RL A2

Parth Bhardwaj

Link for GIFs: https://drive.google.com/drive/folders/1-rAjnIXzMme_UHL0o5aVjhbqGM2OlAm1?usp=sharing

Cliff environment:

Decay schedule tried: epsilon greedy, softmax

Epsilon Decay methods attempted: decay factor ^ episode, 1/ sqrt (episode), linear decay of which the inverse square root worked best

Best Q_Value among seeds found by comparing Value function of start state

Optimistic Initialization for better exploration

For SARSA and ESARSA

Gamma = 0.99, Max Episodes = 500, Max Steps = 1000

Optimal alpha for SARSA = 0.55 with mean reward 169.8

Optimal alpha for ESARSA = 0.45 with mean reward 170.4

The equilibrium is extremely unstable: changing alpha by even 0.001 changes the reward drastically hence finetuning difficult

Another unexpected result was that even under optimal conditions, increasing number of epochs leads to divergence

Weird behaviour for Q learning: optimal policy was found in case of alpha = 0.55, Max Episodes = 5000 and Gamma = 1, but with a catch. Plainly calling Q learning with these parameters results in reward –1000. But when I write the loop for searching optimal alpha by varying it from 0.1 to 0.55 in steps of 0.05, the best reward turns out to be 173.4 for alpha = 0.55. This has probably to do something with how environment is getting reset in cliff.py file which I can't alter but it is interesting anyways.

Frozen Lake environment:

Used similar techniques as part 1

Optimal gamma for both = 1

Optimal alpha for Q learning: 0.2

Q learning takes much lesser episodes to converge.

MC (0,5), Q learning (0,3), Q learning (0,5) all converge in 20K episodes

Optimal epsilon for Monte Carlo (0,5) = 0.66 without any decay

For Q learning epsilon decay 1/sqrt(episode) worked well

Plain Monte Carlo starting from (0,3) didn't work well for any hyperparameters. One option was to use incremental monte carlo with varying alpha as hyperparameter, another was to use policy different from epsilon greedy i.e. biasing greedy policy choose the down right side, both showed convergence. In code I have written the latter one.

Results : Mean reward = 0.7 for start state (0,3) and 0.75 for start state (0,5).

Importance Sampling

Monte Carlo-

Initially I tried optimistic Q initialisation to allow more exploration but that surprisingly prevented optimal convergence and gave convergence to sub-optimal goal hence it was removed. While training epsilon greedy target policy was used which was finally converted to deterministic greedy. Parameters for MC: episodes = 20k, gamma = 0.85, epsilon = 0.3 with linear decay till 0.01. W was clipped to avoid overflow. The convergence to optimal state was slow in the case of noise = 0 but it eventually did happen. One reason is that there isn't negative penalty on just staying at a blank place.

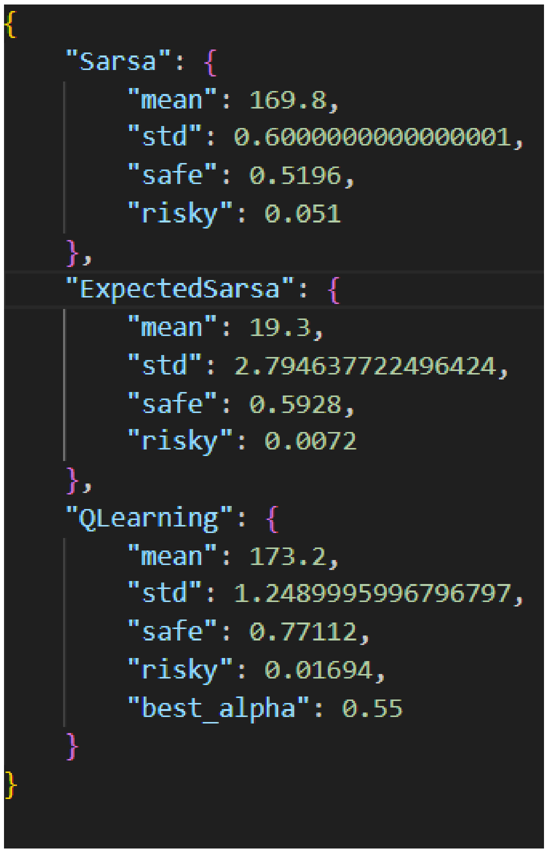
TD:

Since it wasn't specified in the problem statement whether we use SARSA inspired off sampling or Q-Learning inspired off sampling, I tried both and got convergence to optimality in Q Learning inspired one, where we take G = G + gamma* max(Q[next_state]) along with epsilon greedy policy in training finally converted to deterministic

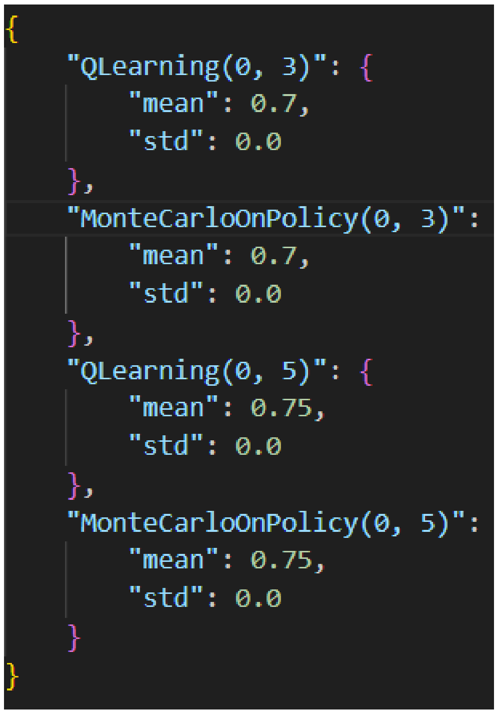
Episodes = 20k, Optimal gamma = 0.85, alpha = 0.02 and epsilon = 0.4 decayed linearly till 0. All three noises comfortably converged.

In both cases best policy was chosen comparing the mean reward during evaluation

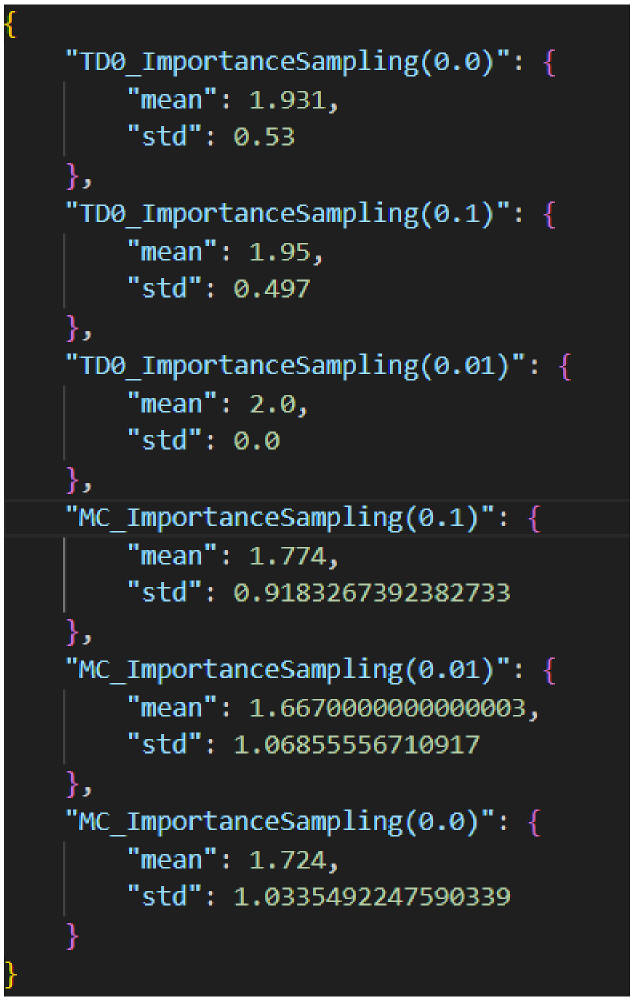
Results:



```
{
    "Sarsa": {
        "mean": 169.8,
        "std": 0.6000000000000001,
        "safe": 0.5196,
        "risky": 0.051
    },
    "ExpectedSarsa": {
        "mean": 19.3,
        "std": 2.794637722496424,
        "safe": 0.5928,
        "risky": 0.0072
    },
    "QLearning": {
        "mean": 173.2,
        "std": 1.2489995996796797,
        "safe": 0.77112,
        "risky": 0.01694,
        "best_alpha": 0.55
    }
}
```



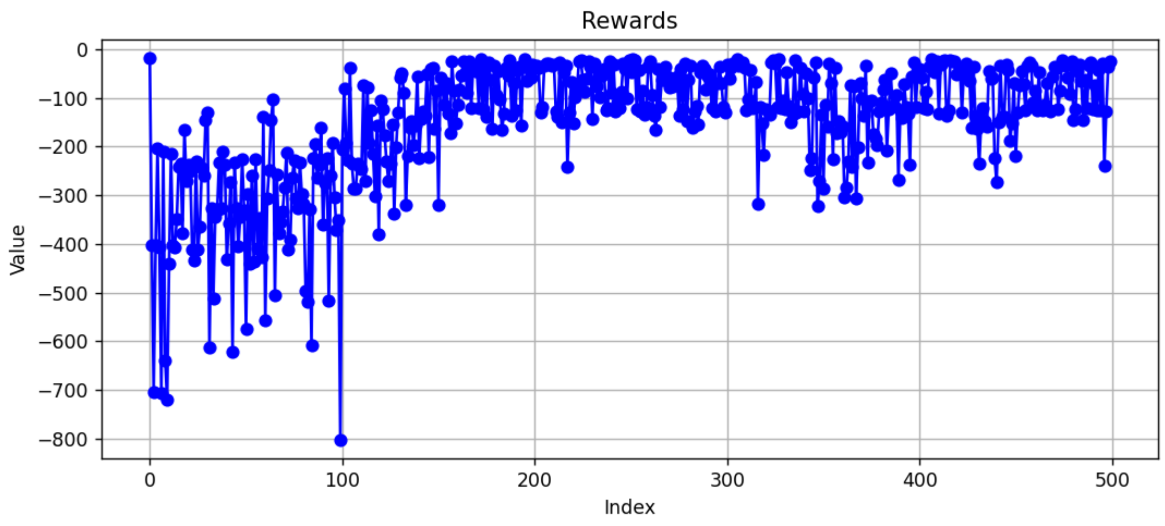
```
{
    "QLearning(0, 3)": {
        "mean": 0.7,
        "std": 0.0
    },
    "MonteCarloOnPolicy(0, 3)":
        "mean": 0.7,
        "std": 0.0
    },
    "QLearning(0, 5)": {
        "mean": 0.75,
        "std": 0.0
    },
    "MonteCarloOnPolicy(0, 5)":
        "mean": 0.75,
        "std": 0.0
    }
}
```



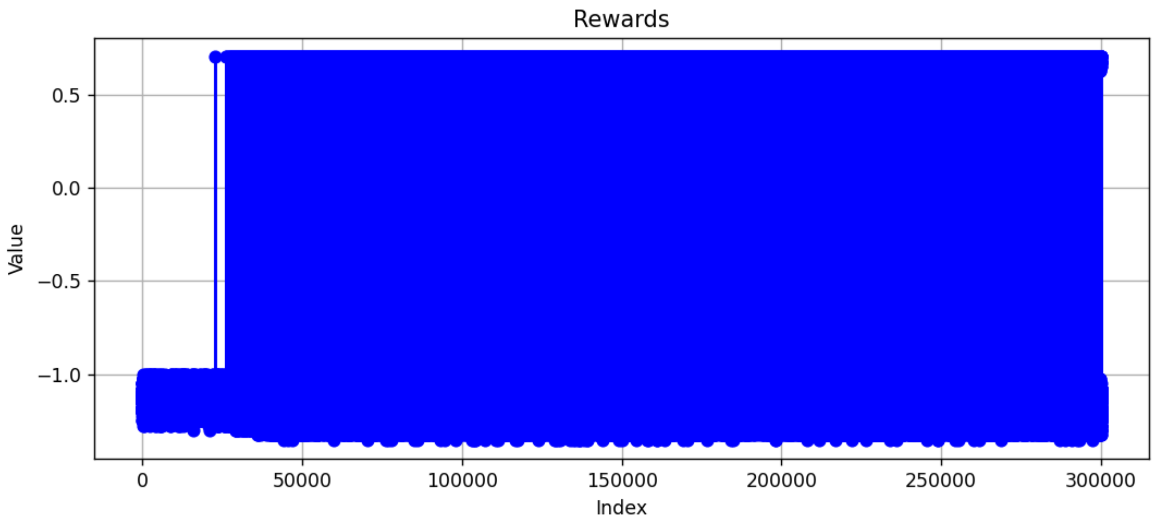
```
{
    "TD0_ImportanceSampling(0.0)": {
        "mean": 1.931,
        "std": 0.53
    },
    "TD0_ImportanceSampling(0.1)": {
        "mean": 1.95,
        "std": 0.497
    },
    "TD0_ImportanceSampling(0.01)": {
        "mean": 2.0,
        "std": 0.0
    },
    "MC_ImportanceSampling(0.1)": {
        "mean": 1.774,
        "std": 0.9183267392382733
    },
    "MC_ImportanceSampling(0.01)": {
        "mean": 1.6670000000000003,
        "std": 1.06855556710917
    },
    "MC_ImportanceSampling(0.0)": {
        "mean": 1.724,
        "std": 1.0335492247590339
    }
}
```

Plots:

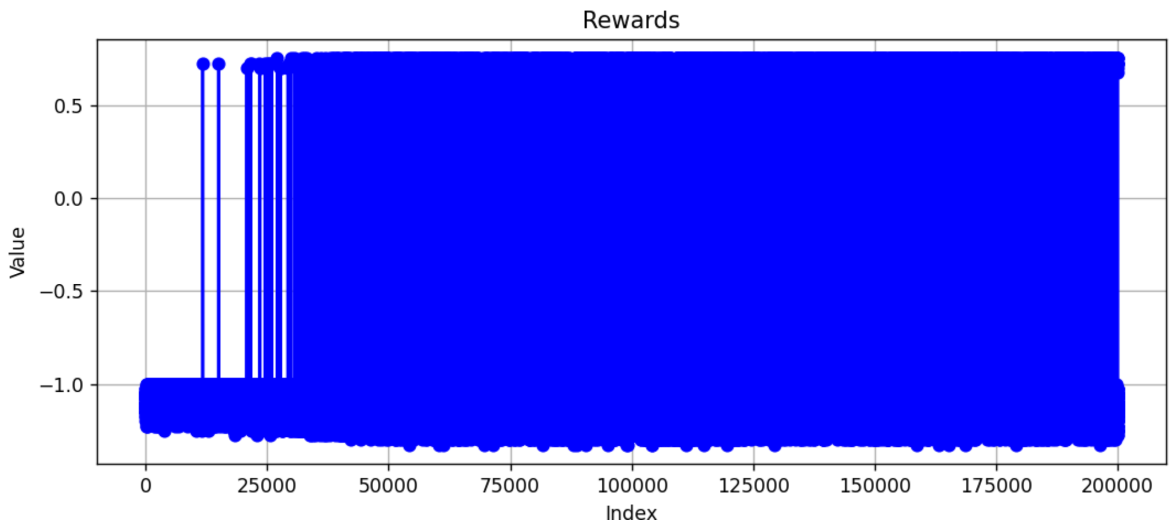
Expected Sarsa Plot:



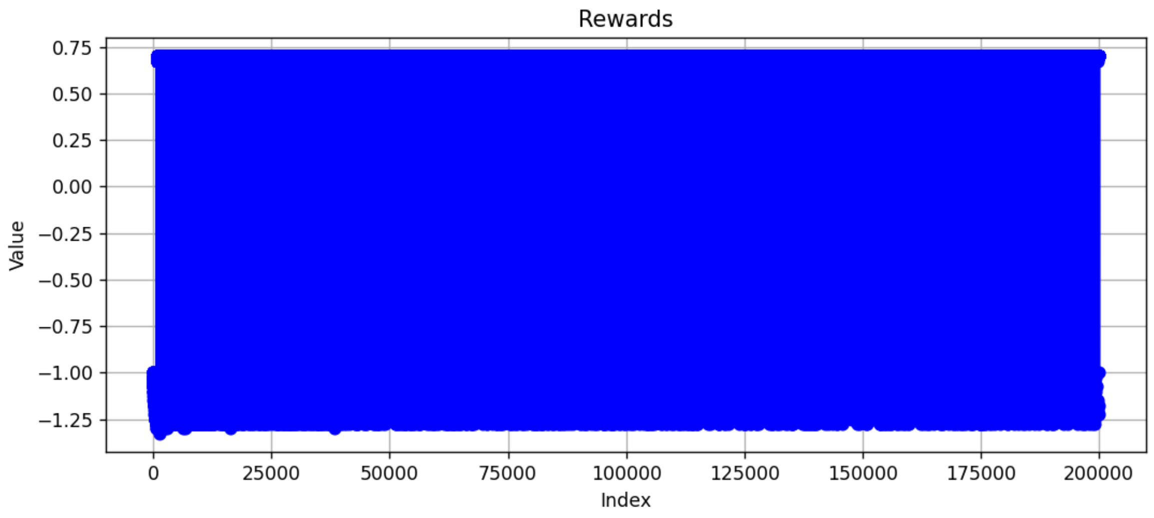
Frozen_lake_mc_(0,3)



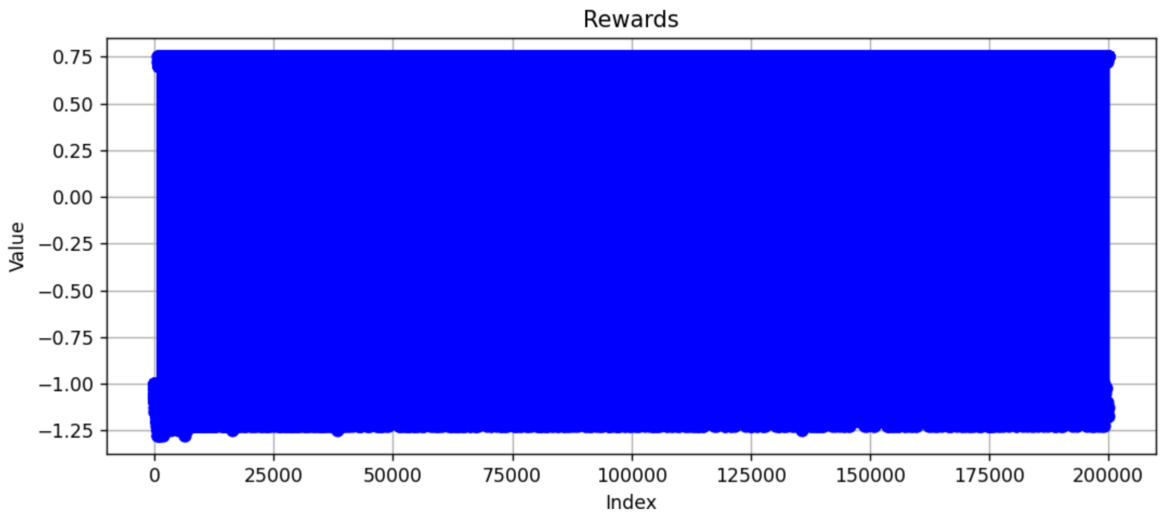
Frozenlake_mc_(0,5)



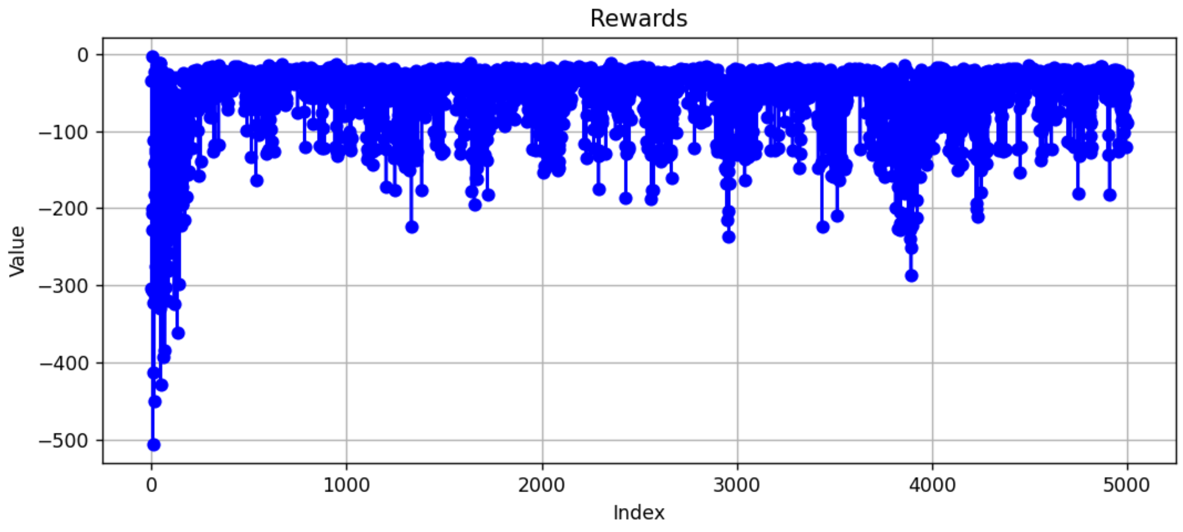
Frozenlake_qlearning_(0,5)



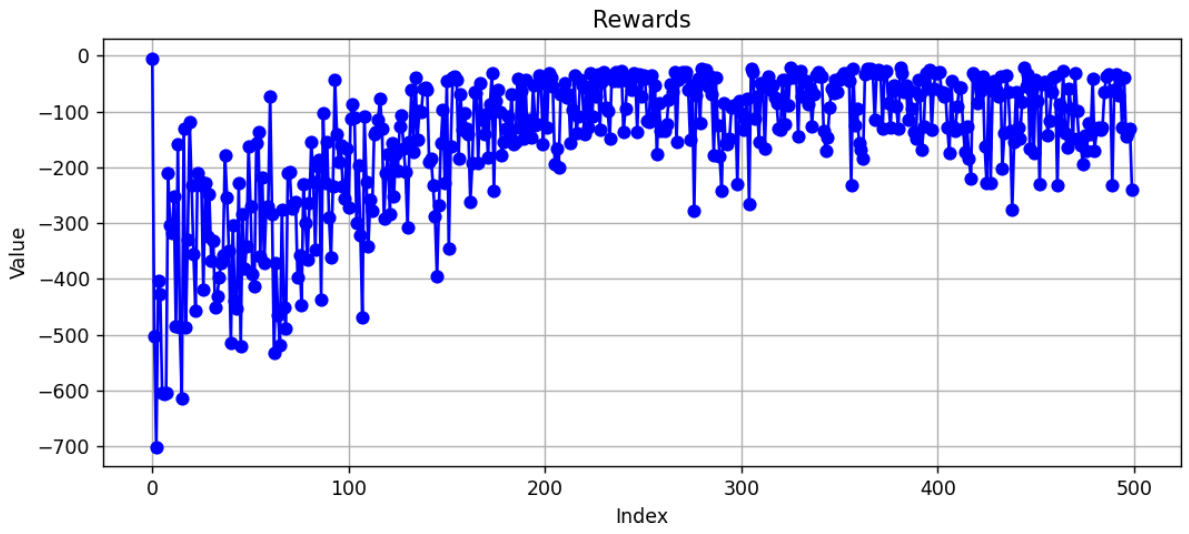
Frozenlake_mc_(0,3)



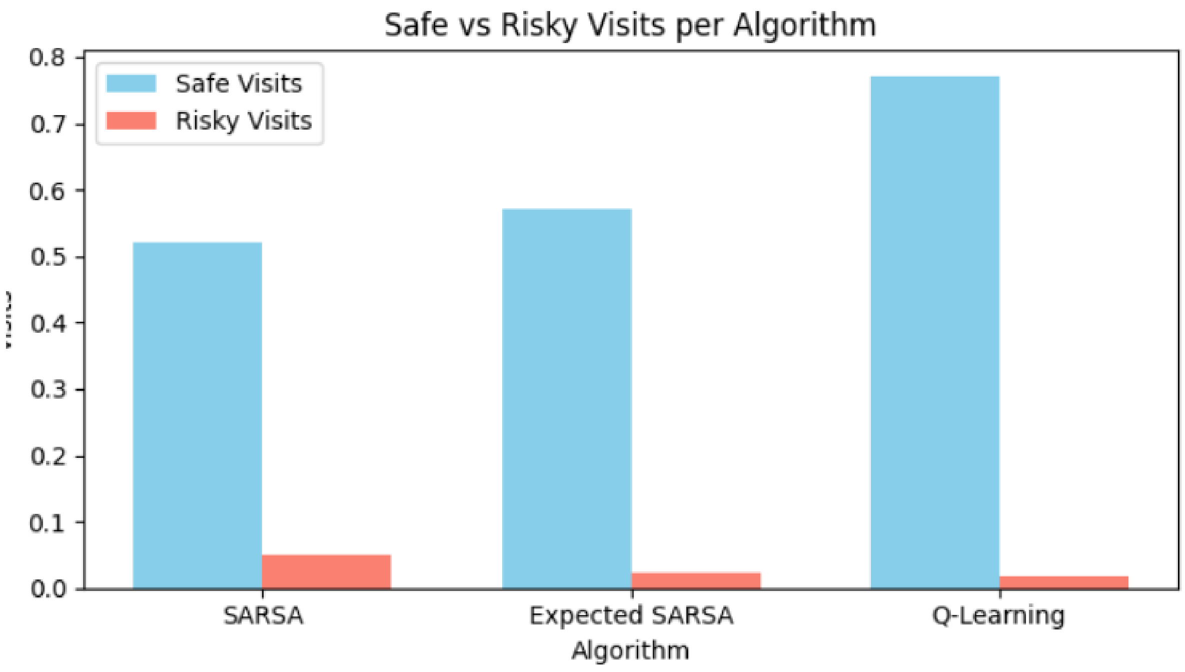
Qlearning_plot



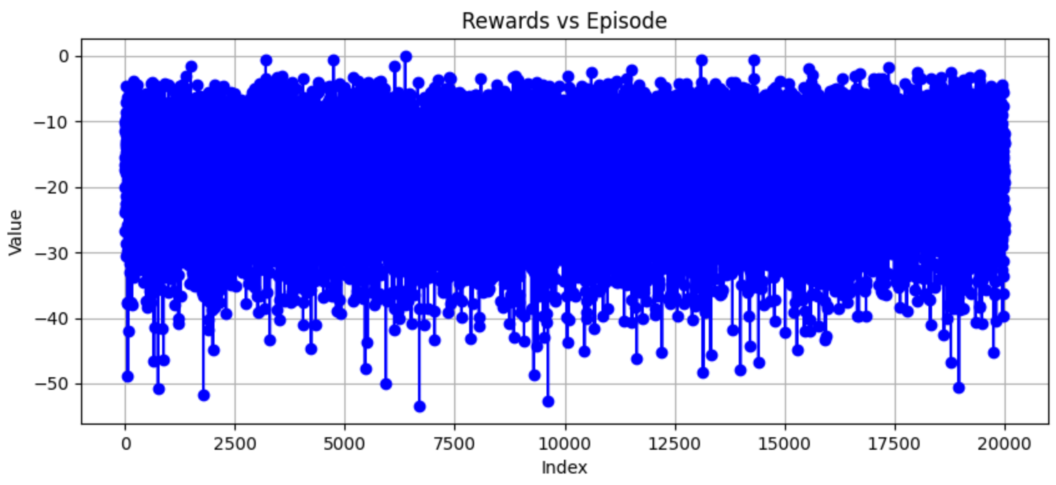
Sarsa_plot

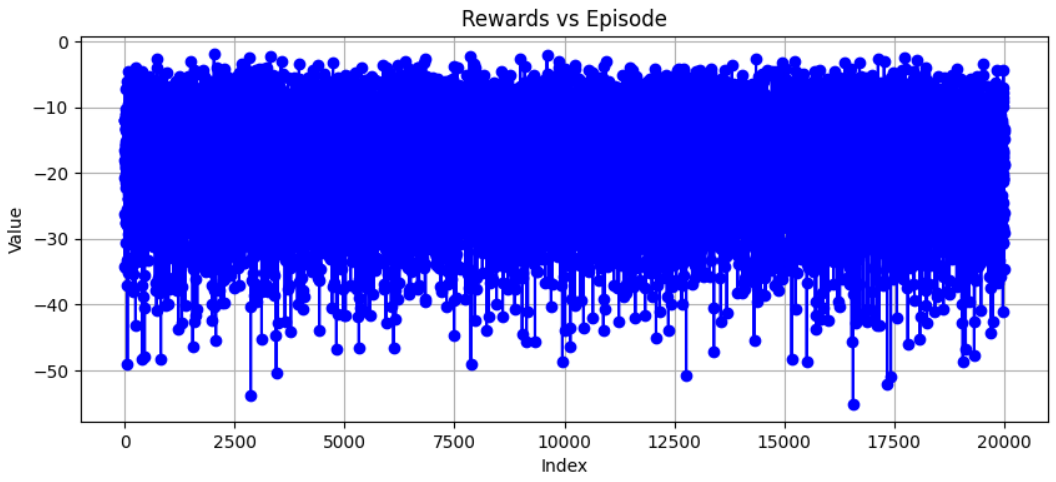


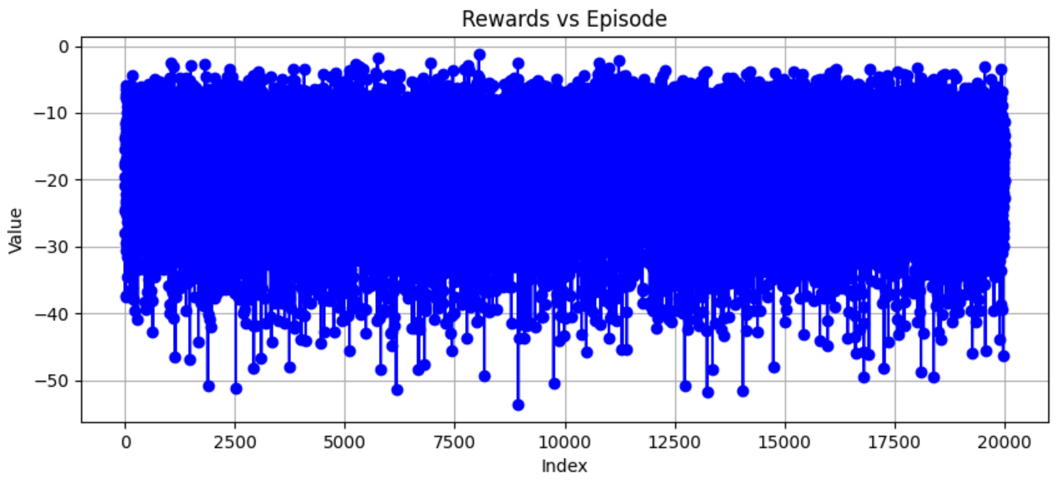
Visit plot:



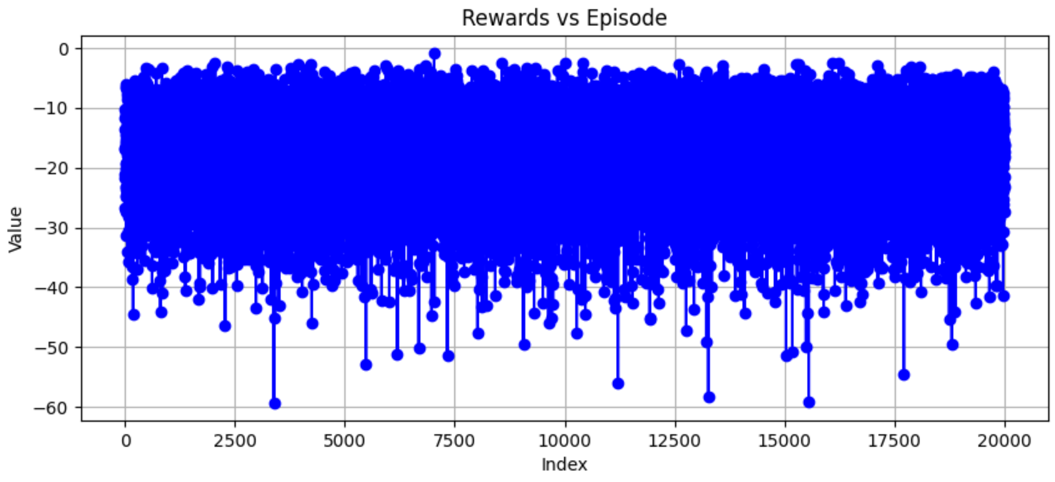
Monte carlo for 0.0, 0.01,0.1 respectively:

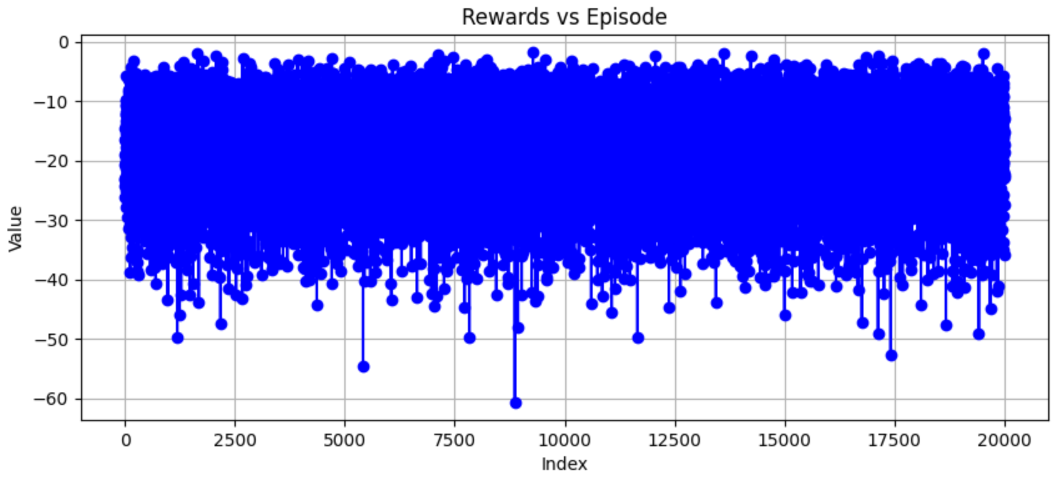


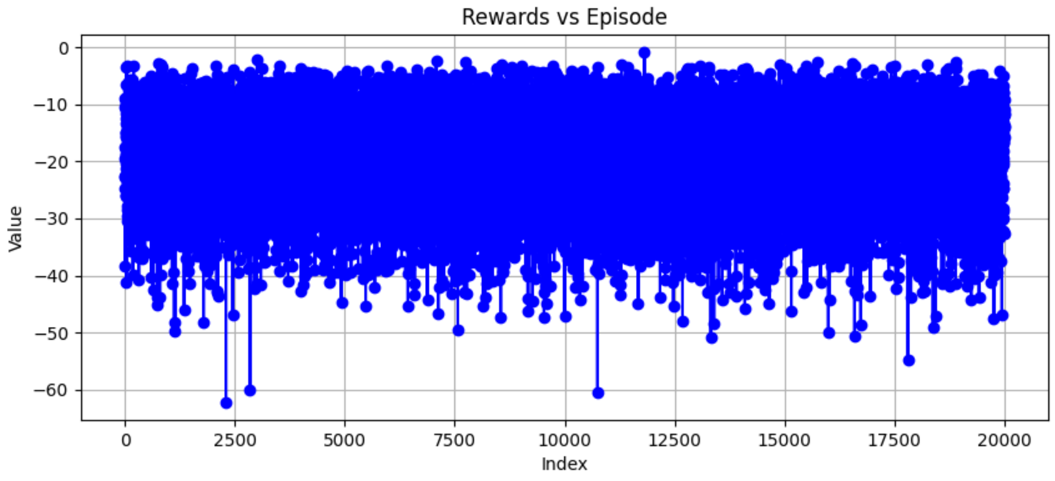




TD for 0.0, 0.01,0.1 respectively:


Rewards vs Episode
Value
Index
0
−10
−20
−30
−40
−50
−60
0
2500
5000
7500
10000
12500
15000
17500
20000


Rewards vs Episode
Value
Index
0
−10
−20
−30
−40
−50
−60
0
2500
5000
7500
10000
12500
15000
17500
20000


Rewards vs Episode
Value
Index
0
−10
−20
−30
−40
−50
−60
0
2500
5000
7500
10000
12500
15000
17500
20000