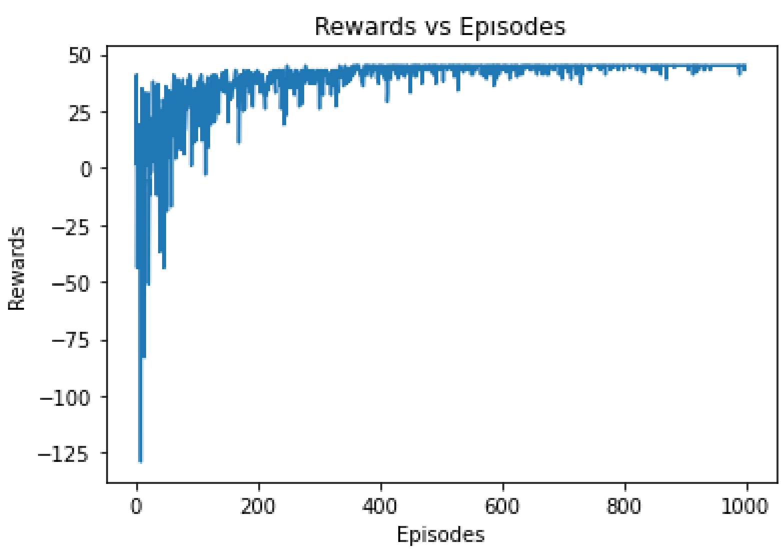
REINFORCEMENT LEARNING CSE4/573 ASSGN 1

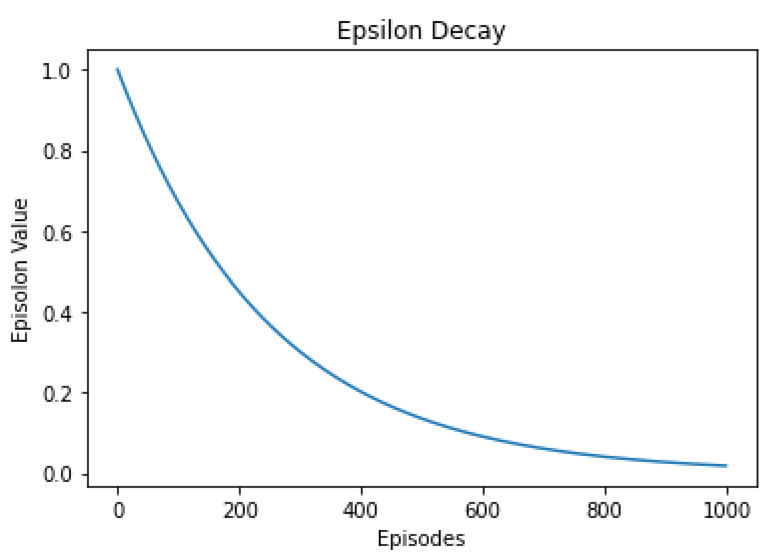
HIMADRI ROY
50374112

I have changed my environment from 10x10(submitted as checkpoint 1) to 4x4 for simplicity.

1) Q-LEARNING FOR DETERMINISTIC SYSTEM



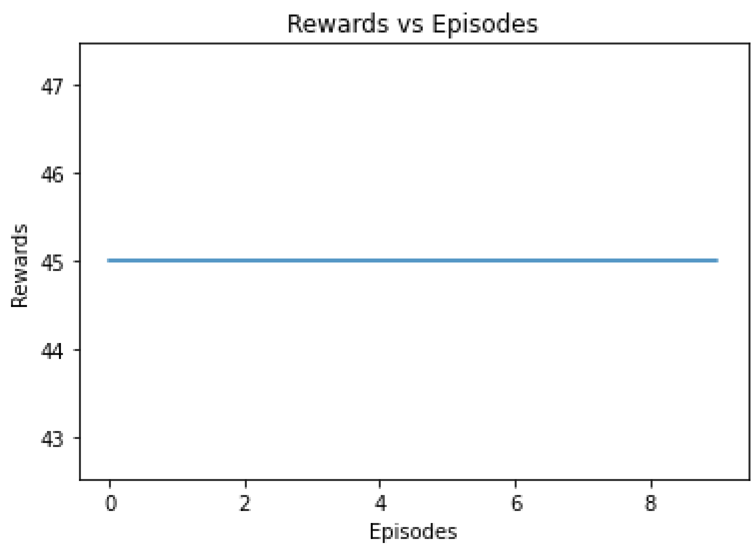
Reward vs episode for Q-Learning in deterministic Environment



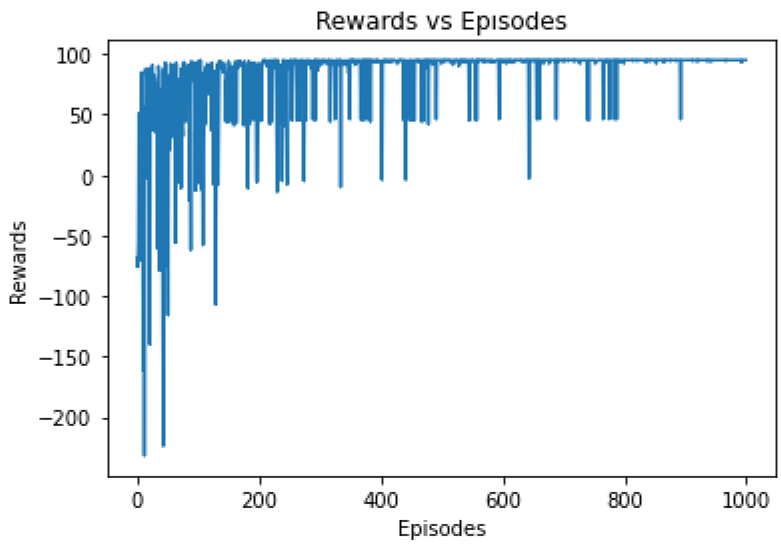
Epsilon Decay vs Episode for Q-Learning deterministic Environment

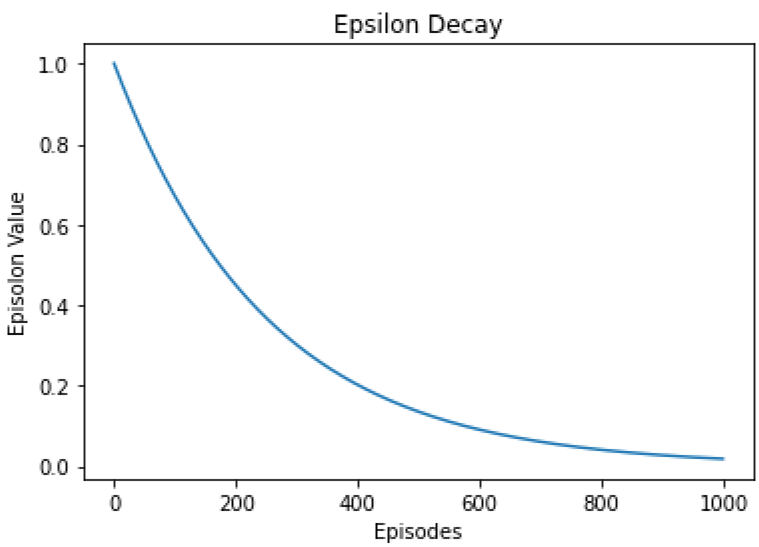
```
Q Table:
 [[ 1.5824     -0.07680141 -0.45346731 -0.06771774]
 [ 2.33840964 -1.36151548  4.29757852 -0.6808883 ]
 [ 7.11666197  3.48048478  8.83936231 -1.88356325]
 [16.39994692  7.3907548   7.4136713   3.27863901]
 [ 3.87199987 -0.10226127  4.304       1.55324983]
 [ 7.48105258 -0.65593887  8.84        1.25492648]
 [12.89548742  3.8721849  16.4         4.03358683]
 [29.          8.64338901 16.36422846  8.46662912]
 [ 8.11999997  1.29179007  8.00300834  3.71549787]
 [15.19930782  2.56295831 10.62042498  2.38287657]
 [26.99935921  7.31424531 25.24414323  6.28630672]
 [50.         16.10422232 28.79471975 12.98357133]
 [ 8.00136009  3.85135712 15.19999999  7.99086183]
 [15.17554002  8.05393228 27.          8.06812771]
 [26.96240629 13.05844037 50.         15.09401053]
 [ 0.          0.          0.          0.        ]]
```

1.1) Evaluation of Q-Learning in Deterministic

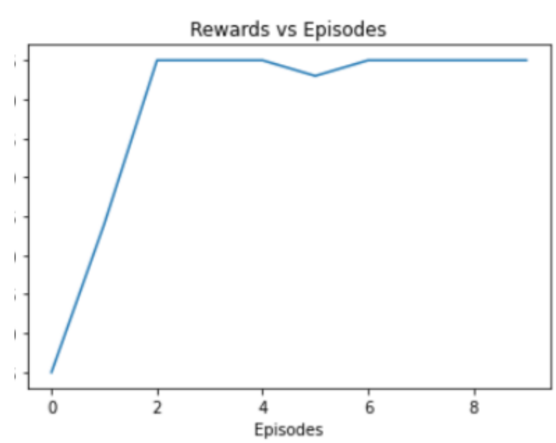


2) Q-LEARNING FOR STOCHASTIC SYSTEM





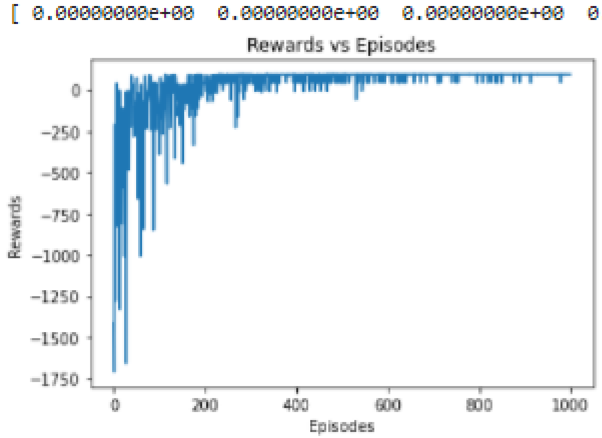
2.1) Q-LEARNING STOCHASTIC EVALUATION

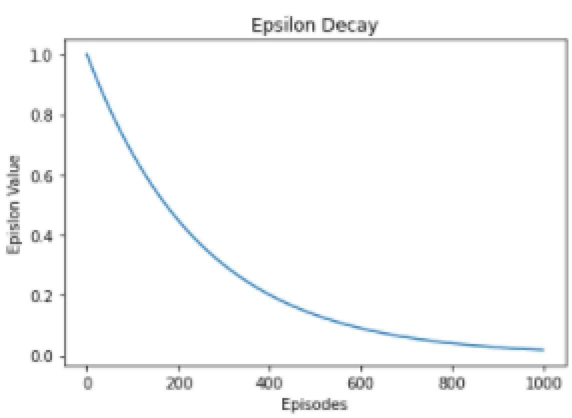


3) SARSA - DETERMINISTIC

3.1) Training

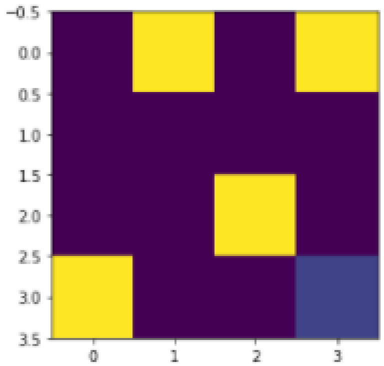
```
Model Ready! Training finished.

Total Episodes:  1000
Total Timesteps:  11853
Total Penalties:  9813
Average timesteps per episode: 11.853
Average penalties per episode: 9.813
SARSA Table:
 [[ 5.23047547e+00 -4.82939618e+00 -4.93538716e+01 -8.21121451e-01]
 [ 8.04499712e+00 -6.23538892e+01 -8.48357450e+00 -1.12266115e+01]
 [-7.43189626e-01 -2.01037619e+01 -5.67819597e+01 -6.56000971e+01]
 [ 1.10244456e+01 -5.95953113e+01 -5.81273636e+01 -2.01173115e+01]
 [ 2.29137412e+00 -2.80196424e+00  1.01435510e+01  9.97874693e-02]
 [ 1.85902111e+01 -5.09258717e+01 -2.21579869e+00  7.14805340e-01]
 [-3.76439642e+01 -1.21637068e+01  1.52366368e+01 -5.99507981e+00]
 [ 4.82921618e+01 -5.88568371e+01  4.29304677e+00 -4.59839768e+00]
 [-5.82253789e+01 -2.97831973e+00  1.44388655e+01 -3.94992663e+00]
 [ 3.38729380e+01  3.05208767e+00 -2.11090287e+01  2.18487620e+00]
 [ 5.53277694e+01 -4.39834716e+00  2.46989059e+01 -1.13207664e+00]
 [ 9.99980637e+01  4.06302181e+00  1.14795582e+01 -3.69322088e+01]
 [-6.14499195e+01 -6.37257759e+00  2.53098842e+01 -5.43705016e+01]
 [ 2.57237858e+01  1.27722115e+01  5.89991456e+01 -4.17937059e+01]
 [ 5.15034486e+01 -2.10768853e+01  1.00000000e+02  2.52241983e+01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]
```



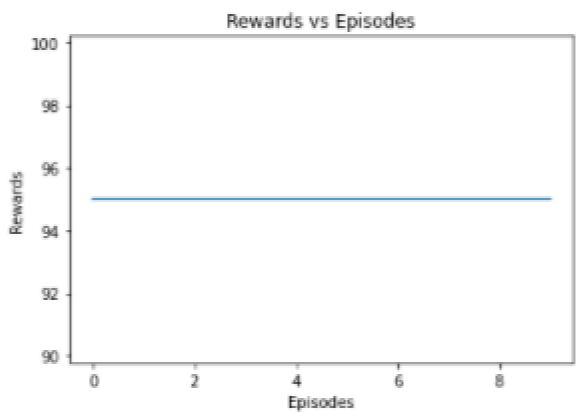


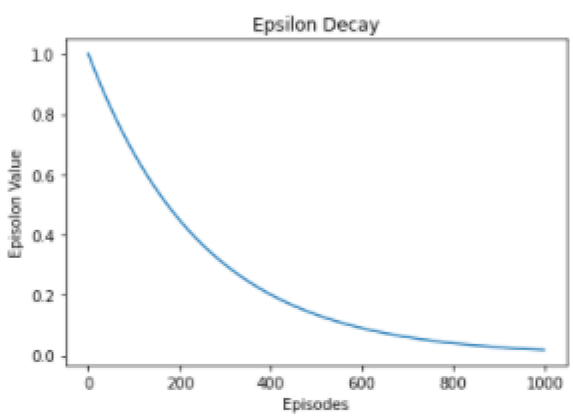
3.2) Test

```
Episode: 10
```




```
5
Testing Complete.

Total Episodes: 10
Total Timesteps: 60
Total Penalties: 50
Results after {episodes} episodes:
Average timesteps per episode:  6.0
Average penalties per episode:  5.0
```





Rewards per episode remains constant for a deterministic environment

4) SARSA - STOCHASTIC

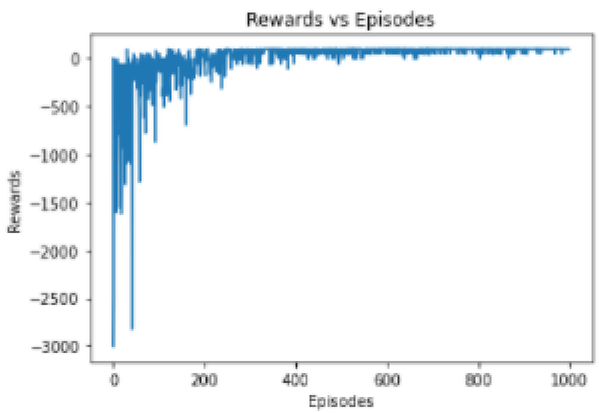
For stochastic nature I've add this block

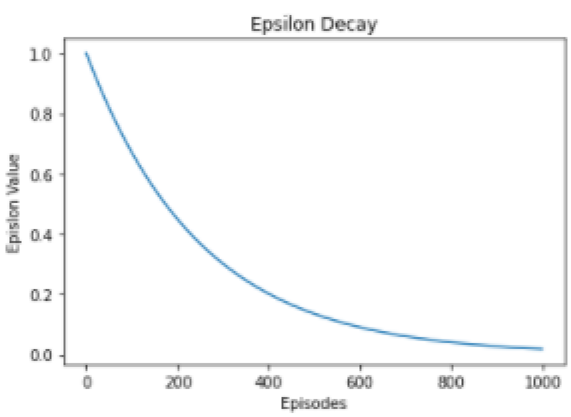
```
def stochastic(self, action):
    if action == 0 :
      return np.random.choice([0,1,2,3], p=[0.91, 0.03, 0.03, 0.03])

    elif action == 1 :
      return np.random.choice([0,1,2,3], p=[0.03, 0.91, 0.03, 0.03])

    elif action == 2 :
      return np.random.choice([0,1,2,3], p=[0.03, 0.03, 0.91, 0.03])

    elif action == 3 :
      return np.random.choice([0,1,2,3], p=[0.03, 0.03, 0.03, 0.91])
```

Thus the agent might or might not end up where it has decided to due to the probabilities being attached to each actions

4.1) Training - Stochastic SARSA

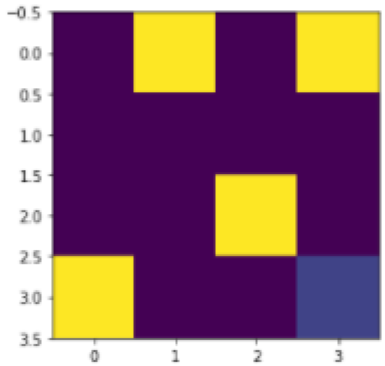
```
Model Ready! Training finished.

Total Episodes:  1000
Total Timesteps:  15085
Total Penalties:  12415
Average timesteps per episode: 15.085
Average penalties per episode: 12.415
SARSA Table:
 [[  5.45549782  -0.40742184 -48.73816466  -3.49182519]
 [  8.68532519 -56.75307519  -7.3036529   -6.14684497]
 [ 17.61300548 -12.89515166 -57.54909039 -60.71994012]
 [ 25.35872192 -60.02921886 -54.68438205  -7.31712145]
 [  2.77630271  -0.3907359   10.7817381    3.96397795]
 [ 10.72848221 -46.12565229  19.63934821   3.17755657]
 [-15.76804824   5.90541775  34.39985399   7.23963954]
 [ 58.9999921  -40.41020789  30.77404746  16.18499027]
 [-51.22375994  -2.94737027  13.24232184  -6.8171628 ]
 [ 30.84212405  -0.49054285 -32.11002736  -3.55374826]
 [ 39.52076226   2.84207849  58.97475499   2.35789844]
 [100.          31.79226469  54.61519189 -16.32720865]
 [-54.27974526  -7.19227058  22.27910458 -54.61798554]
 [ 11.67169262   3.83198009  57.75527859 -49.71456538]
 [ 39.05290892 -31.88910315  99.99999995  10.09054588]
 [  0.           0.           0.           0.        ]]
```



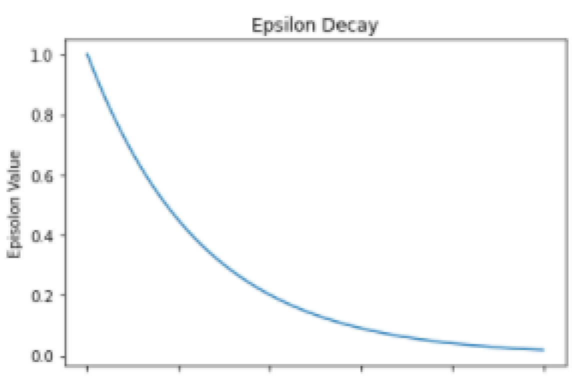


4.2) Test - Stochastic SARSA

```
--
Episode: 10
```



```
5
Testing Complete.

Total Episodes: 10
Total Timesteps: 72
Total Penalties: 62
Results after {episodes} episodes:
Average timesteps per episode:  7.2
Average penalties per episode:  6.2
```



- We can see from the rewards vs episodes graph that for stochastic nature the reward is not constant for all episodes as there is a random factor involved with the agent

HYPERPARAMETERS (Q-Learning)

- **Epsilon Decay Rate (ε)**

```
epsilon= np.exp(-4*i/episodes)
```

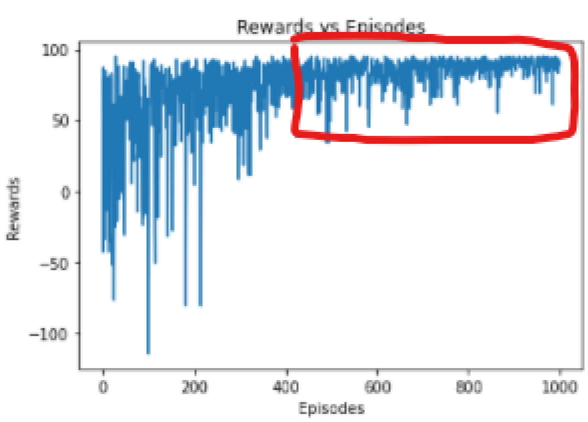
By adding the above line to the code epsilon decays over the time as the number of episodes proceed. Thus (1-ε) increases thus increasing the greedy factor. This can be also inferred from the graph. At the end of the run the rewards accumulated by the agent seems to become constant as the agent tends to be greedy due to the low epsilon at the end.

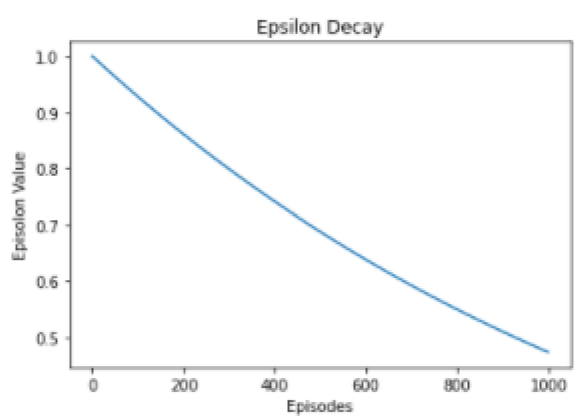

```
epsilon= np.exp(-4*i/episodes)
```

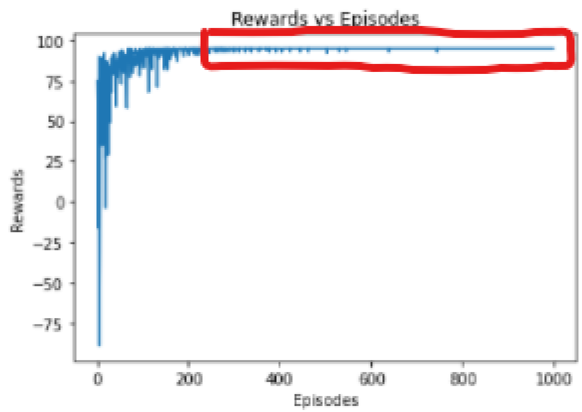
In the above equation we can tune the decay to be shallow or steep by changing 4 to any other value. Lower value will make the decay shallow (it will decay slowly over the time)

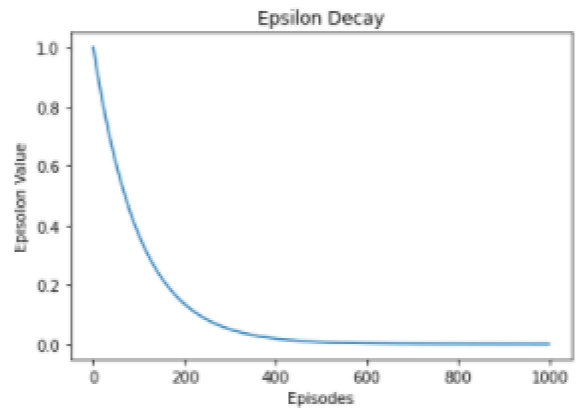
As we can see for this decay we get the following graphs for Rewards vs Episodes

```
epsilon= np.exp(-0.75*i/episodes)
```










```
epsilon= np.exp(-10*i/episodes)
```

For a = -10 the agent starts taking greedy actions in earlier episodes compared to others. One drawback of having a **very high 'a' value** is that **epsilon** will **decay very fast** and this will cause the agent to **not explore the environment** and miss out on better or more optimized paths. This is because greedy actions are taken more.

For my environment I found **4** to be the best value as the agent goes through most of the parts of the environment

The following value of ε works the best `epsilon= np.exp(-4*i/episodes)`

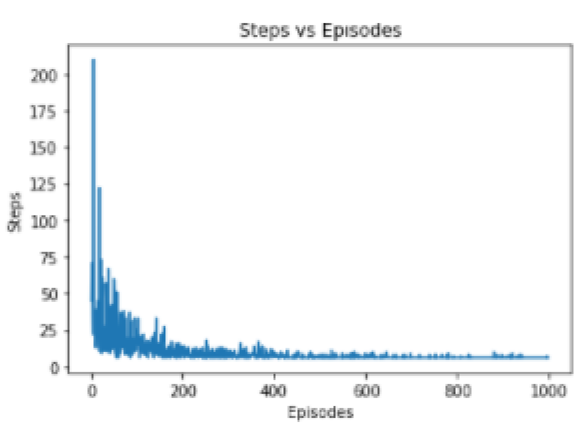
- <u>**Discount Factor(gamma ɣ)**</u>

ɣ = 0

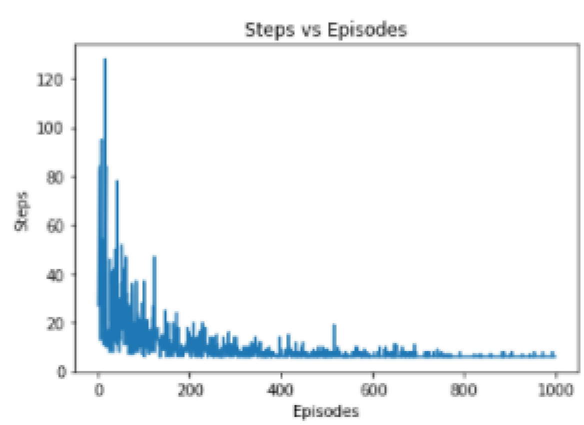
The agent **starts ignoring or worrying about the distant rewards** and concentrates more on the immediate rewards. Agent becomes short-sighted.

For ɣ=0 the training wasn't completed as there is no convergence in the agent's learning so it keeps moving around looking for immediate rewards. It might end up at Terminal state as the environment is very small.

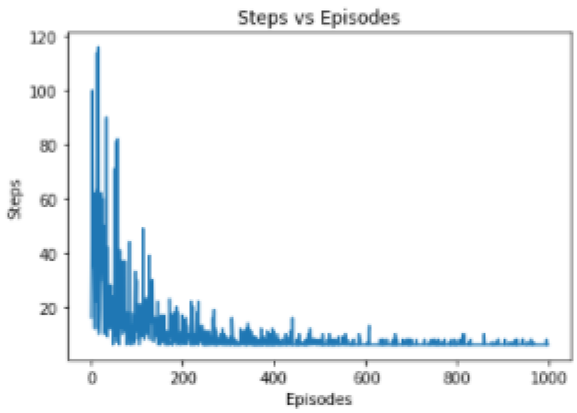
For ɣ=0.9



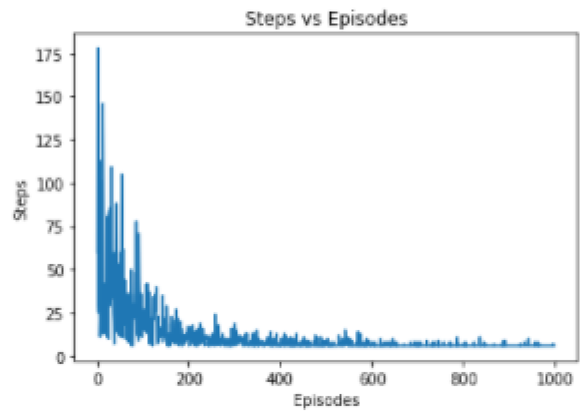
For ɣ = 0.1



For Ɣ = 0.65(Works best for my environment)



For Ɣ = 0.005

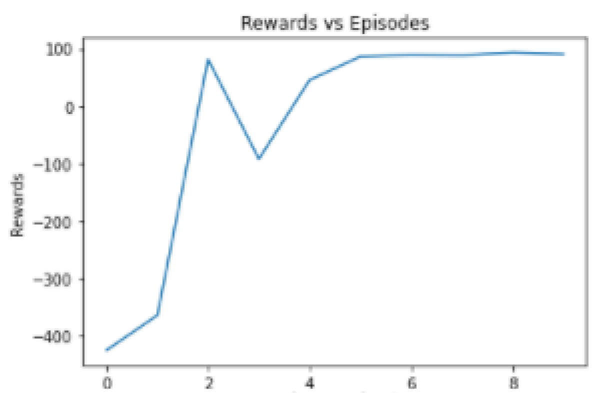


If we consider from 0-200 on episode axis, we notice that for a very high Ɣ the number of steps taken by the agent is much less and grouped up compared to the other graphs.
For a high Ɣ, the agent will give more weightage to the future or distant rewards and less about immediate rewards. Thus reducing the number of steps in the coming episodes and finding an optimal path that involves less steps.

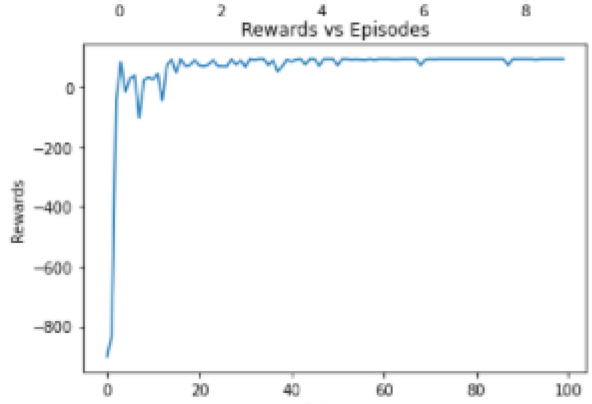
Ɣ = 0.65 is the best value for this environment.

- **<u>Number of Episodes</u>**

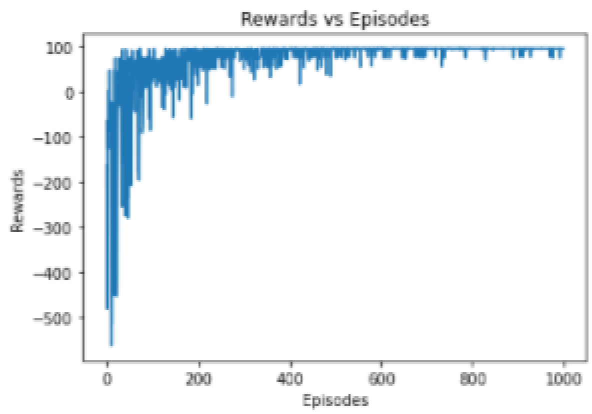
```
Episodes 10
Steps per episode 29.6
```




```
Episodes 100
Steps per episode 11.49
```



```
Episodes 1000
Steps per episode 9.964
```



From the above results if we consider the average steps, we notice that it reduces as the number of episodes increases. This is because it is able to figure out the environment better.

When the agent was trained with only 10 episodes, it was observed that it took the maximum steps as well as the rewards are all over the place

As the **number of episodes increases** the **avg. steps taken by the agent also decreases**, denoting that the agent has begun gathering enough data to navigate through the environment.

However it was observed that if the **number of episodes were increased more than 3000** episodes it gives **almost the same training result as 1000,2000 episodes**. We can say that the number of episodes required to train the agent must depend upon the size of the environment. **My environment being small 2000-3000 episodes was enough.**

GITHUB LINK