**Q-Learning + MLP Validation:**

Exploration Method = Completely random strategy (e-greedy with epsilon= 1 and without decay)

Maximum number of time-steps per episode = 2000

Experience Replay Batch Size = 10

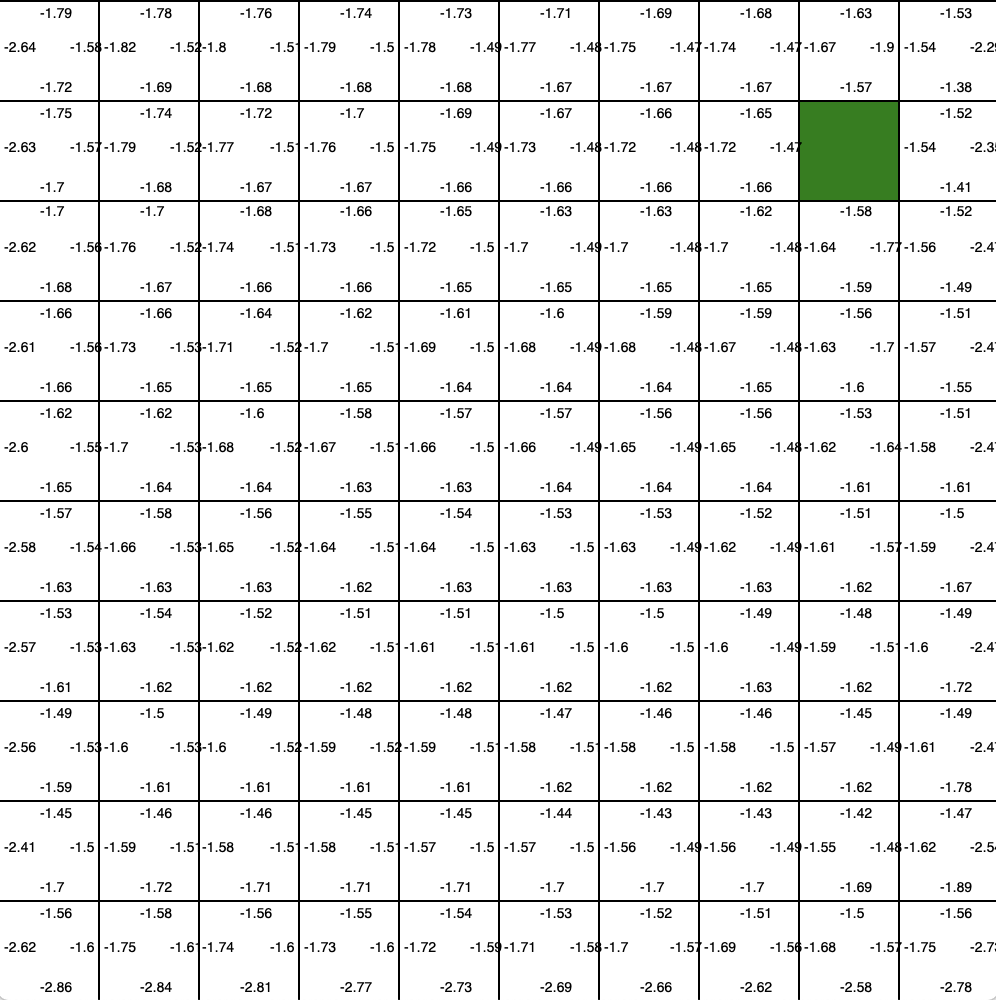
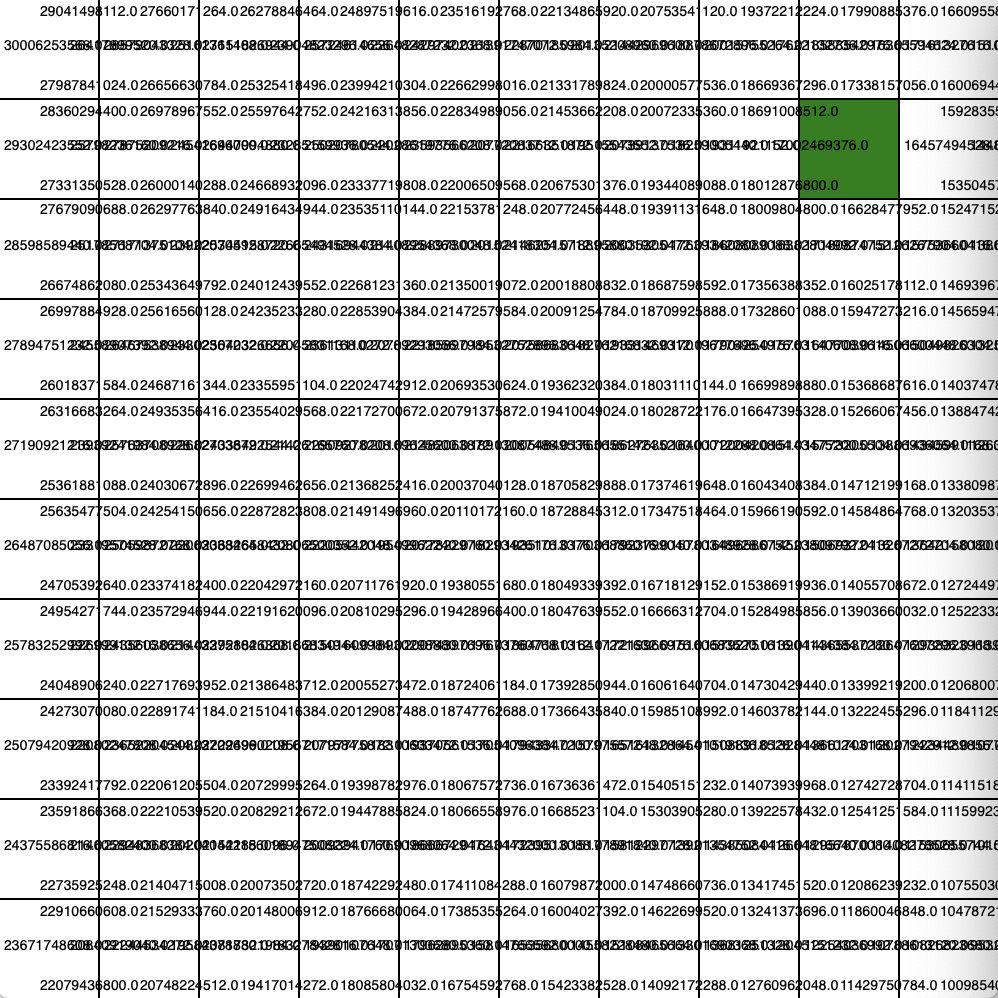
Target Q-Network Update = After each episode

Discount Factor = 0.98

MLP inputs = X state coordinate and Y state coordinate (normalized)

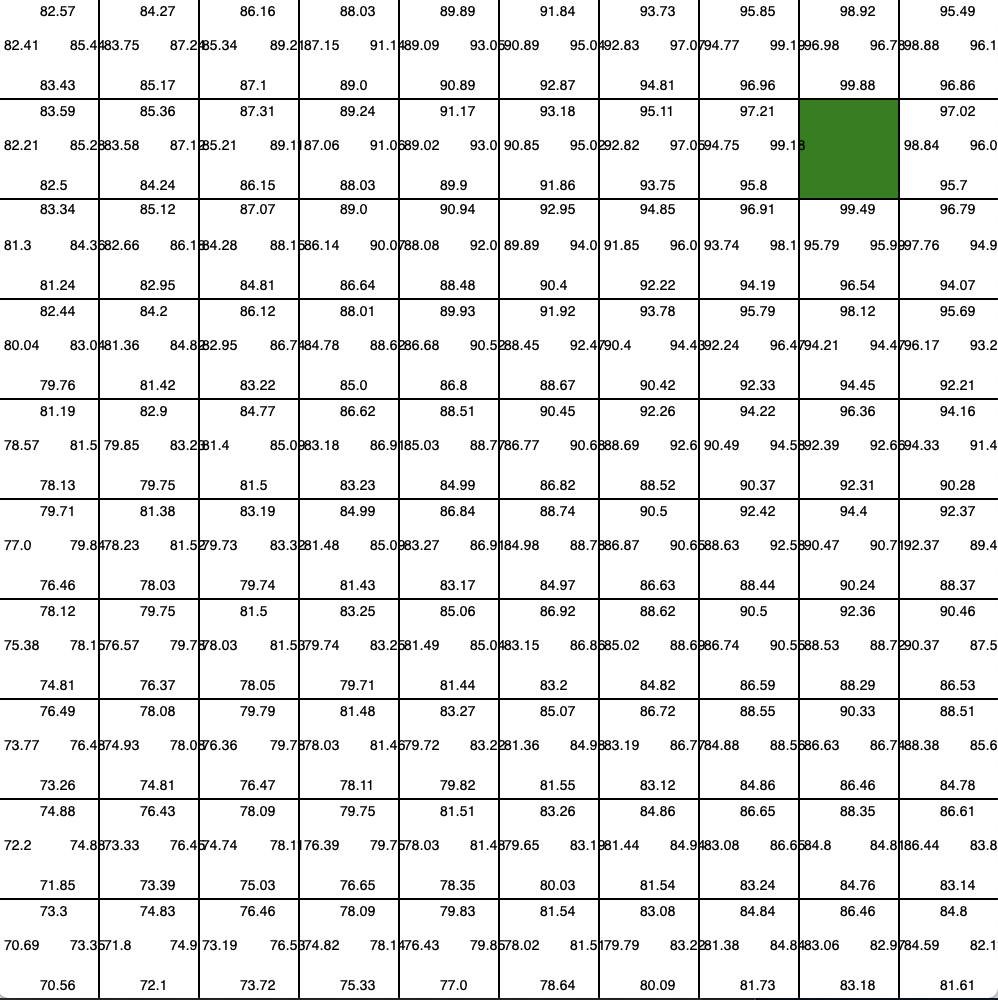
**No Walls:**

First of all, I tried to validate my Q-Learning algorithm, based on an MLP as Function Approximator, in the Grid World without walls. The problem was that using the MLP layer structure 2 (Input layer) / 24 (1st Hidden layer) / 24 (2nd Hidden layer) / 4 (Output layer) the results didn’t make any sense at all:



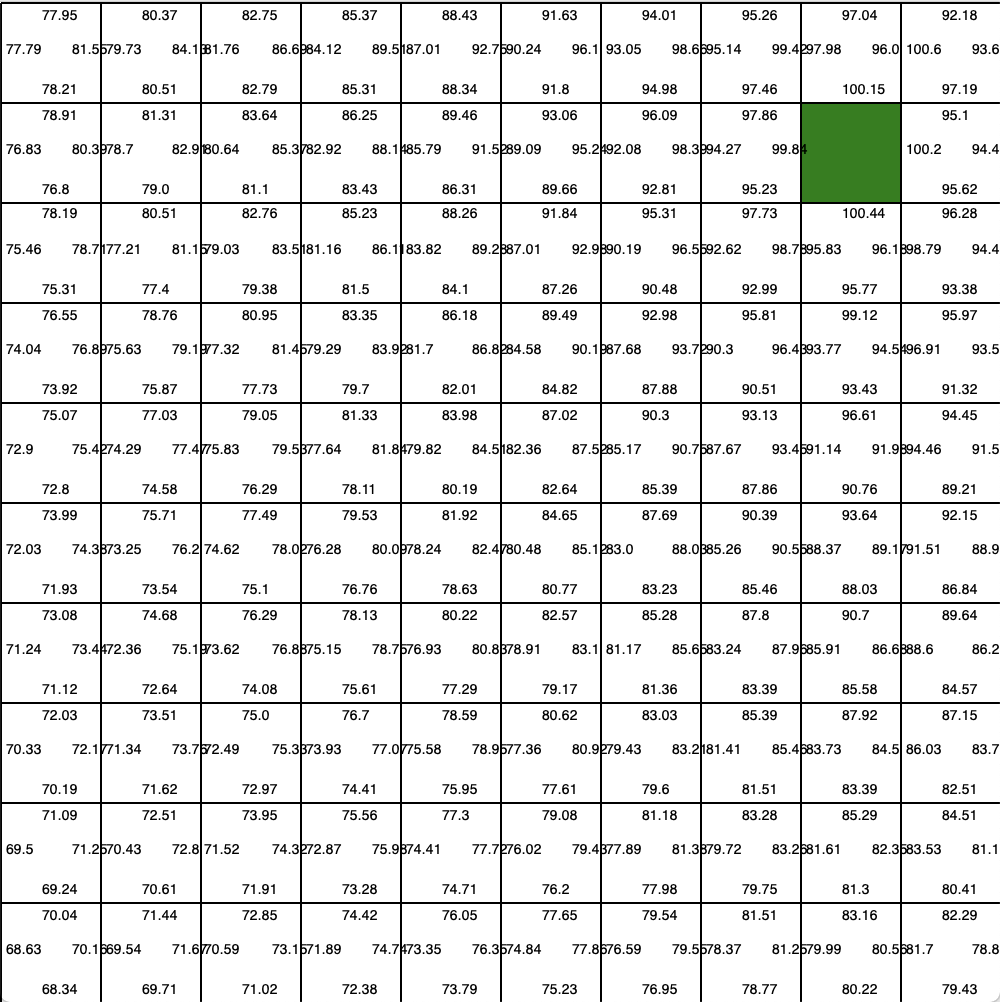
* 2/24/24/4 MLP Structure
* RMSProp Optimizer
* ReLu Activation Function
* 200 Episodes
* Learning Rate = 0.001
* 2/24/24/4 MLP Structure
* Adam Optimizer
* ReLu Activation Function
* 200 Episodes
* Learning Rate = 0.001

But then, simplifying the MLP layer structure to 2 (Input layer) / 10 (Hidden layer) / 4 (Output layer), the results were much better:



* 2/10/4 MLP Structure
* Adam Optimizer
* Sigmoid Activation Function
* 2500 Episodes
* Learning Rate = 0.001

The next step was to try to get a representation of the Q-Table (Q-Function) quite close to what could be an optimal one. I changed the learning rate LR = 0.001 🡪 0.01 and I limited the number of episodes to the moment in which the difference between the actions leading to the goal state and 100 was less than 0.5:



* 2/10/4 MLP Structure
* Adam Optimizer
* Sigmoid Activation Function
* Until Error < 0.5
* Learning Rate = 0.01

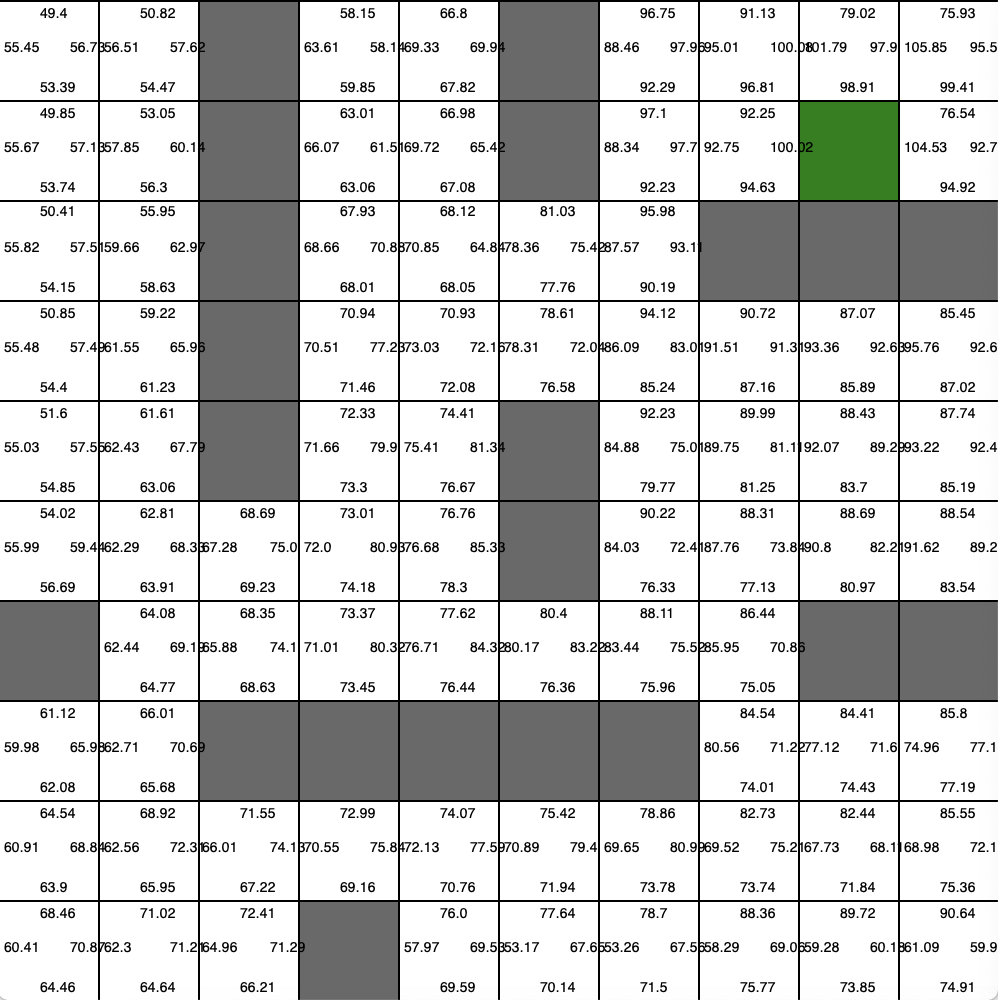
My questions at this point are:

* Why the first MLP layer structure was behaving so bad?
* When should I update the Target Q-Network? Because if I do it every time-step, it is only learning from 1 batch of experiences from the Experience Replay Memory. But if I update it every episode, it is possible that in some episodes the MLP is not giving reasonable results and then the agent gets stuck. Limiting the number of time-steps is a solution. But do you have any recommendation?
* Another thing I observed was that using sigmoid function instead of ReLu as activation function in the MLP made the agent learn faster. What’s the reason for that?

**With Walls:**

After validating the Q-Learning algorithm in the Grid World without walls I tried again what I hadn’t been able to accomplish, to make the agent able to reach the goal position with all the walls placed in the grid based on the MLP Q-Function. But this time it worked, I’m quite sure that the reason is the Discount factor value change 0.9 🡪 0.98 and the simplification of the MLP structure.

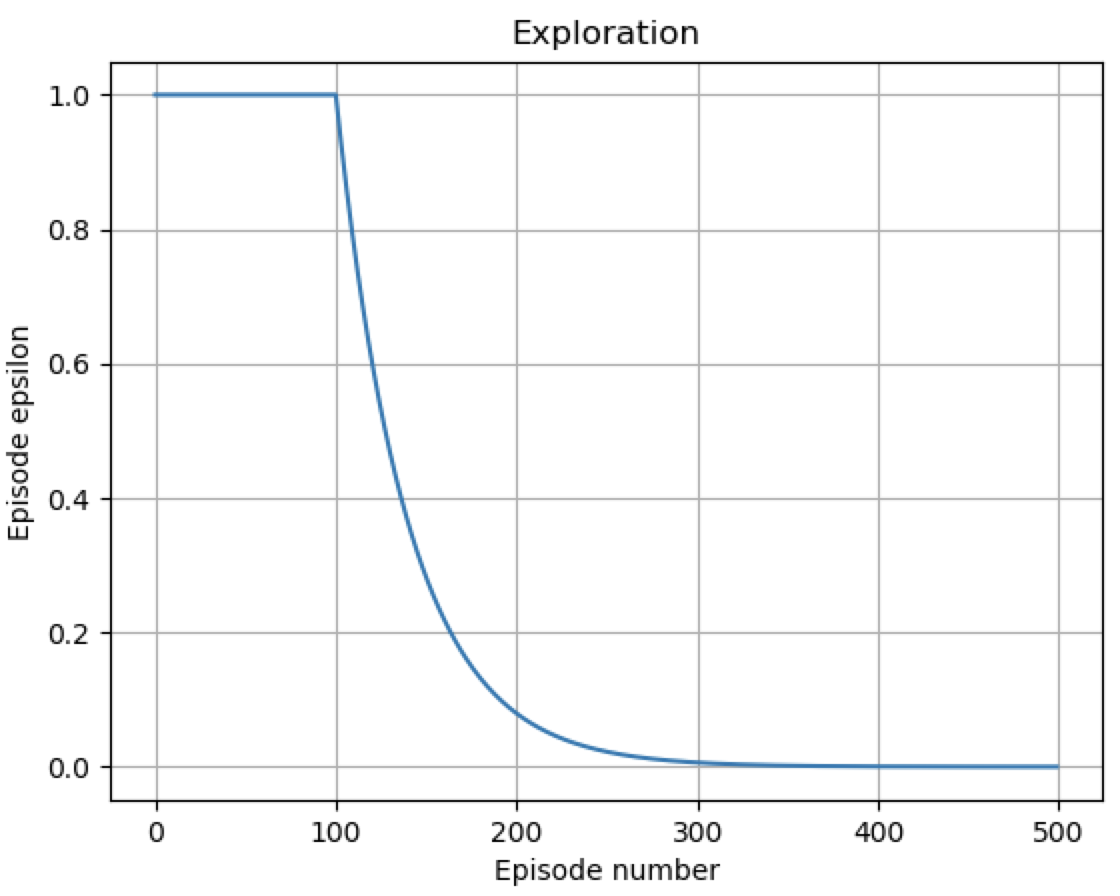
These were the results I obtained:



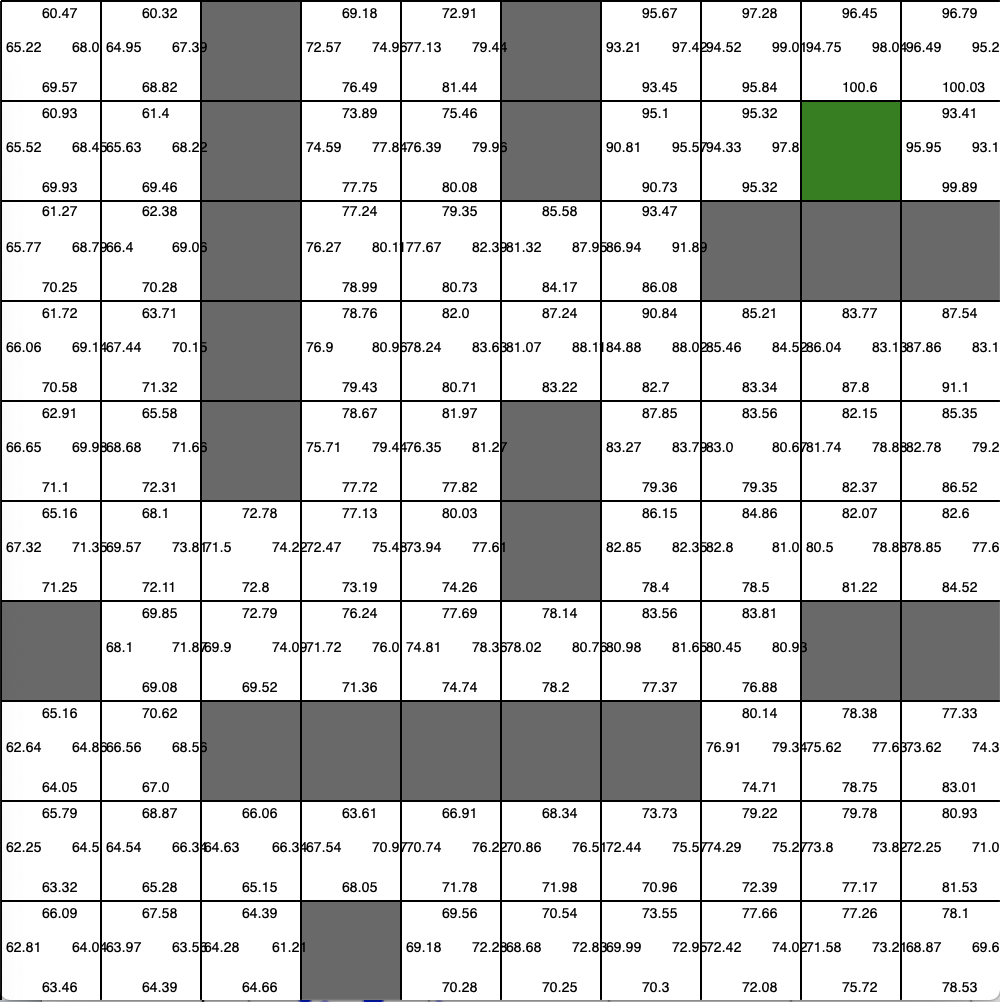
* 2/10/4 MLP Structure
* Adam Optimizer
* Sigmoid Activation Function
* 500 Episodes
* Learning Rate = 0.01

Finally, as the algorithm seemed to be working, even with walls, I tried to train and test my agent. The exploration method I used in this case was an epsilon-greedy strategy with an epsilon value of 1 and no decay until the 100th episode (So, completely random). After that episode, I decayed the epsilon exponentially with a decay rate of 0.975 until epsilon = 0 where the agent was being tested.

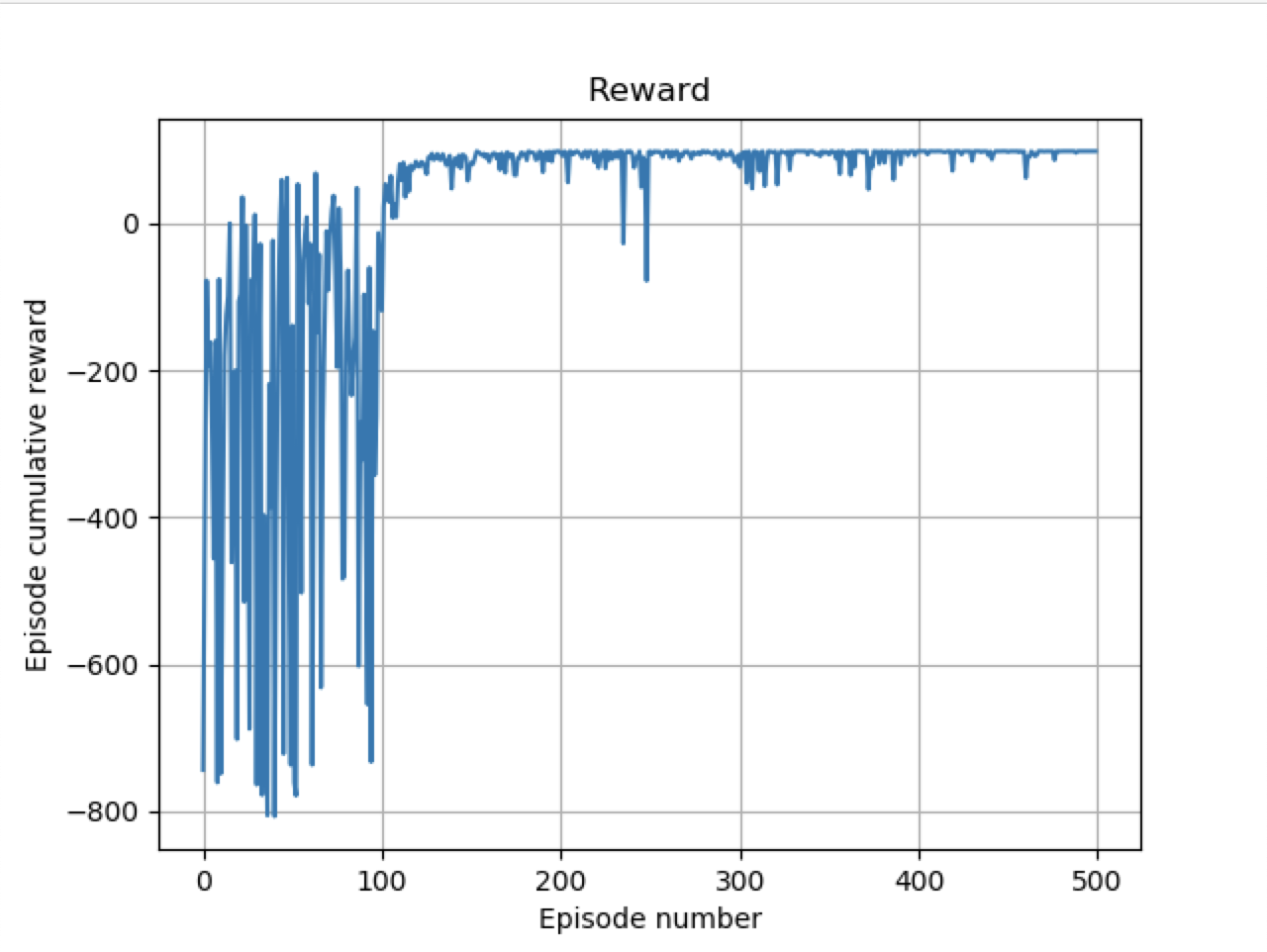
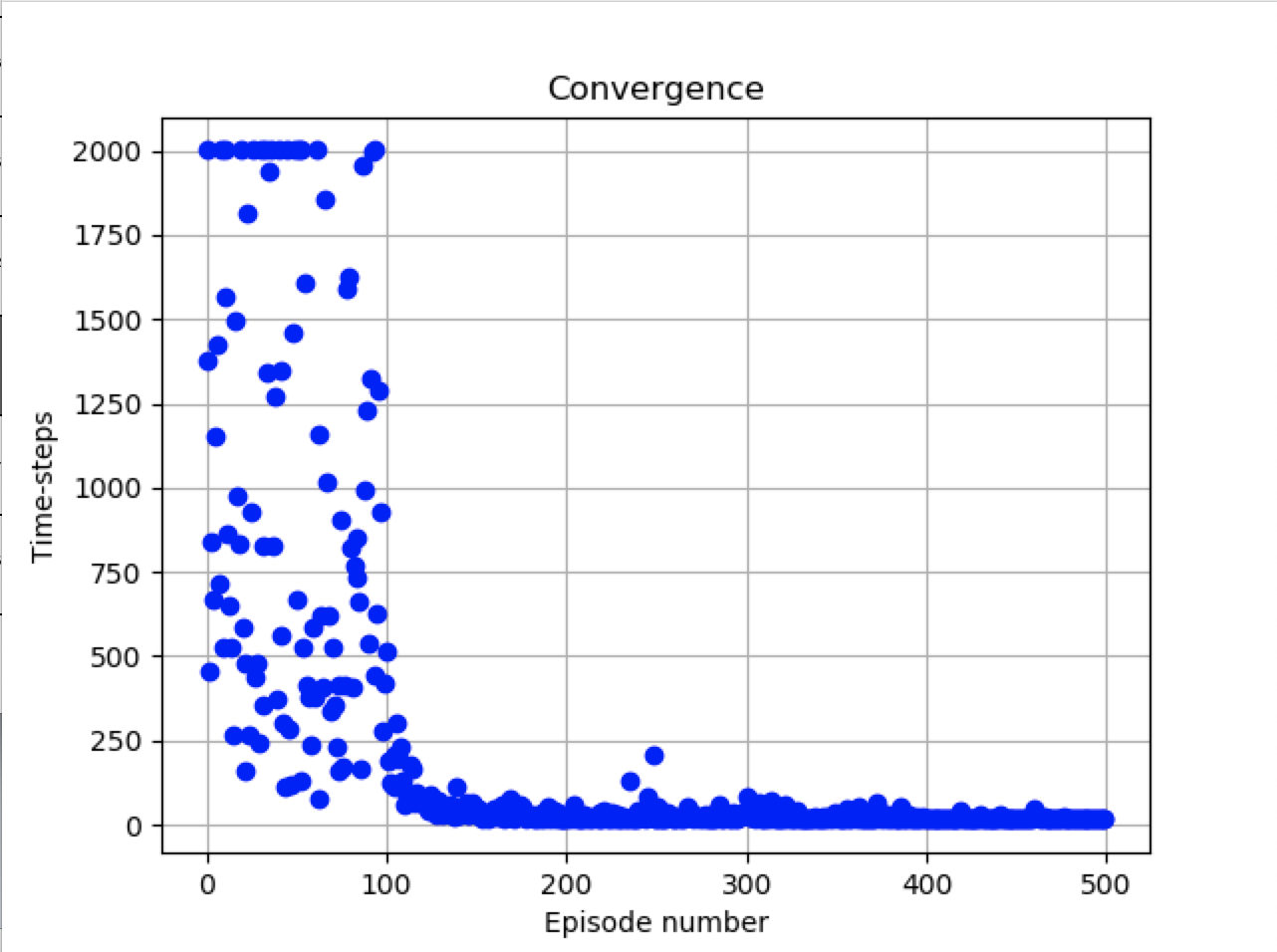
Exploration method:



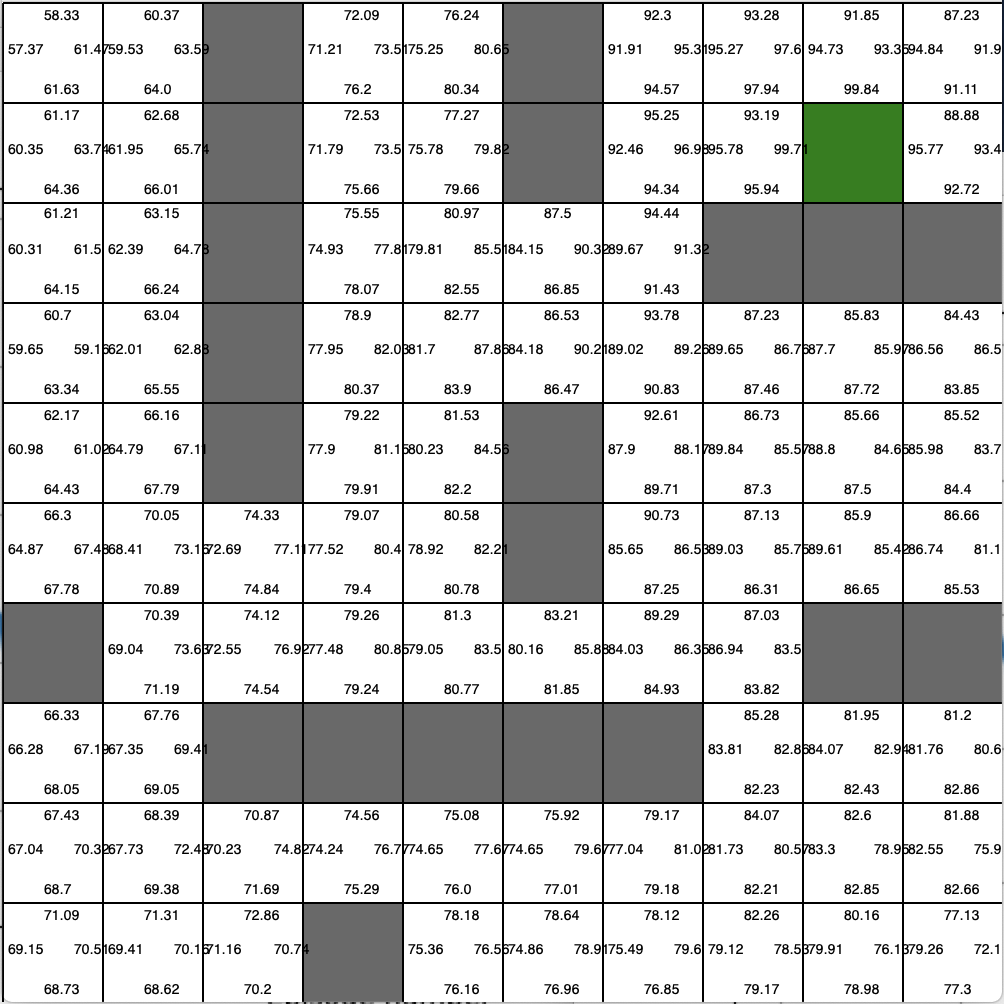
These were the results I obtained:



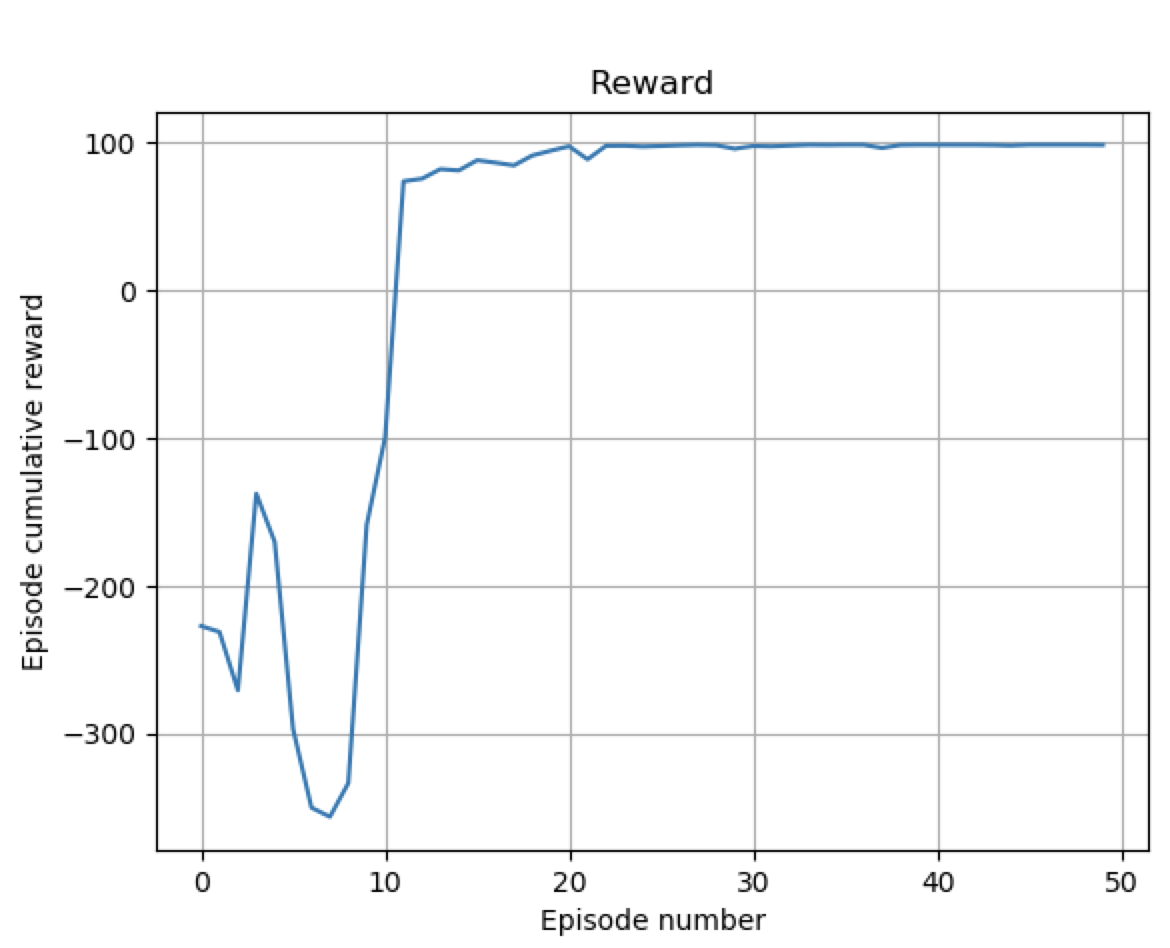
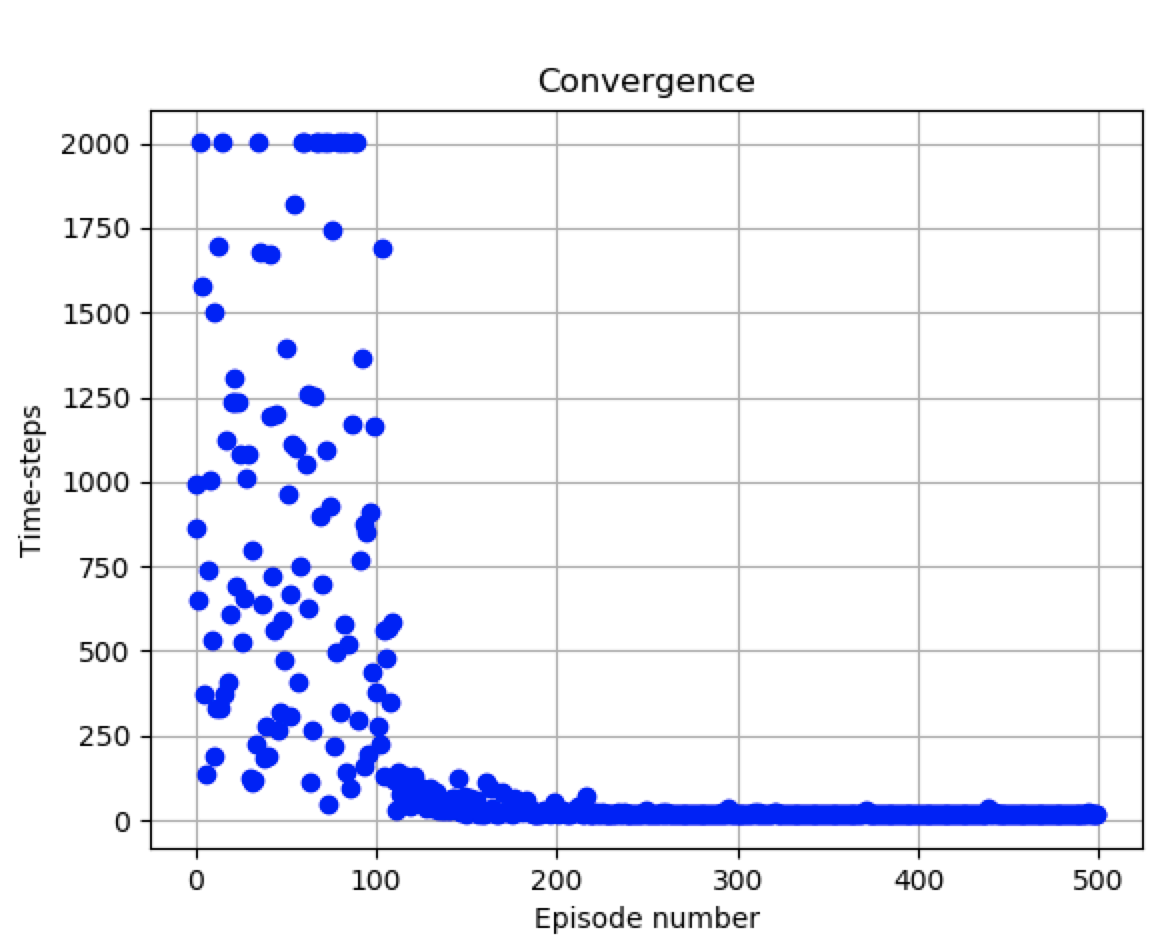
* 2/10/4 MLP Structure
* Adam Optimizer
* Sigmoid Activation Function
* 500 Episodes
* Learning Rate = 0.01



In this first experiment, the agent behaviour ended converging. The same did the reward function as well. The interesting thing was that the solution found was, indeed, the optimal one (16 steps). But that is not always happening. Running another experiment, in order to show the reward function evolution using its 10 time-steps mean, the agent converged but into a suboptimal solution (18 steps). These are the results:



* 2/10/4 MLP Structure
* Adam Optimizer
* Sigmoid Activation Function
* 500 Episodes
* Learning Rate = 0.01



My only question is: Do you think I am in the right direction?

Now I will implement the multi-objective agent that should collect other pieces before going to the goal. In this case:

* Which information should I use as input for the MLP?
* Should I change a lot the MLP layer structure?
* Should I use a kind of observation matrix of the agent as input of the MLP apart of the state coordinates?