Assignment 1 - Defining Solving RL Environments

Anonymous Author(s)
Affiliation
Address
email

Abstract

This report aims to define and analyse a grid environment which follows OpenAI Gym standards. The environment can be configured to be deterministic or a stochastic one.

1 Checkpoint 1: Defining RL environments

1.1 Describe the deterministic and stochastic environments, which were defined (set of actions/states/rewards, main objective, etc).

The deterministic environment is a 4x4 grid(16 states) with (0,0) as the start point and goal at (3,3). The action space is of size 4, i.e., down, up, left, right. The max allowed time-steps is 10.

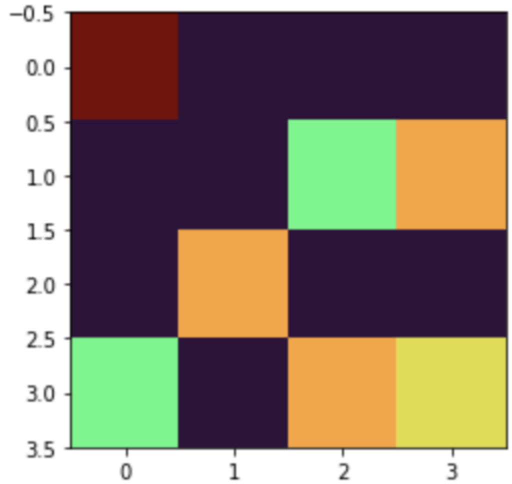
Our agent is a farmer with a donkey, the goal of the farmer is to reach the shed while loading all the bales of hay along the way on the donkey. The farmer wants to reach the shed quickly so he wants to avoid any patches of grass the donkey might start grazing.

There are 3 bales of hay/rewards in the grid(worth +1 each) and the 4th reward is the shed/goal itself(worth +3).There are 2 grass patches/obstacles in the grid(worth -1 reward each), further, after max time-steps have elapsed the environment will generate a reward of -3 for every subsequent action.

In the stochastic env, the only difference is that sometimes the donkey doesn't listen to the farmer and the requested action/step doesn't execute.

1.2 Provide visualizations of your environments.

Red - Agent
Green - Obstacle
Orange - Reward
Yellow - Goal



1.3 How did you define the stochastic environment?

The stochastic environment was defined by adding a 'stochastic' parameter to the environment. If this is set (self.stochastic = 1) there's 95% probability of the requested action being executed & a 5% probability that no action will be executed. The probabilities are based on what was recommended in the lectures.

1.4 What is the difference between the deterministic and stochastic environments?

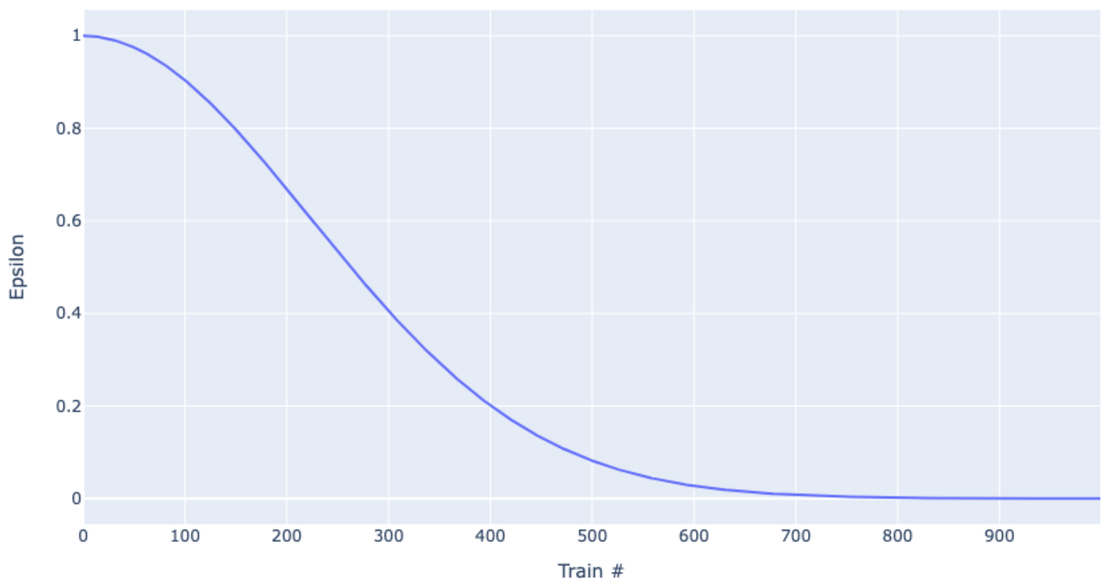
In the stochastic env, sometimes the donkey doesn't listen to the farmer so a requested action doesn't execute and the farmer stays in the same state. If the farmer ends up remaining in the same state, he collects the reward of that state again.

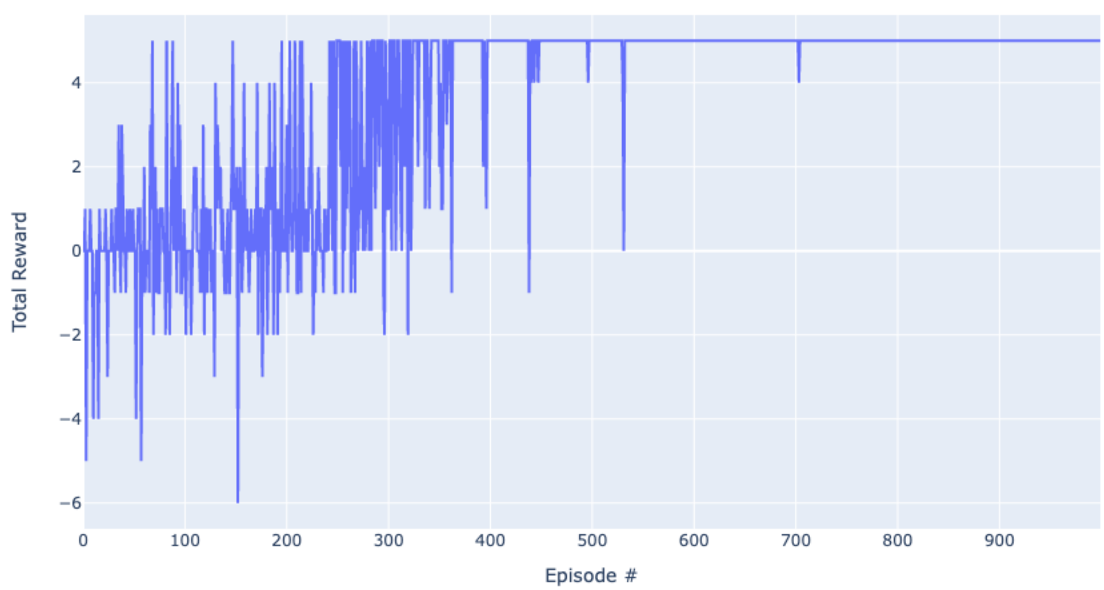
1.5 Safety in AI: Write a brief review explaining how you ensure the safety of your environments.

We ensure safety by using the np.clip() method. Using this method we clip the agent state list after every action such that the coordinates of the resultant position of the agent after taking an action always stays within the bounds of our environment.

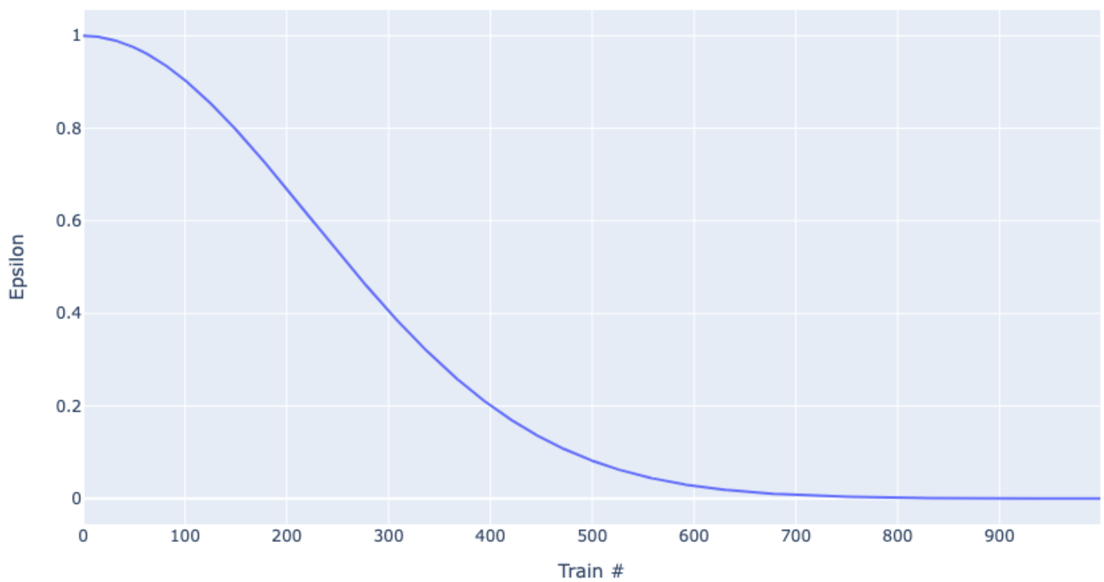
2 Final submission: Tabular learning

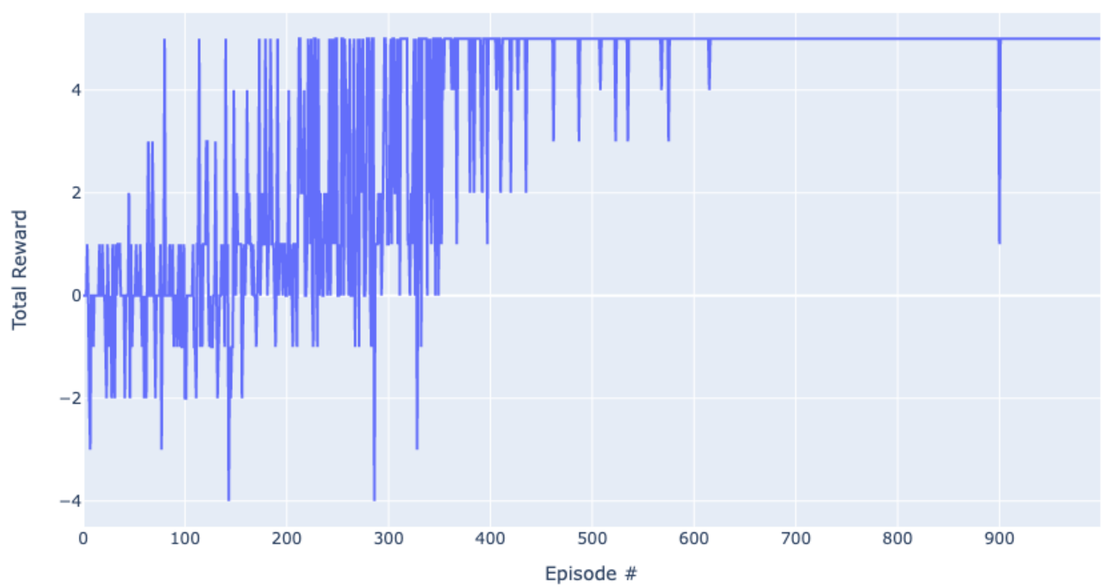
2.1 Applying Q-learning to solve the deterministic environment defined in Part 1. Plots should include epsilon decay and total reward per episode.



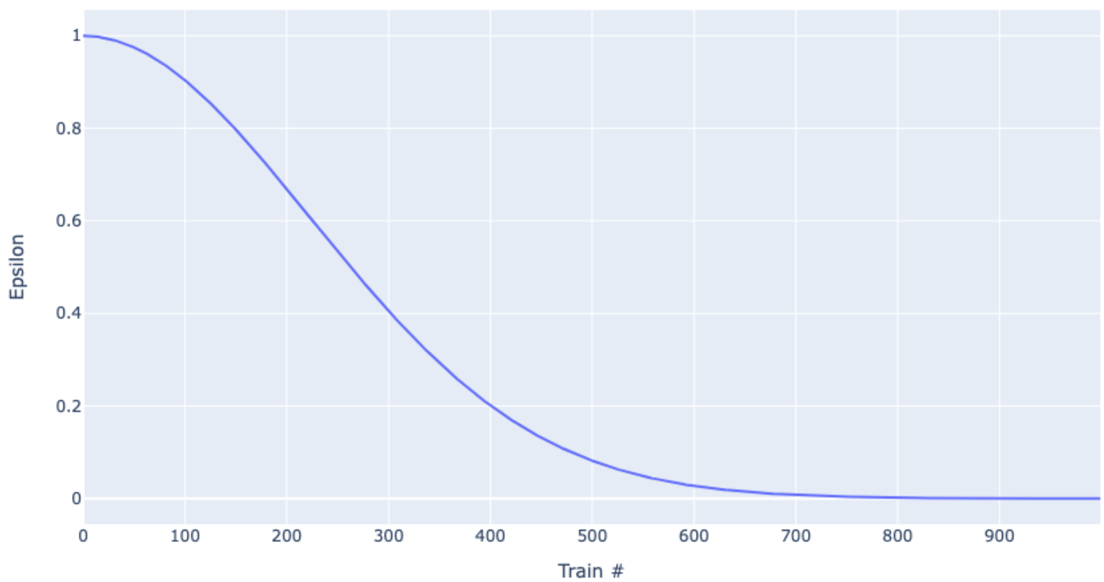


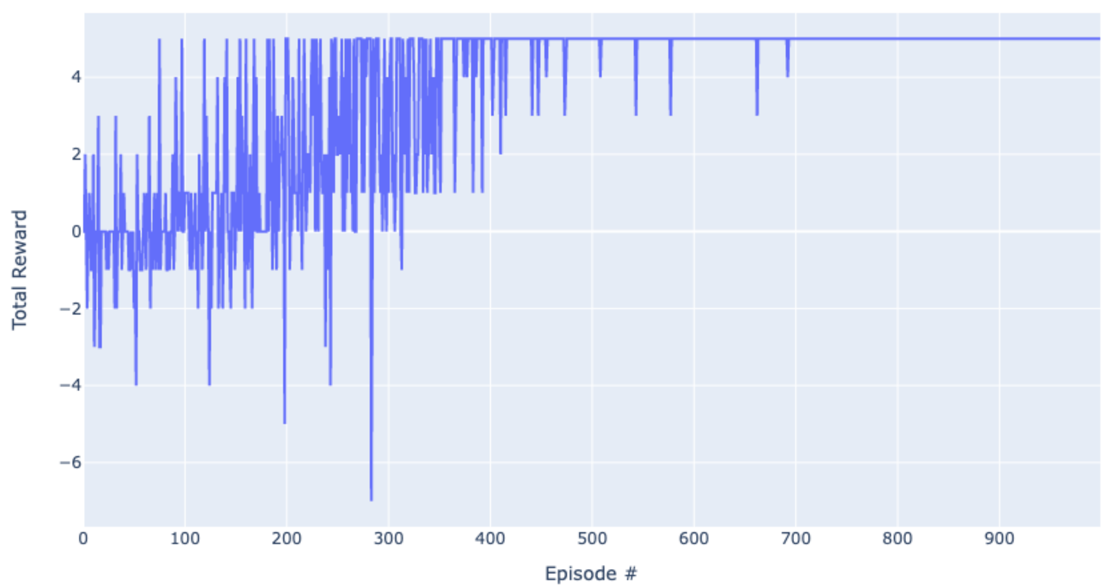
2.2 Applying Q-learning to solve the stochastic environment defined in Part 1. Plots should include epsilon decay and total reward per episode.



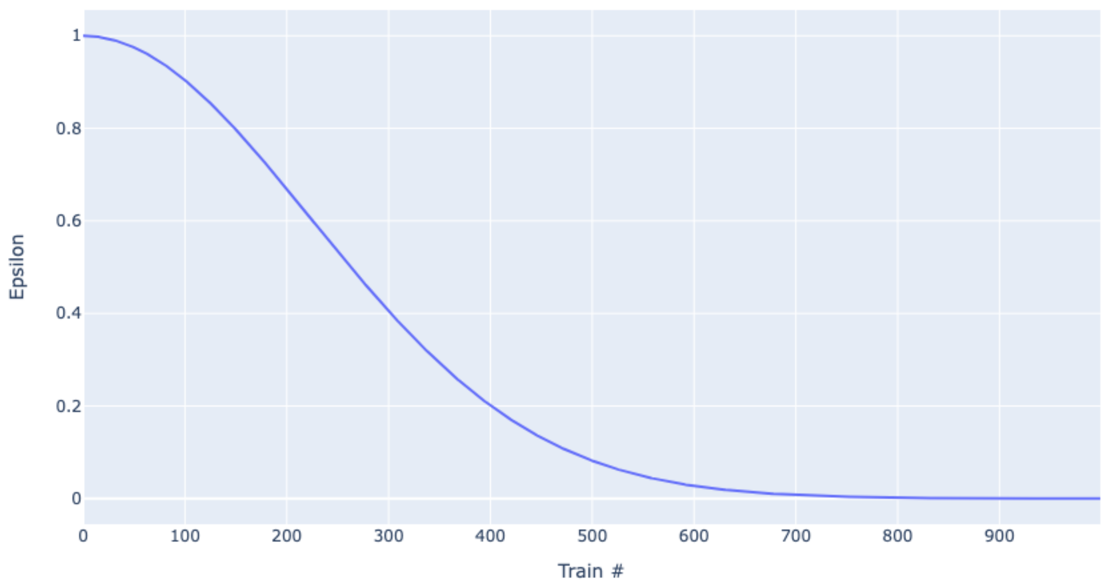


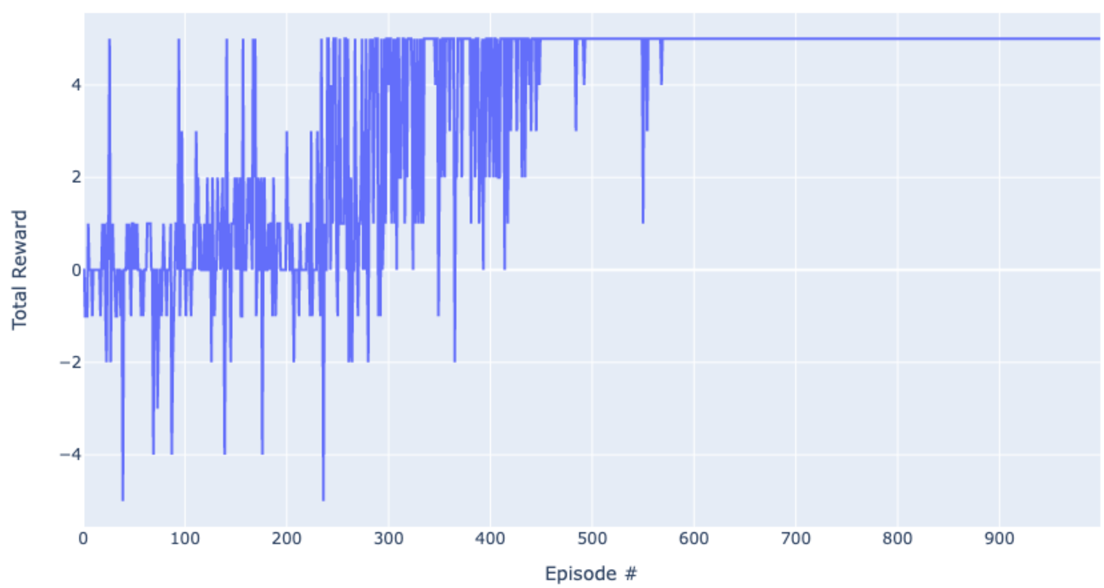
2.3 Applying any other algorithm of your choice(SARSA) to solve the deterministic environment defined in Part 1. Plots should include total reward per episode.



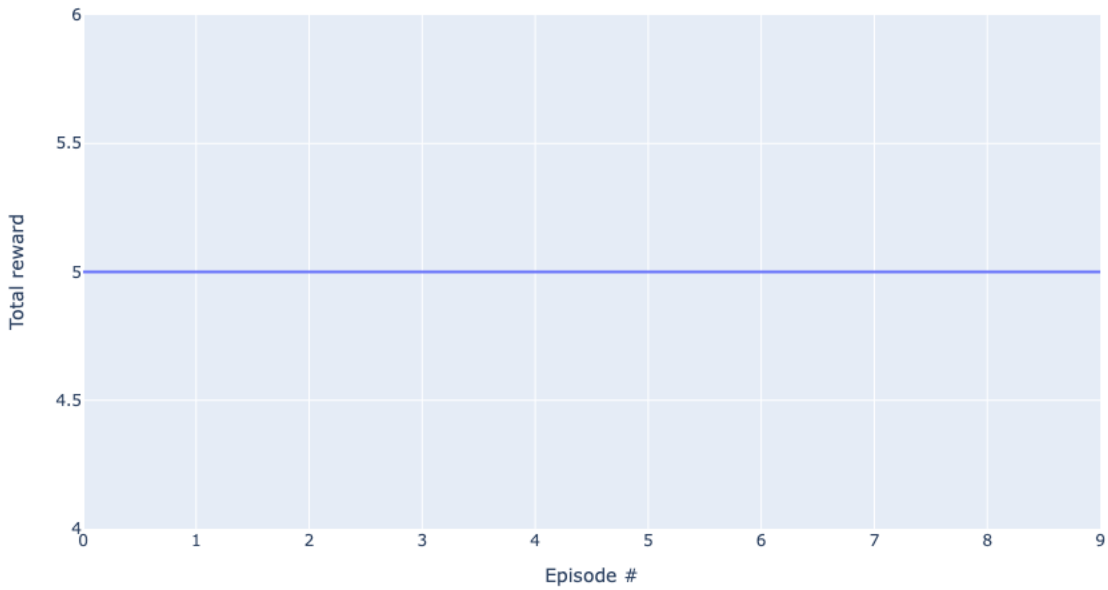


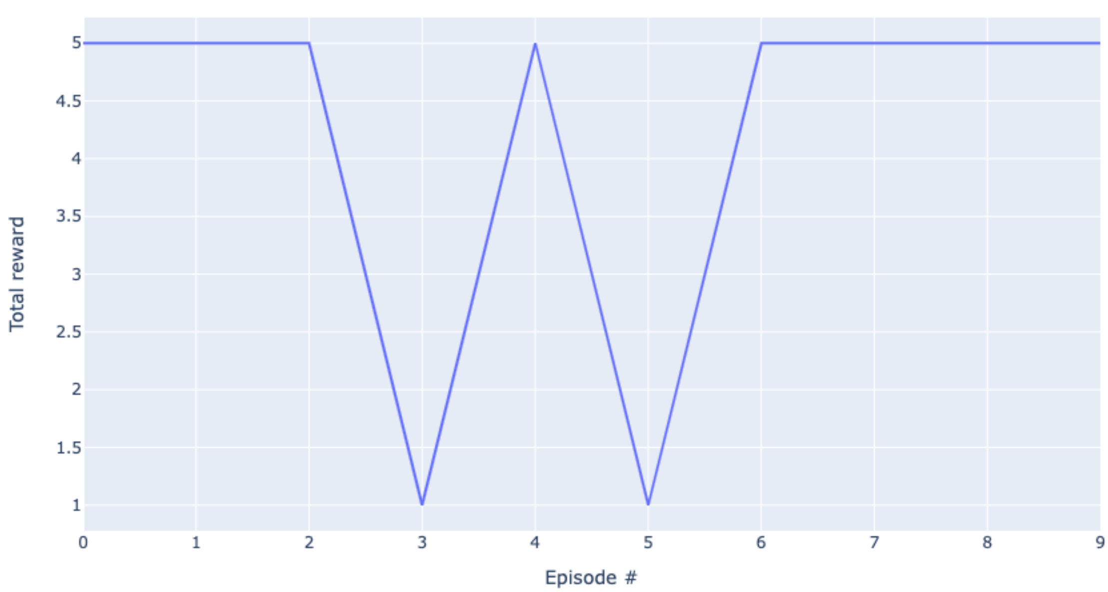
2.4 Applying any other algorithm of your choice(SARSA) to solve the stochastic environment defined in Part 1. Plots should include total reward per episode.



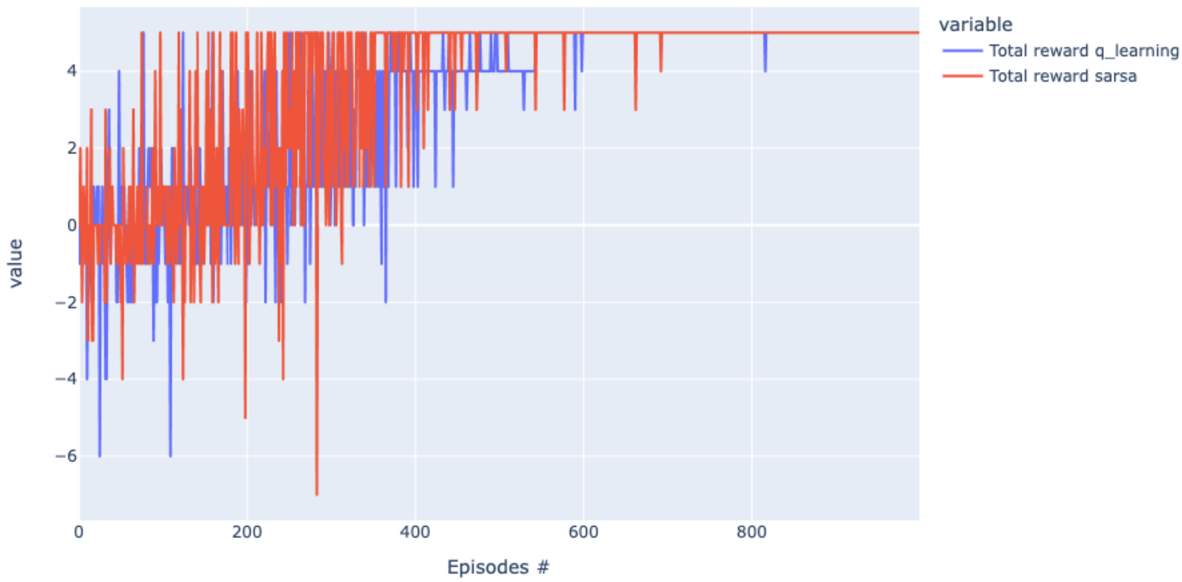


2.5 Provide the evaluation results. Run your environment for at least 10 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.

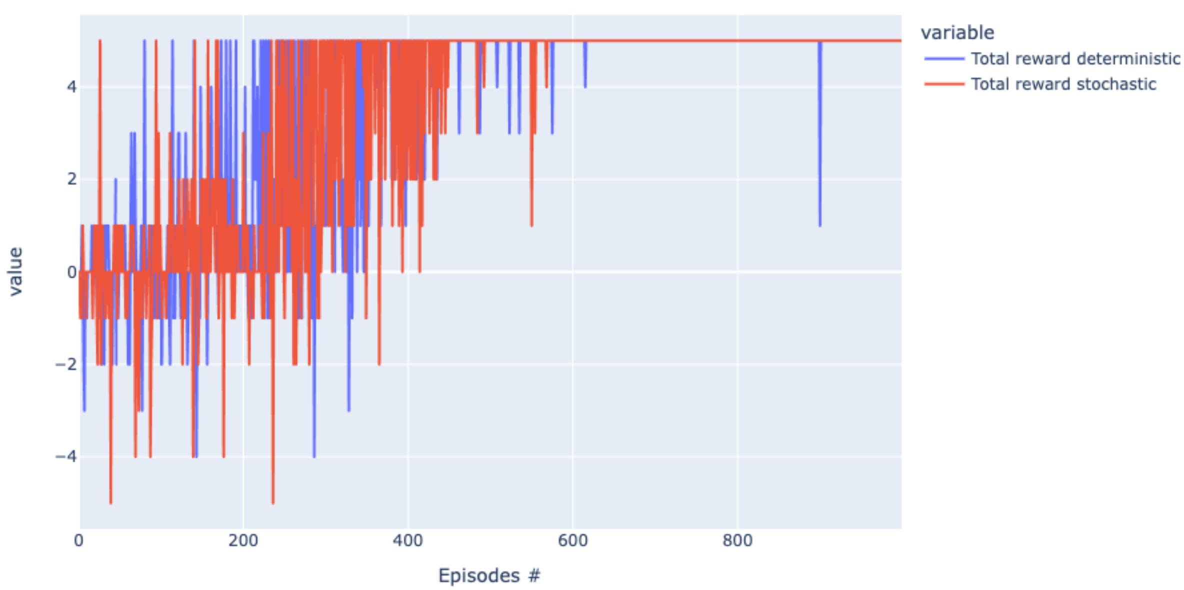




2.6 Compare the performance of both algorithms on the same deterministic environment (e.g. show one graph with two reward dynamics) and give your interpretation of the results.



2.7 Compare how both algorithms perform in the same stochastic environment (e.g. show one graph with two reward dynamics) and give your interpretation of the results.



2.8 Briefly explain the tabular methods, including Q-learning, that were used to solve the problems. Provide their update functions and key features.

2.8.1 Q-learning

In this off-policy tabular method, we use a greedy approach to converge towards the optimal policy.

Update func - qtable[state][action] = qtable[state][action] + alpha*(reward + discountfactor * maxq - qtable[state][action])

Here, maxq is the greedily chosen q value from the end state after we take an action.

2.8.2 SARSA

In this on-policy tabular method, we update the Q-table with policy determined decisions.

Update func - qtable[state][action] = qtable[state][action] + alpha*(reward + discountfactor * policyq - qtable[state][action])

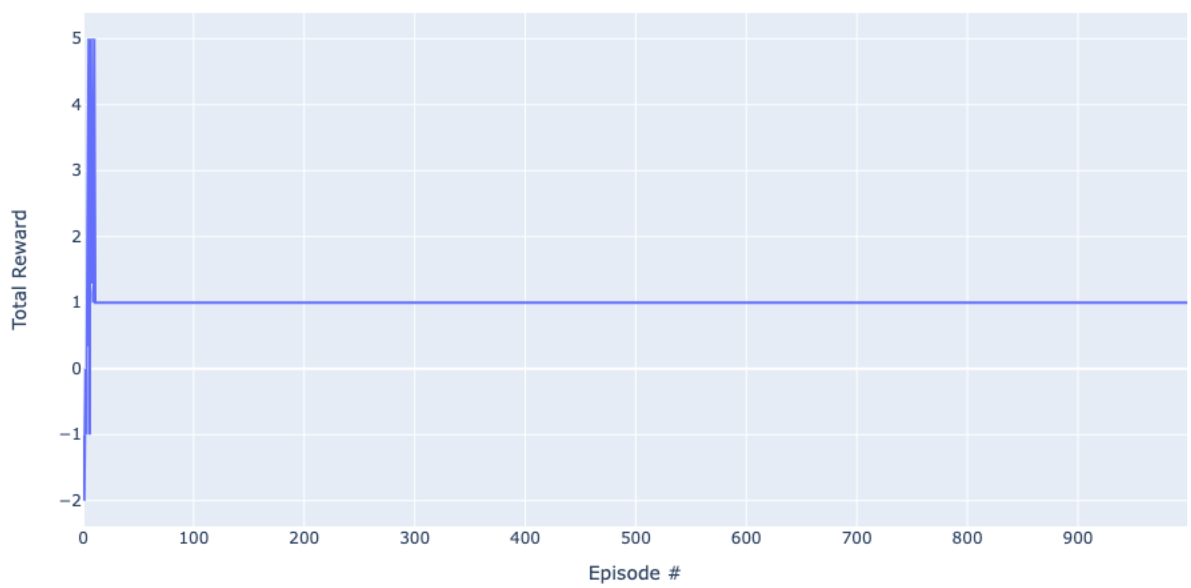
Here, the only difference from q-learning is that we choose 'policyq' based on the next action our policy(-greedy here) determines.

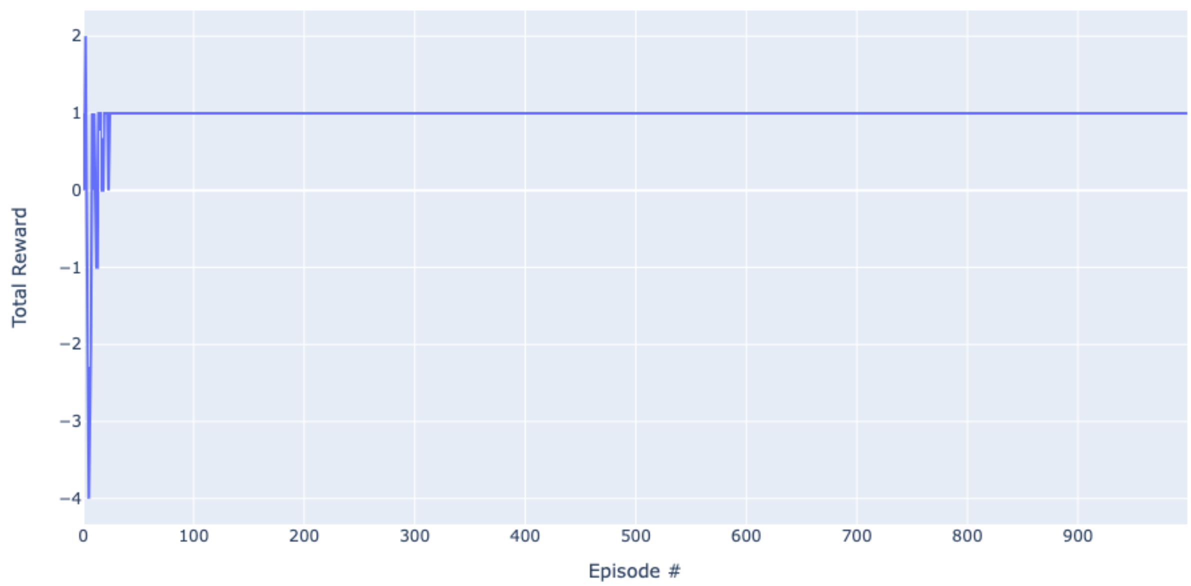
3 Hyper parameter tuning

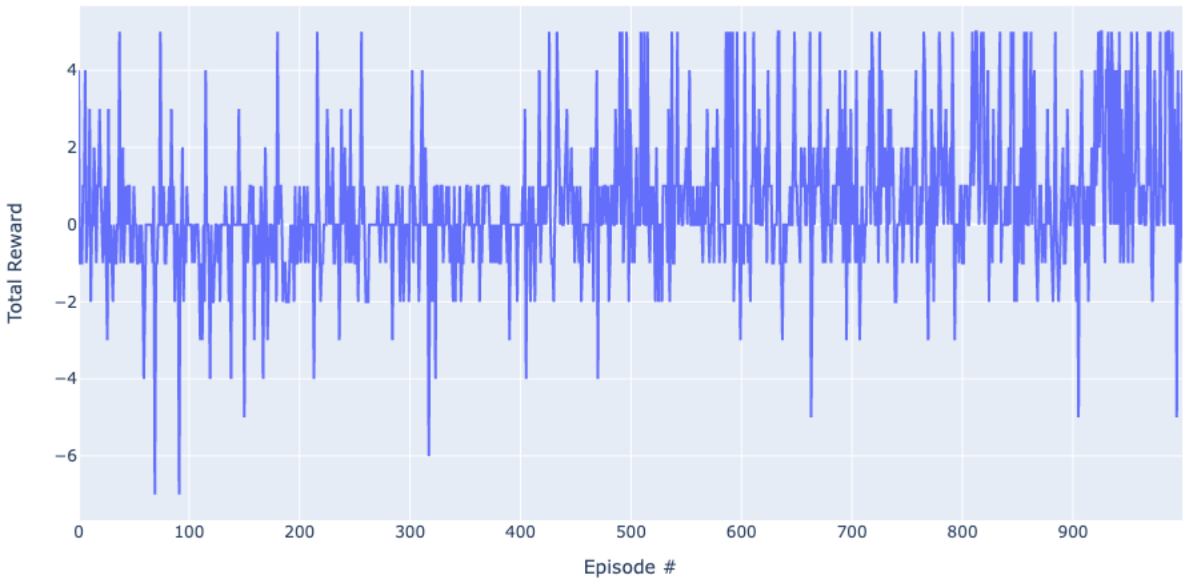
3.1 Epsilon decay rate

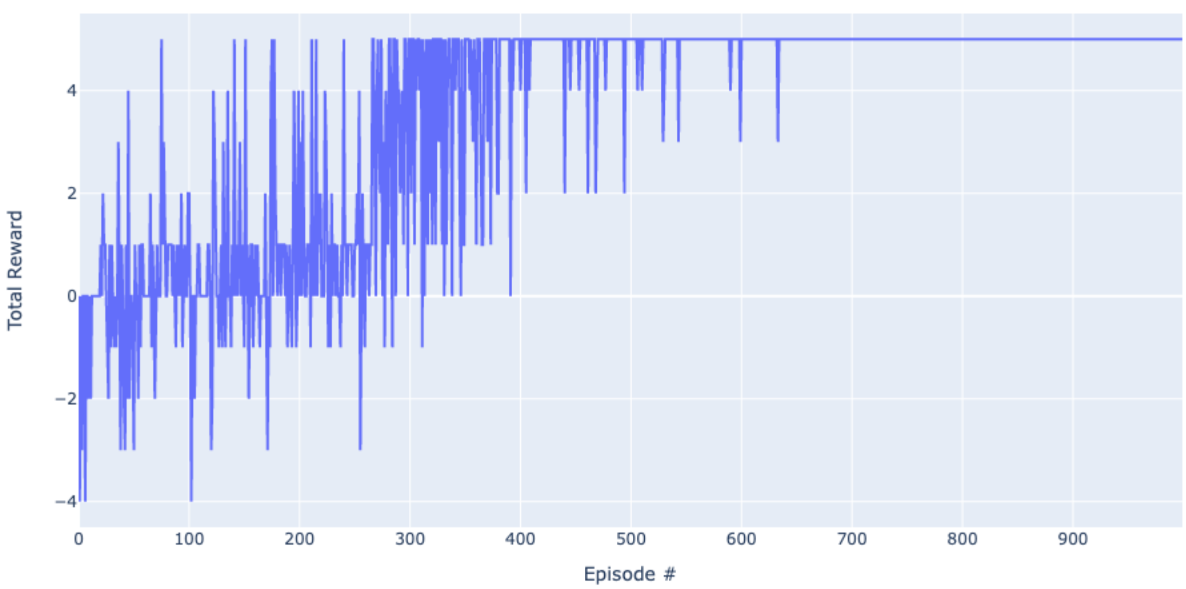
Please refer the jupyter notebook for more graphs, code etc.

When my epsilon was 0.03, my agent wasn't reaching the goal since the agent was exploring for long enough to find the best possible path as seen in the corresponding graph(steep slope). With the decay rate 0.000005, the slope was very gentle and my agent again wasn't reaching the goal as epsilon was still big enough towards the latter episodes to cause frequent spurts of random behaviour(exploration).




Total Reward
2
1
0
−1
−2
−3
−4
0
100
200
300
400
500
600
700
800
900
Episode #



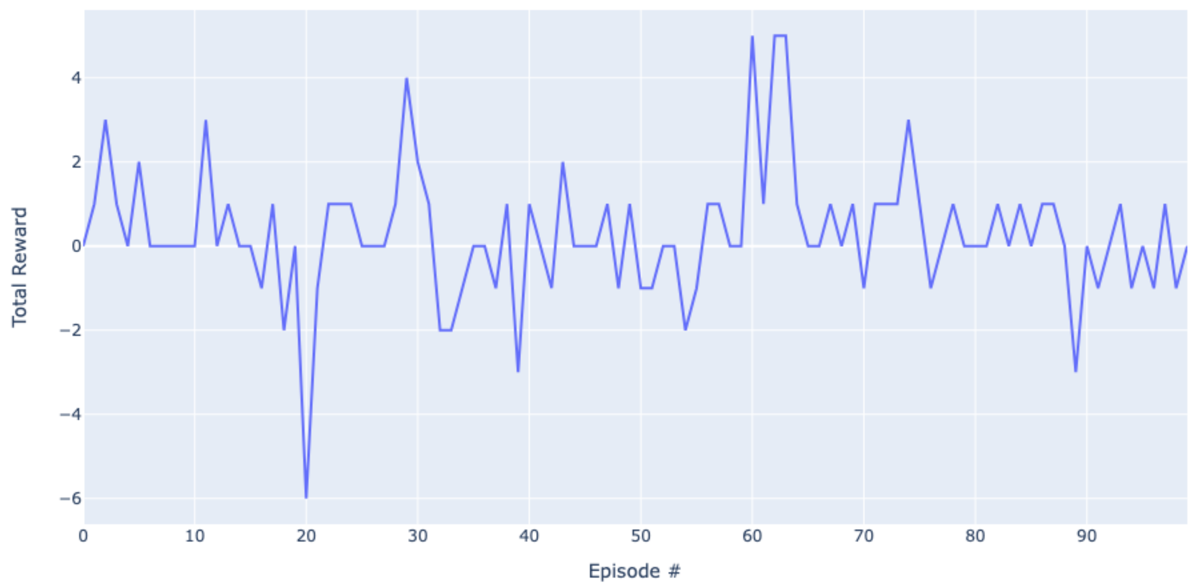


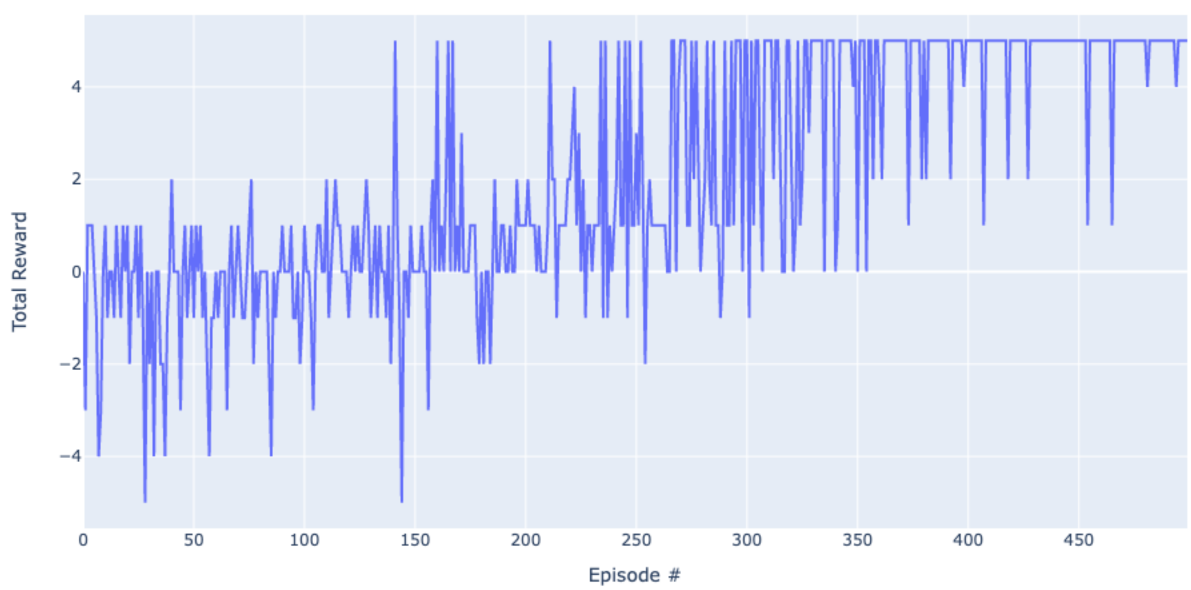
For epsilon=0.00002, the slope produced was much smoother causing a gradual transition to exploitation. Epsilon was converging to 0, roughly around 70% through the episodes(700) which seemed like the right spot to only do exploitation. So this, was chosen as the decay rate for epsilon.

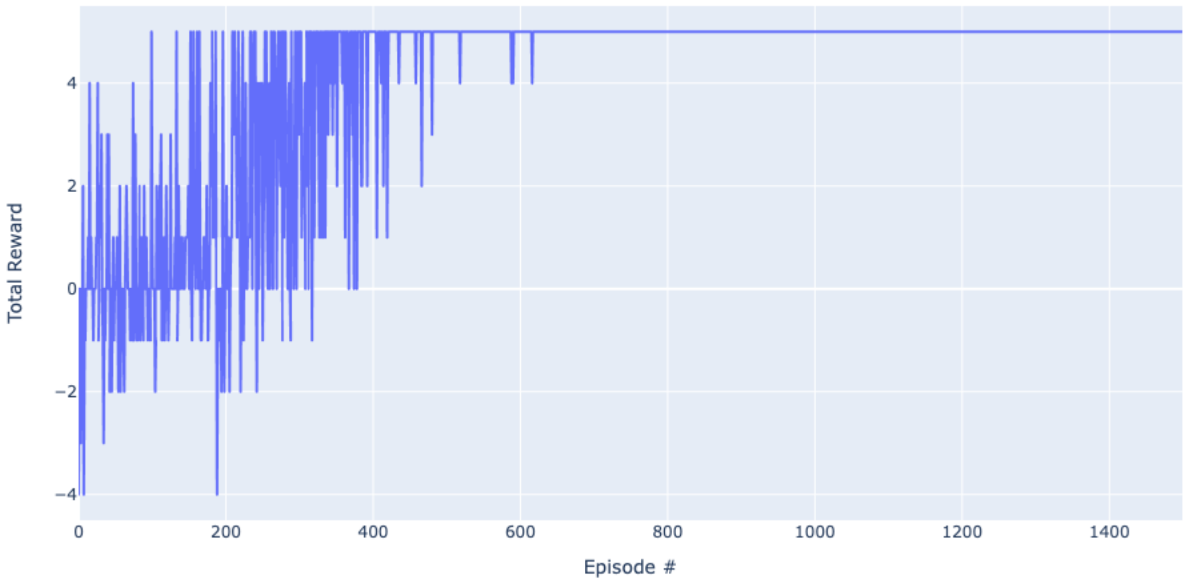
Note - I have used the exponential decay formula to calculate epsilon for each training episode.

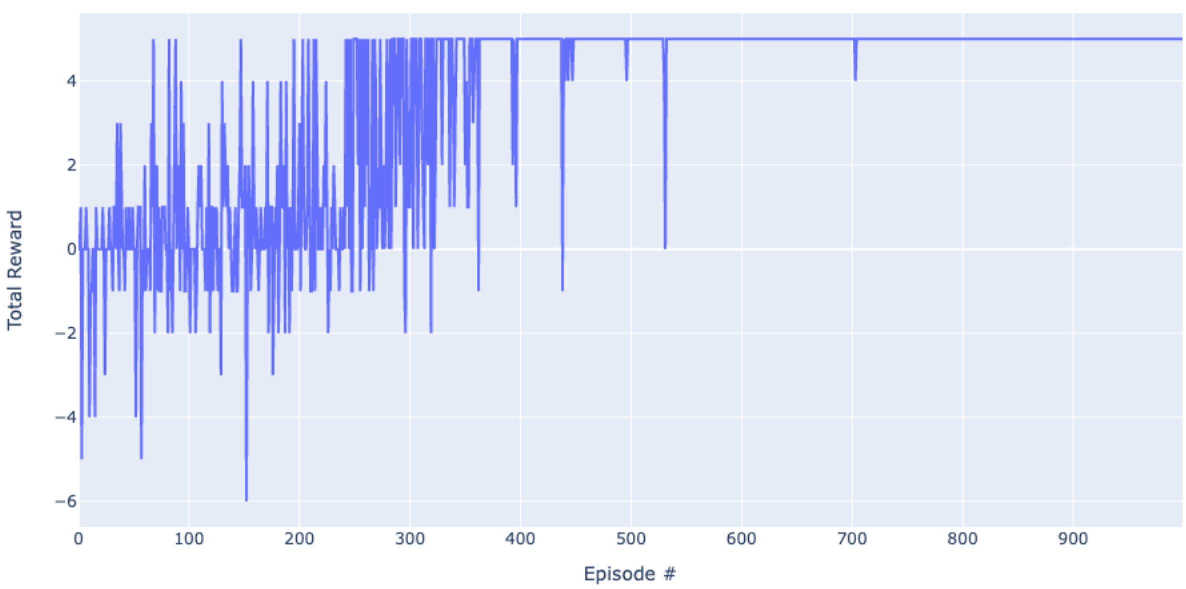
3.2 Number of episodes

First we must note that number of episodes and epsilon decay are related in this case since we use exponential decay. For this experiment the epsilon decay rate was fixed to 0.00002 which was decided at optimum earlier.









When we train for 100 episodes we can see that rewards haven't convert much even in the latter iterations, for 500 episodes the rewards seem to have just converged towards the end episodes.

Third graph is 1500 iterations, where the rewards seem to converge around 600 mark so we just seem to be wasting iterations at this point. Finally for 1000, iteration we see that the rewards converge the same mark (a later fluctuation might be due to randomness in the epsilon greedy policy). Since that's also doesn't happen for this training after roughly 70