Reinforcement Learning - Mid Semester Course Project

Hadar Pur, Ron Azuelos

Submitted as Mid Semester project report for the RL course, IDC, 2023

1 Introduction

The goal of our project is to summarize the first part of the Reinforcement Learning (RL) course, in which we learn about some basic approaches to solve RL problems. More particularly, we were instructed to solve maze problems of 3 different sizes. First, we used dynamic programming (DP) to find a solution for the 5x5 maze problem, considering it as a planning problem. Second, we used three different methods to solve the 15x15 maze problem: Monte Carlo and SARSA, which are on-policy methods, and Q-Learning which is an off-policy method. Last, to solve the 25x25 maze problem we chose the Q-Learning algorithm with the addition of two optimizations: using ε-decay, and modifying the rewards of two cells, one with a positive reward and the other with a negative one, based on experiments we performed. For each method we tested various different parameters, and let the agent run for a changing number of episodes. The results are presented in Section 3.

2 Solution

2.1 General approach

According to the requirements of our assignment, our approach to solve the problems included the implementation of the following algorithms: Policy-iteration (DP method), Monte Carlo, Q-learning and SARSA.

2.1.1 Dynamic Programming

In this method, we solve the problem as a planning problem, assuming we know the model (i.e., the probabilities of doing each action from each state) and the corresponding rewards. Our goal is to find the best policy, by performing a policy improvement step after a series of policy evaluations. This entire process is named Policy Iteration. At first, we calculate the values for each state, based on Bellman equation (see Equation 1).

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')] \tag{1}$$

This process is repeated until the delta (δ), which is the maximum difference between a current value and an updated value of any state, is smaller than a pre-defined θ.

Next, we perform a policy improvement step, that calculates the best current action for each state, and then updates the policy. The best actions are considered as the ones that maximize the values of the state that they leads to. This process is repeated until the policy is considered stable, i.e. there are no changes between the best actions of the newly generated policy and the previous one. We used this method to solve the 5x5 maze, as requested.

2.1.2 Monte Carlo

This on-policy method is based on generating large amount of "games", starting from a random policy and then gaining knowledge about the possible states and actions, based on the agent's experience. Each game is called an episode, which is a set of steps that the agent takes in the maze until it reaches the goal, according to the current policy. Each episode is followed by the calculation of the G-values for all state-action pairs, based on their first visit in the state. Then, we update the policy by setting the preferred actions to be the ones that maximize the Q-values from each state. We used this method to solve the 15x15 maze.

2.1.3 Q-Learning

This algorithm is an off-policy method, where the agent starts from the initial state and plays until it reaches the goal. It repeatedly chooses an action based on a certain policy, performs the step according to the stochastic environment, and reaches to a new state. Then, it updates the Q-values of the states based on the actions that maximize the Q-value of the next state, according to the formula seen in Equation 2.

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]^{1} \tag{2}$$

We used the basic form of Q-learning to solve the 15x15 maze. We extended the algorithm to also perform ε-decay when solving the 25x25 maze, as it did not converged within the maximum amount of episodes (i.e., 500 episodes).

2.1.4 SARSA

This algorithm is an on-policy method, which has the same principal as Q-Learning. The main difference between the two algorithms is that SARSA updates the Q-values based on the Q-value of the next state and action, according to the *current policy*, using the formula shown in Equation 3.

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \tag{3}$$

We used this method to solve the 15x15 maze.

2.2 Design

2.2.1 Utilities

Our code contains a few general methods, as well as the implementations that we used to solve all maze problems. The following specifications are included in the maze classes we created, which inherits from the basic *MazeEnvCast* class that was provided with the exercise.

- **Encoding and Decoding of states**. Since the states are represented by a tuple of (X, Y), the value-functions and policies would be programmatically represented by 3D matrices (e.g. $|X| \cdot |Y| \cdot |A|$). We wanted to avoid these representations and use 2D matrices (e.g $|S| \cdot |A|$), which are much more comprehensive. To do so, we created 2 methods that helps us map the (X, Y) representation of a state in the maze, to a sequential index. That way, we can get an index that corresponds with a specific state, which will be used to access the data structure, such as the policy and value functions. The X-axis is represented as the column index and Y-axis is represented as the row index. The encode method calculate the $X \cdot M + Y$, where M is the size of the maze. For example, the state in cell (2, 4) in a 5x5 maze (3rd column, bottom line) will be calculated as $2 * 5 + 4 = 14$, and will be considered as the 14th state. The decode method extracts the Y value as the modulo of the encoded state with M, and the X value as a floor-division of the encoded state with M. For example, the state encoded as 7 in a 5x5 maze, is (2,1) as $7 mod 5 = 2$ and $\lfloor \frac{7}{5} \rfloor = 1$.
- **Stochastic**: As mentioned in the instructions, the environment needs to be stochastic, meaning that if the agent chooses to perform an action, the environment will allow him to follow this action with a probability of 0.9, and will perform any of the other 3 actions with a probability of $\frac{0.1}{3}$ for each action. We extended the provided deterministic environment by creating 3 stochastic maze classes, *StochasticMazeEnvCast5x5*, *StochasticMazeEnvCast15x15*, and *StochasticMazeEnvCast25x25* that inherits from the original classes. These new classes contain a revised *step()* method, that gets an action

[1] α is the algorithm's learning rate

code from the agent, and chooses the actual action based on the probabilities mentioned above using *random.choices()* from the *NumPy* module.

- **Action space**: We used the provided action space that is represented as a list of 4 characters, ['N','S','E','W'], representing the 4 possible directions the agent can move: North, South, East and West respectively.

2.2.2 Dynamic Programming

In our code, the policy is represented by a 2D list. The first dimension represents the states, which includes 25 values in the case of 5x5 maze, that will be accessed using the encoded states' values. The second dimension represents the actions, and therefore contains 4 values, which represent the probabilities to perform each action, correlative to the action space. Initially, we set those probabilities to be $\frac{1}{|A|}(= 0.25)$, which will be later updated during the policy improvement step.

The value-functions are also represented in a similar manner as a 2D list. However, the values of the in the second dimension represent the current values for the state-action pair. Starting from zeros for all state-action pairs, we iterate the states and evaluate their values,based on their neighbors, and update the value for each state. We chose to set the θ to a value of 0.0001 to ensure better precision of the value calculation. We also examined various discount factors. The experiment results are presented in Section 3.1. We update the policy by calculating the benefit from doing every action from every state, and choosing the action that maximize the benefit. For those actions, we set their values to be 1, and 0 for every other action. We consider the policy to be stable if the current best action and the updated best action are the same.

2.2.3 Monte Carlo

As requested, we let the agent running for a maximum of 500 episodes, with each episode limited to 5000 steps. We used an initial policy of ε soft policy, with two different values of ε experimented: 0.1 and 0.2. During each episode, we documented the states, actions and rewards, and at the end of the episode, we used those values to calculate the G-value based on the first visit in the state. We also updated the policy by setting $1 - \varepsilon + (\frac{\varepsilon}{|A|})$ for the action that maximize the Q-value for every state, and $\frac{\varepsilon}{|A|}$ for the rest of the actions.

2.2.4 Q-Learning and SARSA

As before, we limit the agent to a maximum of 5000 steps per episode. To prevent a lack of exploration, we used an ε-greedy policy with various of εs that are detailed in Sections 3.3 and 3.4. We implemented the ε-greedy policy by randomly choose a value between [0, 1] and perform an action according to the value.

3 Experimental results

3.1 Dynamic Programming

We examined the Policy Iteration algorithm using 3 different discount factors (γ), the first experiment with $\gamma = 0.3$, the second experiment with $\gamma = 0.5$, and the first experiment with $\gamma = 1$. All experiments were using $\theta = 0.0001$. All three experiments ended after two iterations of policy iteration and reached convergence. We noticed that larger values of γ lead to higher state values, for each state. Using larger γ values lead to another notable outcome which is much more distinctive difference in state values. This difference can be seen in Figures 1, 2 and 3, which demonstrates, using colors, the actual values of each state, where darker cells (states) has values closer to 0 and brighter cells has the largest values (ranging from 0.56 to 1.83 in our case, depending on the experiment). We also noticed that the agent of the first experiment ($\gamma = 0.3$) took much more steps until reaching the goal, and looked less "confident" overall.

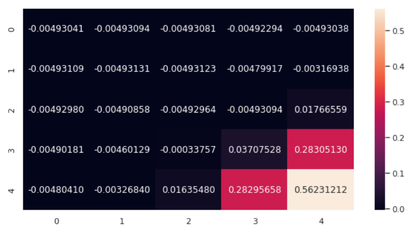

Figure 1: Experiment 1
Policy Iteration, Maze 5x5
$\gamma = 0.3$

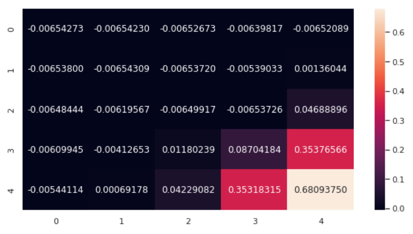

Figure 2: Experiment 2
Policy Iteration, Maze 5x5
$\gamma = 0.5$

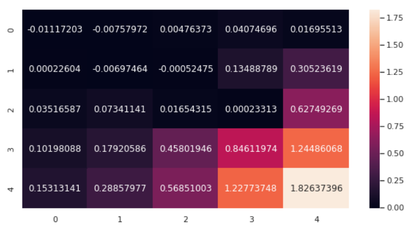

Figure 3: Experiment 3
Policy Iteration, Maze 5x5
$\gamma = 1$

3.2 Monte Carlo

In the case of Monte Carlo algorithm, we have a few parameters to explore: the discount factor, the number of episodes, and ε. We began our exploration in the first experiment, using $\varepsilon = 0.1$, $\gamma = 0.9$, and let the agent "play" for 100 episodes. In Figure 4, we can see that the total cumulative reward of each episode was very low during the entire learning process, along with a large amount of steps, meaning that the agent did not learn the maze properly. The average number of steps was 4967, and the average reward was -2.197. The relevant video can visualize that despite its struggles, the agent was able to successfully solved the maze. However, the agent did had some difficulties at certain cells, from which it was probably "rescued" by the stochastic actions.

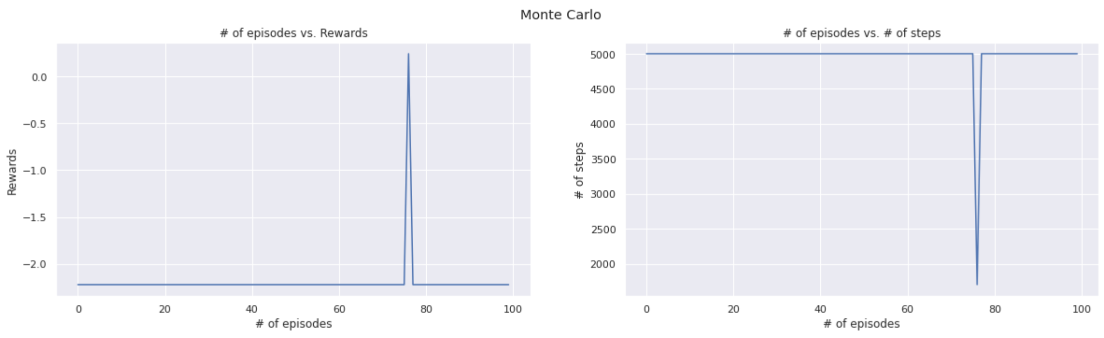

Figure 4: Experiment 1 (Monte Carlo, Maze 15x15, $\varepsilon = 0.1$, $\gamma = 0.9$, 100 episodes)

In experiment 2-3, we used $\varepsilon = 0.1$, $\gamma = 0.95$, and let the agents run for 100 and 300 episodes respectively. For both experiments, the average number of steps and the average reward showed improvement comparing to the first experiment, but the learning process still was not stable enough. In experiment 4, we used $\varepsilon = 0.2$, $\gamma = 0.95$, and 300 episodes of learning. As we can see in Figure 5, the learning process improved, the average number of steps decreased to 442, and the average reward increased to 0.743.

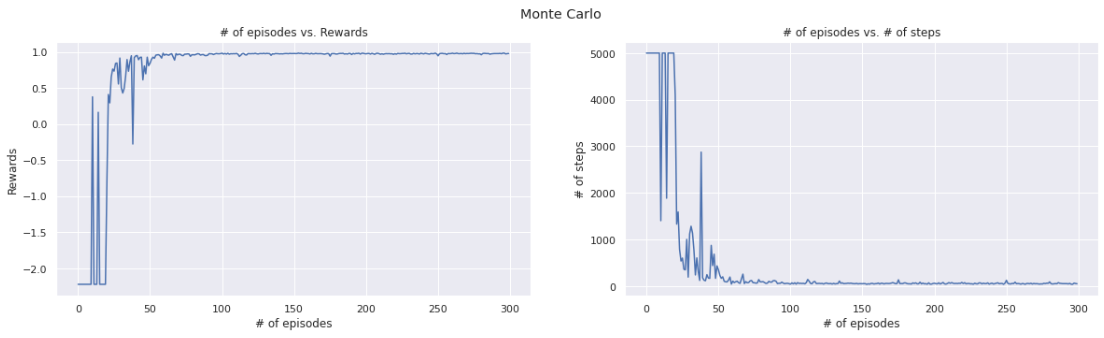

Figure 5: Experiment 4 (Monte Carlo, Maze 15x15, $\varepsilon = 0.2$, $\gamma = 0.95$, 300 episodes)

Although experiment 4 did reached a stable solution, in experiments 5-6 we increased the number of episodes of learning to 500, and explored its effect with $\gamma = 0.95$ and $\gamma = 0.99$ respectively, looking for an additional improvement. In experiments 6 we saw a minor improvement in the average number of steps and the average reward, as seen in Figure 6. Also, we can see that the learning process becomes relatively stable after only 100 episodes.

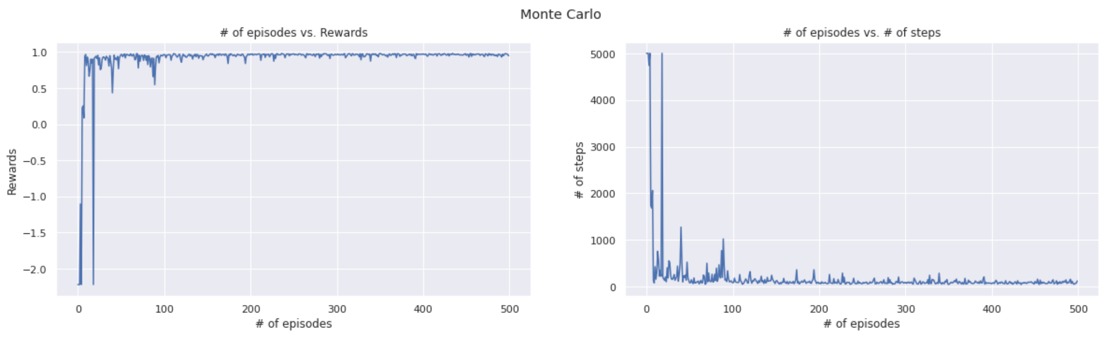

Figure 6: Experiment 6 (Monte Carlo, Maze 15x15, $\varepsilon = 0.2$, $\gamma = 0.99$, 500 episodes)

We noticed that starting from experiment 3, where the number of episodes is larger then 100, the algorithm becomes stable enough, and manages to solve the maze even halfway through the learning process. We assume, according to the graphs, that after 100 episodes the agent collects enough information to successfully solve the maze, and it has a descent amount of episodes to apply what it learned and try to improve its strategy.

3.3 Q-Learning

In the case of Q-Learning algorithm, we have the same 3 parameters as Monte Carlo, along with another parameter – α – which is also referred to as the learning rate. We performed 6 experiments using different values for each parameters. We started with the following values in experiment 1: $\varepsilon = 0.1$, $\gamma = 0.9$, $\alpha = 0.2$, and 100 episodes. Figure 7 presents the learning process of experiment 1, which was decent but not stable enough. Overall, it produced an average of 460 steps and an average reward of 0.795. The relevant videos can verify that the agent could not solve the maze at the middle of the learning process, and hardly succeeds to solve it at the end.

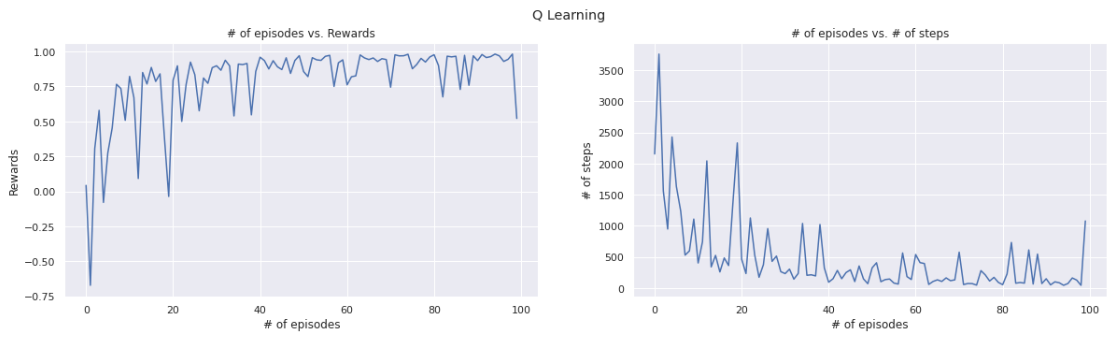

Figure 7: Experiment 1 (Q-Learning, Maze 15x15, $\varepsilon = 0.1$, $\gamma = 0.9$, $\alpha = 0.2$, 100 episodes)

In experiments 2-3 we let the agent run for 200 episodes, and set $\gamma = 0.95$, $\alpha = 0.2$, and $\varepsilon = 0.1$ and $\varepsilon = 0.01$ respectively. Both experiments showed better average results than the first experiment, however, not significantly. Experiment 4 was able to produce great results, which can be seen in Figure 8, as we let the agent to run for 300 episodes, with the same parameters as experiment 3. The average number of steps dropped to 185 and the average reward increased to 0.917.

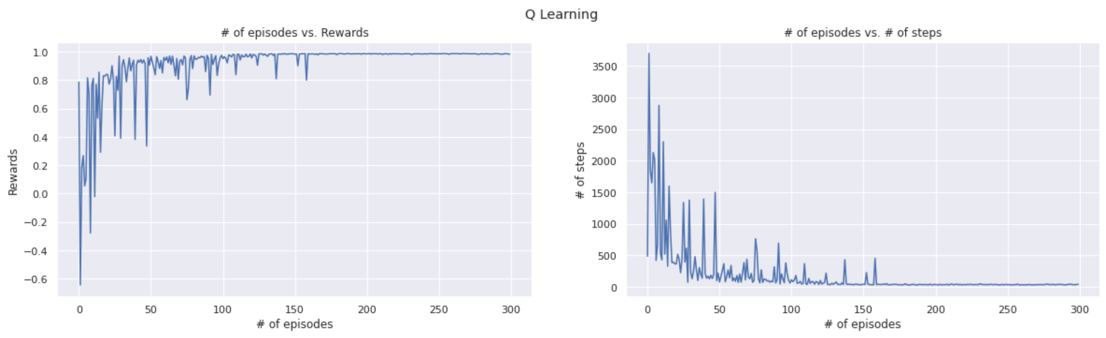

Figure 8: Experiment 4 (Q-Learning, Maze 15x15, $\varepsilon = 0.01$, $\gamma = 0.95$, $\alpha = 0.2$, 300 episodes)

Although we were satisfied with those result of experiment 4, we wanted to explore the effects of γ and α. In experiment 5, we set $\gamma = 0.99$ and $\alpha = 0.1$ to be in experiment 5. This experiment provided worse result comparing to the previous one, as it took the agent more episodes to become stable and it could not solve the maze in the middle of the learning process. We wanted to see if this was due to insufficient training, so in experiment 6 we increased the number of episodes to 500 (with other parameters the same as experiment 5), and got similar results as the ones observed in experiment 4, which can be reflected in Figure 9 that has similar characteristics to Figure 8.

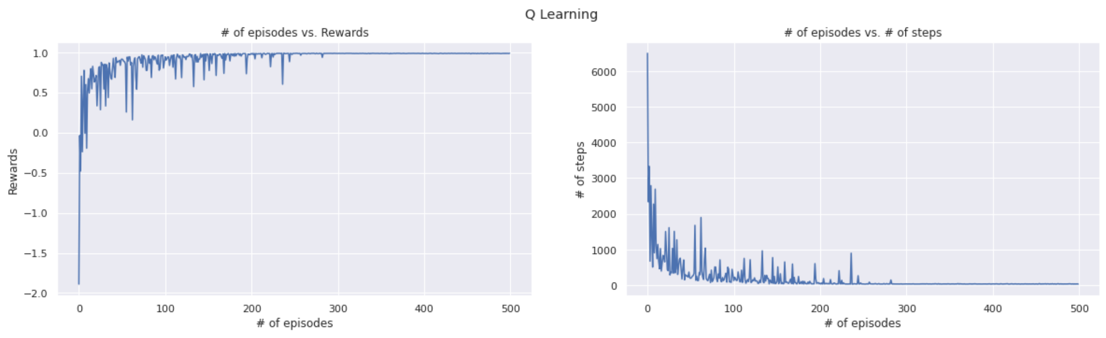

Figure 9: Experiment 6 (Q-Learning, Maze 15x15, $\varepsilon = 0.01$, $\gamma = 0.99$, $\alpha = 0.1$, 500 episodes)

3.4 SARSA

Due to the similarity between SARSA and Q-Learning, SARSA uses the same parameters as we mentioned in Section 3.3. We used similar, but not identical, values for the SARSA experiments and performed the same amount of experiments. As we could expect, changing parameters had the same influence at the results which were relatively similar. Starting with $\varepsilon = 0.2$, $\gamma = 0.9$, $\alpha = 0.2$, and 100 episodes, experiment 1 produces an unstable learning process, as presented in Figure 10. The agent could not solve the maze neither at on the middle of the process nor at the end, although it does improves along the episodes.

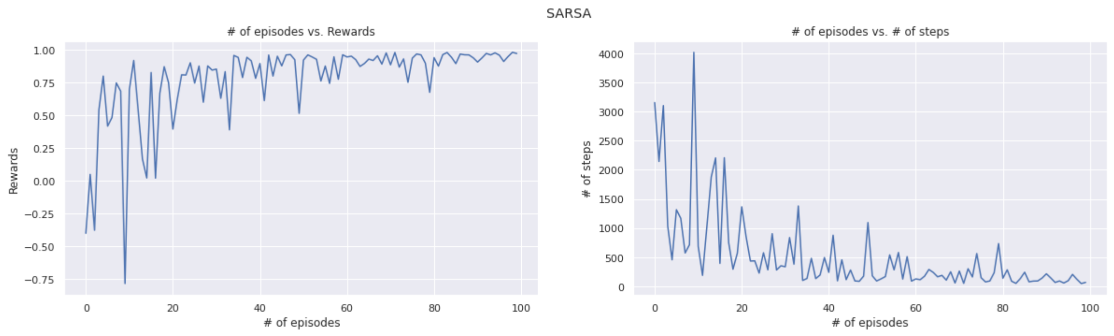

Figure 10: Experiment 1 (SARSA, Maze 15x15, $\varepsilon = 0.2$, $\gamma = 0.9$, $\alpha = 0.2$, 100 episodes)

In experiments 2-3 we stayed with the same parameters values but increased the number of episodes twice, to 200 and 300 respectively. Those changes showed us that higher number of episodes provides better results for the same parameters, regarding the average steps number and average rewards. In experiment 3, the agent even managed to solve the maze in the middle of the learning process although it experienced some struggles. For the next experiment we decreased both α and ε to 0.1 to test their influence, and the results are presented in Figure 11. These changes produced worse result, as it provided an average steps number of 332 (comparing to 208 in experiment 3), and an average reward of 0.852 (comparing to 0.907). For both experiments 3 and 4, the agent managed to solve the maze in the middle of the process, but it seems like it managed to do so only due to the stochastic model.

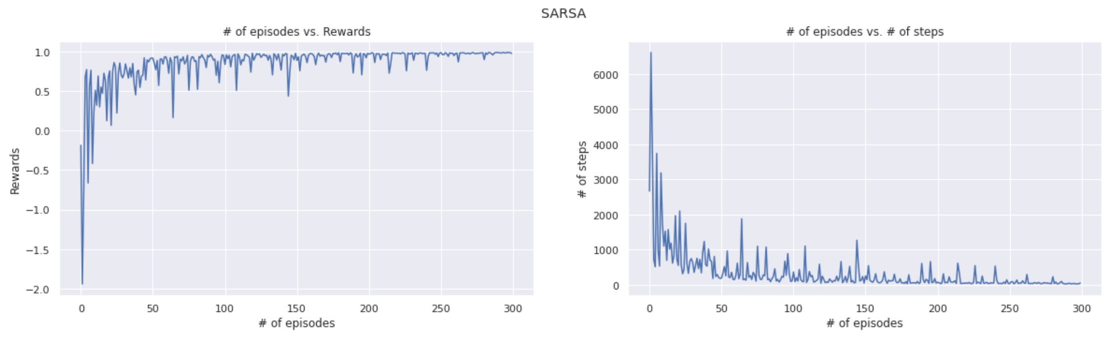

Figure 11: Experiment 4 (SARSA, Maze 15x15, $\varepsilon = 0.1$, $\gamma = 0.9$, $\alpha = 0.1$, 300 episodes)

In experiment 5 we only changed the γ comparing to experiment 4, and set it to be 0.99. There was an improvement comparing to experiment 4, but it was not significant, as experiment 3 still provided better results, with higher ε and α. The final experiment was the most successful one, which used the same parameters as experiment 5 along with an increase of the episodes number to 500. In Figure 12 we can see that the learning process of the agent becomes stable much earlier in the process, and we noticed that it can solve the maze in the middle of the process without relying on the stochastic model. The average steps number was 195 and the average reward was 0.913.

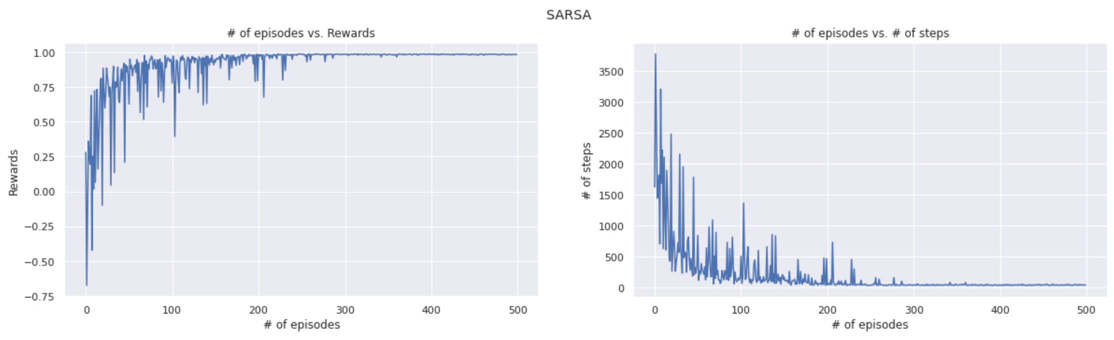

Figure 12: Experiment 6 (SARSA, Maze 15x15, $\varepsilon = 0.1$, $\gamma = 0.99$, $\alpha = 0.1$, 500 episodes)

3.5 Q-Learning 25X25

We chose to solve the 25x25 maze with the Q-Learning algorithm. We started the first experiments with the basic algorithm and provided it the following values: $\varepsilon = 0.1$, $\gamma = 0.9$, $\alpha = 0.1$, and 100 episodes. Not only did the learning process was not stable but it also did not present any improvement in terms of steps number and rewards, as we can see in Figure 13.

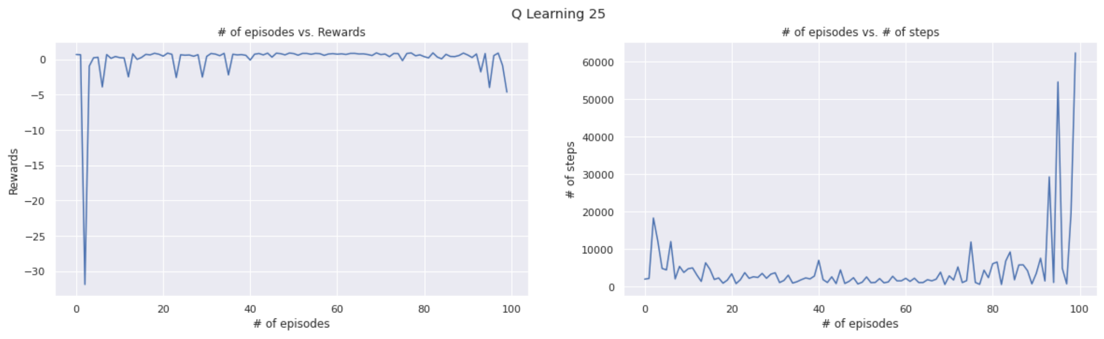

Figure 13: Experiment 1 (Q-Learning, Maze 25x25, $\varepsilon = 0.1$, $\gamma = 0.9$, $\alpha = 0.1$, 100 episodes)

In the next experiment, we kept using the basic algorithm and set $\varepsilon = 0.5$, to enable more exploration. We indeed saw a minor improvement, however, the agent still couldn't solve the maze at all. In experiment 3, we implemented ε-decay method, to enable much more exploration at the beginning, and reducing it as long as the agent completes more episodes. Starting with $\varepsilon = 0.5$ and decaying to 0.01, the agent presented results that were similar to the results of experiment 2, which are presented in Figure 14.

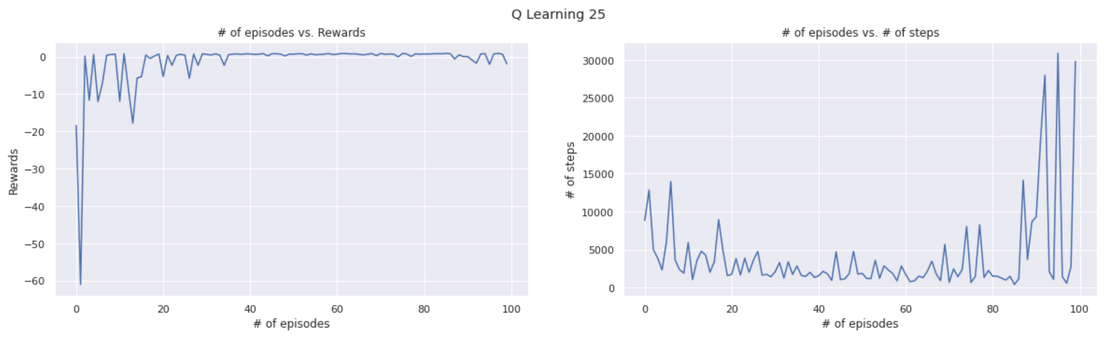

Figure 14: Experiment 3 (Q-Learning, Maze 25x25, $\varepsilon = 0.5$ to 0.01, $\gamma = 0.9$, $\alpha = 0.1$, 100 episodes)

Experiment 4 extended the setting of experiment 3, with $\gamma = 0.95$, $\alpha = 0.2$ and changing the rewards in to chosen cells. We wanted the agent to avoid the left part of the maze, to limit the amount of ways to the goal the agent will need to explore. Therefore, we set the reward of cell (2, 6) to be -5. Cell (5, 7) got a "positive" reward of 0.1. Although the learning process appears to be more stable, the agent still could not solve the maze. In experiment 5 we changed the location of the strategic cells and the rewards values, compared to experiment 4. We set the "negative" cell to be (2, 5) with a reward of -10, and the "positive" cell moved to (4, 4) with a reward of 0. Figure 15 presents the learning process of experiment 5, that appears stable with an average steps number of 755, and average reward of -2.652. This experiment was the first experiment in which the agent managed to solve the maze at the end of the process.

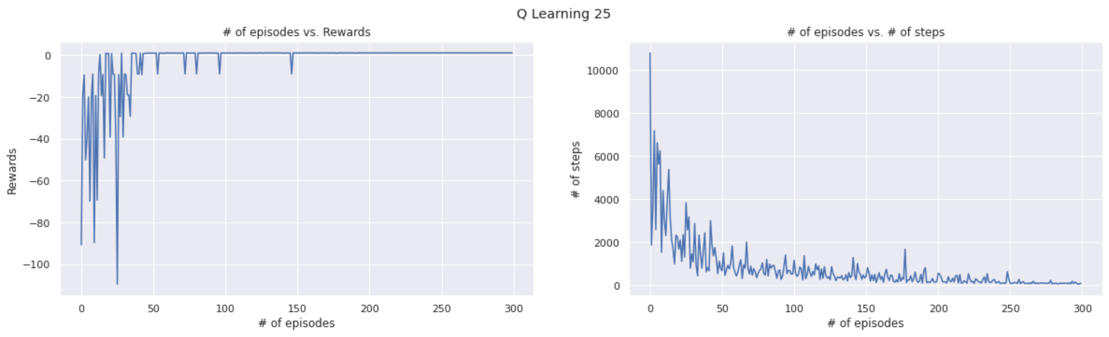

Figure 15: Experiment 5 (Q-Learning, Maze 25x25, $\varepsilon = 0.5$ to 0.01, $\gamma = 0.95$, $\alpha = 0.2$, 300 episodes)

We then realized that the cell with the "positive" reward needs to be located closer to the goal, and guide the agent to use the preferred path, which is virtually goes through the "diagonal" of the maze that connects the start cell and the goal cell. We chose cell (16, 17) with a reward of 0 for the "positive" cell and did not change the "negative" cell. The results presented in Figure 16 shows a better learning process that converges earlier, But at this point the agent still managed to solve the maze only at the end of the learning process.

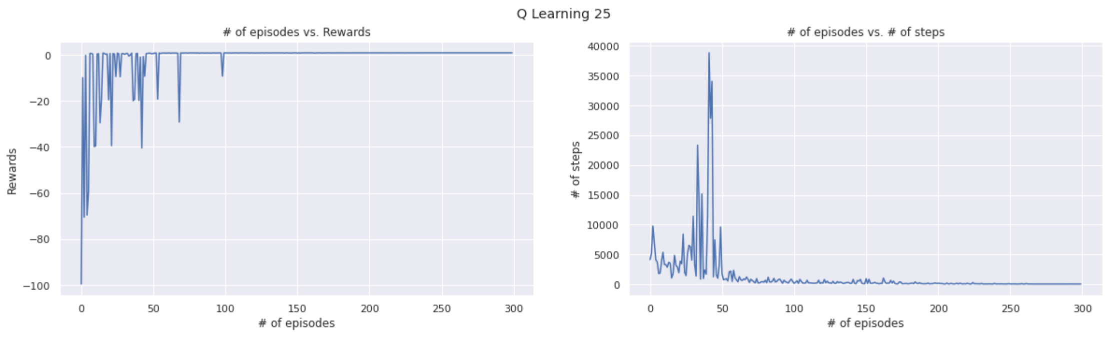

Figure 16: Experiment 6 (Q-Learning, Maze 25x25, $\varepsilon = 0.5$ to 0.01, $\gamma = 0.95$, $\alpha = 0.2$, 300 episodes)

Finally, we increased the number of episodes to 500 in experiment 7, to enable the agent to apply its knowledge. This, as expected, improved the learning process as shown in Figure 17. Also, the final video of experiment 7 emphasizes the agent's certainty in its knowledge. This experiment was also the first time that the agent managed to solve the maze, with some struggles, in the middle of the process, which is reasonable since it is equal to 250 episodes. In addition, the average reward was almost twice as better as the one provided by experiment 6.

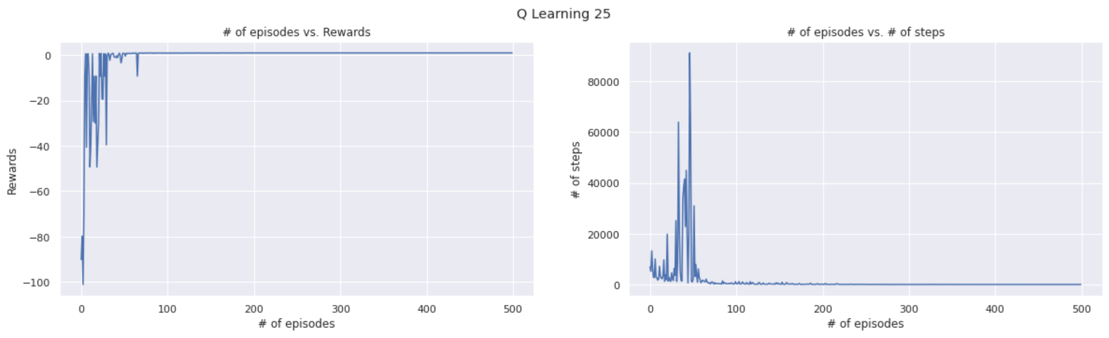

Figure 17: Experiment 7 (Q-Learning, Maze 25x25, $\varepsilon = 0.5$ to 0.01, $\gamma = 0.95$, $\alpha = 0.2$, 500 episodes)

4 Discussion

Dynamic Programming approach is more suitable to smaller versions of the maze problem, since it requires massive calculations for each state. Hence, it was only implemented and evaluated for the 5x5 maze version. Our experiments showed that for higher γ values we get more accurate and distinguishable results. In all of our cases, the agents were successful with the γ values we provided, but we assume that for larger mazes it will be a more significant factor, as we could see in the case of 25x25 maze comparing to the smaller versions. For the rest of the algorithms, we conclude that it took the agent about 100 episodes to make a decent learning process, and the experiments that allowed more training episodes helped the agent to improve its knowledge. Those experiment, understandably, were the ones that was more successful. In addition, we noticed that in some cases increasing the ε value allowed the agent to perform more exploration, and therefore also provide better results. For cases with higher amount of states such as the 25x25 maze, the basic Q-learning algorithm was not enough to solve the maze, and we were forced to implement and test some optimizations. In our case, using ε-decay technique, along with changing the rewards for some cells, helped us to reach a solution. In larger problems, however, it is likely that some extra measures will be required.

5 Code

You can access our public Colab notebook at the following link:

`https://colab.research.google.com/drive/1uBy5tjp8wKzgXZ2-xh-SQUSKpCsSQPww?usp=sharing`