**4 – Q-Learning**

1. The reward function according to the reward specification is a cumulative reward -

For goal not reached at step T-

For goal reached at step T-

Setting the reward at the goal state as a relatively high positive value (instead of a 0) affects the agent. The high reward for goal state rewards a case of reaching the goal state after up to 100 actions higher than any case of not reaching the goal state in this time horizon. Also, it enables us to grant negative rewards for each state reached which is not the goal state, instead of 0 reward in previous settings. This way the agents is encouraged to reach goal in fewer steps (higher reward).

The required cumulative reward is set to . The car is required to reach the peak in steps at max. Since the cumulative reward in monotonically decreasing by 1 in each step for which the goal isn’t reached, if by the 175th step the car is not at the peak of the mountain, the cumulative reward is . Reaching the goal state at a later step will add to the reward, meaning we will not reach the target of .

1. We implemented the following evaluation criterion –

Randomly selecting 10 starting states, each starting in position −0.5 (bottom of valley), and with a small uniformly distributed velocity. We ran the greedy policy with respect to the current Q function (without exploration) for each iteration.

The average success rate was defined as–

Before applying the learning process, the average success rate of reaching the top was 0.

1. We Implemented the Q-learning algorithm and calculated the performance as defined in section (2).

The update of was implemented by the relation seen in lectures –

In our problem is linearly dependent on –

being the features of the problem. Therefore –

For the terminal state we get –

The results for running the algorithm until finding solution (finding solution being – reached performance (=success rate) = 1) are displayed below.

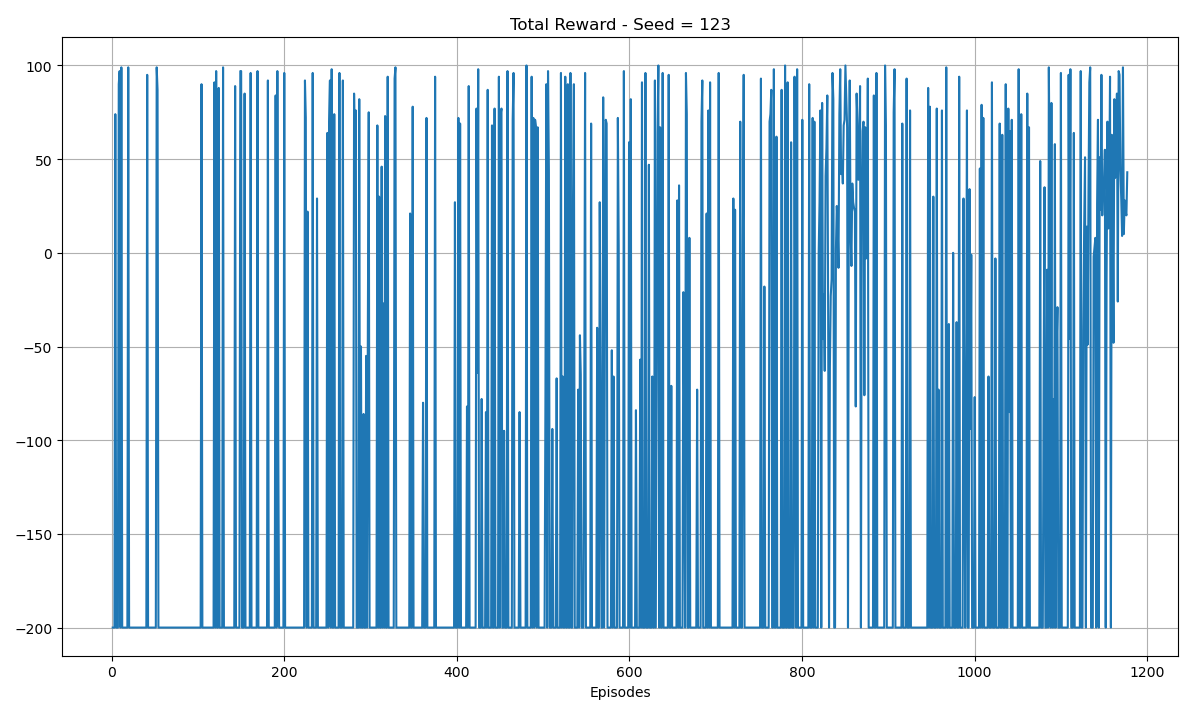
Since initial values of are randomly selected, in addition to a random factor of selecting starting states, the run time until reaching solution varied between runs.

In results displayed below, solution was found after-

* First Seed = 123: 1179 episodes (first run) and after 1829 episodes (second run). We chose to display for two runs to show variance of running time.
* Second Seed = 234: 199 episodes (third run).
* Third Seed = 345: 419 episodes (forth run).

For all runs we chose -

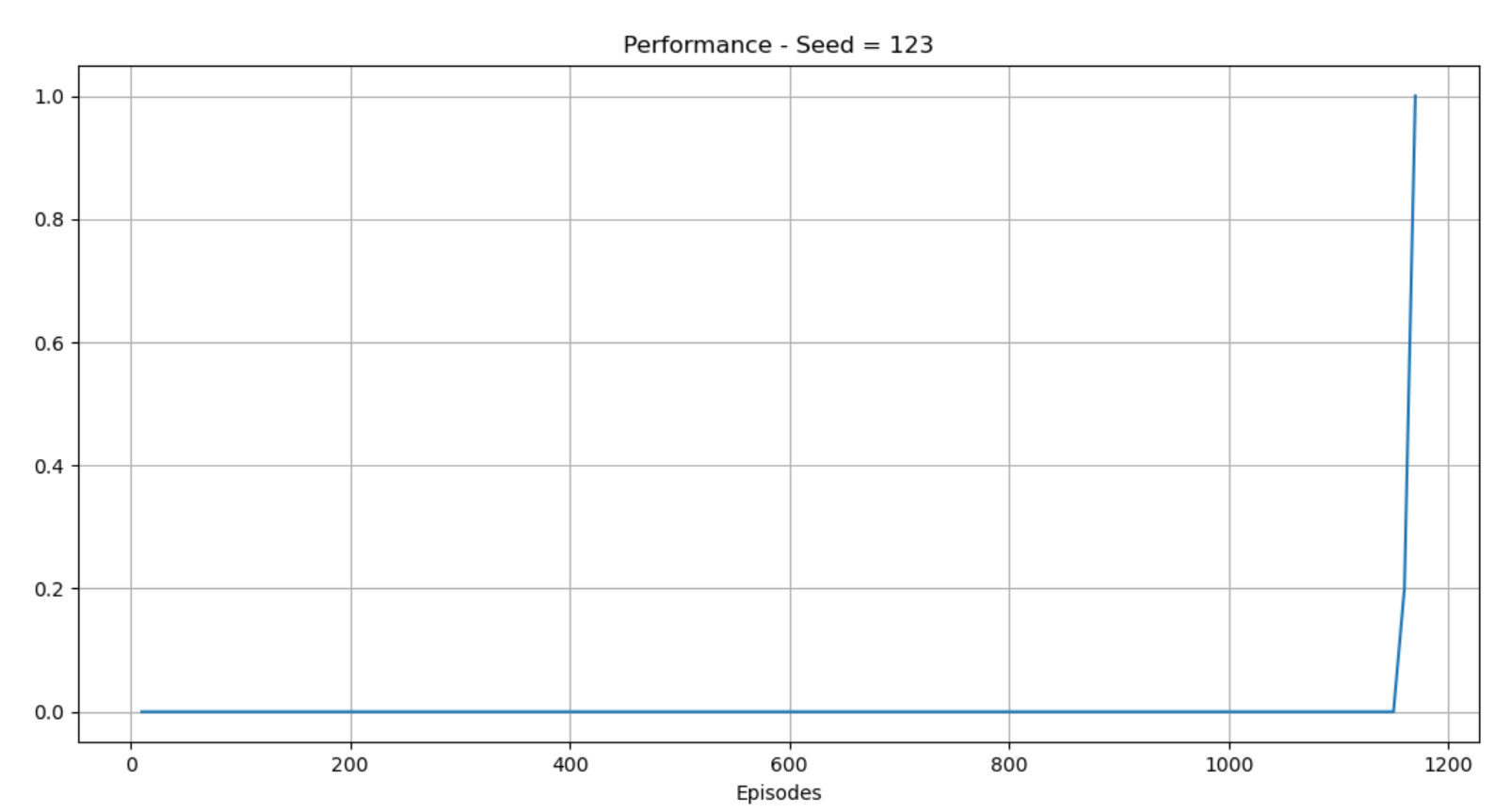
* First run – Seed = 123, 1179 episodes of training.



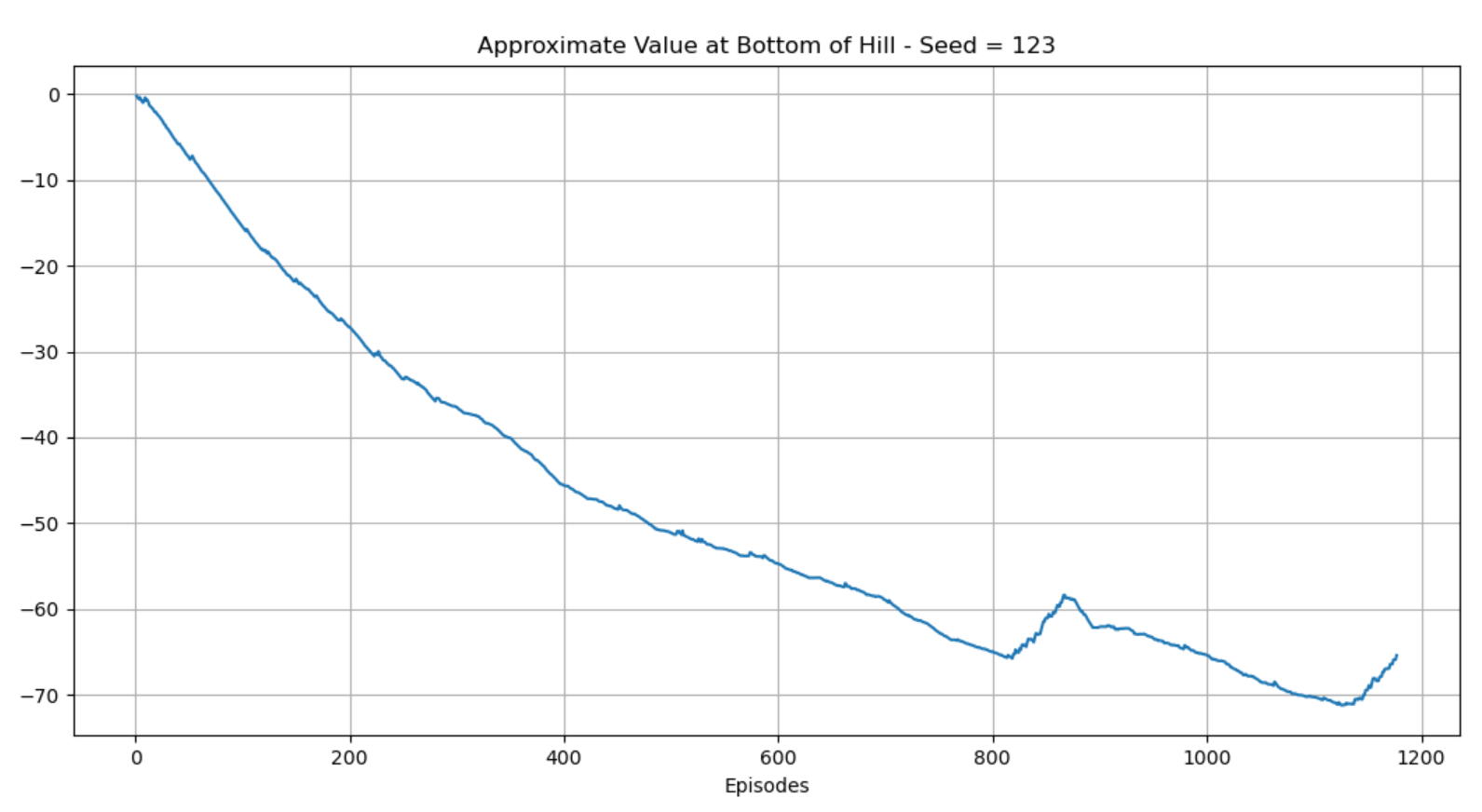
As seen above, a large portion of the training episodes resulted with reward -200. This reward is result of not reaching the top in the episode (max number of steps in episode was set to 200).

Also seen above are episodes with high reward early-on in the training process. Since each episode randomly selects a starting state, there is some probability of choosing a state close to the terminal state. For these episodes reward is high (up to 100) however the evaluation criteria are set for the bottom hill state, meaning high reward in plot above does not indicate closeness to solution. If, however, for a majority of consecutive episodes we see reward this may indicate with some probability the rewards for bottom hill state being .

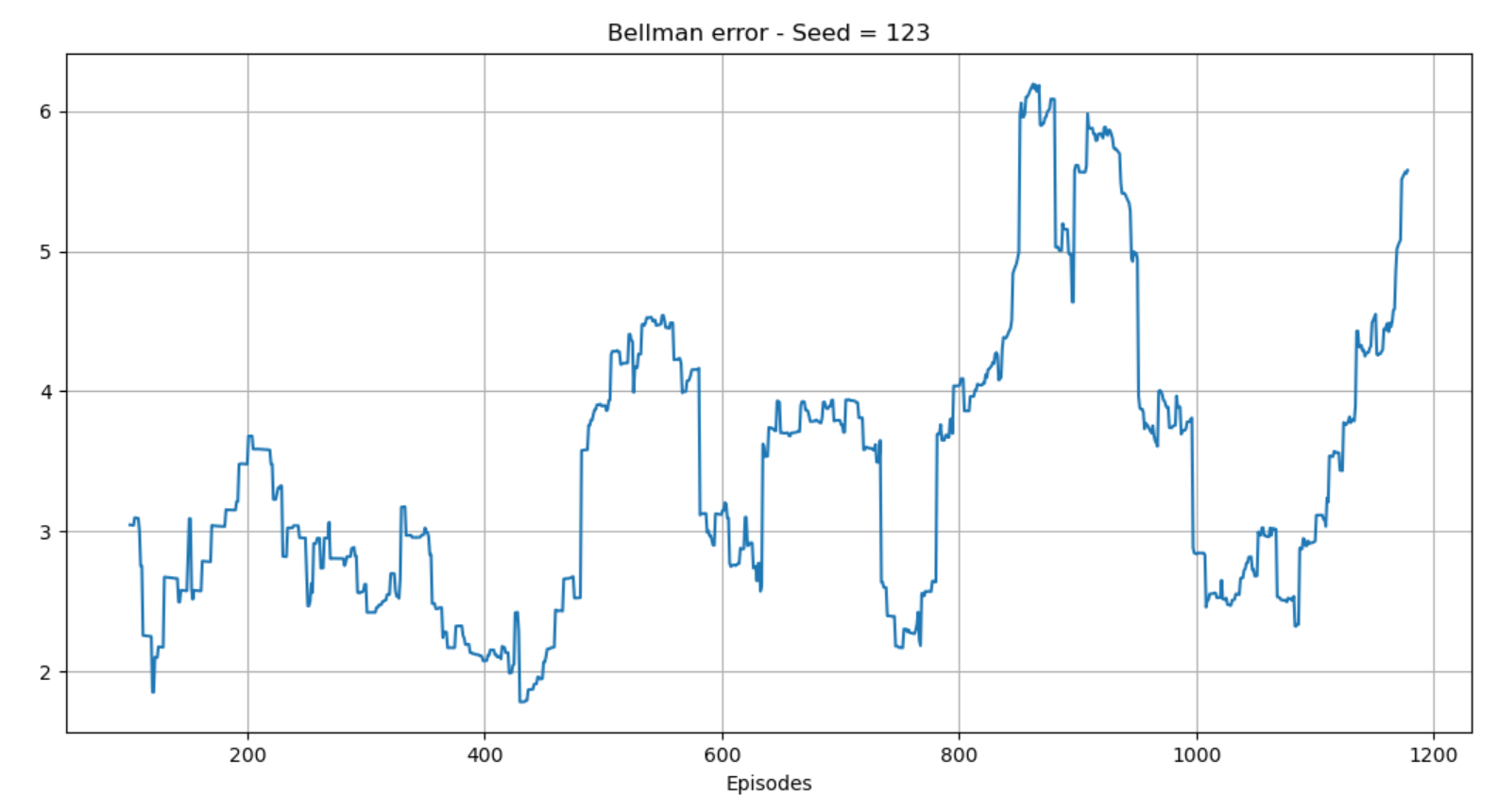
As seen in the last ~50 training episodes, the reward was , meaning the car reached the top for these episodes. Also, the reward for last ~50 training episodes was , indicating with some probability that we are close to reaching solution.



Performance was calculated every 10 episodes. Performance definition is defined in section (2).



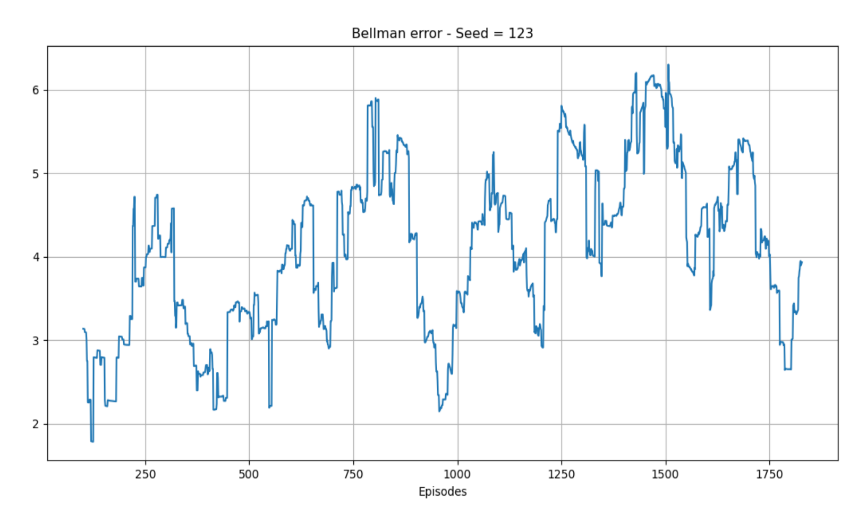
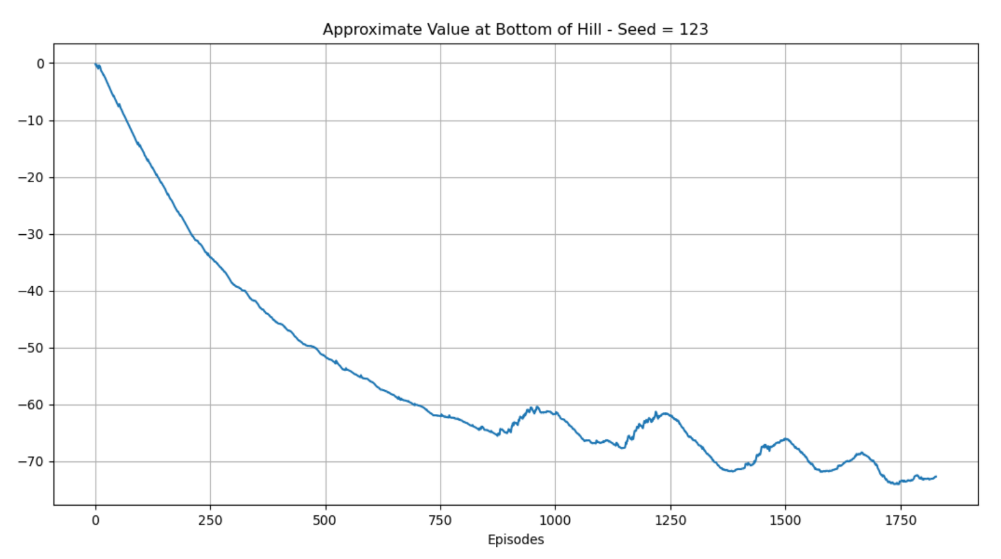
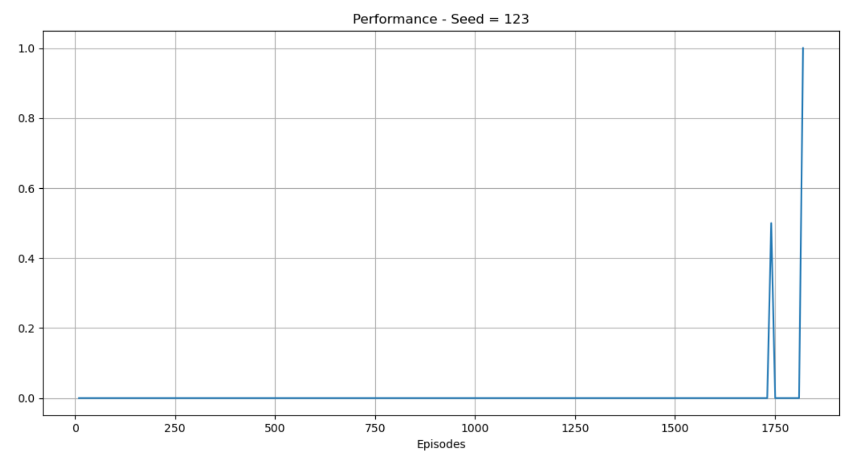
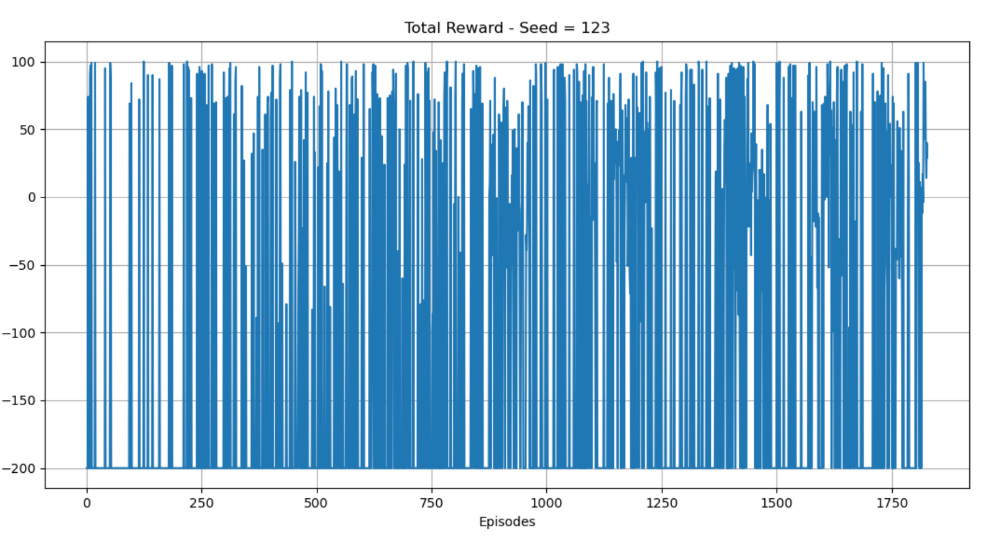
The value of the bottom hill state converges to about . We are required to reach reward of and above (for location of bottom hill will small velocity), therefore we expect the training process to converge to this value (or larger).



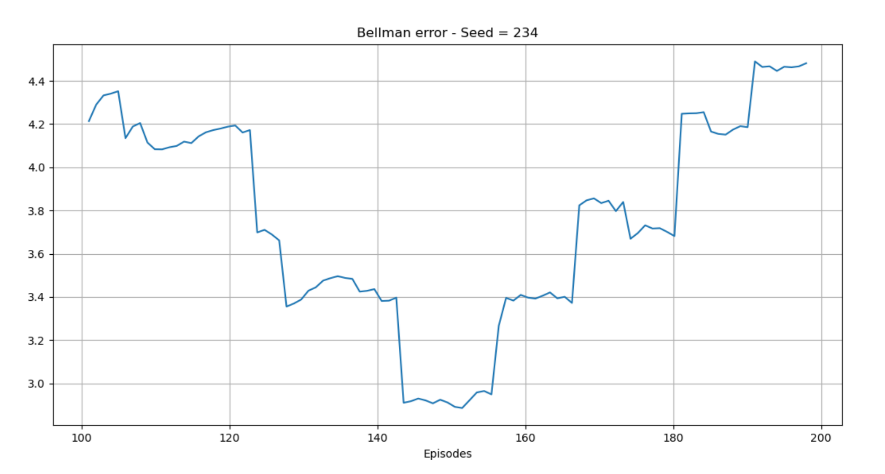
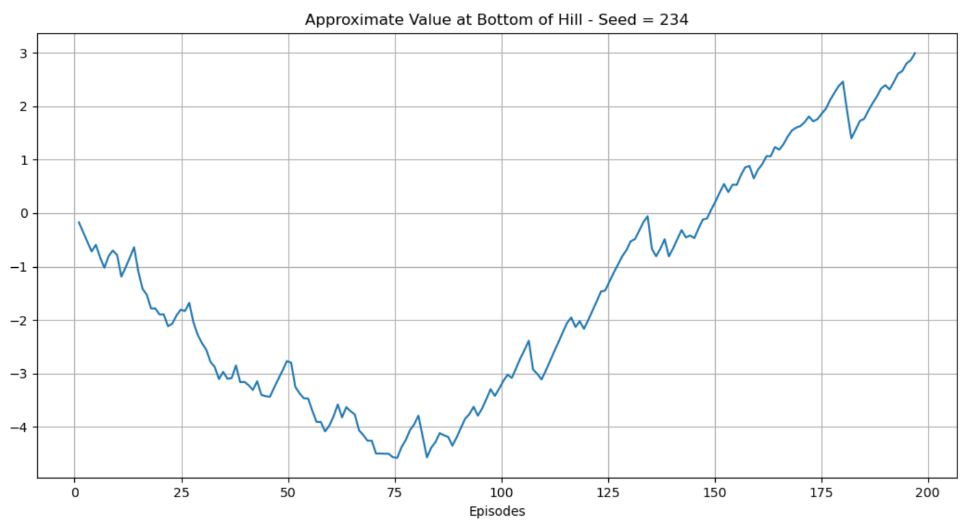
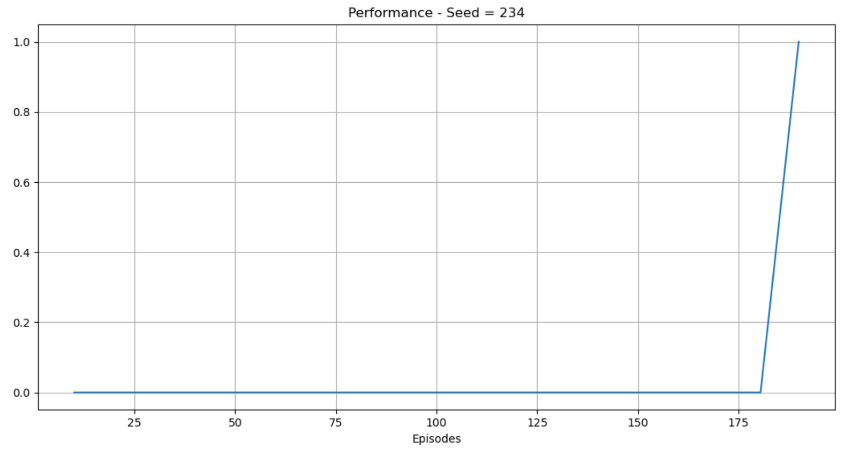
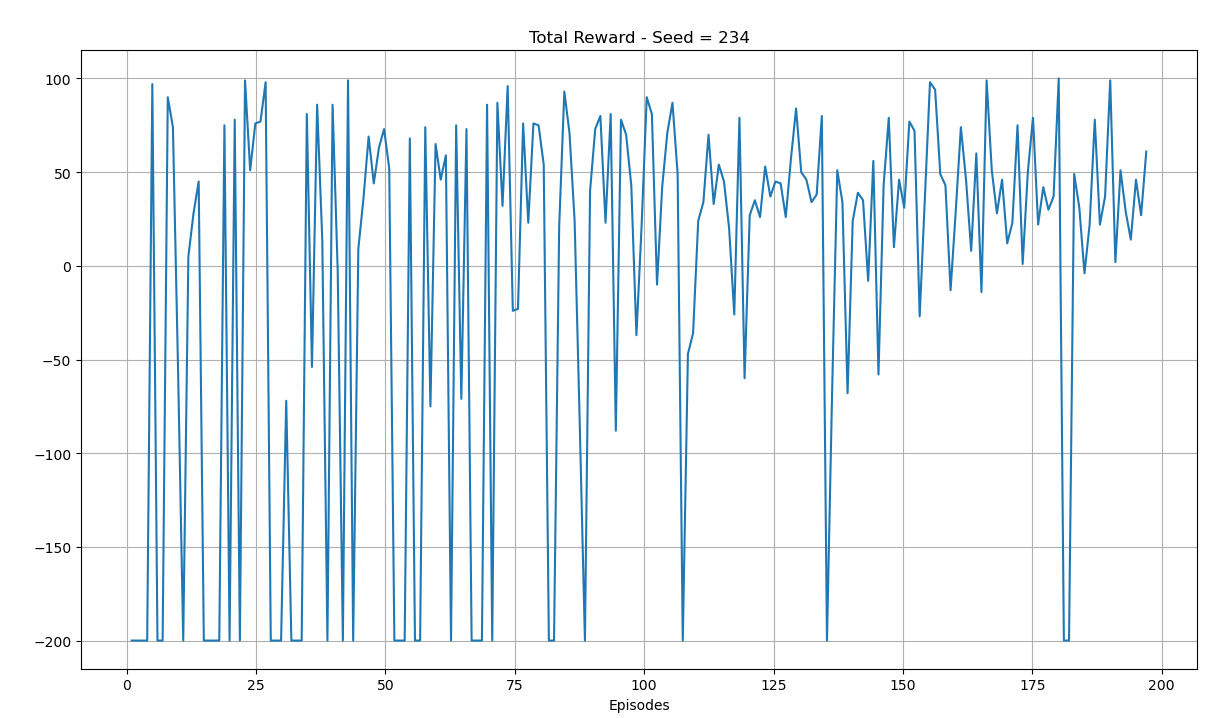
Bellman error displayed is average over 100 last episodes.

In each episode Bellman error is average over all steps of-

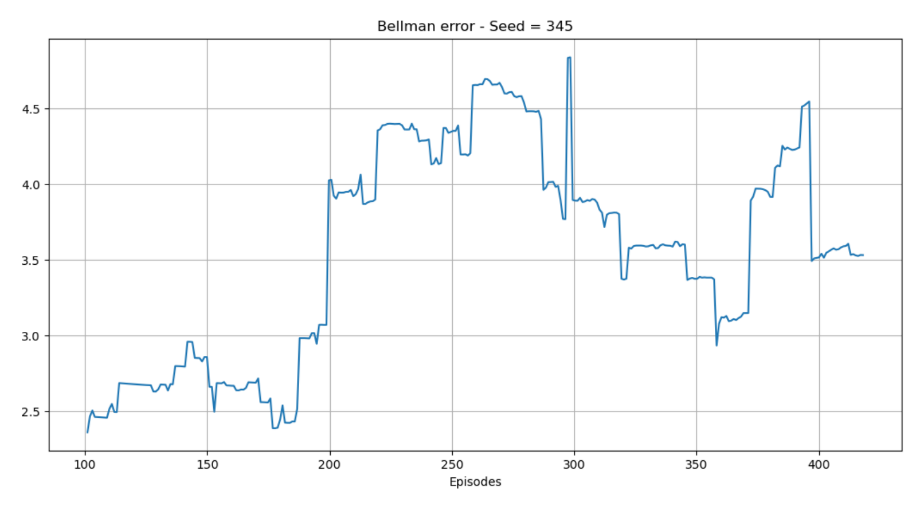
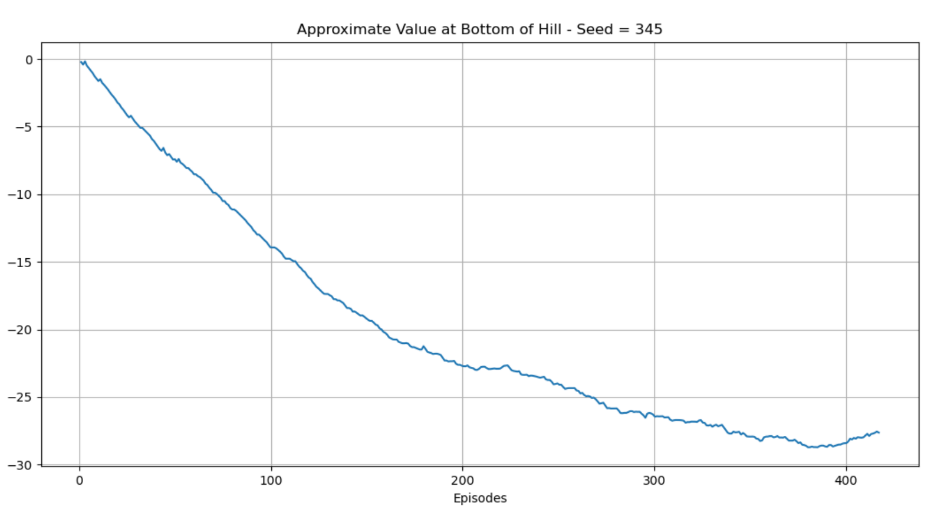
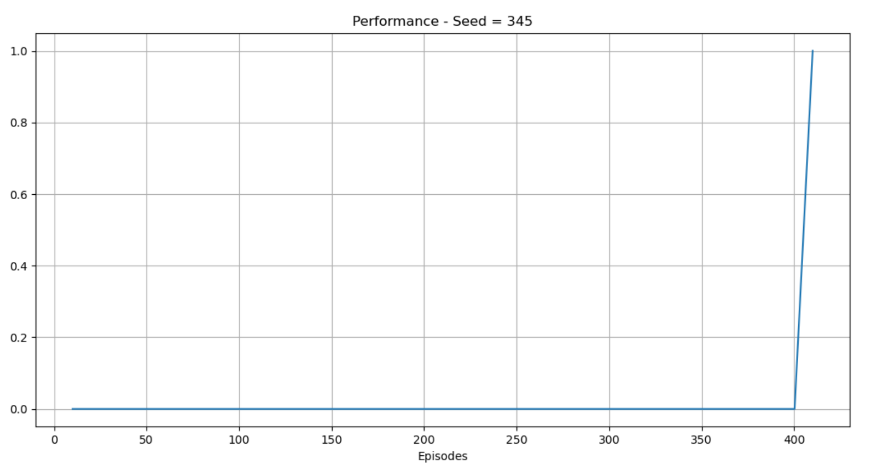
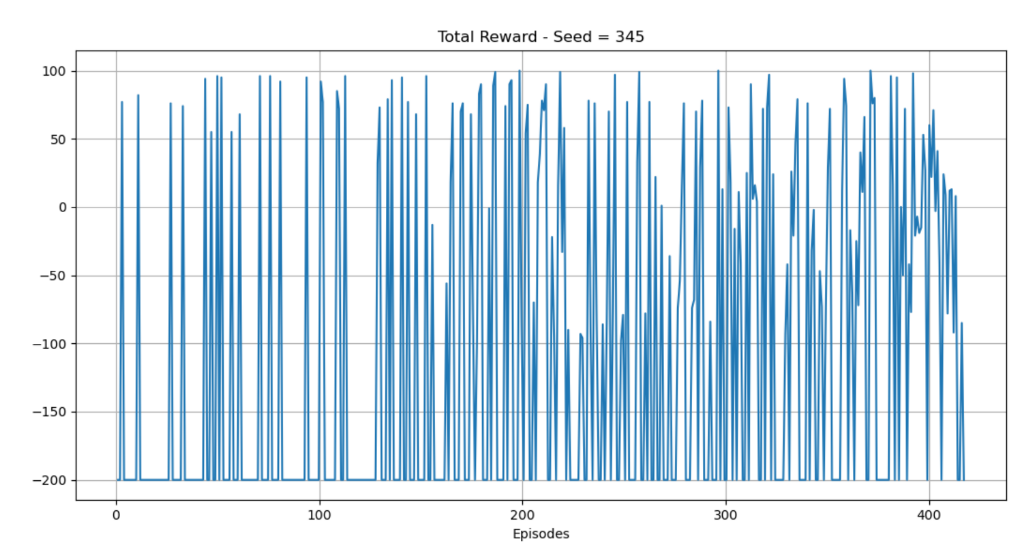
* Second run – Seed = 123, 1829 episodes of training



* Third run – Seed = 234, 199 episodes of training



* Forth run – Seed = 345, 419 episodes of training

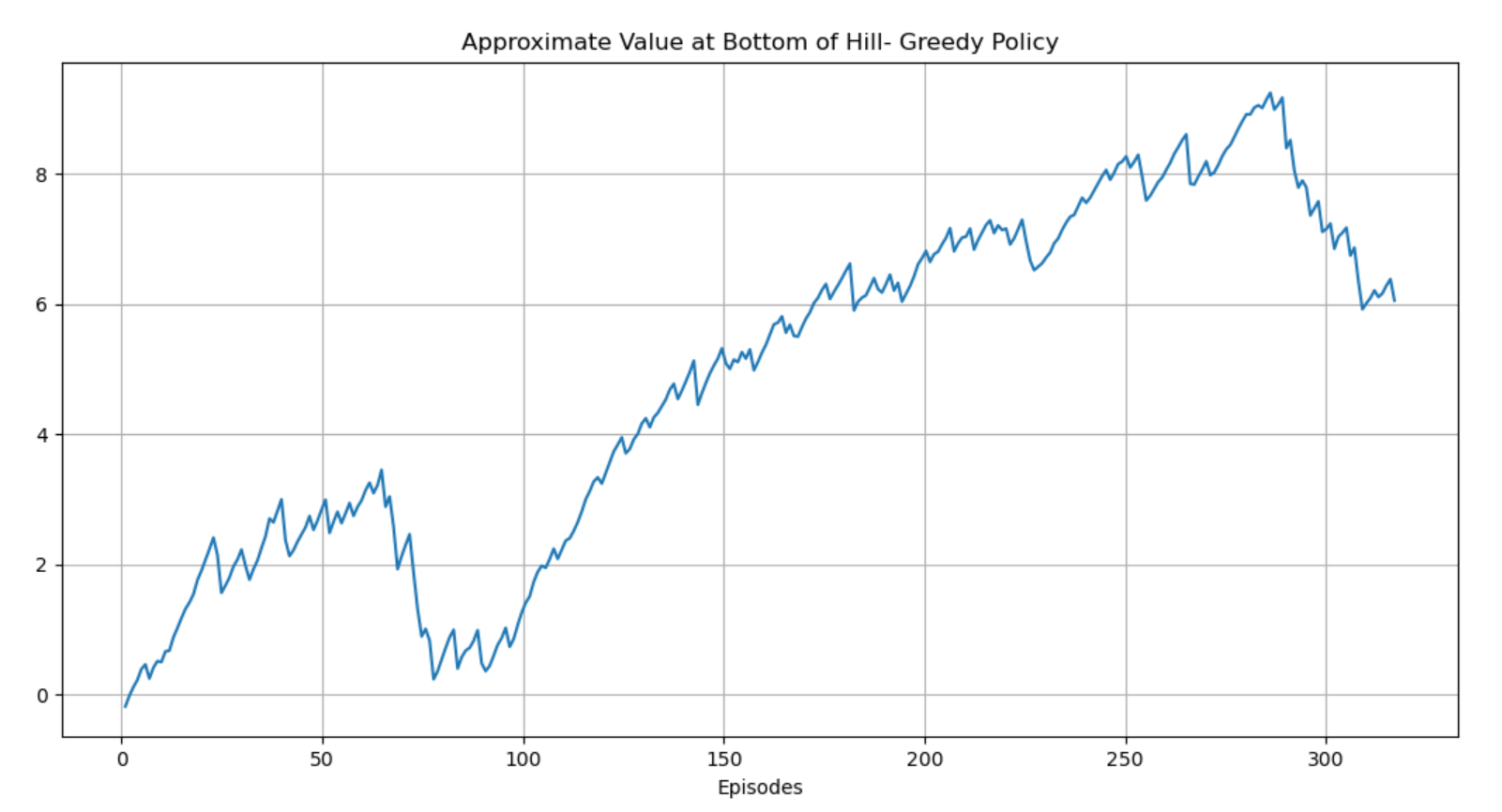


In all seeds we observed no converges to 0 for the Bellman error, however it is bonded (for all cases - ). The converges of the bellman error to 0 is not promised as the Q-Learning in our case is for approximate values of (weighted features) and all convergence theorems have been proved for un-approximated Q function.

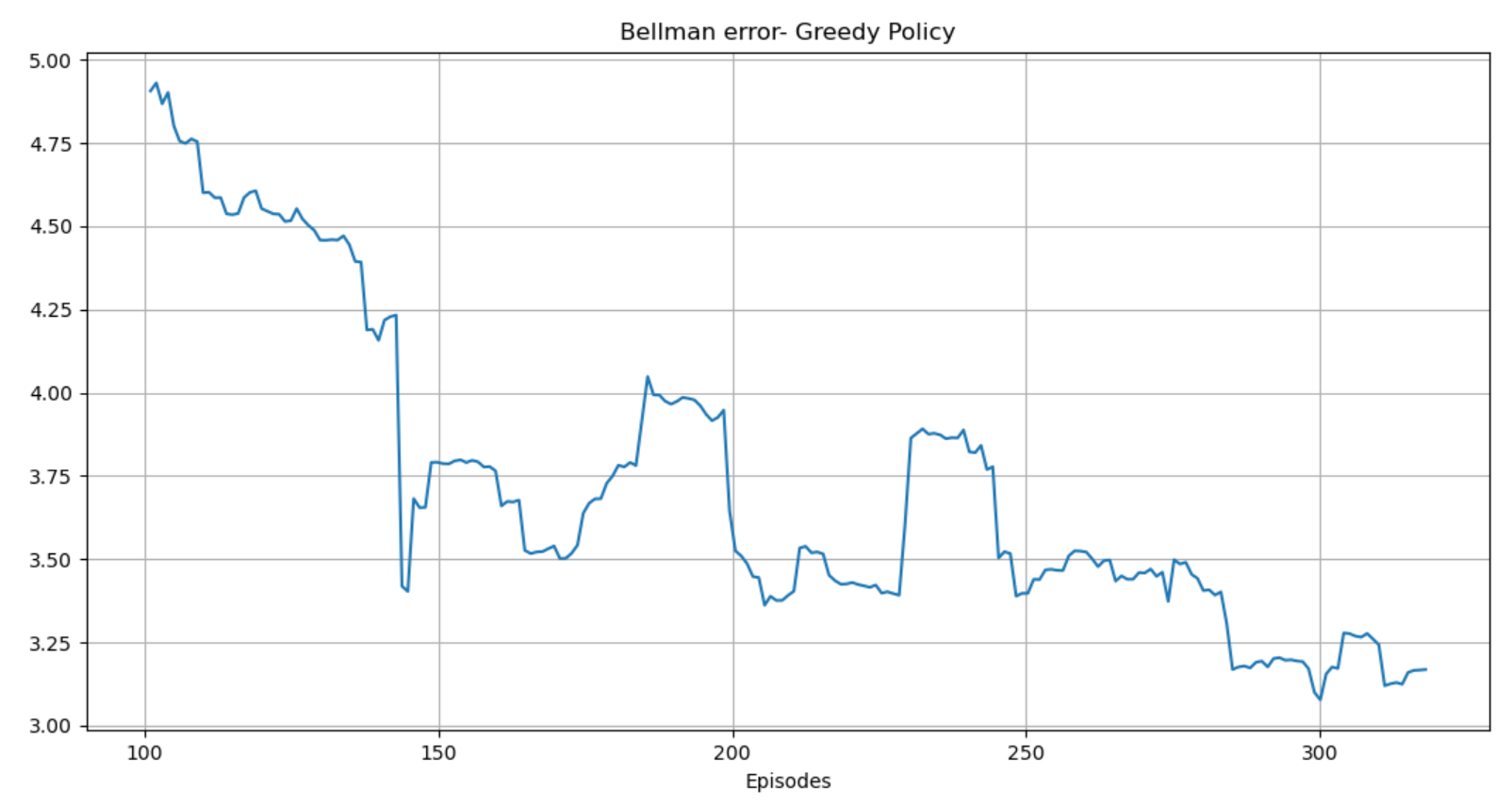
1. We ran the greedy policy (no exploration - ) until solution was found (criteria for finding solution is described in previous sections (performance = 1)).

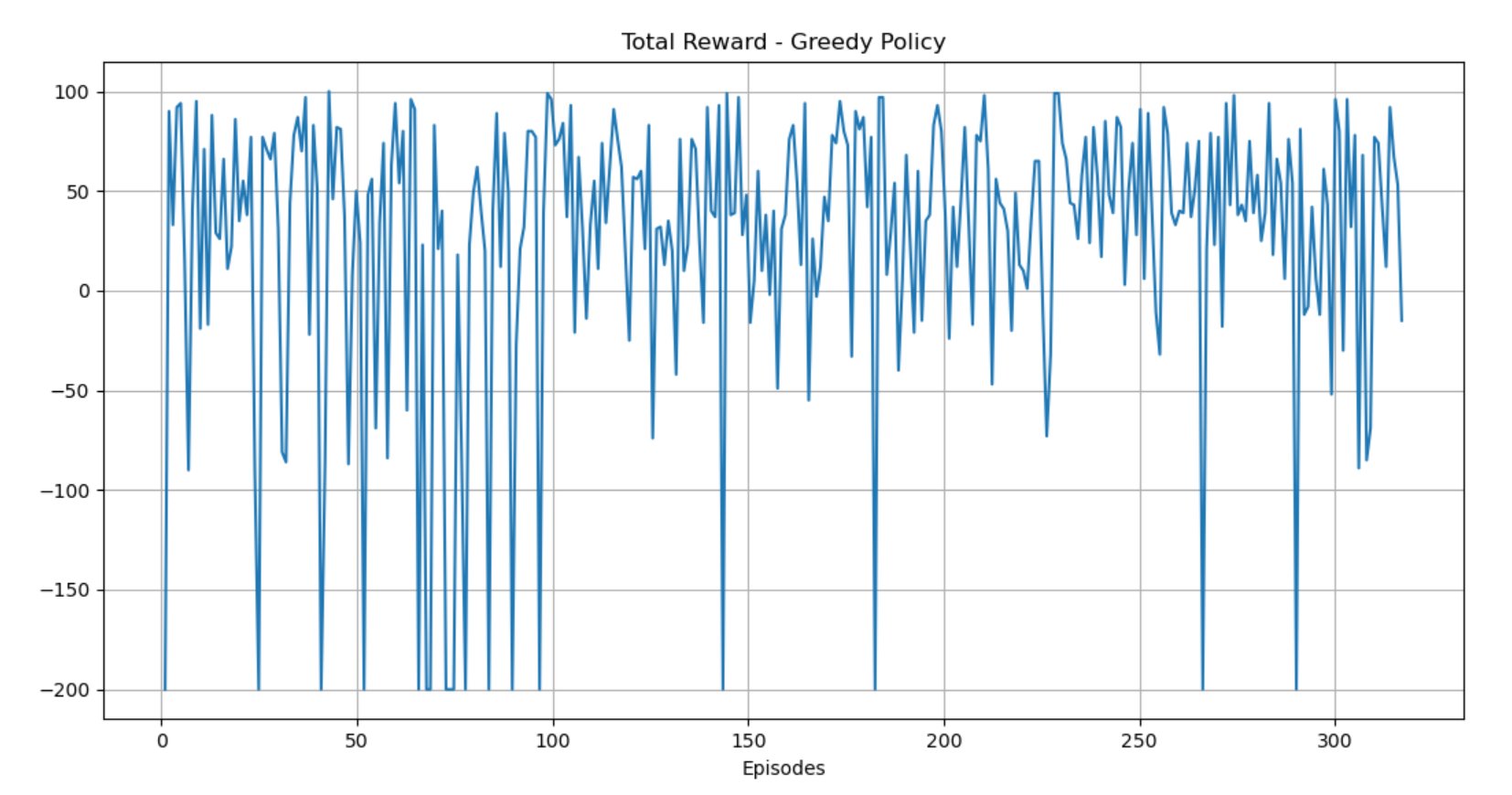
We used same - and seed was 123.

It took the greedy policy 319 iterations to reach solution.

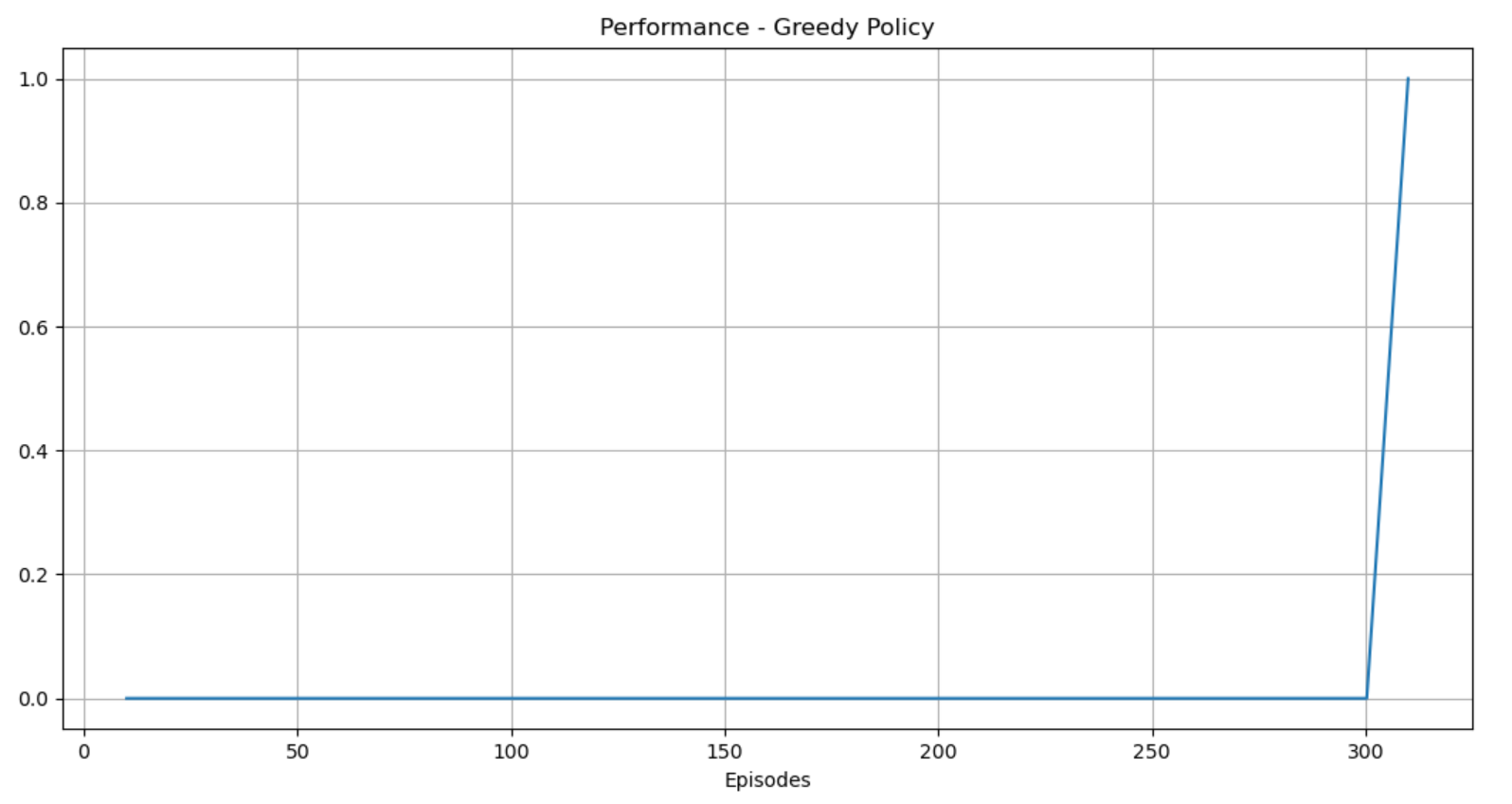


As seen above, the greedy policy reached a solution consisting less than 100 steps to reach the top (expected cumulative reward for bottom hill state is positive). This value is reasonable since we are searching for a solution consisting less than 175 steps.

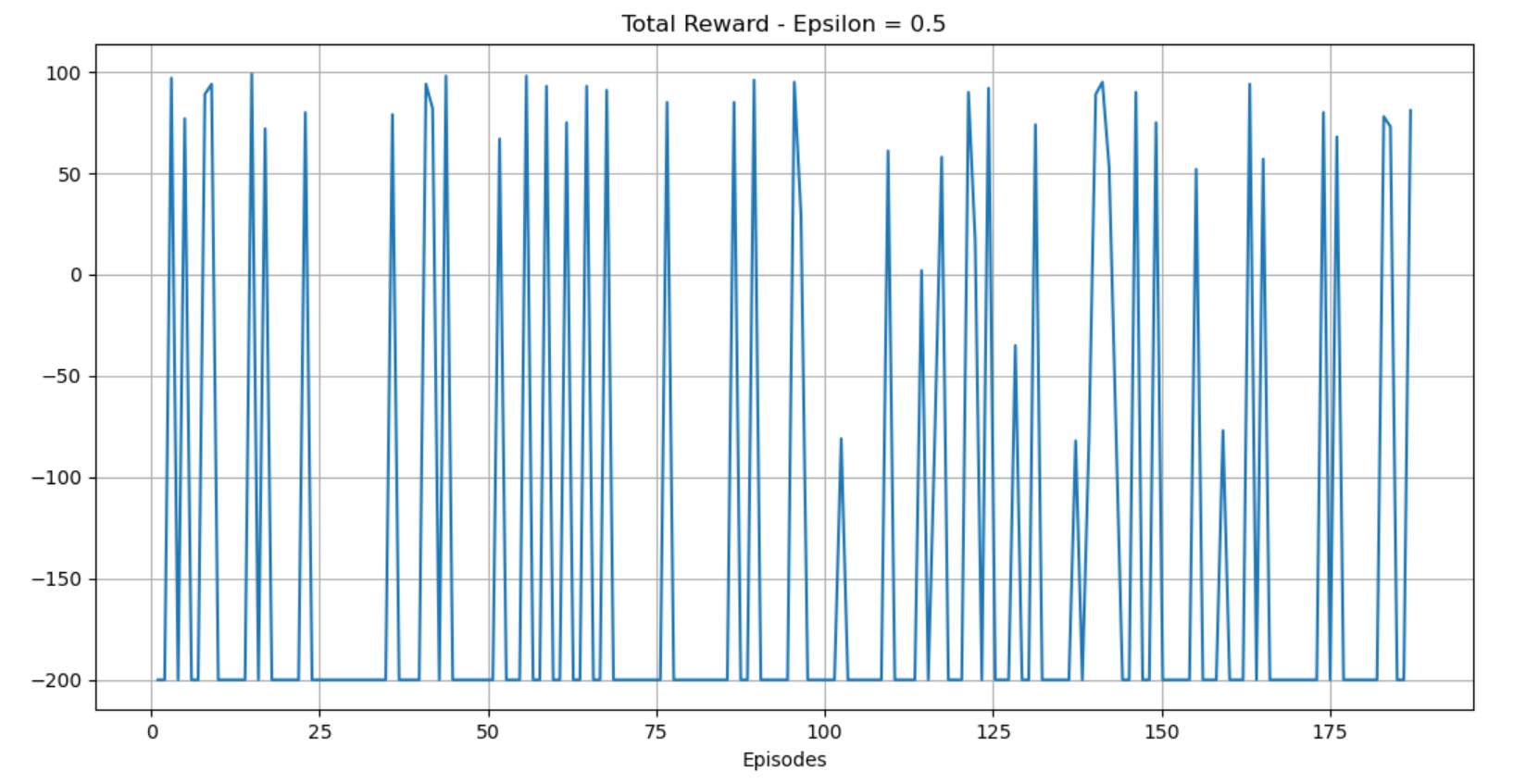




As seen, most episodes reached the top from early on in the training process (reward ).

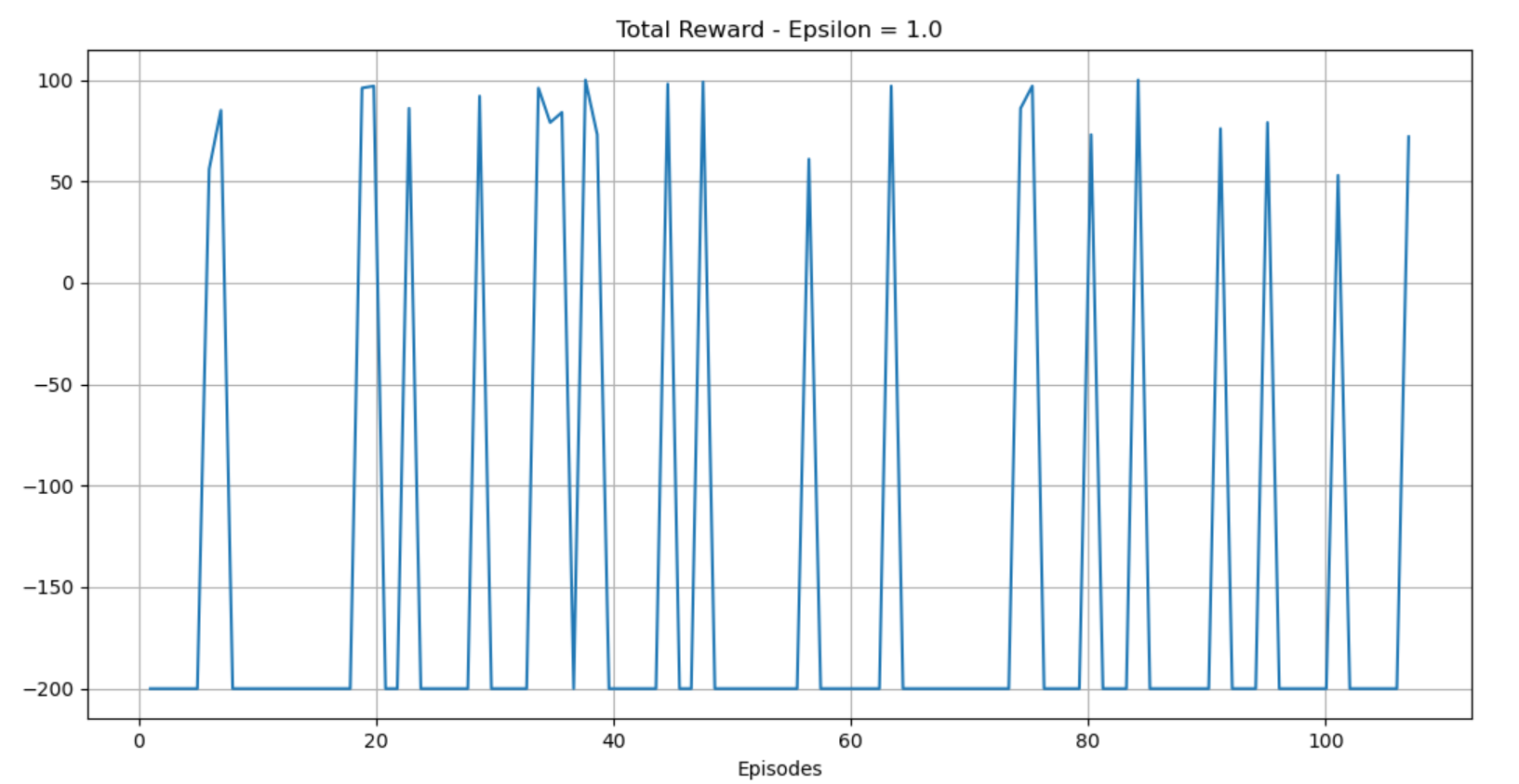


1. We ran the Q-Learning algorithm for 5 different values - 1.0, 0.75, 0.5, 0.3, 0.01. Plots of total rewards over episodes for each displayed below. The best value is due to its short training run time till finding solution-



Solved in 189 episodes.

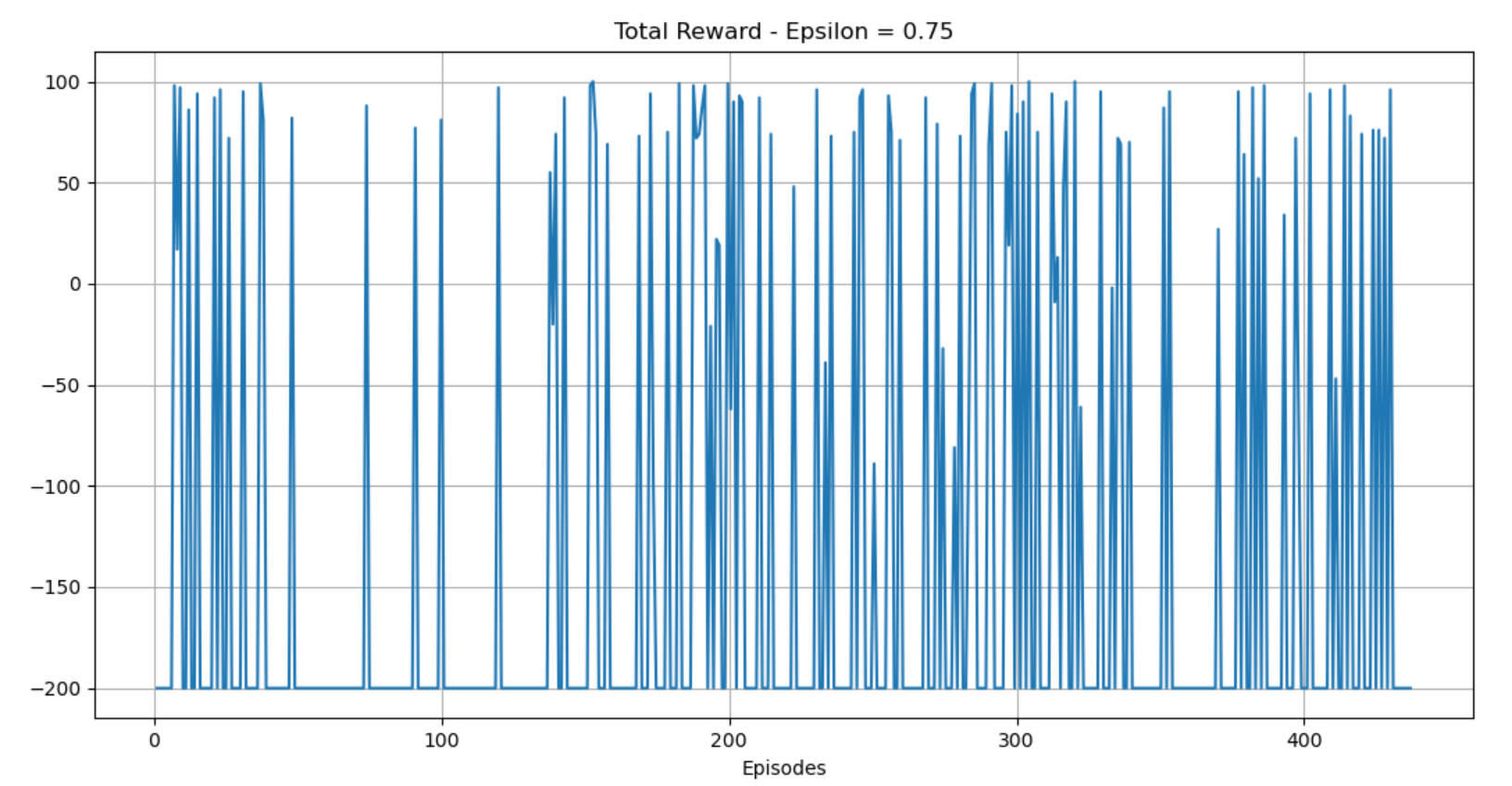
also found solution in short training time, as seen below-



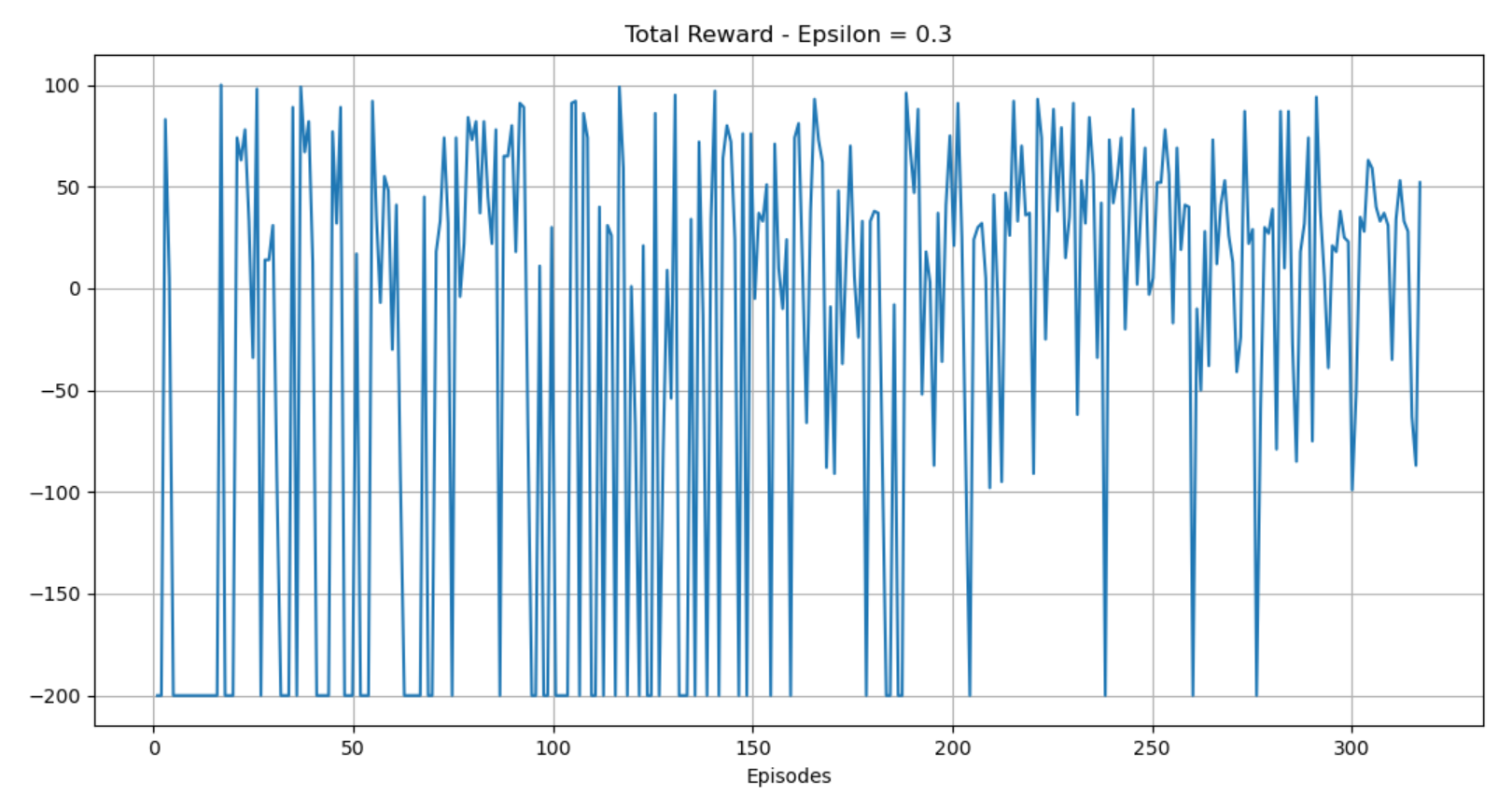
Solved in 109 episodes.

In oppose to the greedy policy, the reward for most episodes was (meaning most episodes did not reach the top). However, convergence to the solution took less episodes thanks to the exploration process. This could be explained due to relatively small state space. Therefore, the exploration method reaches goal state in short time. However, since the training process stops when reaching solution, the policy reached may be far from optimal – solution in up to 175 steps, while solution of less than 100 steps exists.

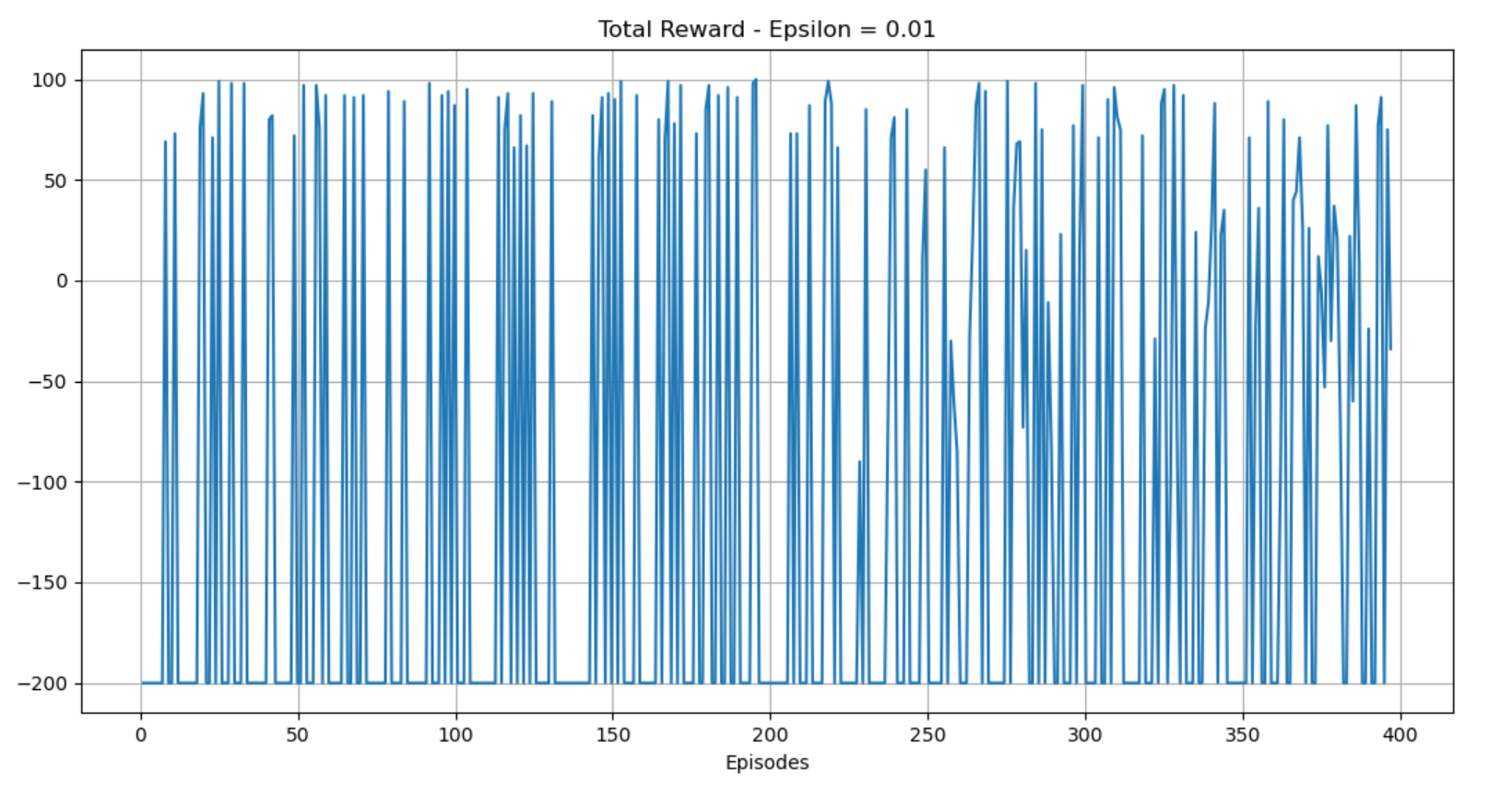
Other values-



Solved in 439 episodes.



Solved in 319 episodes.



Solved in 399 episodes.

**5 – Bonus**

The advantage of training while beginning only from the bottom hill state is that the bottom hill state is the state we calculate performance by. We are interested in solution for this state (or very close states of it). By limiting our agent, we achieve updating of the bottom hill state in each training iteration rather than “wasting” episodes on states which aren’t critical to finding solution.

The downfall of limiting our agent to start only from the bottom of the hill is that many states will remain unreached. One of the unreached states may be the goal state, holding the positive reward. If no positive reward is obtained, the training process is useless- all iterations will only decrease the values of the states (reward -1).

Our suggestion to overcoming this issue is to initially apply an exploration time period (), and then applying an -greedy policy as before with gradually decreasing values.

We implemented the following update rule for -

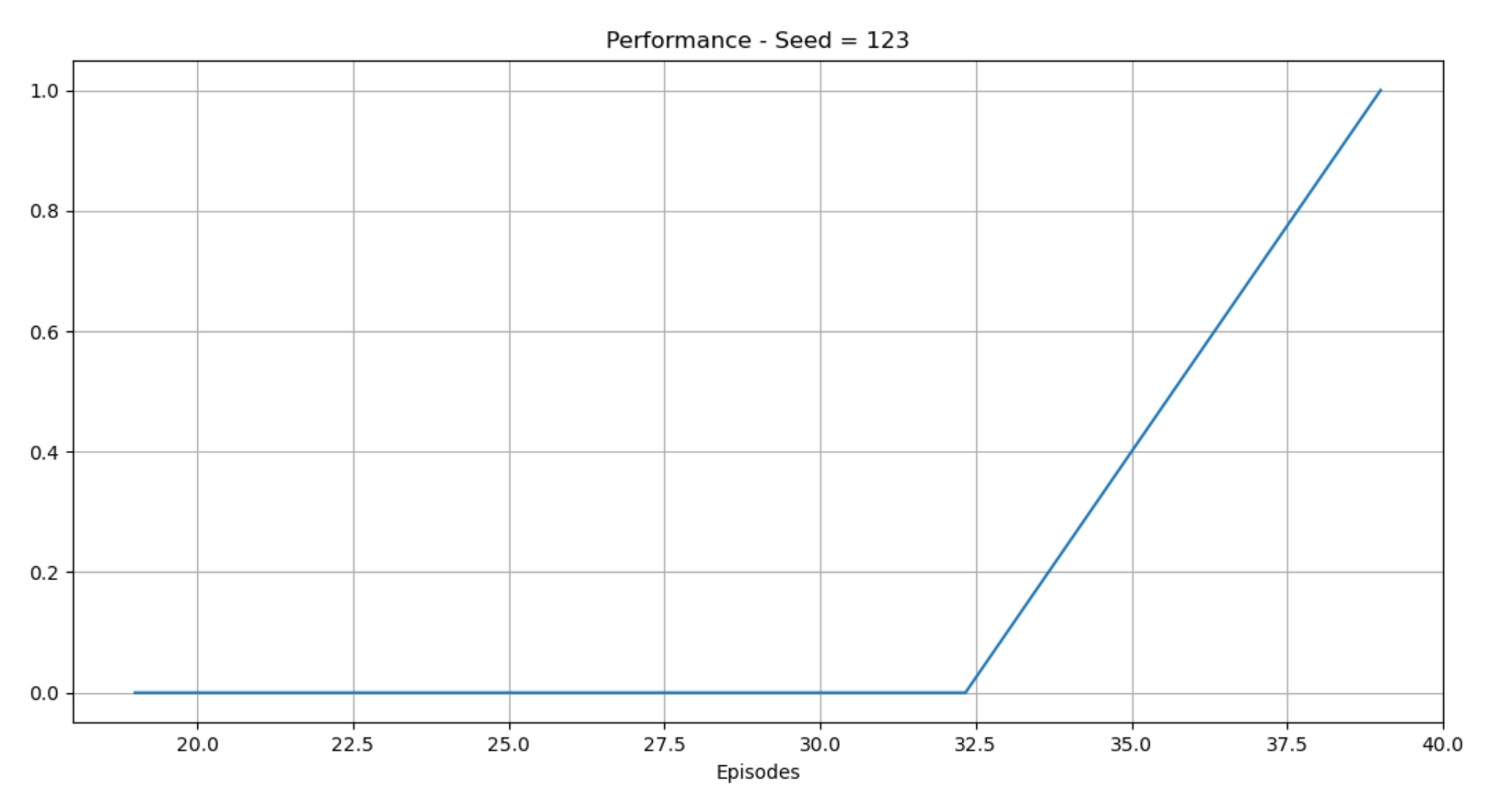
Episodes 0-9: initial (solely exploring)

Each tenth episode update –

When reached

In the settings - we reached solution after 39 episodes. This is the shortest time period of finding solution in compare to the methods of part 3.

Performance plot demonstrating our success -



As seen above solution was found after 39 episodes. We continued the training process after the solution was found for 300 episodes and got the following results:

