Reinforcement Learning

Michał Jóźwik, Lukas Ostrovskis, Martynas Krupskis, Denis Tsvetkov

March 2022

1 Running the code

You need to install numpy before running the code. The code in main.py will find the path in the *toy_ maze.txt* and print it's length to the console.

2 Development

Q: How do you ensure that the agent is not biased towards selecting the same action in *getBestAction()* over and over when it has not learned anything yet?

A: To ensure that the agent is not biased towards selecting the same action in **getBestAction()**, we first check whether all the action values within the valid action list are equal. If they are, we select the action randomly, otherwise we choose the best one.

Q: Implement the agent's cycle in RunMe (selecting actions, executing them, updating the Q-table, ...). Explain your cycle.

A: We train the agent for a defined number of steps (30000 for the toy maze and 130000 for the easy one). The agent starts taking steps until it reaches its goal and then its position is reset and the number of steps left that it can make is reduced. For each step, we get the current state of the agent, use the e-greedy algorithm to select the next action, perform that action to obtain the next state, and finally use these obtained values to update the Q-table for the old state.

Q: Run your agent. You should be able to observe that the number of steps the agent takes per trial does not decrease. Make a plot of the average of 10 runs (one run is one launch of RunMe) to show that the agent indeed does not learn. The x-axis should represent the number of trials, the y-axis should represent the average number of steps in that trial. Make this plot for both mazes given. Explain your plots.

A: As it can be seen in the plots below the average number of steps to reach a target per trial randomly fluctuates between 100 and 2500 for the toy maze and 4000 and 11000 for the easy one but with no clear pattern of downtrend. Thus, we can conclude that the agent is in fact not learning anything yet.

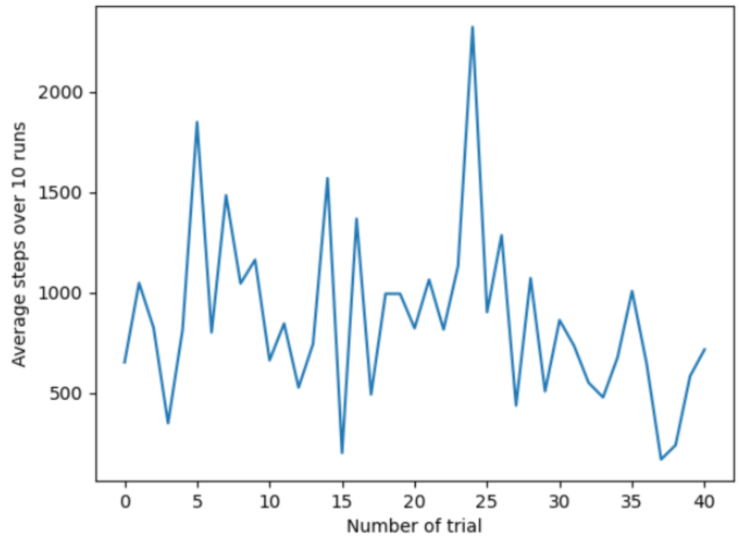

Figure 1: Average number of steps per trial for the toy maze for 30000 steps

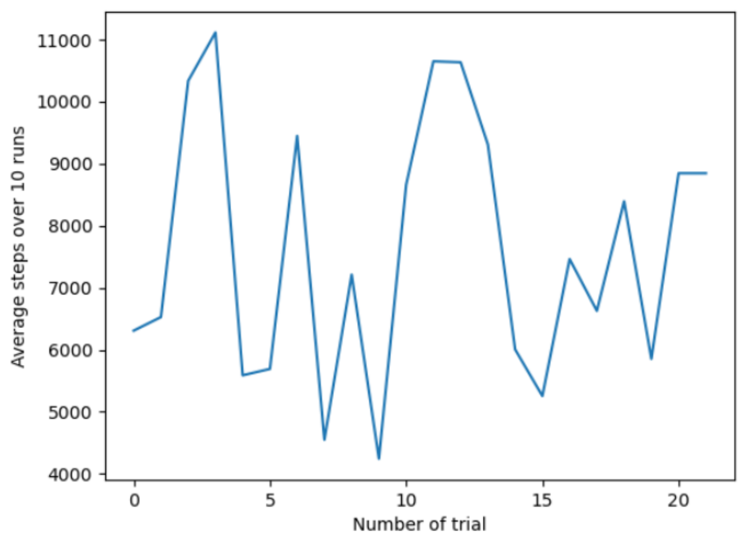

Figure 2: Average number of steps per trial for the easy maze for 130000 steps

Q: Implement updateQ() in MyQLearning. Set $\alpha = 0.7$, $\gamma = 0.9$, and $\epsilon = 0.1$. This should be a simple implementation of Q-Learning. Explain your method.

A: This is how our **updateQ()** method works:

- Get the **Q(s', a')** for each possible action **a'** from next state **s'** and store that into **action_values** variable.
- Get the maximum action value from next state **s'** using **action_values[argmax(action_values)]** and store it to **Q_max**.
- Get the **Q** value of the current state **s** and the action **a** and store it to **Q_old**

- Compute $\mathbf{Q\_new} = \mathbf{Q\_old} + \boldsymbol{\alpha} \cdot (\boldsymbol{r} + \boldsymbol{\gamma} \cdot \mathbf{Q\_max} \textbf{ - } \mathbf{Q\_old})$, formula given to us in the lecture and in the assignment itself.
- Set **Q(s, a)** to **Q_new** computed in step 4.

Q: Now run your agent again on the mazes. You should be able to see that the numbers decrease over time during a run. If you don't, first verify if you allow the robot to try long enough. Again, make a plot of the average of 10 runs to show that the agent indeed does learn. Make this plot for both mazes given. Explain your plots.

A: In figures 3 and 4 we can see that our agent is now able to learn both mazes, as there is a clear downtrend for the average number of steps (it sometimes goes up but that is due to the randomness of $\boldsymbol{\epsilon}$). Furthermore, both graphs are converging to an optimal value in around 35 trials for the toy maze and 45 for the easy one.

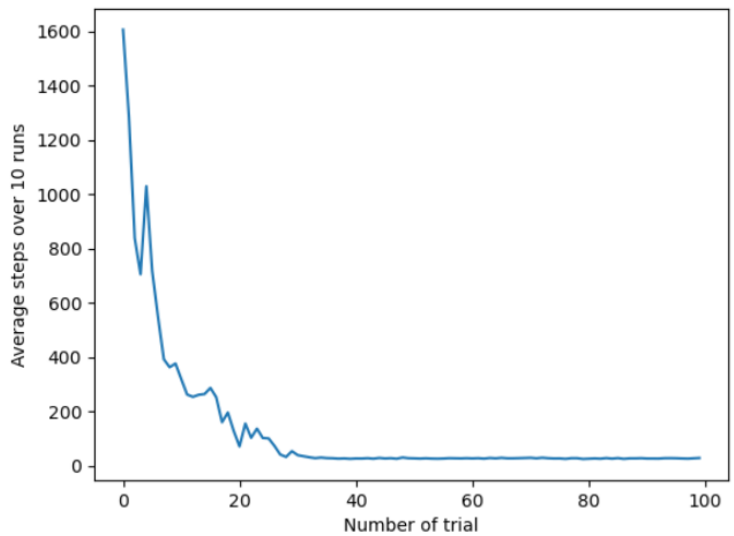

Figure 3: Average number of steps per trial for the toy maze for 30000 steps[1]

[1]We introduced an additional stopping criterion of 100 trials (only for the plotting but not the final version of the algorithm) so that the learning and convergence of the algorithm are clearer. This applies to all plots except figures 1, 2, 17, and 18.

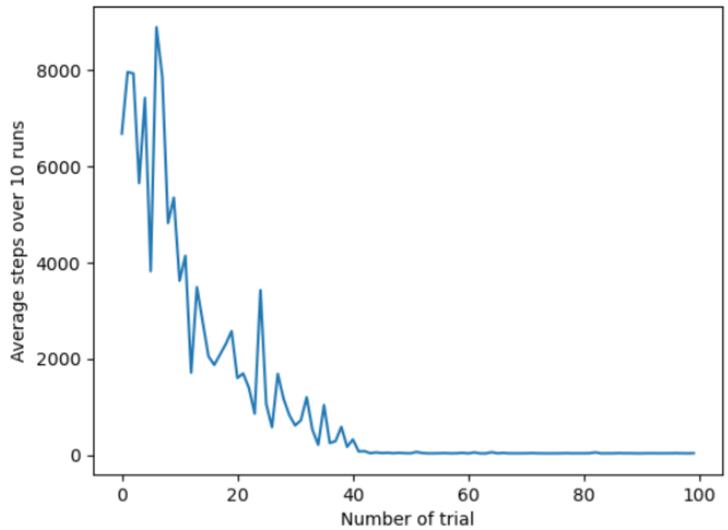

Figure 4: Average number of steps per trial for the toy maze for 130000 steps

3 Training

Q: Play around with the ϵ parameter of your action selection algorithm. Choose a value of ϵ between 0 and 1 at least four times. For each of those values, make a plot of 10 run averages (similar to above) to study the effect of varying epsilon. Explain your plots.

A: We have chosen to generate plots with epsilon values of 0.1, 0.3, 0.6 and 0.9. Not to repeat ourselves the effects of high and low epsilon values are explained in question 8, meaning that going from 0.1 to 0.9 our agent is prioritizing exploration over exploitation more and more, therefore more and more random and non-optimal steps are likely to get included in the path. In figures 3, 5, 6, 7, 4, 8, 9, and 10 corresponding to epsilon values of 0.1, 0.3, 0.6, and 0.9 for both mazes this can be seen, as the optimal path we find is getting worse as more and more random and non-optimal steps are introduced into our path from start to target location.

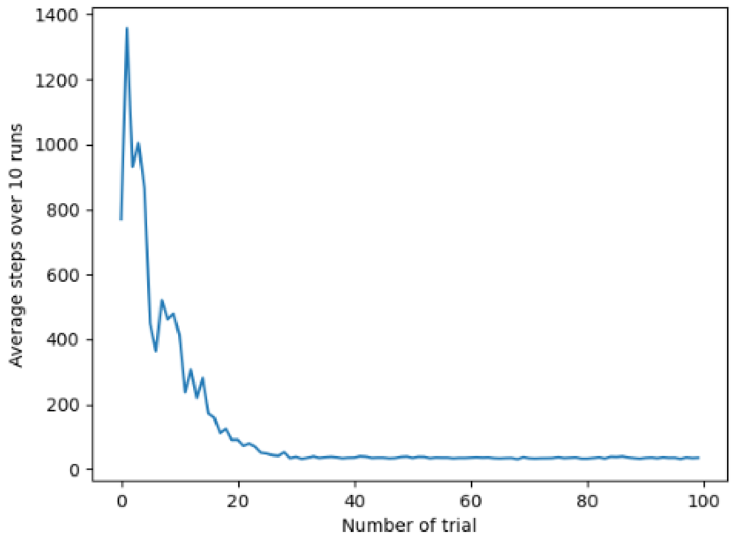

Figure 5: Average number of steps per trial for the toy maze for $\boldsymbol{\epsilon = 0.3}$ and 30000 steps

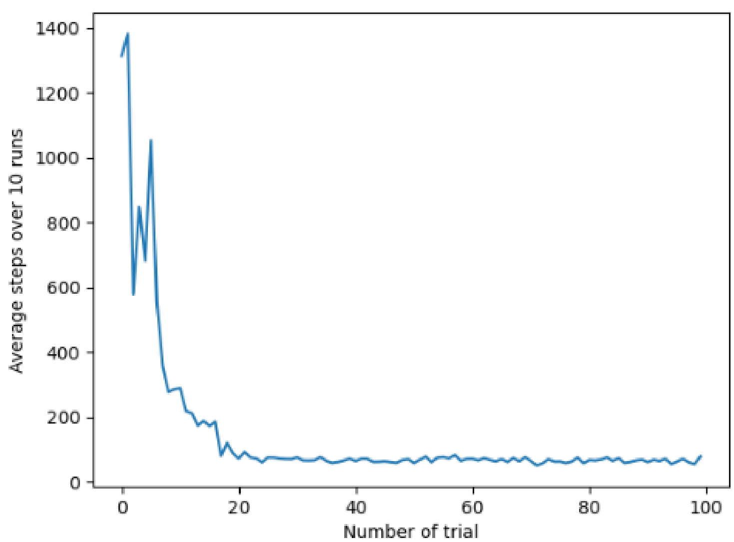

Figure 6: Average number of steps per trial for the toy maze for $\boldsymbol{\epsilon = 0.6}$ and 30000 steps

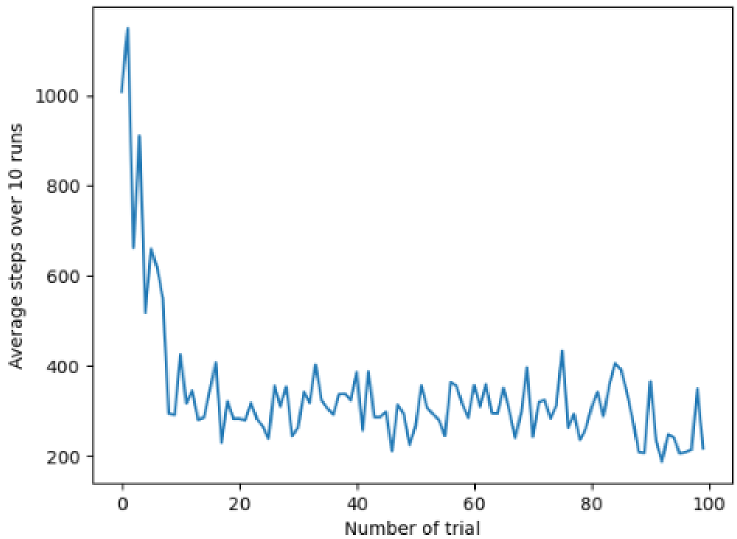

Figure 7: Average number of steps per trial for the toy maze for $\boldsymbol{\epsilon} = \mathbf{0.9}$ and 30000 steps

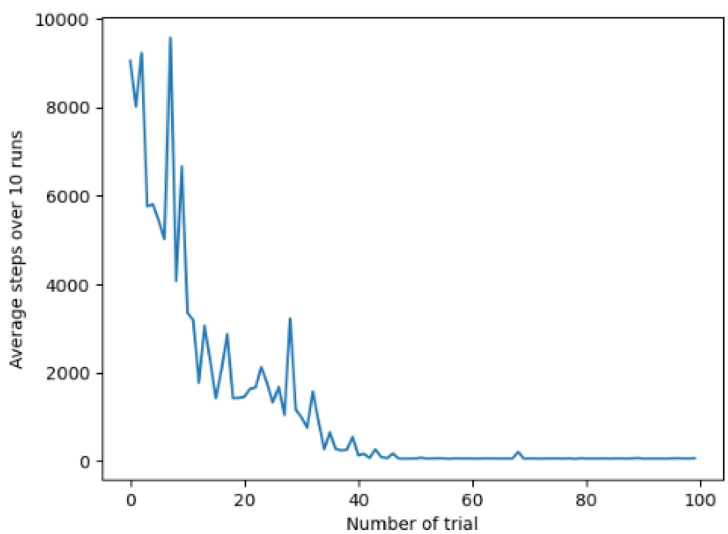

Figure 8: Average number of steps per trial for the easy maze for $\boldsymbol{\epsilon} = \mathbf{0.3}$ and 130000 steps

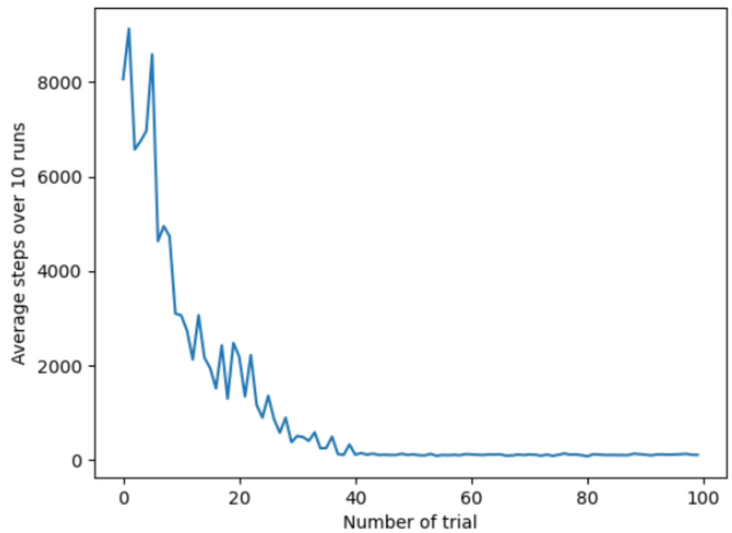

Figure 9: Average number of steps per trial for the easy maze for $\boldsymbol{\epsilon} = \mathbf{0.6}$ and 130000 steps

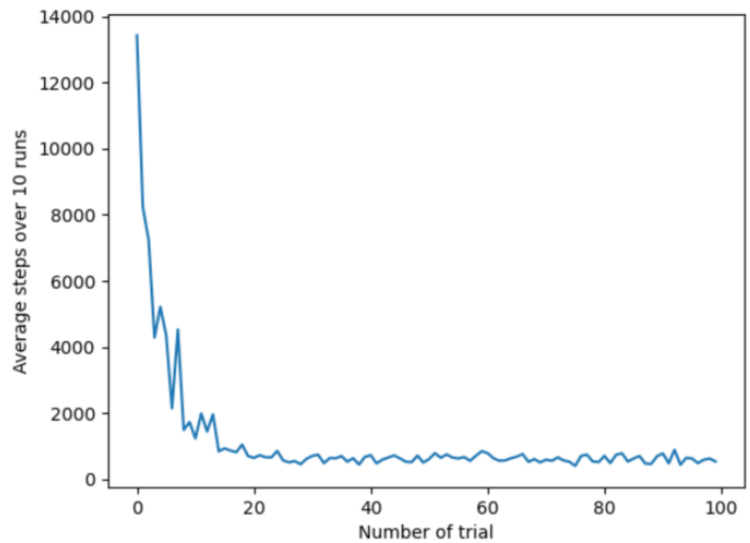

Figure 10: Average number of steps per trial for the easy maze for $\boldsymbol{\epsilon} = \mathbf{0.9}$ and 130000 steps

Q: What are the trade-offs between a high and a low ϵ?

A: The trade-off between a high and a low epsilon is the trade-off between exploration and exploitation. The agent must exploit the actions that have given it a high reward in the past to obtain a lot of reward but to discover such actions, it also has to try actions it has not tried before - it has to explore. High epsilon is equivalent to relying heavily on exploration, thus the agent will explore a larger solution space, but won't be as effective when learning from it. Low epsilon is equivalent to relying heavily on exploitation, thus the agent will exploit the solutions that have already led it to rewards, but this might result in the agent getting stuck in a local optima.

Q: Now, play around with the learning rate α of your algorithm. Again, choose a value of α between 0 and 1 at least four times. Make a plot for each of these values. Explain what you notice. Why is that the case?

A: Theoretically lower alpha values should take a larger number of trials to converge to an optimum, but in our experimentation (look at figures 11, 12, 3, 13, 14, 15, 4, 16 corresponding to the alpha values of 0.1, 0.4, 0.7, and 1 for both of the mazes) there does not seem to be any correlation between alpha values and the number of trials it takes to converge to an optimum. After some experimentation, we were able to conclude that this is because $(\boldsymbol{r + \gamma \cdot} \textbf{Q\_max - Q\_old})$ in each iteration is really small, in the scale of $10e-6$. This means that multiplying this value by an alpha of 0.1 or 0.9 has negligible influence on the value by which the Q(s, a) is updated.

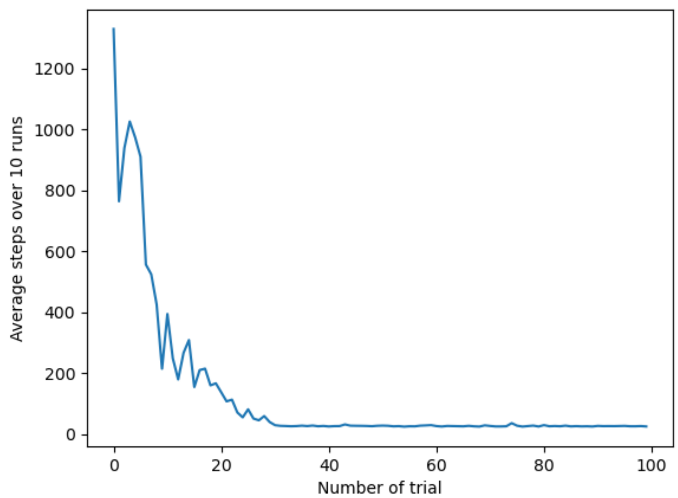

Figure 11: Average number of steps per trial for the toy maze for $\boldsymbol{\alpha = 0.1}$ and 30000 steps

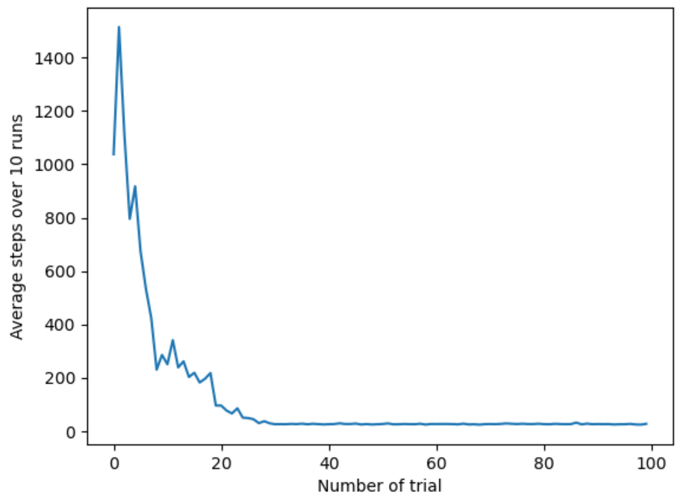

Figure 12: Average number of steps per trial for the toy maze for $\boldsymbol{\alpha = 0.4}$ and 30000 steps

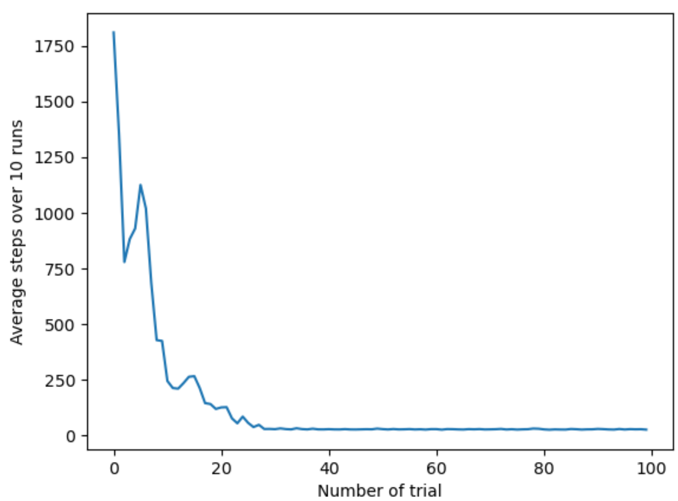

Figure 13: Average number of steps per trial for the toy maze for $\boldsymbol{\alpha = 1}$ and 30000 steps

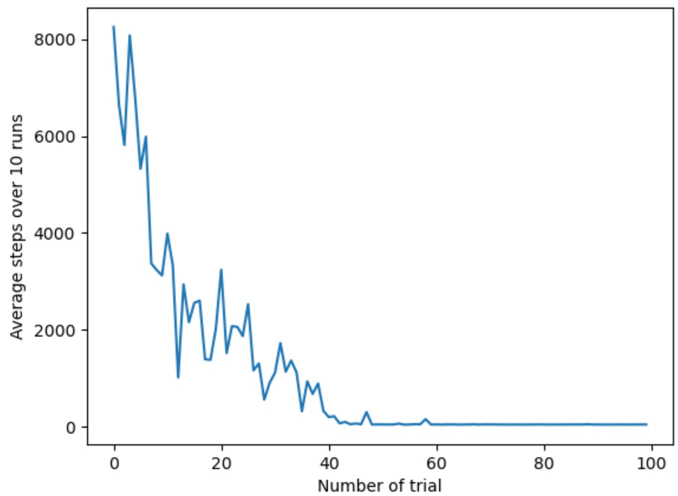

Figure 14: Average number of steps per trial for the easy maze for $\boldsymbol{\alpha = 0.1}$ and 130000 steps

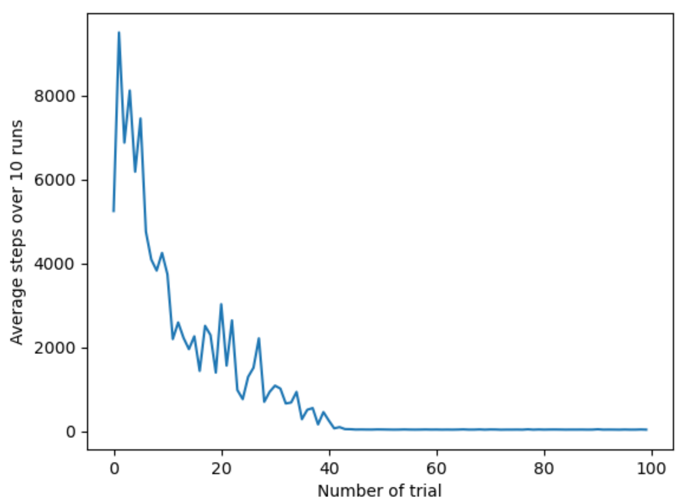

Figure 15: Average number of steps per trial for the easy maze for $\boldsymbol{\alpha = 0.4}$ and 130000 steps

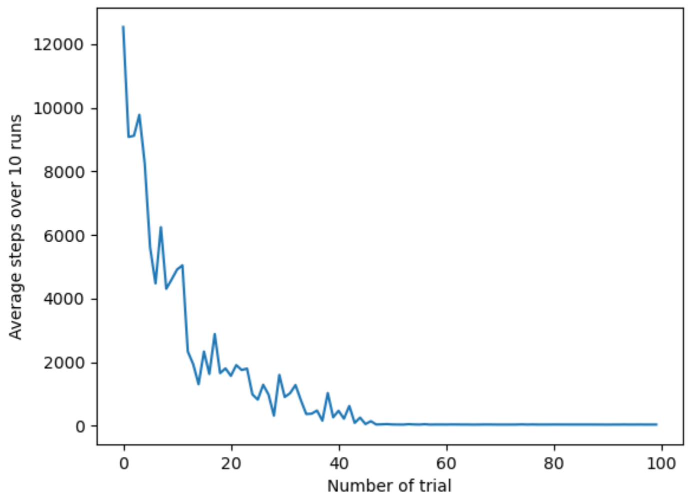

Figure 16: Average number of steps per trial for the easy maze for $\boldsymbol{\alpha = 1}$ and 130000 steps

4 Optimization

Q: Add a second reward, sized 5, at the location (9,0) (top right). Remember to add this location to the condition to reset the agent (it is a second goal). What do you observe? How can you explain this?

A: After adding a second reward, sized 5, our algorithm switches to converging to the coordinate with the new reward of 5, meaning that it was unable to converge to an optimal reward. This is easy to explain, as in question number 7 the optimal epsilon value is 0.1. This means that our algorithm currently prioritizes exploitation over exploration and it simply cannot find that (9, 9) contains a bigger reward, as finding the reward of 5 at (9, 0) is easier.

Q: Invent a way to mitigate the previous problem by modifying the values of ϵ (they can change with each trial). Explain your method. Show with a plot that your agent converges to the optimal solution.

A: Our algorithm was able to find a path to a coordinate with a reward of 10 with an epsilon value of around 0.5. Increasing the epsilon value further also increases the number of steps we have to take to find the reward of 10.

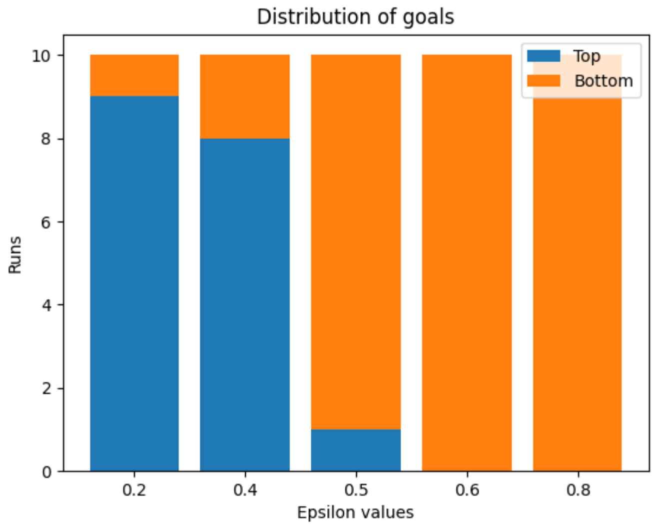

Figure 17: The distribution of the goals reached depending on the value of epsilon. The bottom is the optimal goal (the one at coordinate (9, 9) with reward 10).

Q: Can you, using your previous solution, experimentally find a value of γ for which the optimal solution is to go up for the smaller reward, rather than go down for the larger one? Explain and show the plots for different values of γ that you try.

A: Gamma is a parameter responsible for the importance of future rewards, so 0 will make the agent myopic, meaning that it will only consider current rewards, and for gamma values approaching 1 the agent will take long-term reward into account more and more. In our maze, reaching a reward of 5 is easier, as it requires less exploration, but reaching a reward of 10 is more rewarding, therefore high gamma value means optimizing for a reward of 10. If we want to go up for a smaller reward then we have to become more short-sighted for future rewards and try to reach a target that is "easier" to reach. This is evident in the figure below, where high gamma values result in the majority of runs going down (in the maze) and reaching a higher reward, but low gamma values result in going up for the smaller reward.

As it is visible in our plot, the turning point for the agent is the gamma value of around 0.75. For all gamma values below it, it tends to always end up at the top goal with the small reward, at 0.8 it starts terminating at the bottom goal more frequently, and for all values greater than or equal to 0.9 it tends to always end up at the bottom goal with the big reward.

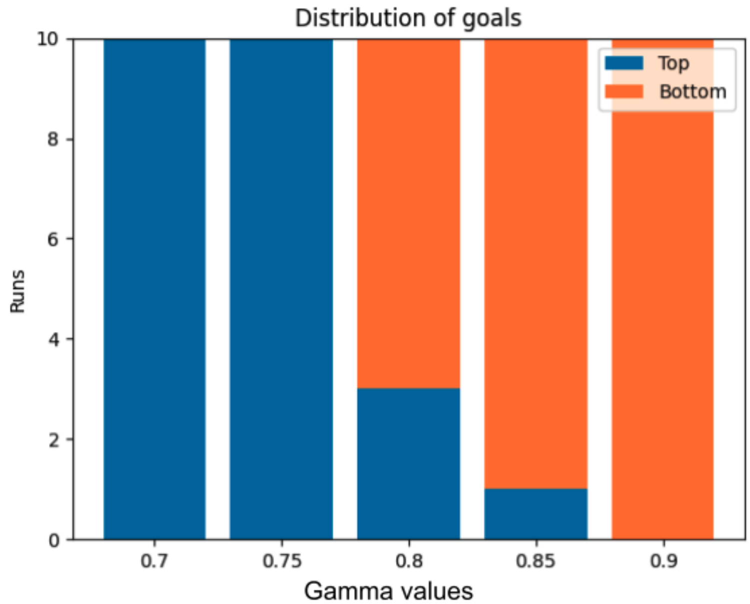

Figure 18: Top and bottom goal terminations over 10 runs for different gamma values. The bottom is the optimal goal (the one at coordinate (9, 9) with reward 10).

5 Reflection

Q: What are the possible downsides of having a greedy algorithm concerning its action selection?

A: The greedy actions may not always be optimal in the long run, as sometimes a slight loss in the short term may lead to an overall better solution. This could lead to being stuck in the local optimum instead of finding the global optimum that we're looking for.

Q: When and why can reward functions cause problems in society?

A: The reward function is generally designed by humans, who have a certain goal in mind and they want the agent to get to that goal most efficiently. In most cases, they also have a general idea of how the agent should perform, but it also happens that the system discovers unexpected ways to obtain a lot of rewards. This could be considered a reward function exploitation and it could lead to unintended negative consequences, which were not thought of by the designers. As currently more companies opt for Machine Learning solutions we need to be aware that the main stakeholders of most of these projects are humans.

Q: Can you think of a way to overcome these problems for reward functions? If not, why do you think that the problem is unsolvable?

A: The easiest way to mitigate this problem is to carefully design the reward function and ensure that there are means to mitigate the problems that arise. Engineers have to be prepared to use these means to adjust it whenever there is an unintended solution. In general, it might not always be possible to avoid all the possible exploitation techniques, but this might also lead to the discovery of better-unexpected solutions. There is no silver bullet when it comes to machine learning, but there are definite ways to make it more controlled. We need to make our code readable and modifiable to possibly act whenever there are changes needed.