Group 23 Assignment 3

Andrej Erdelsky — Dean Polimac — Mateja Zatezalo — Lucas Fatas

March 2021

1 Development

1.1 E-greedy Selection

In order to prevent the algorithm from choosing the best option at all times when it has not learned anything yet. Instead we let it choose a random action in a case where all adjacent action values are zero. This is because random actions are not based on any preexisting action values.

1.2 Agent Cycle

For each agent cycle, we start by storing the current state. Then we get our next action based on the current state by using our e-greedy selection algorithm. This action is then used to get the new state of the agent after performing it. The new state is subsequently used to retrieve its respective reward, which will be used in our Q-learning algorithm in the next step. After updating our action values through this learning algorithm, we have to check if the agent has arrived at its goal state, and if it has, we reset it back to its starting position.

1.3 Stopping Criterion

We have indeed implemented the agent cycle with a stopping criterion of 30000. Depending on the provided maze, we plan on adjusting this criterion.

1.4 Initial Run Without Learning

Running the algorithm without having implemented Q-Learning yields the results which can be seen on Figures 1, and 2. The figures represent the results of toy, and easy maze, respectively. Even though, the two figures differ in the number of trails taken, and the number of steps per trail, they both show that there is no strict decrease in the number of steps between consecutive trails. The fact that this is the case, leads to the conclusion that the algorithm indeed does not learn. As if it did learn, the performance would progressively get better after each trail, resulting in a strict decrease in the number of steps taken.

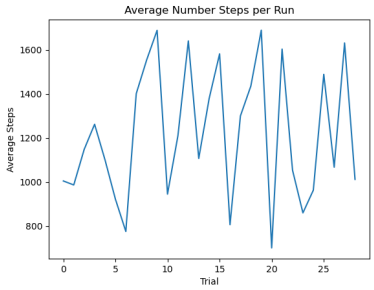

Figure 1: Results on Toy Maze without Q-Learning

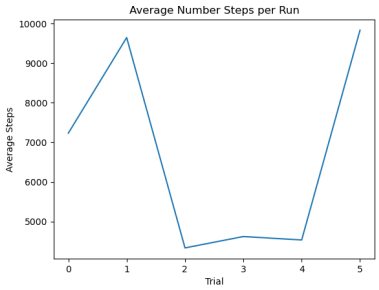

Figure 2: Results on Easy Maze without Q-learning

1.5 Implementing Q-Learning

Our Q-learning method first uses the previous position of our agent and our performed action to retrieve a q value from the maze (Q_{old}). We then retrieve the largest q value from all the possible combinations of our next state, combined with all the possible actions at the current state (Q^s_{max}). By using these values along with the reward amount r and constants α and γ, we can calculate the Q_{new} value using the equation 1. Constants α, and γ represent the speed of learning, and the discount applied to the reward, respectively. Once we obtain this value, we set it for the corresponding action and previous position pair.

$$Q_{new} = Q_{old} + \alpha \cdot (r + \gamma \cdot Q^s_{max} - Q_{old}) \tag{1}$$

1.6 Running The Agent With Q-Learning

Having implemented Q-Learning the results of running the algorithm now are represented with the Figures 3, and 4, where they depict the results of toy, and easy maze, respectively. It can be clearly seen from both Figures that there is a strict decrease in the number of steps taken by each consecutive trail. This implies that the algorithm manages to learn as it preforms progressively better until it reaches some optimum, either being the global or local minimum. These results are achieved when running the algorithm with an increased stopping criteria of 200 thousand steps, α set to 0.7, γ set to 0.9, and ϵ - which determines the probability of selecting a random path over the best path - set to 0.1 as suggested by the assignment.

2 Training

2.1 Changing Epsilon

The results of using three different values for epsilon can be seen on Figures 10, 6, 7, where the values are 0.1, 0.001, and 0.4, respectively. There is no significant

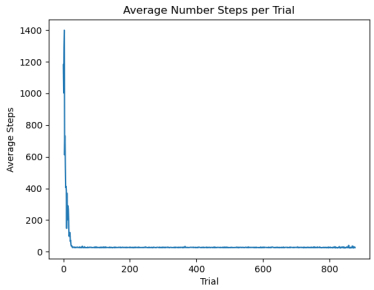

Figure 3: Results on Toy Maze with Q-Learning

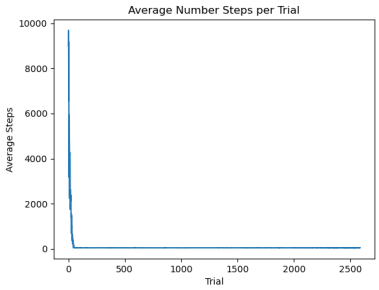

Figure 4: Results on Easy Maze with Q-learning

difference between the three plots, but one thing that can be seen is that a higher epsilon causes more variation. This makes sense since its value corresponds to the amount of randomness when choosing an action. Even though varying epsilon did not have a huge impact, using settings its value to 0.001 gave the best results.

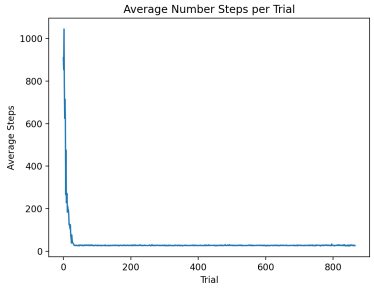

Figure 5: Epsilon of 0.1 on Toy Maze

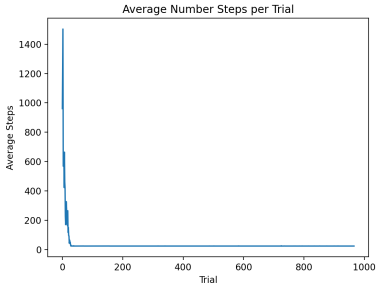

Figure 6: Epsilon of 0.001 on Toy Maze

2.2 Changing Learning Rate

As mentioned above in section 1.6, the learning rate is expressed with variable α in Equation 1. For the learning rate we plot results given by α when set to 0.3, and 0.9. This was done while keeping the value of γ at 0.001. Figures 8, and 9 show that there is no noticeable difference between the results produced using these two values. The only value for α which impacts the number of average steps was 0. This returns a similar plot as in Figure 1 when the algorithm was run without Q-Learning. This is because with a learning rate of 0, the algorithm does not actually learn. The learning rate we ended up using was 0.9.

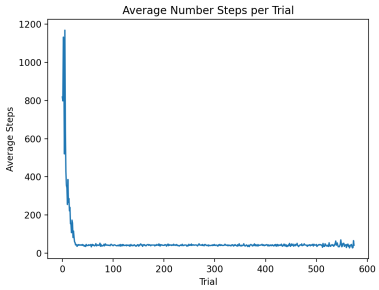

Figure 7: Epsilon of 0.4 on Toy Maze

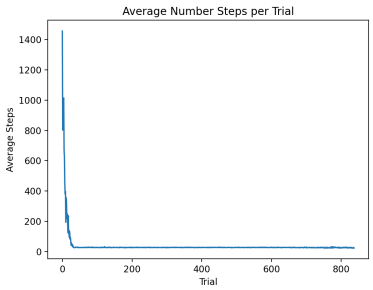

Figure 8: Alpha of 0.3 on Toy Maze

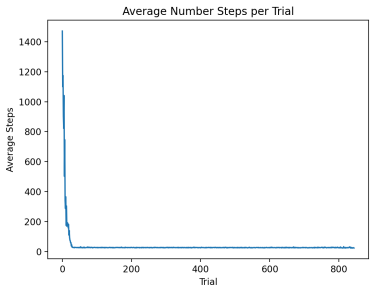

Figure 9: Alpha of 0.9 on Toy Maze

2.3 High vs Low Epsilon

In case of the given mazes, a lower epsilon - being 0.001 - caused the algorithm to have less variance, and is the best option. With a more complex maze, a low epsilon could result in less exploration which may lead to finding a sub-optimal path. This would likely cause the algorithm to converge to a higher average number of steps. Even though a high epsilon increases the variance, it also decreases the chances of getting stuck in a local minima.

3 Optimization

3.1 Adding second reward

Adding a second reward at the top right corner of the maze led to the following. With epsilon is set to 0.1, it causes the average number of trials to be slightly larger than when using one reward.

3.2 Changeable Epsilon Value

In the previous problem the average number of trials was 32. Varying ϵ led us to a more optimal solution. We vary ϵ by decrement it after each trial by a small value of 0.00005. By doing this, we got an optimal path length of 21, while the average number of trials also decreased to 26.

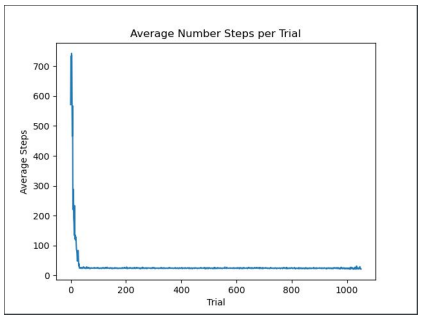

Figure 10: Epsilon of 0.1 with 2 rewards

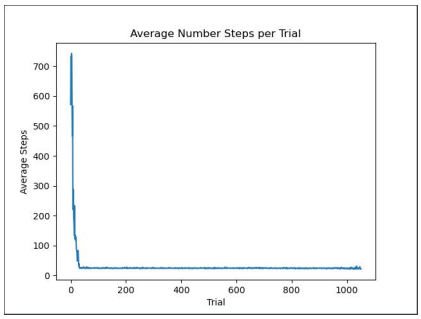

Figure 11: Decreasing epsilon with every trial

3.3 Experimenting with Gamma Value

After experimenting with the γ value, we can conclude it does not have much impact on the optimal solution. Figures 12, and 13 show that the result quickly converges supporting this idea.

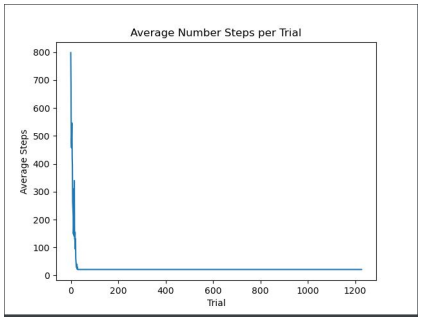

Figure 12: Gamma value of 0.9

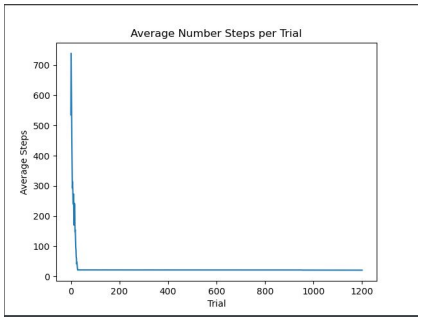

Figure 13: Gamma value of 0.001