# ASSIGNMENT #4 (Model-Free)

**Alireza Mostafizi**

---

I coded MDP Planning Algorithm in Python 2.7. Attached to the email you can find three python codes. One is the MDP maker for the parking lot domain (MDP Maker.py) which you can use to create a MDP, using the probabilities and rewards, to feed the Q-learning algorithm or the policy simulator with. The other one is the code for Part III that simulate the learning environment (HW4.py). I also included calculating optimal policy with value iteration in this part of code to see how the final learned policy differ from the optimal one. The other code is the policy simulator for Part II which you can feed with the MDP and see the value of the policy. Please keep in mind that to have the code running you have to have "NumPy" installed for matrices operations. In the next sections I briefly elaborate the details.

## *Part I: Designing the MDPs*

Here is a brief description of the MDP for the parking lot domain which I mostly copied from HW2, except some minor changes:

**States**

- If we assume n parking spots for each row, the MDP has $8n + 2$ states. Like suggested in the assignment, I considered $8n$ states, plus a "FINAL" state and a "NO" state.
    - The "FINAL" state you can get to only if the car is parked or had an accident, and it acts as a terminal state. It has +0.01 reward, and once you get there, you will remain there.
    - The "NO" state is the state where you get to when you make an inappropriate decision. For example, in case the car is parked and the "DRIVE" decision is made, you will get to "NO" state which has -0.01 reward, and once you get there, you will remain there.

To depict how the states are mapped to the numbers, let's assume $n = 5$:

| | State Numbers | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| Cell | A5 | | | | A4 | | | | A3 | | | | A2 | | | | A1 | | | |
| Occupancy | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O |
| Parked | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P |

| | State Numbers | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| Cell | B1 | | | | B2 | | | | B3 | | | | B4 | | | | B5 | | | |
| Occupancy | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O |
| Parked | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P |

*NO = NOT OCCUPIED  O = OCCUPIED  NP= NOT PARKED  P = PARKED*

*State 41 is the FINAL state, and state 42 is the NO state.*

**Occupancy Probability**

There are some simple assumptions I made to prepare the MDP.

- The probability of ADA spot being empty is entered in the program.
- The closer the spot is to the store, the lower the probability of the spot being empty.
    - Therefore, the program asks for the probability of closest spot ($A_2$ and $B_2$) being empty (AKA Min probability) and probability of the farthest spot ($A_n$ and $B_n$) being empty (AKA Max Probability)
    - I assumed that the probability of other spots not being occupied linearly decrease from max to min probability, based on their distance to the store.

**Rewards**

Here are assumption about the rewards. Most of them are suggested by the assignment.

- There is a fine for parking in ADA spot, which you can enter in program
- There is a huge cost for accident, which you can enter in program
- There is a relatively small cost for driving from each cell to the next one, which you can enter in the program
- You can enter the reward of parking in the closest spot ($A_2$ and $B_2$), known as Max reward, and parking in the farthest spot ($A_n$ and $B_n$), known as Min reward.
    - The other rewards are drawn from linear interpolation between max reward and min reward, based on their distance to the store.

**Transition Matrices**

As suggested by the assignment, I considered three actions:

- 1. Drive to the next spot
    - ACTION 1 in the program
    - Cannot be taken when the car is parked
    - Will take you to the next cell considering the circulation pattern
        - Note that there are 4 states for each cell. This action will take you to the next cell's state with NO PARKED condition. Ending up in OCCUPIED or NOT OCCUPIED state is based on the probability of occupancy.
- 2. Park
    - ACTION 2 in the program
    - Cannot be taken when the car is parked
    - Will take you from NOT PARKED state in each cell to the PARKED state in the same cell.
        - The probability of ending up in OCCUPIED or NOT OCCUPIED condition is based on the probability of occupancy.
- 3. Exit
    - ACTION 3 in the program
    - Cannot be taken when the car is not parked
    - And If the car is parked (or had an accident), the next action will definitely be the exit

**Procedure**

I created the general-purpose simulator in which you can simulate a learning agent. Basically, what you have to do is to create the corresponding MDP for the parking lot through "MDP Maker.py," using the parameters and rewards of interest. Then you can run the algorithm with mentioned MDP to see how the learning agent comes up with the optimal policy. To compare the policy to the optimal one, I included the planner from HW2 which calculates the optimal policy using value iteration. To test the results, I considered two MDPs as following:

NOTE: MDPs' probabilities and rewards are the same as the ones I used for HW2

**First MDP Description**

```
N (>3): 10
COST of Driving (+): 5
COST of Accident (+): 800
FINE for ADA Spot (+): 200
Probability of ADA spot NOT being occupied (0-1): 0.05
Reward for the closest spots (NOT THE ADA) (+): 100
Reward for the farthest spots (+): 10
Probability of the closest sport NOT being occupied (0-1): 0.05
Probability of the farthest spot NOT being occupied (0-1): 0.85
Output File Name: Output1
MDP input generated: Output1.txt
```

**Second MDP Description**

```
N (>3): 10
COST of Driving (+): 5
COST of Accident (+): 800
FINE for ADA Spot (+): 200
Probability of ADA spot NOT being occupied (0-1): 0.05
Reward for the closest spots (NOT THE ADA) (+): 30
Reward for the farthest spots (+): 30
Probability of the closest sport NOT being occupied (0-1): 0.5
Probability of the farthest spot NOT being occupied (0-1): 0.5
Output File Name: Output2
MDP input generated: Output2.txt
```

As you can see the difference between these two parking lot MDPs is that in the first one, closer spots have higher rewards and lower probabilities to be empty. However, in the second one, all the spots have the same reward and the same probabilities. Other than that, fines for accident and parking in ADA spot and cost of driving is the same. Thus, we should expect that in the second scenario, optimal policy is to park wherever is empty. On the other hand, in the first one, agent should learn how to balance the cost of driving and the reward of the parking.

The following Tables show the optimal policy through value iteration algorithm for these two MDPs.

### First MDP Optimal Policy

| A10 | | | | A9 | | | | A8 | | | | A7 | | | | A6 | | | | A5 | | | | A4 | | | | A3 | | | | A2 | | | | A1 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O |
| NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P |
| 1 | 3 | 1 | 3 | 1 | 3 | 1 | 3 | 1 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 1 | 3 | 1 | 3 |

| B1 | | | | B2 | | | | B3 | | | | B4 | | | | B5 | | | | B6 | | | | B7 | | | | B8 | | | | B9 | | | | B10 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O |
| NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P |
| 1 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 1 | 3 | 1 | 3 |

### Second MDP Optimal Policy

| A10 | | | | A9 | | | | A8 | | | | A7 | | | | A6 | | | | A5 | | | | A4 | | | | A3 | | | | A2 | | | | A1 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O |
| NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P |
| 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 1 | 3 | 1 | 3 |

| B1 | | | | B2 | | | | B3 | | | | B4 | | | | B5 | | | | B6 | | | | B7 | | | | B8 | | | | B9 | | | | B10 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O | NO | NO | O | O |
| NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P | NP | P |
| 1 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 3 |

*NO = NOT OCCUPIED  O = OCCUPIED  NP= NOT PARKED  P = PARKED*

As it is shown in the above tables, optimal policy for these two MDPs are different. As expected, the second optimal policy is to park wherever you can. However, in the first one, it is not optimal to park on A10, A9, A8, and B10 since they are too far from the store and they have low rewards.

## *Part II: Policy Simulation*

The following table shows the performance of these basic policies for both of the MDPs mentioned in Part I. For the policies which has probabilistic actions, you can enter the value in the program interface. Here are the policies:

- **POLICY 1**: Select PARK action with probability $p$ and DRIVE with probability of $1 - p$
- **POLICY 2**: Select DRIVE whenever the spot is occupied. Otherwise, PARK with probability of $p$ and DRIVE with probability of $p - 1$

To improve the **first** MDP's policy, I manipulated the second policy as following:

- **POLICY 3**: Select DRIVE whenever the spot is occupied. Otherwise, PARK with probability of $p(s)$ and DRIVE with probability of $p(s) - 1$
    - $p(s)$ in each state is the probability of the next spot being empty

To improve the **second** MDP's policy, since the rewards and the probabilities for the spots are exactly the same, above policies are not good. The best policy would be:

- **POLICY 4**: Definitely PARK in the first spot you find.

Here are the performance measurements for these policies (Average rewards) calculated with 1000 trials for each policy, and 50 actions for each trial since generally after 50 actions it gets to the terminal state.

| Average Rewards (1000 trials) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Policies | | | | | | | |
| | Policy 1 | | | Policy 2 | | | Policy 3 | Policy 4 |
| | $p = 0.25$ | $p = 0.5$ | $p = 0.75$ | $p = 0.25$ | $p = 0.5$ | $p = 0.75$ | | |
| MDP 1 | -324 | -196 | -142 | -6.4 | 12.3 | 13.8 | 15.4 | 11.1 |
| MDP 2 | -413 | -380 | -387 | -8.4 | 13.7 | 21.4 | 12.4 | 24.9 |

As it is obvious in the table above, policy 1 is very trivial, therefore the rewards are extremely negative since it is probable to end up in an accident. Policy 2 alleviate this problem, and the rewards are higher. The higher the probability of getting parked sooner, the higher the rewards are. Policy 3 is better than the other two since it take the probability of the next spot being empty into consideration, and since MDP 3 has different probabilities for each spot, it improves the performance of that MDP the best. Policy 4 is made specifically for MDP 2 and it improves the performance of MDP 2 the best.

## *Part III: Reinforcement Learning*

As I mentioned before, I implemented a model-free reinforcement learning agent through Q-learning. For all the purposes in this assignment, $\beta$ is considered to be 0.99. I tried two different $\alpha$, 0.1 and 0.5, to see how learning curve will change.

Just like Part II, the value of policy is calculated with 1000 trials. Each trial goes on for 50 steps where generally the agent gets to the terminal state.

The process of learning is simulated up to 10000 trials at most.

**Explore/Exploit Strategy:**

- Epsilon-Greedy:

  This was the first strategy I implemented, but it turned out that it is not very functional in this domain. I compared the final learned policy with the optimal policy after 10000 iterations, and there was a huge difference between policies, especially at the end of the second row, which did make sense. Reaching the final states in this domain requires lots of "drive" actions, and since sometimes the greedy action is to drive, and some other times the greedy action is to park, it is not practical to implement epsilon greedy to explore those final states.

- Uniform:
  The second strategy I implemented was a naïve agent. It basically tries to seek all the possible actions, but again, with this domain and this strategy, first row was almost thoroughly observed, however, the second row and final states have not been explored at all. This strategy did not converge to the optimal policy after 10000 trials either.

- Drive-More:
  So, I came up with this exploring strategy, and it perfectly converge to the optimal policy after some iterations. The basic idea is that:
  - In each state, agents are willing to drive to the next spot with probability of 0.7.
  - Otherwise, they will choose the action that they have tried the least before.

  This strategy seems to work fine in terms of exploring and converging to the optimal policy.

Since the last strategy works the best, I just included the results of that. So all the following learning curves are based on that exploring strategy.

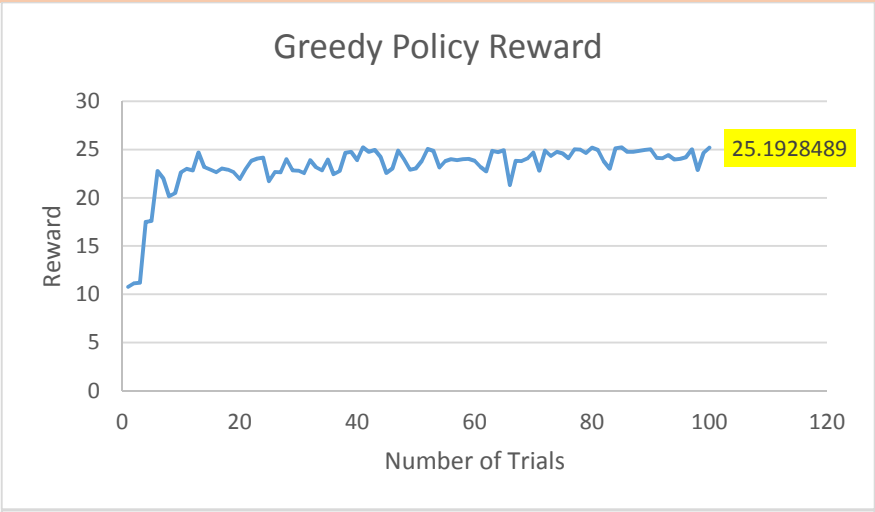
- ***The learned policy in high number of trials (around 10000 in total) is almost the same as optimal policy calculated using value iteration presented in the first part.***

- ***For the low number of trials, learned policies for the first row of parking spots are pretty close to the optimal policy, however, as the algorithm fails to explore the second row, for that part, it is not close to the optimal policy.***

The following pages include the learning curves for two different mentioned MDPs (in Part I), with different number of trials.

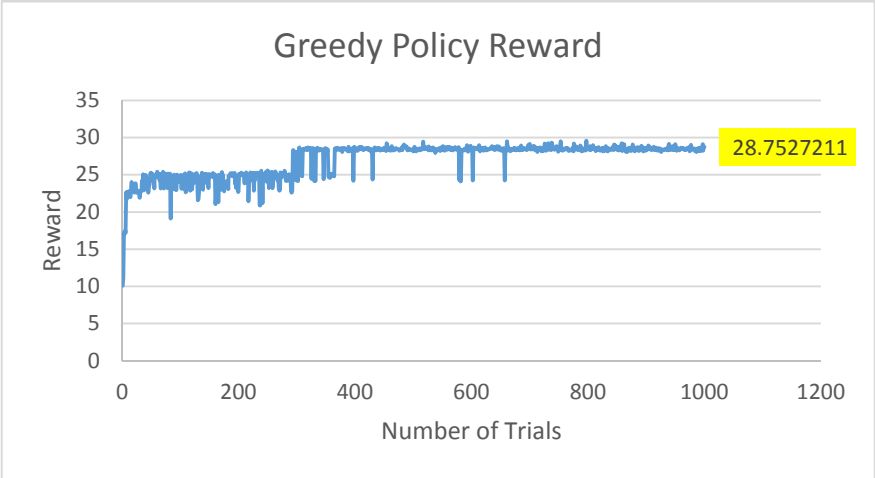
# MDP 1

$\alpha = 0.1$



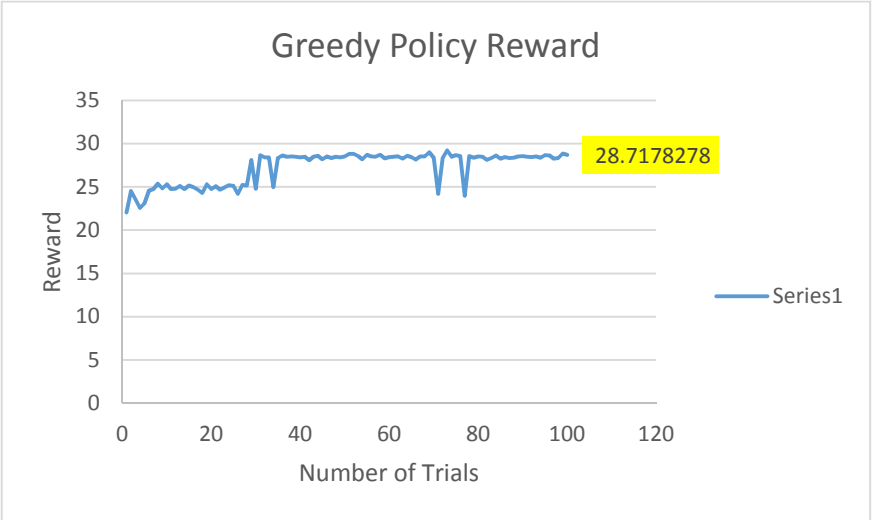

**N = 100**

- Due to low number of trials (100*100) the algorithm did not converge to the value of optimal policy (28.7)




**N = 100**

- Since the number of trials in total is high (1000*100), the greedy policy converged to the optimal policy and the final value is 28.7.
- The shape of the learning curve was almost as we could expect

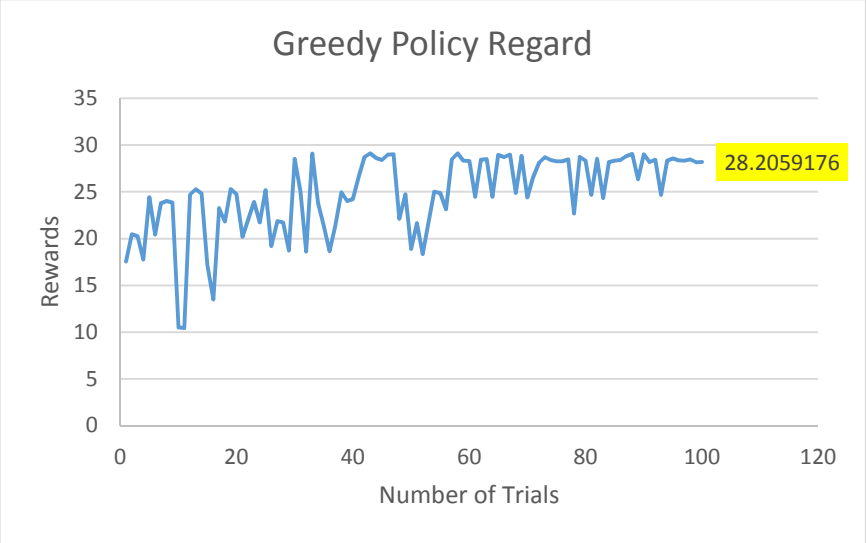



**N = 1000**

- Large number of trials resulted in a policy fairly close to the optimal policy.

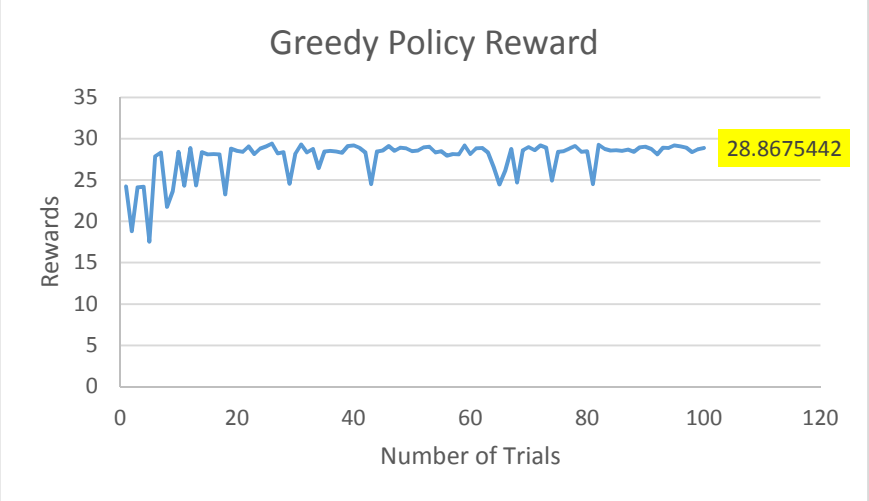
## MDP 1

$\alpha = 0.5$




**N = 100**

- The shape of the learning curve is noisy since we increased the learning rate (alpha)

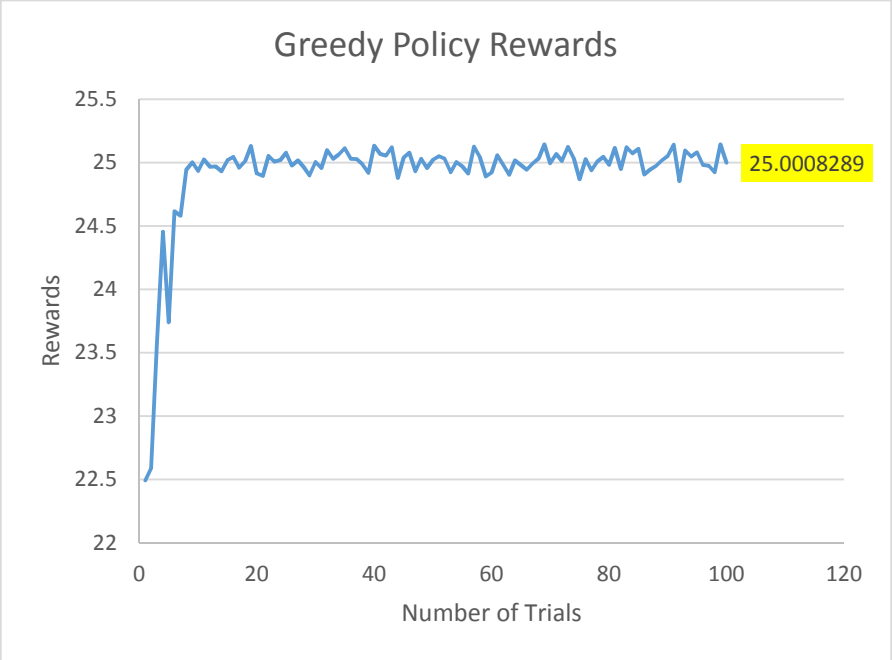



**N = 1000**

- It converged closer to the value of the optimal policy since the number of trials is higher
- The shape is still noisy due to high value of learning rate

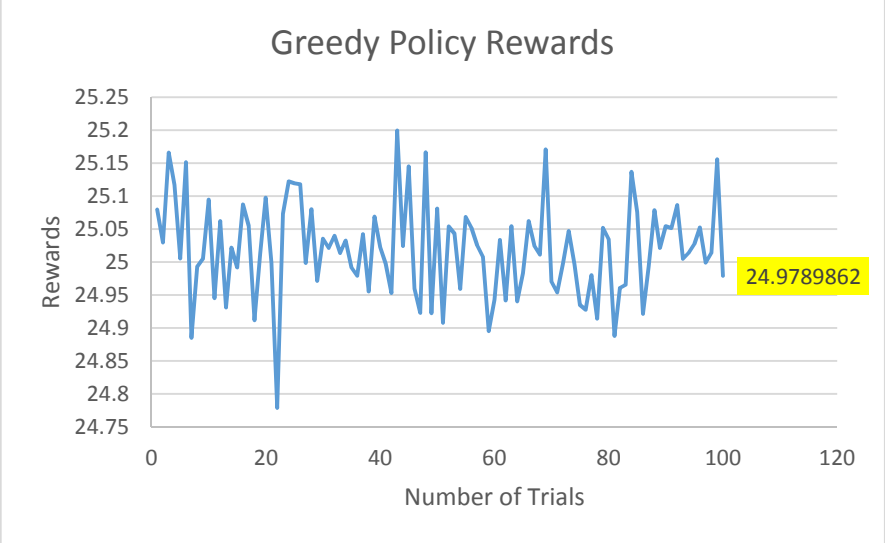
# MDP 2

$\alpha = 0.1$




**N = 100**

- The number of trials for this domain looks fine since it has converged to the value of optimal policy (25)
- It seems that since the description of the second MDP is easier, convergence to the optimal value happens faster than the other MDP
- The shape of the learning curve was almost as we could expect




**N = 1000**

- Since N is high, the value is oscillating around the optimal value (25)