COMP 138 RL: Assignment 2

Luxi Liu

October 19, 2022

1 Introduction

Monte Carlo Methods are a genre of learning that does not require prior knowledge of the environment, instead, it only require experience. It is useful in many scenarios where it is easy to generate experience. Monte Carlo increments episode-by-episode, rather than step-by-step. Through general policy iteration, Monte Carlo methods sample and average returns for each state-action pair.

2 Problem Statement

For this project, we try to learn to cross the finish line for a racetrack. Given a track, we average the value of each state-action pair through iteration, after learning , we evaluate the result by starting from a random starting state (the starting state set is represented by a single row on the track), taking greedy action at each step, and evaluate the reward we get at the end of the episode.

For this project, the direction of moving at both x and y directions are restrained to be only positive, meaning we are not allowed to go backwards. At each state, velocity at both x and y direction can change by +1, -1, or 0. We refer to this change in velocity *delta*. Velocities are not allowed to be 0 at both directions, except for at the starting state. To ensure this, if a velocity change leads velocities at both directions to be 0, we keep generating random deltas until the velocities are not both 0s.

To clarify definitions:
A state here is a position on the track.
An action is a change in velocity.
An episode is defined as the race car starting from a random position from the start line, then take an action at each state, until either it finishes the track successfully, or went out of bound.

For evaluating performance, we choose to use the *average return for the best state-action pair at the starting state*. This makes sense in both successful and failed case: if we reach the goal in an episode, a better route would have less

steps, thus the return for fist state would be discounted less; if we failed i.e. the race car went out of bound, it is better to get closer to the goal than not, so the penalty would be discounted more.

3 Experiment

Two MC methods were used in comparison.

1. The basic MC ES where each episode is randomly generated
2. The On-Policy first-visit MC control using epsilon greedy

For each of these methods, we first run n episodes using the algorithm, then we run greedy policy on the learnt result, to evaluate how good is the "best action" learnt at each state.

To decrease result variation caused by randomness, for each number of episode, we run the algorithm 20 times around that number, and take the average. For example, to evaluate how a policy performs at around 1000 episodes, we ran the algorithm with 1000 episodes for 20 times, each time using a different random seed and starting from scratch, than take the average of the return for the best state-action pair at the starting state.

We set the rewards for successfully cross the finish line to be 10, and the penalty of going out of bound to be -100.

We represent the race track as a matrix, for example:

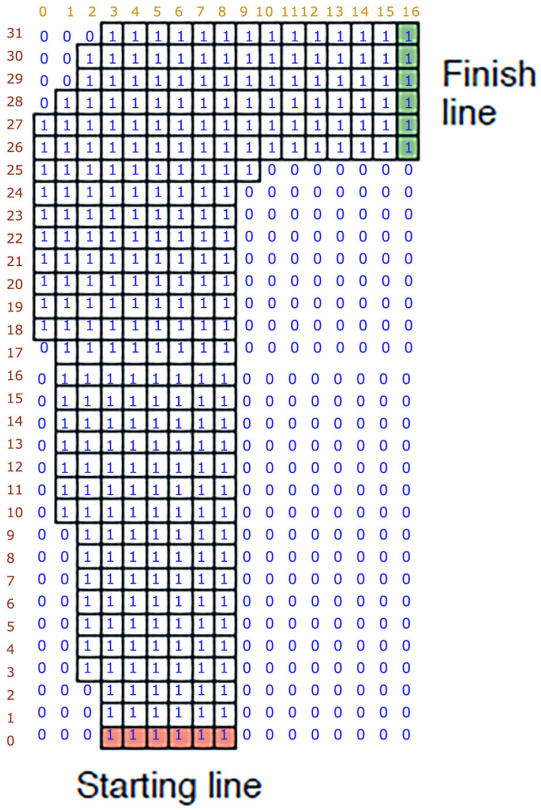
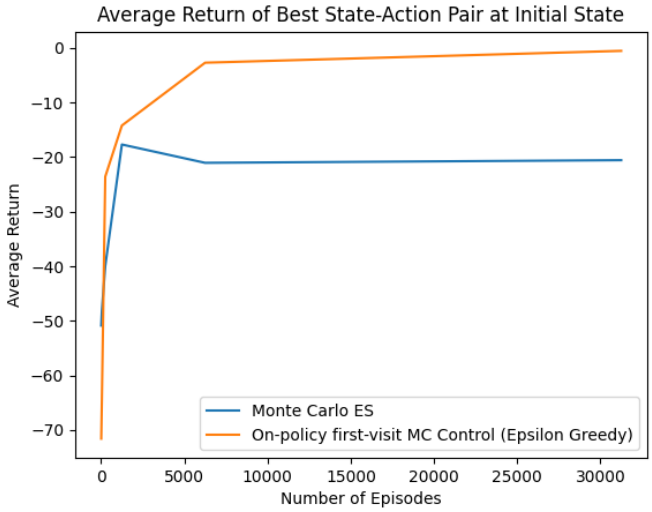


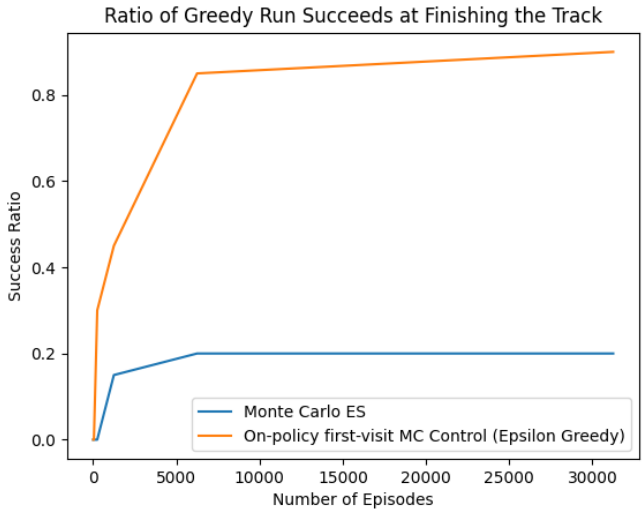
Figure 1: Racetrack 1

3.1 Without Noise

First we ran the two methods mentioned above without any noise. i.e. for method 1 the basic MC ES, delta is always randomly picked, for method 2 MC with epsilon greedy, we only use epsilon to decide whether a step should explore or exploit.

For episode number = [10, 50, 250, 1250, 6250, 31250] (each averages over 20 runs with different random seed), the result is as following:





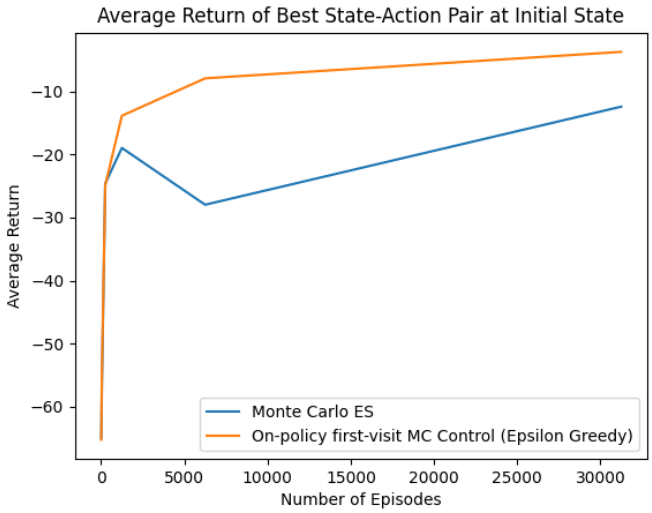
We see that Epsilon Greedy MC is much more efficient than the entirely random one, this might be due to that there are too many combinations of state-action pair, therefore for a relatively large track, it is very hard for the entirely random MC method to converge.

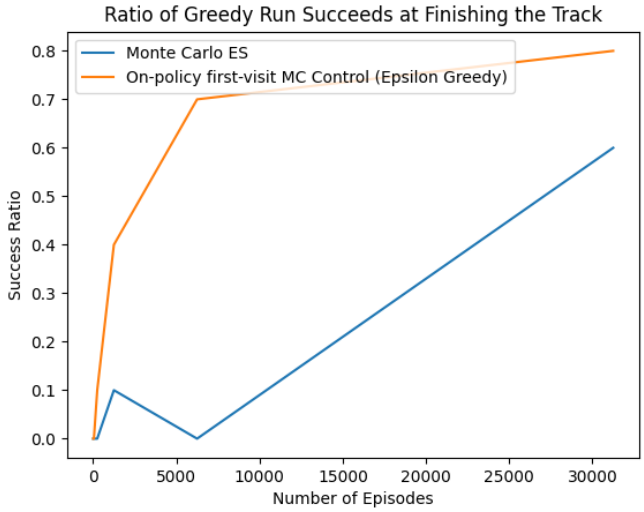
3.2 With Noise

3.2.1 Noise Probability 0.1

Next we introduce noise into our learning phase. With probability 0.1 at each time step the velocity increments are both zero.

For episode number = [10, 50, 250, 1250, 6250, 31250] (each averages over 20 runs with different random seed), the result is as following:





We see that with noise at 0.1, the learning slows down, we can compare success rate as an example:

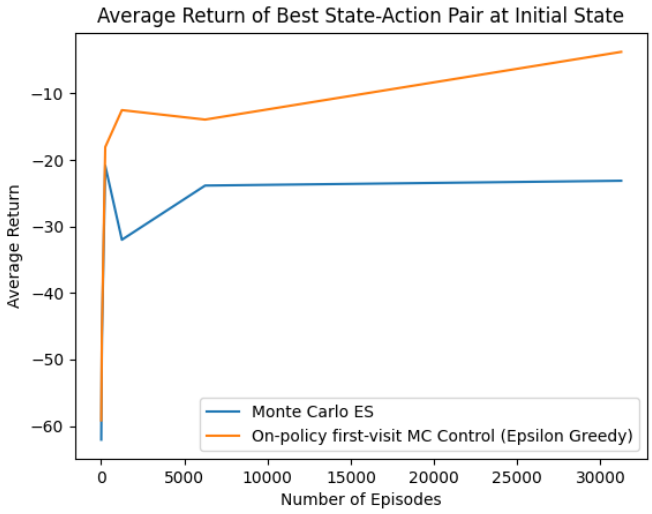
Episode Numbers
[10, 50, 250, 1250, 6250, 31250]

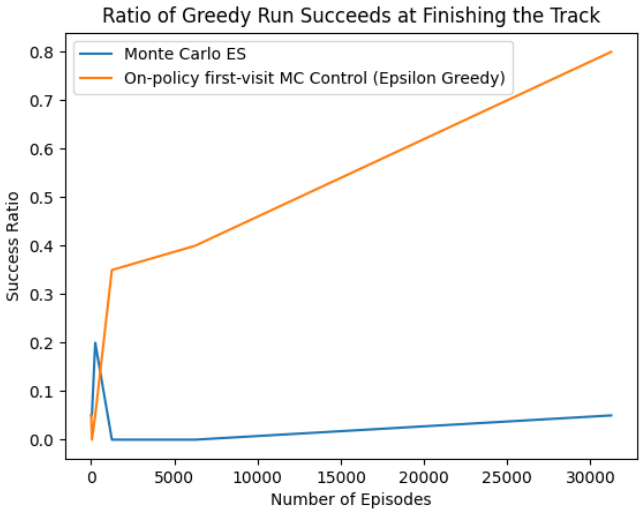
Without noise
greedy success rate list for Monte Carlo ES
[0. 0. 0. 0.15 0.2 0.2]
greedy success rate list for Monte Carlo Epsilon Greedy
[0. 0. 0.3 0.45 0.85 0.9]
Noise = 0.1
greedy success rate list for Monte Carlo ES
[0. 0. 0. 0.05 0. 0.5]
greedy success rate list for Monte Carlo Epsilon Greedy
[0. 0. 0.2 0.4 0.6 0.75]

In most cases success rates of the with noise runs are lower at the same episode number, compared to the version without noise. We also see a surprising surge in the random algorithm's performance at around 30,000 episodes. I suspect this is just a experiment variation due to randomness. If I have better computing power and longer time to work on this project, ideally I want to average over a lot more than 20 runs at each episode number.

3.2.2 Noise Probability 0.2

Next we try turning noise level a little higher.

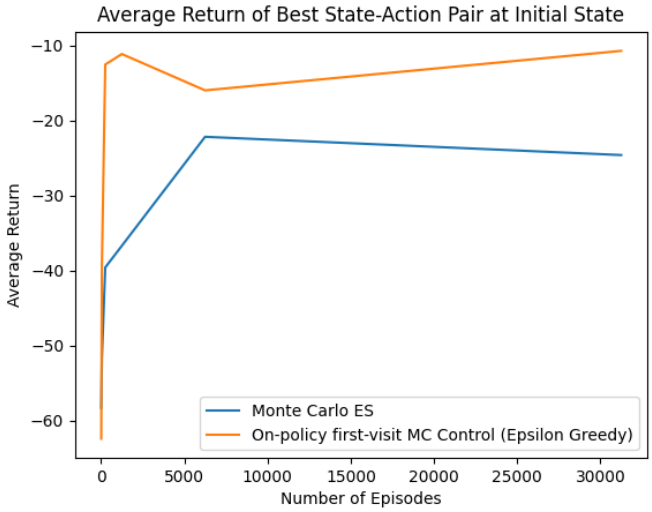


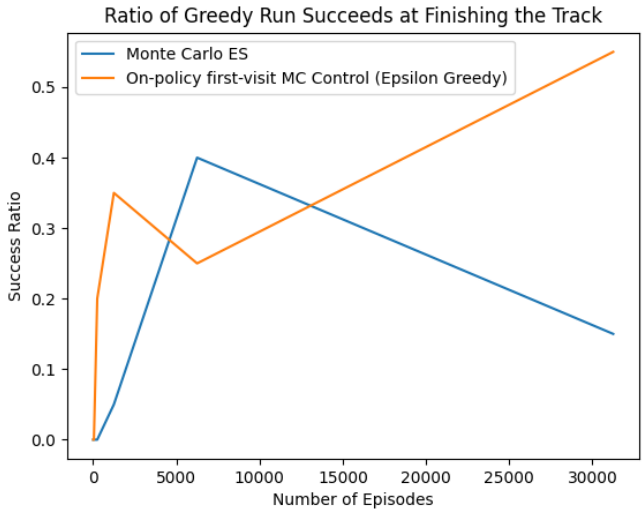


Again, the learning gets slower, but eventually also reaches high success rate (around 0.8) for Epsilon-Greedy MC Method.

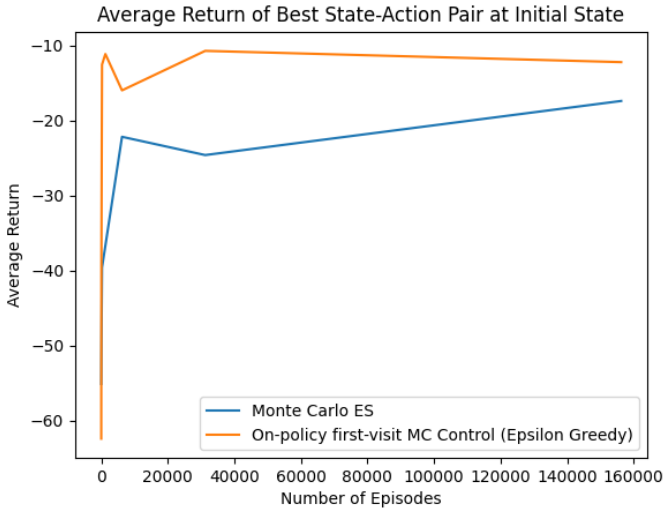
3.2.3 Noise Probability 0.3

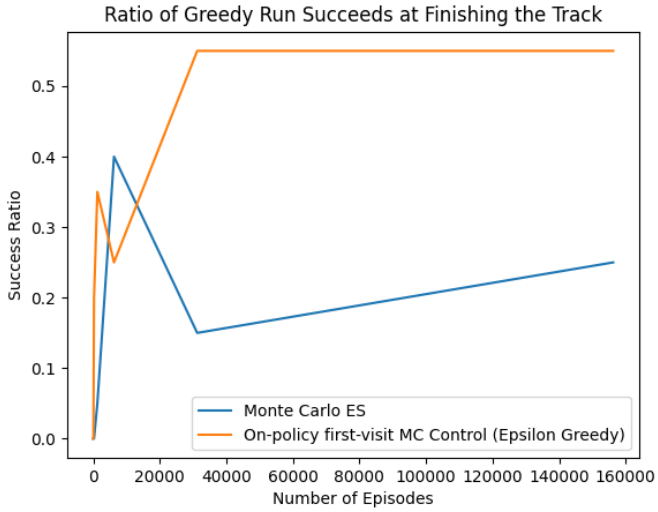
At noise level 0.3, we start to see significant drop in success rate and returns for Epsilon Greedy MC. This might be due to the noise level start to cause significant distractions in the learning process, it might take much longer episode to reach high success rate like the previous experiments.





Thus, we try longer episode. The following is the result from running episode number [10, 50, 250, 1250, 6250, 31250, 156250], each for 20 times.





It turns out that success rate and reward for epsilon greedy monte carlo do not converge to a higher number as I expected, it stayed relative the same after around 30000 episode, namely, the learning does not improve our policy after that point. I guess the noise level is too high for the algorithm to learn the optimal route.

Next, we demonstrate some sample trajectories for both methods with noise 0.1.

Episode number 10000, Sample Trajectory 1
Random MC trajectory:
state = (0, 7) best action = (1, 0) return = -38.742
state = (1, 7) best action = (1, 0) return = -43.047
state = (3, 7) best action = (1, 0) return = -47.830
state = (6, 7) best action = (1, 0) return = -53.144
state = (10, 7) best action = (-1, 0) return = -59.049
state = (13, 7) best action = (1, 0) return = -65.610
state = (17, 7) best action = (1, 0) return = -72.900
state = (22, 7) best action = (-1, 0) return = -81.000
state = (26, 7) best action = (0, 0) return = -90.000
state = (30, 7) best action = (0, 1) return = -100.000

Epsilon Greedy MC trajectory:
state = (0, 7) best action = (1, 0) return = 1.351
state = (1, 7) best action = (1, 0) return = 1.501
state = (3, 7) best action = (-1, 0) return = 1.668
state = (4, 7) best action = (1, 0) return = 1.853
state = (6, 7) best action = (0, 0) return = 2.059
state = (8, 7) best action = (-1, 0) return = 2.288
state = (9, 7) best action = (0, 0) return = 2.542
state = (10, 7) best action = (0, 0) return = 2.824
state = (11, 7) best action = (0, 0) return = 3.138
state = (12, 7) best action = (0, 0) return = 3.487
state = (13, 7) best action = (0, 0) return = 3.874
state = (14, 7) best action = (0, 0) return = 4.305
state = (15, 7) best action = (1, 0) return = 4.783
state = (17, 7) best action = (0, 0) return = 5.314
state = (19, 7) best action = (1, 0) return = 5.905
state = (22, 7) best action = (-1, 0) return = 6.561
state = (24, 7) best action = (-1, 1) return = 7.290
state = (25, 8) best action = (0, 1) return = 8.100
state = (26, 10) best action = (-1, 1) return = 9.000
state = (26, 13) best action = (1, 1) return = 10.000

Episode number 10000, Sample Trajectory 2
Random MC trajectory:
state = (0, 3) best action = (1, 0) return = -28.243
state = (1, 3) best action = (0, 0) return = -31.381
state = (2, 3) best action = (1, 0) return = -34.868
state = (4, 3) best action = (-1, 0) return = -38.742
state = (5, 3) best action = (1, 0) return = -43.047
state = (7, 3) best action = (-1, 0) return = -47.830

state = (8, 3) best action = (1, 1) return = -53.144
state = (10, 4) best action = (1, -1) return = -59.049
state = (13, 4) best action = (0, 0) return = -65.610
state = (16, 4) best action = (1, 0) return = -72.900
state = (20, 4) best action = (0, 0) return = -81.000
state = (24, 4) best action = (0, 1) return = -90.000
state = (28, 5) best action = (1, 1) return = -100.000

Epsilon Greedy MC trajectory:
state = (0, 4) best action = (1, 0) return = 2.059
state = (1, 4) best action = (0, 0) return = 2.288
state = (2, 4) best action = (1, 0) return = 2.542
state = (4, 4) best action = (1, 0) return = 2.824
state = (7, 4) best action = (0, 1) return = 3.138
state = (10, 5) best action = (0, -1) return = 3.487
state = (13, 5) best action = (1, 0) return = 3.874
state = (17, 5) best action = (-1, 0) return = 4.305
state = (20, 5) best action = (-1, 0) return = 4.783
state = (22, 5) best action = (-1, 0) return = 5.314
state = (23, 5) best action = (0, 0) return = 5.905
state = (24, 5) best action = (1, 1) return = 6.561
state = (26, 6) best action = (-1, 1) return = 7.290
state = (27, 8) best action = (-1, 1) return = 8.100
state = (27, 11) best action = (0, 0) return = 9.000
state = (27, 14) best action = (0, 1) return = 10.000

Episode number 30000, Sample Trajectory 1
Random MC trajectory:
state = (0, 4) best action = (1, 0) return = -28.243
state = (1, 4) best action = (1, 0) return = -31.381
state = (3, 4) best action = (0, 0) return = -34.868
state = (5, 4) best action = (-1, 0) return = -38.742
state = (6, 4) best action = (1, 0) return = -43.047
state = (8, 4) best action = (-1, 0) return = -47.830
state = (9, 4) best action = (0, 1) return = -53.144
state = (9, 5) best action = (0, 1) return = -59.049
state = (9, 6) best action = (1, 1) return = -65.610
state = (10, 7) best action = (0, 0) return = -72.900
state = (11, 8) best action = (1, -1) return = -81.000
state = (13, 8) best action = (-1, 0) return = -90.000
state = (14, 8) best action = (0, 1) return = -100.000

Epsilon Greedy MC trajectory:
state = (0, 3) best action = (1, 0) return = 0.798
state = (1, 3) best action = (1, 0) return = 0.886

state = (3, 3) best action = (0, 0) return = 0.985
state = (5, 3) best action = (-1, 0) return = 1.094
state = (6, 3) best action = (0, 0) return = 1.216
state = (7, 3) best action = (1, 1) return = 1.351
state = (9, 4) best action = (1, 0) return = 1.501
state = (12, 5) best action = (0, -1) return = 1.668
state = (15, 5) best action = (-1, 0) return = 1.853
state = (17, 5) best action = (-1, 0) return = 2.059
state = (18, 5) best action = (1, 0) return = 2.288
state = (20, 5) best action = (0, 0) return = 2.542
state = (22, 5) best action = (-1, 0) return = 2.824
state = (23, 5) best action = (-1, 1) return = 3.138
state = (23, 6) best action = (1, -1) return = 3.487
state = (24, 6) best action = (0, 0) return = 3.874
state = (25, 6) best action = (0, 0) return = 4.305
state = (26, 6) best action = (-1, 1) return = 4.783
state = (26, 7) best action = (1, -1) return = 5.314
state = (27, 7) best action = (-1, 1) return = 5.905
state = (27, 8) best action = (0, 0) return = 6.561
state = (27, 9) best action = (0, 0) return = 7.290
state = (27, 10) best action = (0, 0) return = 8.100
state = (27, 11) best action = (0, 1) return = 9.000
state = (27, 13) best action = (0, 1) return = 10.000

Episode number 30000, Sample Trajectory 2
Random MC trajectory:
state = (0, 3) best action = (1, 0) return = 1.216
state = (1, 3) best action = (1, 0) return = 1.351
state = (3, 3) best action = (1, 0) return = 1.501
state = (6, 3) best action = (0, 0) return = 1.668
state = (9, 3) best action = (1, 0) return = 1.853
state = (13, 3) best action = (-1, 0) return = 2.059
state = (16, 3) best action = (-1, 0) return = 2.288
state = (18, 3) best action = (-1, 1) return = 2.542
state = (19, 4) best action = (-1, 0) return = 2.824
state = (19, 5) best action = (1, 0) return = 3.138
state = (20, 5) best action = (0, 1) return = 3.487
state = (20, 6) best action = (1, 0) return = 3.874
state = (21, 7) best action = (1, 1) return = 4.305
state = (22, 8) best action = (0, -1) return = 4.783
state = (23, 8) best action = (1, 0) return = 5.314
state = (24, 8) best action = (1, 1) return = 5.905
state = (26, 9) best action = (0, 0) return = 6.561
state = (28, 10) best action = (-1, 1) return = 7.290
state = (29, 12) best action = (-1, -1) return = 8.100
state = (29, 13) best action = (1, 1) return = 9.000

state = (30, 15) best action = (0, 0) return = 10.000

Epsilon Greedy MC trajectory:
state = (0, 5) best action = (1, 0) return = 2.288
state = (1, 5) best action = (1, 0) return = 2.542
state = (3, 5) best action = (-1, 0) return = 2.824
state = (4, 5) best action = (1, 0) return = 3.138
state = (6, 5) best action = (1, 0) return = 3.487
state = (9, 5) best action = (1, 0) return = 3.874
state = (13, 5) best action = (0, 0) return = 4.305
state = (17, 5) best action = (-1, 0) return = 4.783
state = (20, 5) best action = (0, 0) return = 5.314
state = (23, 5) best action = (-1, 1) return = 5.905
state = (25, 6) best action = (0, 0) return = 6.561
state = (27, 7) best action = (-1, 1) return = 7.290
state = (28, 9) best action = (0, 1) return = 8.100
state = (29, 12) best action = (-1, 0) return = 9.000
state = (29, 15) best action = (0, 1) return = 10.000

4 Bigger Track

Next, we try the experiment on a bigger track:

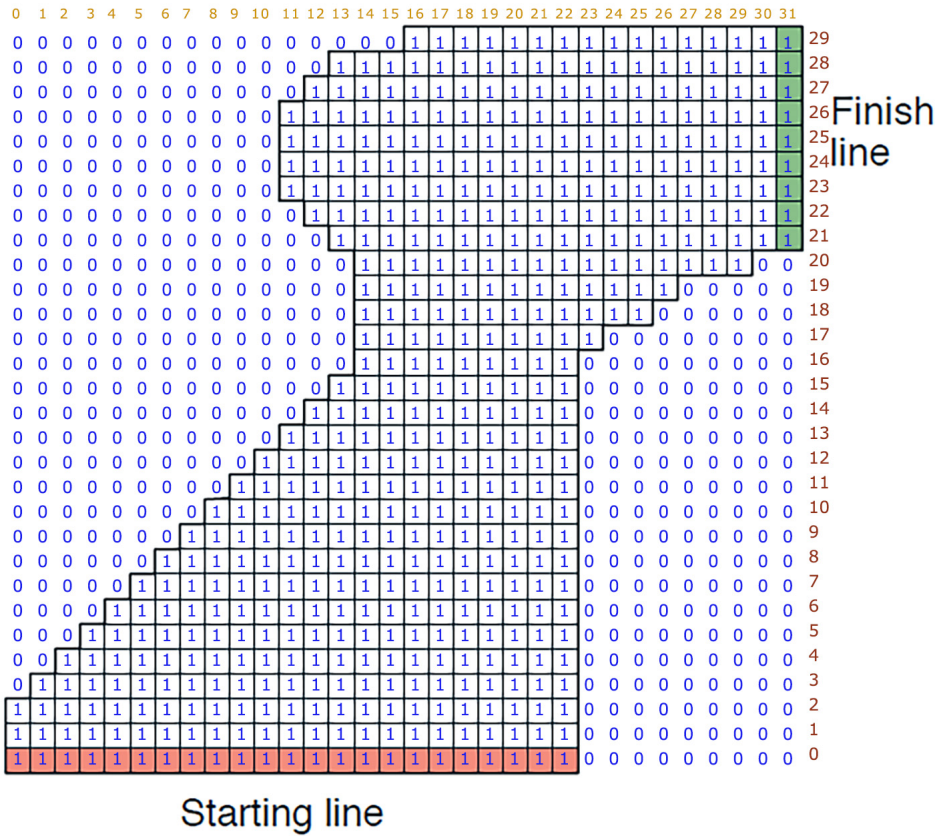
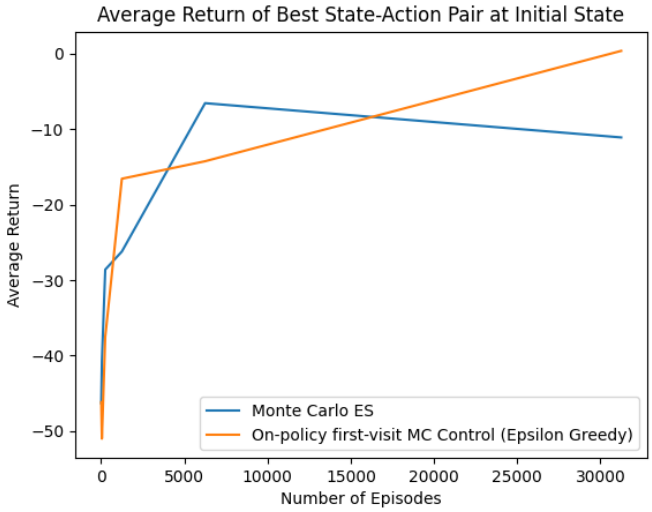
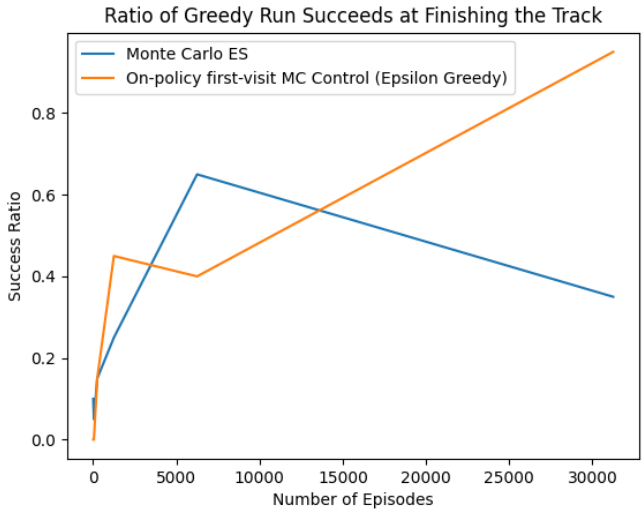


Figure 2: Racetrack 2

Again, we run episode number = [10, 50, 250, 1250, 6250, 31250] (each averages over 20 runs with different random seed), with noise set to 0.1. The result is as following:





Although the shape of the graph we get is different from track1, we also achieve around 0.95 success rate once the episode number is large enough (around 30,000). It looks like a larger track took longer to learn, for instance, the smaller track1 reaches 0.7 accuracy around 6000 episode, but it took much longer for the bigger track2.

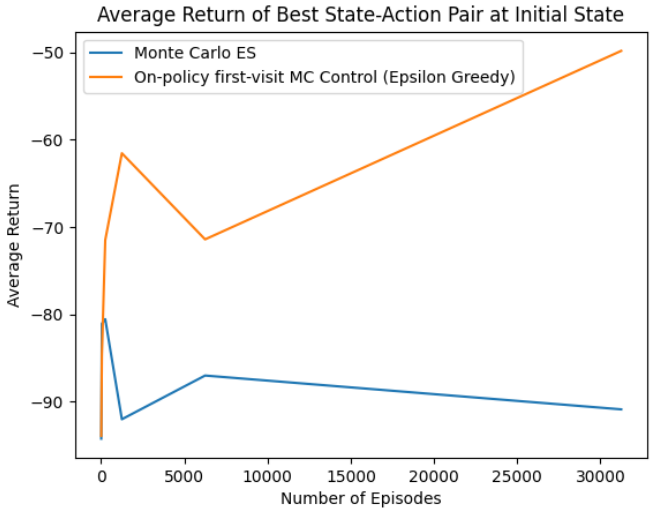
The rest of the experiment is very similar to the result of track1, therefore I will not discuss redundant content here.

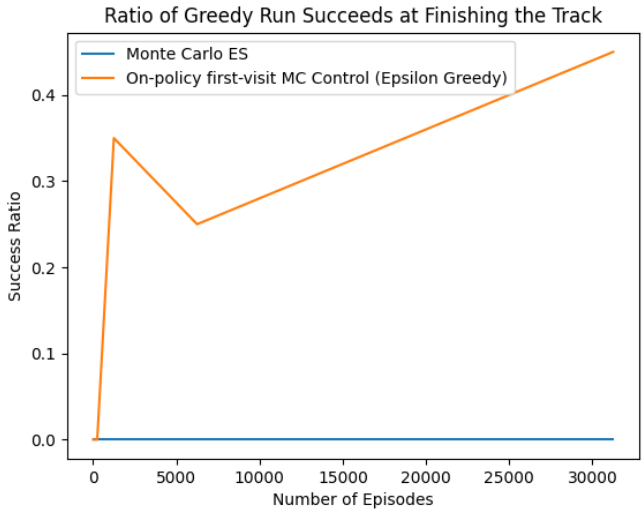
5 Extreme Tracks

If the track or part of the track has very narrow zones, I suspect the entirely random MC method will do even worse, since this kind of "extreme" track is less lenient towards explore, because there is often only one or very few "correct action".

First we try the following narrow track:

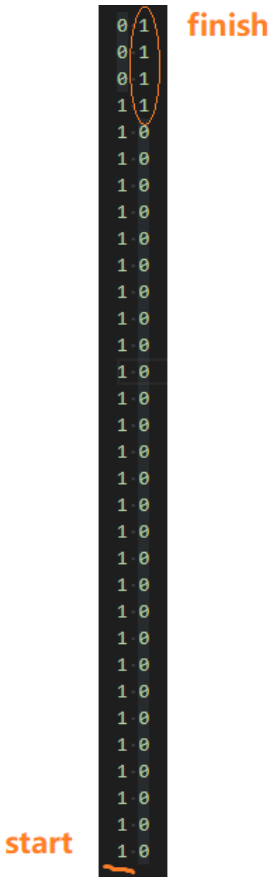
Result is as following:



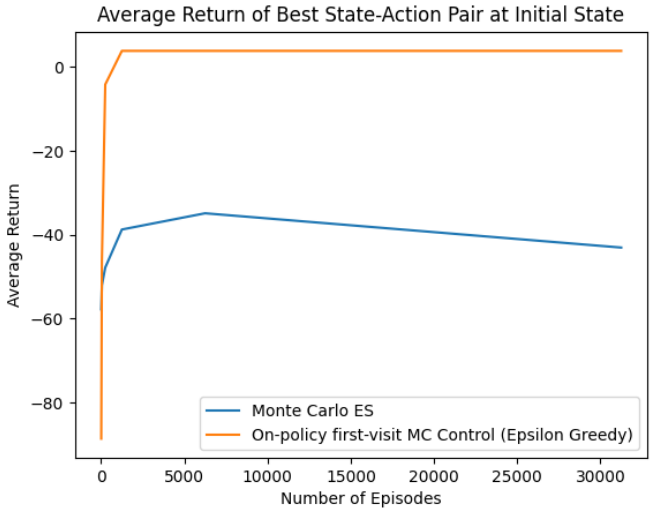


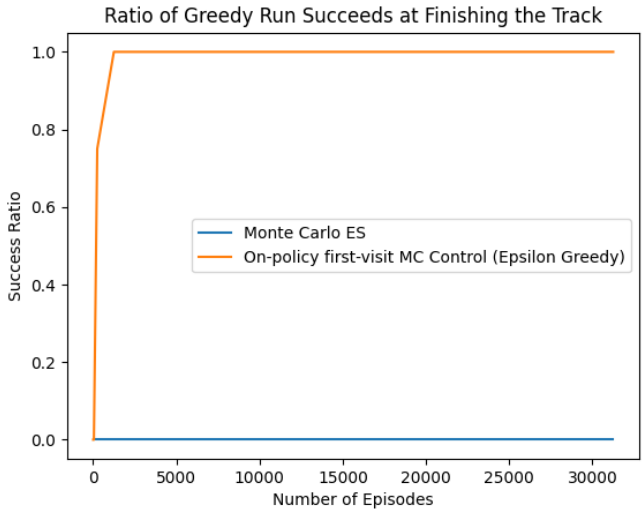
As expected, entirely random MC did very poorly, in fact, the success rate for finishing the track stays at 0.

To confirm our intuition, let's try another narrow track in the vertical shape:



Result is as following:





It confirms our hypothesis that Monte Carlo ES does very poorly with "extreme tracks" because there is often one or very little "correct actions" at each state, if we don't accumulate such "correct actions" greedily, it is very hard for the algorithm to learn further steps.

6 Conclusion

We used two different Monte Carlo methods to learn the racetrack. The epsilon greedy version is much more efficient, even though in theory both should eventually converge. We learned that the noise level affect the speed of learning, if noise level is high enough, it seems that the epsilon greedy method may never find the optimal path.

Monte Carlo is easy to start in many cases including this racetrack problem, since we do not need to build a model first, instead, all "best actions" are learned through experience. This is the strength for Monte Carlo methods, but can also be not so efficient sometime. In the example with track1 and noise level 0.1, the epsilon greedy method took around 6000 episodes to get the success rate of 0.6, and around 30000 episode for success rate to get to 0.75. The entirely random one will presumably took much longer. In general, it could take a long time for Monte Carlo method to converge, especially the entirely random one, if the state-action space is huge.

But I also think that if we just want a relatively ok reward, both methods went from around a really low reward (around -70 to -50) to -20 reward fairly quick (around 1000 episodes).

7 Reference

The experiments and graph drawing referenced the Sutton and Barto Reinforcement Learning: An Introduction textbook.