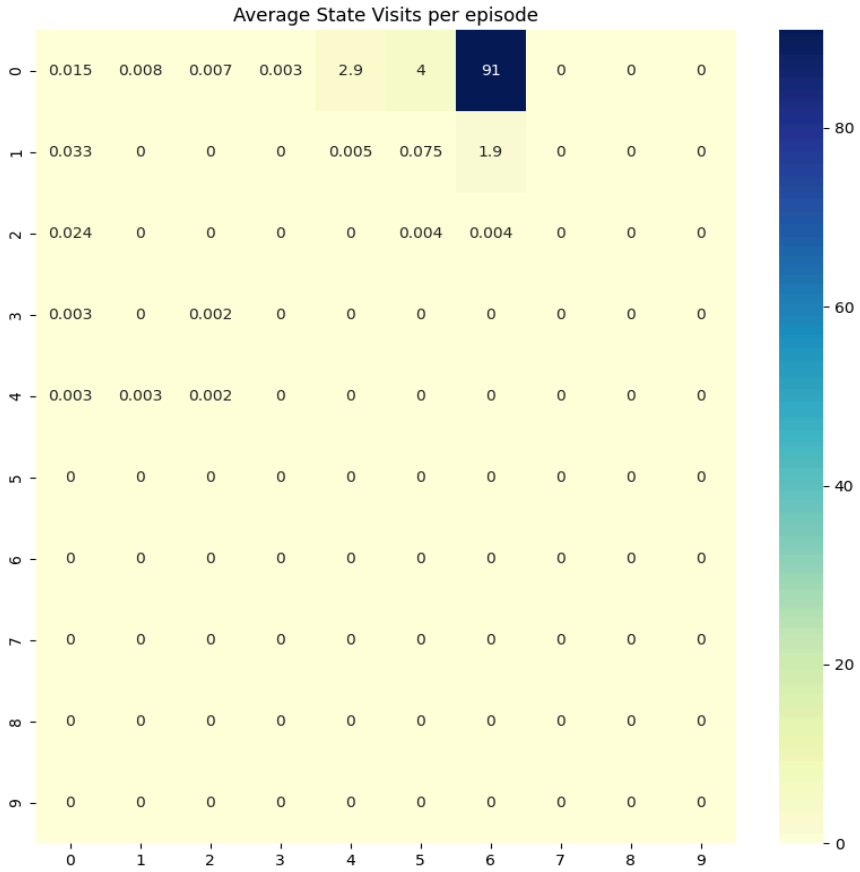


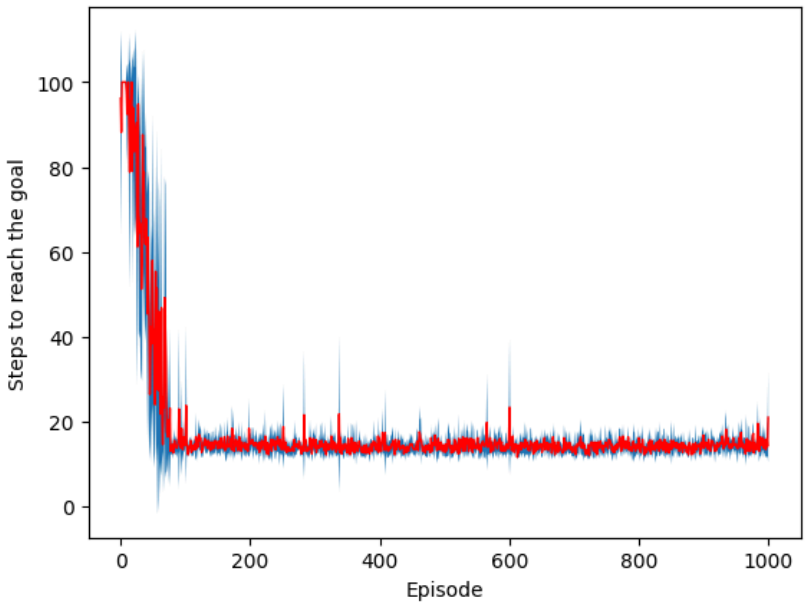


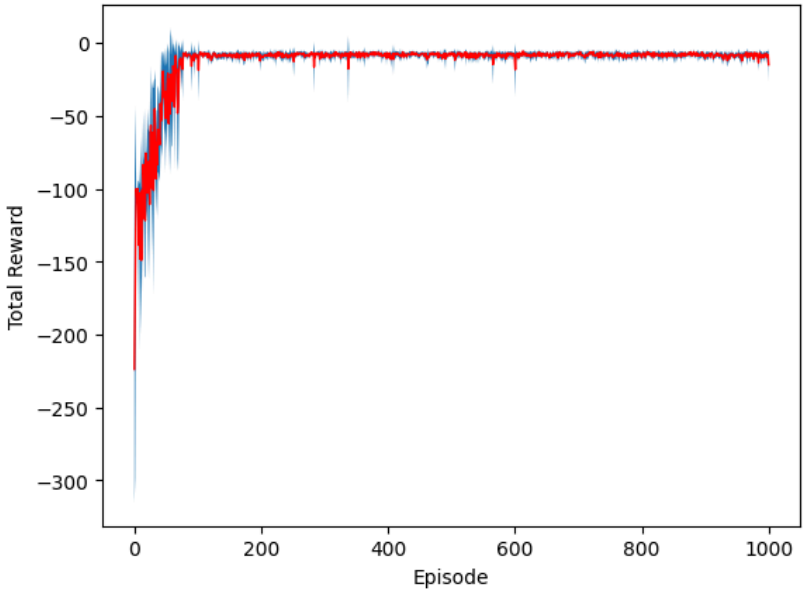
Wind effect introduces stochasticity in this case. Because of that algorithm seems to go for first goal state instead of second one because the second one has a higher chance of getting more negative rewards when affected by the wind(due to restart states etc.). Thus is chooses the first goal state which has the least risk.

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

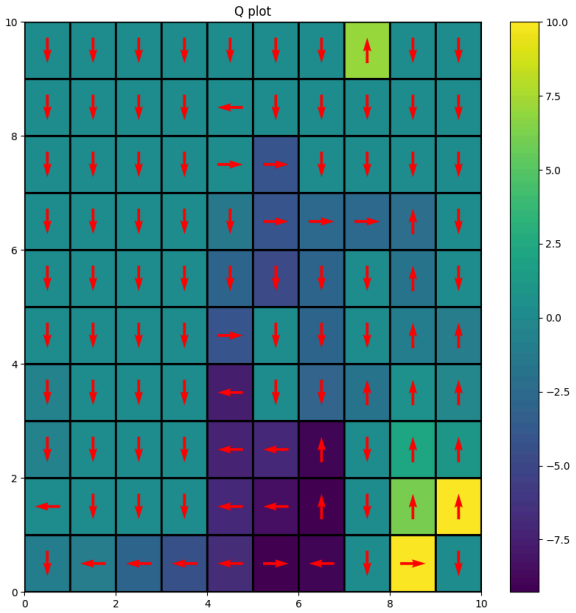


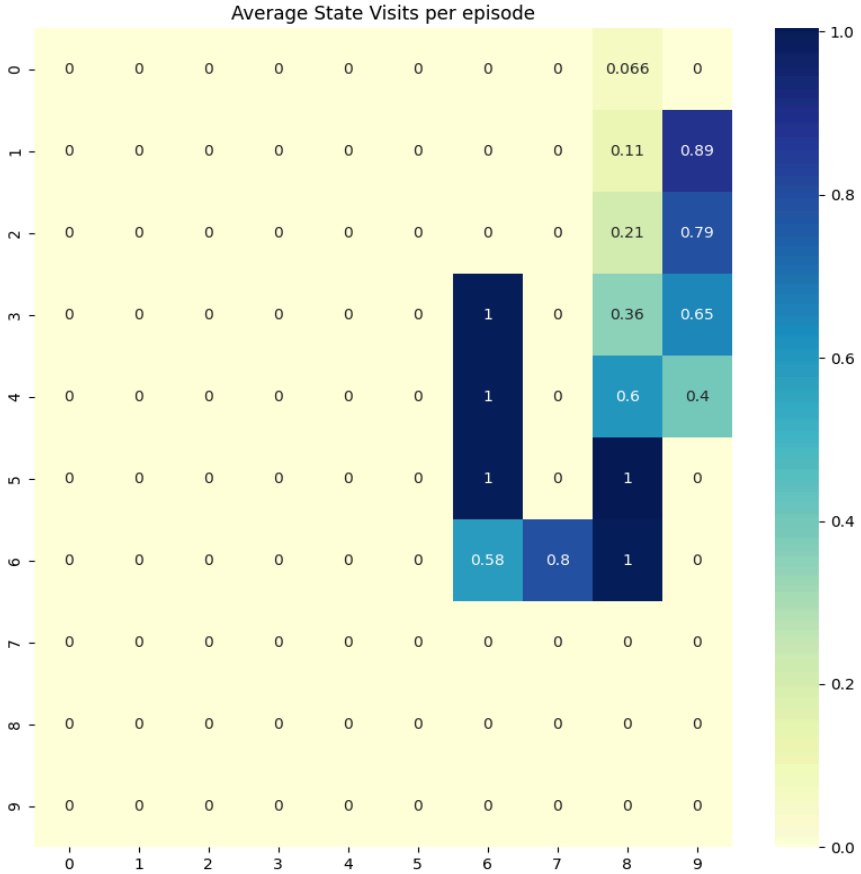


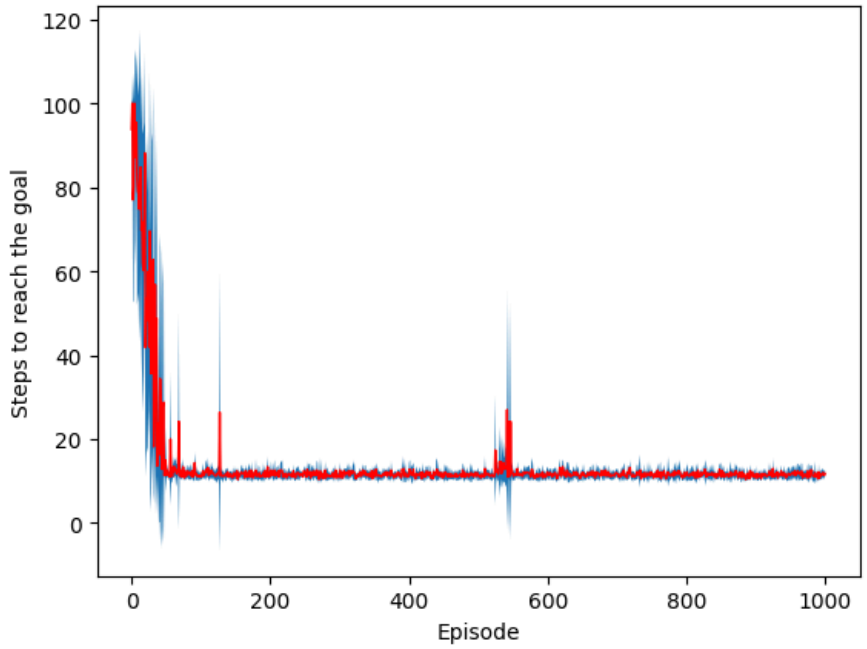

Steps to reach the goal
100
80
60
40
20
0
0
200
400
600
800
1000
Episode

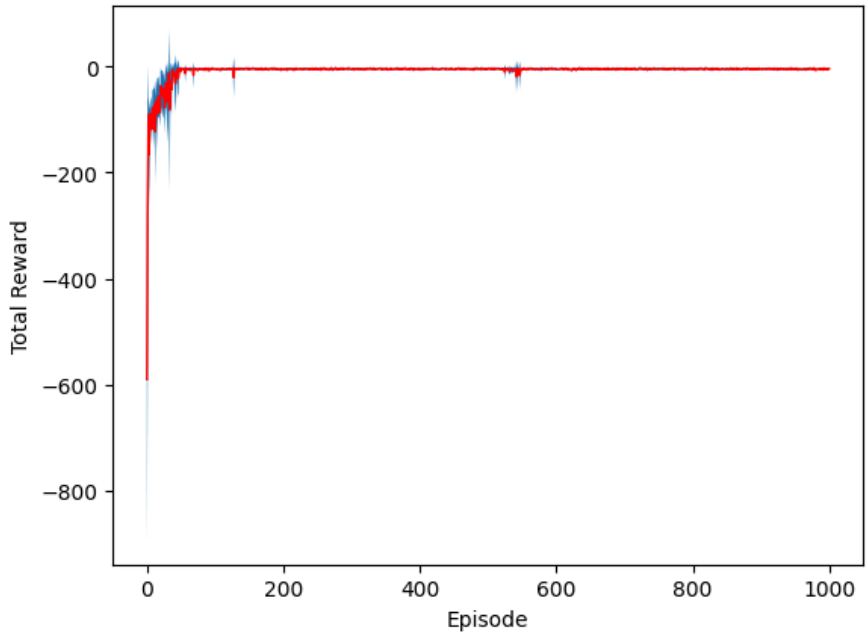

Total Reward
0
−50
−100
−150
−200
−250
−300
0
200
400
600
800
1000
Episode

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

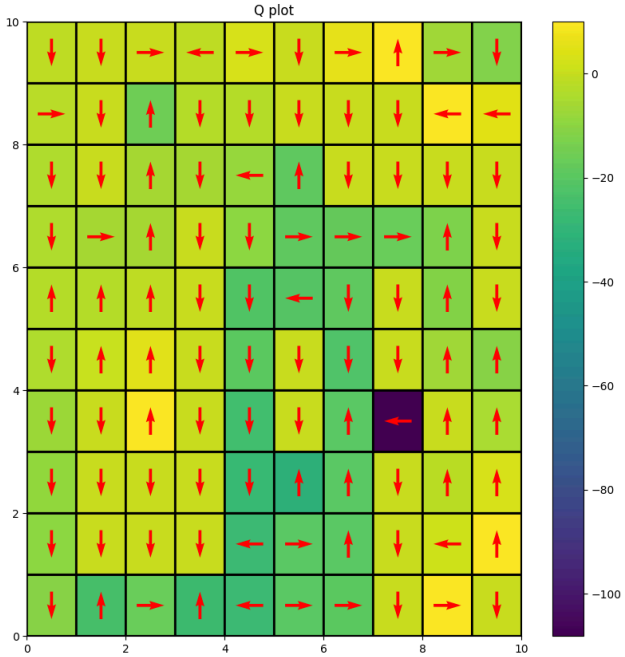


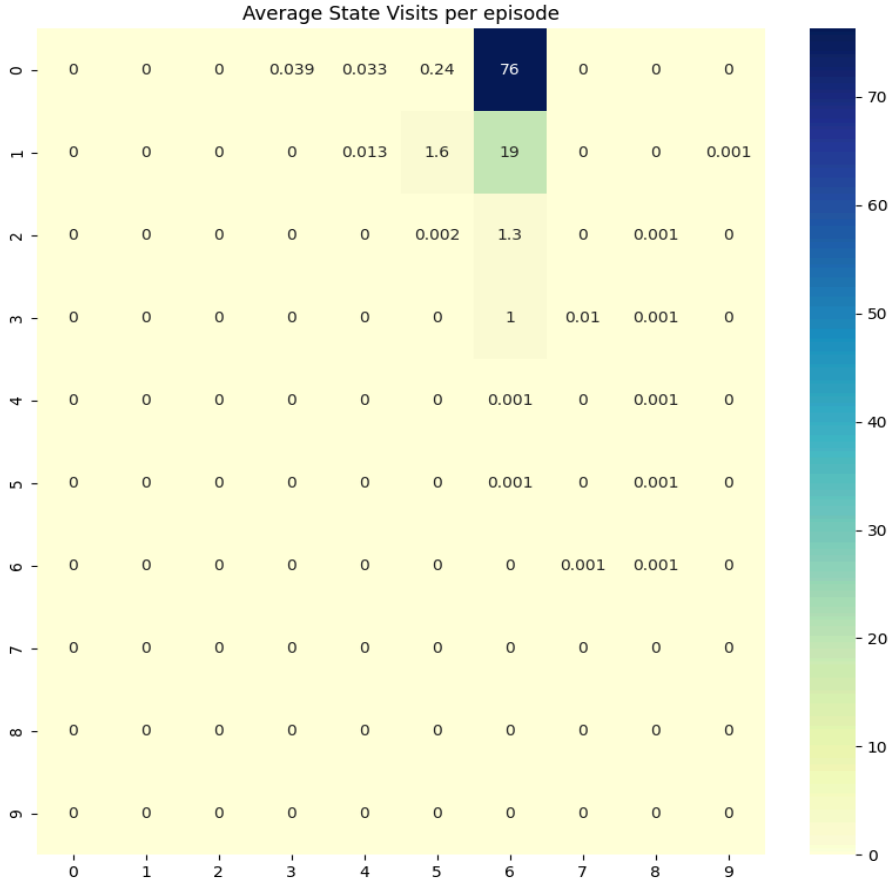


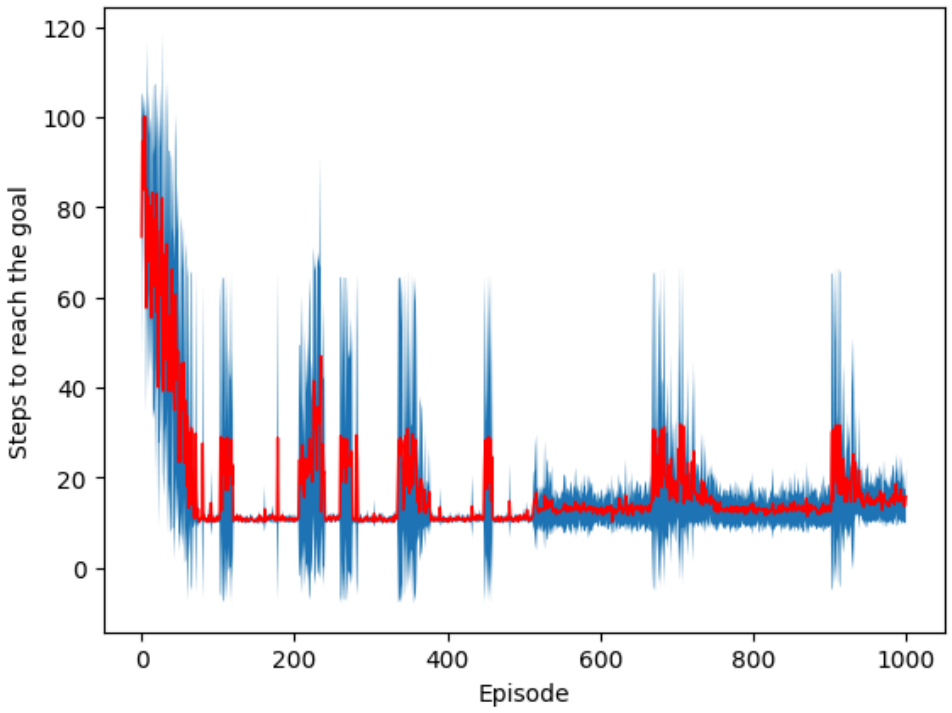

Steps to reach the goal
120
100
80
60
40
20
0
0
200
400
600
800
1000
Episode

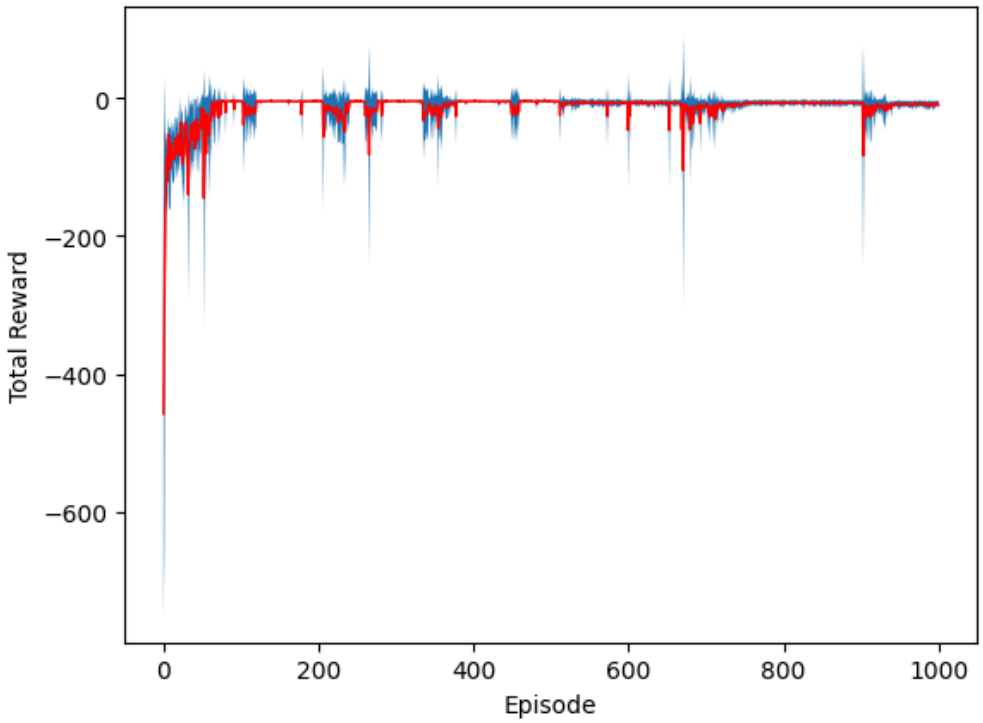

Total Reward
0
−200
−400
−600
−800
0
200
400
600
800
1000
Episode

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






Steps to reach the goal
120
100
80
60
40
20
0
0
200
400
600
800
1000
Episode


Total Reward
0
−200
−400
−600
0
200
400
600
800
1000
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.