Evaluation of Q-Learning and REINFORCE algorithms on the problem of maze-solving

Group 13:
Jan Burakowski (2764440), Mateusz Kielan (2774206),
Kamil Pulchny (2770523), Stanisław Wasilewski (2763732)

October 26, 2024

Abstract

Reinforcement Learning (RL) is a powerful approach for training agents to make sequential decisions in complex environments. This paper investigates the application of two RL algorithms, Q-Learning and REINFORCE, to the problem of maze navigation with a sub-goal and a final goal. The primary objective is to enable an agent to efficiently learn optimal policies for navigating the maze while balancing exploration and exploitation.

We implemented both Q-Learning, a value-based method, and REINFORCE, a policy-based method, within a 10×10 maze environment. The agent is required to reach a sub-goal before proceeding to the final goal, with rewards structured to encourage this behavior. Additionally, we introduced a revisit penalty in the REINFORCE algorithm to discourage the agent from revisiting the same states, promoting efficient exploration.

Our results demonstrate that both algorithms successfully learn to navigate the maze and reach the goal. REINFORCE showed faster convergence to the optimal policy at lower learning rates, benefiting from the revisit penalty that enhances exploration efficiency. Q-Learning exhibited greater stability at higher learning rates, effectively integrating new experiences while maintaining accumulated knowledge.

The study highlights the importance of hyperparameter tuning, particularly the learning rate, in achieving optimal performance for each algorithm. The findings suggest that while both Q-Learning and REINFORCE are effective for maze navigation tasks, their performance characteristics differ, making them suitable for different scenarios.

This aims to contribute to our understanding of how value-based and policy-based RL algorithms perform in structured environments with sub-goals. Future work may involve extending the complexity of the environment by adding multiple sub-goals and exploring additional RL algorithms to further enhance agent performance.

1 Introduction

Reinforcement Learning (RL) is an important area in artificial intelligence, especially for tackling problems that involve making a sequence of decisions in an environment. This research focuses on creating and evaluating RL agents that can solve a maze by balancing exploration and exploitation. The main objective is to enable an agent to navigate a maze efficiently, determining the correct sequence of steps to reach both a sub-goal and the final goal. Designing adaptive agents that can navigate static, goal-oriented environments is a key challenge in RL that this project addresses. The importance of our project lies in the fact that it has applications beyond academic. Maze-solving requires agents to learn how to avoid dead ends, optimize their paths, and reach specific destinations. Training agents in this controlled setting allows us to gain insights that can be applied to real-world scenarios where environments are hazardous or inaccessible to humans, and autonomous decision-making is essential. In this project, we successfully implemented and tested both Q-Learning and REINFORCE algorithms. These agents demonstrated their ability to adapt strategies within the maze by learning and converging to optimal policies. Our comparative study showed that both approaches have distinct advantages: Q-Learning excelled in stability, while REINFORCE achieved faster learning rates. This paper covers the mathematical foundations of the Q-Learning and REINFORCE algorithms. We then examine each agent's performance, focusing on the importance of learning rate and presenting results that illustrate their learning progress. Finally, we compare the number of time steps each algorithm took to converge to the optimal policy, providing insights into their efficiency.

2 Background

2.1 Maze Environment Introduction

In this project, we address the challenge of navigating an agent through a predefined 10×10 maze environment. The maze comprises walls, open paths, a sub-goal, and a final goal. Each cell in the maze can be:

1. **Wall (1)**: An unpassable barrier.
2. **Open Path (0)**: A passable space.
3. **Start Position (S)**: The agent's initial location.
4. **Sub-Goal (G)**: An intermediate goal the agent must reach to claim the reward from the final goal.
5. **End Goal (E)**: The final destination the agent aims to reach.

The agent begins at a specified starting position and must first reach the sub-goal before proceeding to the final goal. The maze's layout introduces complexity by introducing dead ends, and requires the agent to learn an optimal policy that accounts for both the sub-goal and the final goal.

2.2 Justification of Algorithm Choice

2.2.1 Overview of Possibilities

Reinforcement Learning (RL) offers a variety of algorithms suited for different types of problems. Some of the prominent algorithms include:

1. **Value Iteration**: A dynamic programming method that computes the optimal value function by iteratively updating value estimates until convergence.
2. **Policy Iteration**: Alternates between policy evaluation and policy improvement to find the optimal policy.
3. **Q-Learning**: An off-policy Temporal Difference (TD) learning algorithm that learns the optimal action-value function $Q(s, a)$ without requiring a model of the environment.
4. **Policy Gradients**: Optimize the policy directly by estimating the gradient of the expected reward with respect to policy parameters.
5. **Actor-Critic Methods**: Combine value-based and policy-based methods by maintaining both an actor (policy) and a critic (value function).

These algorithms can be broadly categorized into two strategies:

2.2.2 Two Different Strategies: Value-Based and Policy-Based Methods

Value-Based Methods aim to find the optimal value function and derive the policy from it. For example, in Q-Learning, the agent learns the optimal action-value function $Q^*(s, a)$ and selects actions that maximize $Q^*(s, a)$. These methods are well-suited for discrete action spaces and environments where the state and action spaces are manageable.

Policy-Based Methods optimize the policy directly. They parameterize the policy $\pi(a|s; \theta)$ and adjust the parameters θ to maximize the expected cumulative reward. These methods can handle continuous action spaces and stochastic policies, providing flexibility in complex environments.

2.2.3 Our Choice and Justification

We chose to implement Q-Learning and REINFORCE algorithms for the following reasons:

1. **Q-Learning**: As a value-based method, Q-Learning advantages are its simplicity and effectiveness in discrete environments. It does not require a model of the environment and can handle the stochastic part in maze navigation. Its tabular implementation is suitable for our maze's finite state and action spaces.
2. **REINFORCE**: This policy-based method allows us to directly optimize the policy. REINFORCE is particularly beneficial when dealing with stochastic policies and can potentially handle larger or continuous state spaces. Implementing REINFORCE provides insight into how policy optimization performs in comparison to value-based methods in the context of our maze problem.

   By exploring both strategies, we can compare their performance, convergence and stability and see which is more suitable for the given maze.

2.3 Q-Learning

2.3.1 Algorithm Overview

Q-Learning is an off-policy, model-free reinforcement learning algorithm that aims to learn the optimal action-value function $Q^*(s, a)$. The agent updates its estimates of $Q(s, a)$ based on the Bellman Optimality Equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

where:

1. s_t is the current state.
2. a_t is the action taken at state s_t.
3. r_{t+1} is the reward received after taking action a_t.
4. α is the learning rate ($0 < \alpha \leq 1$).
5. γ is the discount factor ($0 \leq \gamma \leq 1$).
6. $Q(s_t, a_t)$ is the current estimate of the action-value function.
7. $Q(s_{t+1}, a')$ is the estimate of future rewards from state s_{t+1}.

The agent selects actions using an ϵ-greedy policy, balancing exploration and exploitation.

$$a = \begin{cases} \underset{a'}{\text{argmax}}\ Q(s, a') & \text{with probability } 1 - \epsilon, \\ \text{random action from } A & \text{with probability } \epsilon. \end{cases}$$

2.3.2 Incorporating the Sub-Goal

To ensure the agent reaches the sub-goal before the final goal, we modify the reward function and environment dynamics:

1. **Reward Shaping**: When the agent reaches the sub-goal and final goal a significant positive reward is provided.
2. **State Tracking**: Maintain a flag indicating whether the sub-goal has been reached.
3. **Penalties**: Impose penalties for every action, encouraging the agent to find most optimal path.

By integrating these modifications, the agent is encouraged to prioritize reaching the sub-goal before proceeding to the final goal.

2.3.3 Mathematical Formulation

1. **State Representation**: Each state s corresponds to a cell in the maze, indexed as $s = (x, y)$, where x and y are the cell coordinates.
2. **Action Space**: The set of possible actions $A = \{\text{up}, \text{down}, \text{left}, \text{right}\}$.
3. **Q-Table Initialization**: $Q(s, a) = 0$ for all $s \in S$ and $a \in A$.
4. Update Rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s,a)\right]$$

   where s' is the next state after taking action a in state s, and r is the immediate reward received. This algorithm is effective in discrete state and action spaces. However, for the larger state spaces, it might face the "curse of dimensionalities" problem as the Q-table will be too large to process. Furthermore, the results indicate high sensitivity to hyperparameter selection.

2.4 REINFORCE

2.4.1 Algorithm Overview

REINFORCE is a Monte Carlo policy gradient algorithm that optimizes the policy directly by estimating the gradient of the expected return with respect to policy parameters θ. The objective is to maximize:

$$J(\theta) = E_{\pi_\theta}\left[R(\tau)\right]$$

where:

1. $\pi_\theta(a|s)$ is the policy parameterized by θ.
2. $R(\tau)$ is the total return of trajectory τ.

The policy parameters are updated using the gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

2.4.2 Policy Gradient Estimation

Using the Policy Gradient Theorem, the gradient of the expected return is:

$$\nabla_\theta J(\theta) = E_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R_t\right]$$

where:

1. $R_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$ is the discounted return from time t.
2. $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is the score function, indicating how the probability of action a_t in state s_t changes with θ.

2.4.3 Incorporating the Sub-Goal

To incorporate the sub-goal we introduced a higher reward for reaching the sub-goal and final goal. Furthermore, for every revisit of an already explored state agent is given a penalty. This method encourages the exploration of new states rather than revisiting old ones.

2.4.4 Policy Parameterization

Policy Parameterization For discrete actions, the policy can be parameterized using a softmax function:

$$\text{softmax}(x_i) = \frac{e^{x_i/T}}{\sum_j e^{x_j \ / \ T}}$$

Where x_i is the score for each action i.

2.4.5 Advantages and Limitations

1. **Advantages**:
   (a) Direct policy optimization.
   (b) Suitable for stochastic policies and continuous action spaces.
2. **Limitations**:
   (a) High variance in gradient estimates.

3 Methodology

3.1 Handling the Sub-Goal and Penalty for Revisiting Fields

To guide the agent towards first reaching the sub-goal and then the final goal efficiently, we employ reward shaping and introduce a penalty for revisiting states.

Reward Shaping We define the reward function as follows:

- **Sub-Goal Reward**: Assign a significant positive reward $r_{\text{sub-goal}} = +20$ upon reaching the sub-goal.
- **Final Goal Reward**: Assign a larger positive reward $r_{\text{final}} = +50$ upon reaching the final goal.
- **Step Penalty**: Apply a small negative reward $r_{\text{step}} = -0.1$ for each time step to encourage quicker solutions.
- **Revisit Penalty (Only for REINFORCE)**: Apply an additional penalty $r_{\text{revisit}} = -1.2$ when revisiting a state.

Revisit Penalty Implementation To discourage the agent from revisiting the same states, we maintain a visit count visit_count(s) for each state s during an episode. When the agent revisits a state (i.e., visit_count(s_t) > 1), we adjust the reward:

$$r_t \leftarrow r_t + r_{\text{revisit}}$$

This penalty reduces the attractiveness of paths that include loops or backtracking.

Impact on Implementation Including the revisit penalty has several benefits:

- **Efficient Exploration**: The agent is incentivized to explore new states rather than returning to previously visited ones.
- **Optimal Pathfinding**: By avoiding loops, the agent is more likely to find shorter and more direct paths to the goals.
- **Faster Convergence**: The agent learns more efficient strategies, which accelerates the convergence of the policy.
- **Balance of Exploration and Exploitation**: The combination of the ϵ-greedy strategy and revisit penalty ensures the agent explores sufficiently while learning to exploit optimal paths. Which will be further proven in the result section

3.2 Experimental Setup

Both the Q-Learning and REINFORCE algorithms are evaluated on three main categories:

1. **Reaching Sub-Goal / End-Goal**
2. **Average Cumulative Reward**
3. **Learning Stability**

During the simulation process, the learning rate is manipulated, and the algorithms are evaluated based on the categories mentioned above. The learning rate is varied across four values for both REINFORCE and Q-Learning: 0.01, 0.1, 0.5, and 0.8. This choice of learning rates is based on evaluating low rates (0.01, 0.1) that promote old rewards and experiences by making smaller updates to the policy, and a high learning rate (0.8) that promotes new rewards and experiences by making larger updates. The value of 0.5 is intended to serve as a middle ground between small and large update sizes. Each simulation was run for 4000 episodes, as this was the optimal number to show convergence.

3.3 Explanation of Evaluation Parameters

Average Cumulative Reward Average cumulative reward is calculated by first summing the rewards over time and then dividing the cumulative sum by the number of episodes completed up to a certain point.

$$\text{Average Cumulative Reward at episode } t = \frac{1}{t}\sum_{i=1}^{t} R_i$$

where R_i is the reward obtained in episode i, and t is the number of episodes completed. We chose to use this metric as it directly represents the agent's performance after a number of episodes. The higher the reward, the better the agent performs.

Learning Stability For the evaluation of learning stability, we use the moving average.

$$\text{Moving Average at episode } t = \frac{1}{W}\sum_{i=t-W+1}^{t} R_i$$

where W is the window size, R_i is the reward at episode i, and t is the current episode. By taking the average over a fixed window size, we can disregard short-term fluctuations in the training process and observe the long-term trends over the set window size. Both the moving average and average cumulative reward are plotted at the end of the simulation.

Reaching Sub-Goal or End-Goal Reaching the sub-goal or end-goal is visualized using heatmaps. The heatmaps show the maze environment with a color-coded scheme that indicates the number of times the agent visited each state during the n episodes.

4 Results

After the simulations described above, the following results were obtained for Q-Learning.

For the learning rate of 0.01, the agent was able to reach the sub-goal; however, due to the small learning rate, it got stuck in a local maximum and wasn't able to reach the end goal, yielding sub-optimal results. Figure 1 illustrates the heatmap showing the agent being stuck at the sub-goal towards the end of the simulation. The average cumulative reward and moving cumulative reward show results where the reward converges on a sub-optimal value (Figure 2).

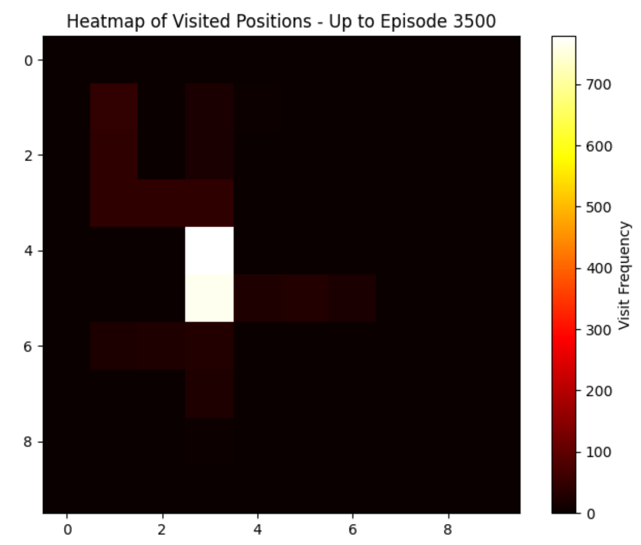
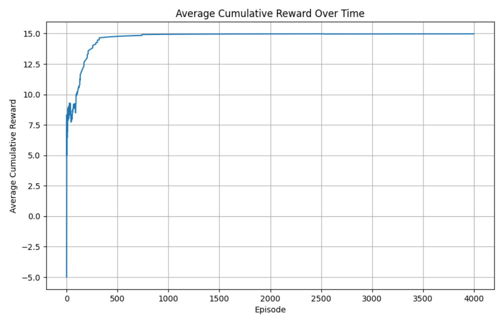
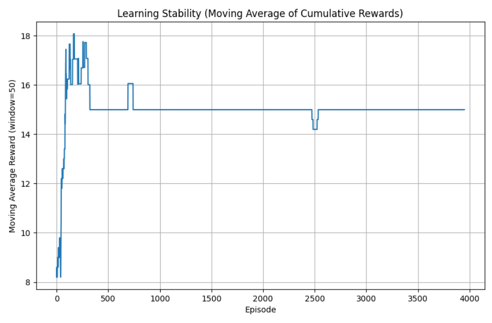


Figure 1: Heatmap of Q-Learning agent with learning rate 0.01.



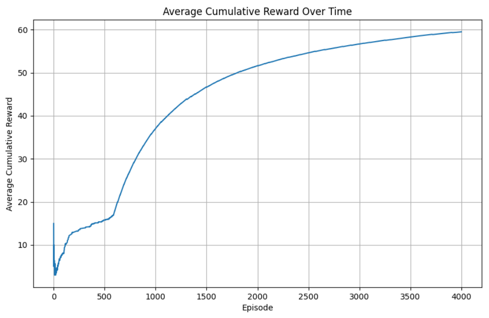
(a) Average cumulative reward.



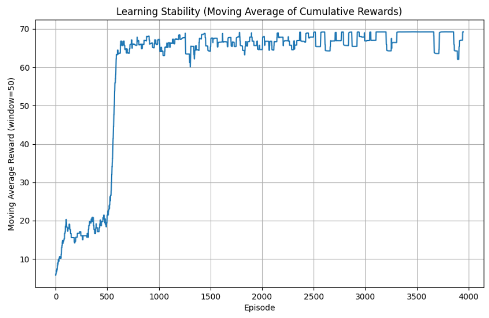
(b) Moving average reward.

Figure 2: Average cumulative reward and moving average of Q-Learning agent with learning rate 0.01.

When testing the Q-Learning algorithm with a learning rate of 0.1, we see the agent find the optimal solution. The average cumulative reward increases at the initial stages and then converges closer towards the 4000th episode (Figure 3). The fluctuations in learning stability indicate that the algorithm still tries to explore after finding the optimal path.



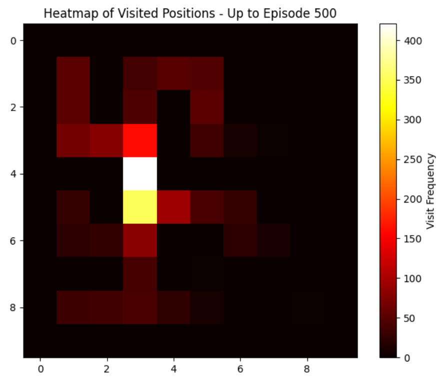
(a) Average cumulative reward.



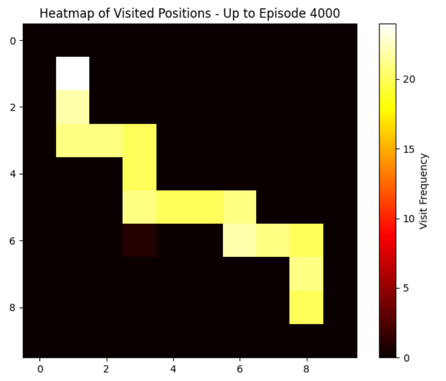
(b) Moving average reward.

Figure 3: Average cumulative reward and moving average of Q-Learning agent with learning rate 0.1.

The heatmap trace shows that the agent finds the optimal path to the end goal. At the beginning, the agent promotes exploration, as seen in the coloring of the heatmaps in Figure 4. Towards the end, the agent promotes exploitation when it finds the optimal path and stays on it. Furthermore, it can be seen that the agent reaches the sub-goal and the end goal, as those parts are significantly highlighted. The fluctuations in learning stability indicate that the algorithm still tries to explore after finding the optimal path.



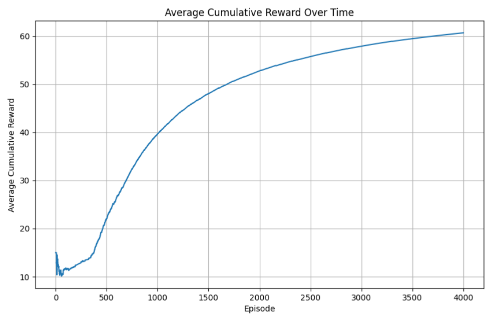
(a) Heatmap until episode 500



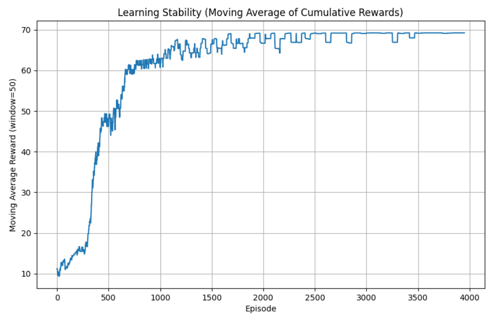
(b) Heatmap until episode 4000

Figure 4: Heatmaps for Q-Learning agents with learning rate 0.1

For the learning rates of 0.5 and 0.8 the agent still manages to converge to the optimal solution however it does so slightly later on than the agent with the 0.1 learning rate. Furthermore, on the (Figure 5 and 6 it can be seen that the learning stability is fluctuating more during the initial process of optimization. The heatmats look similar as the agent still manages to find the optimal solution.

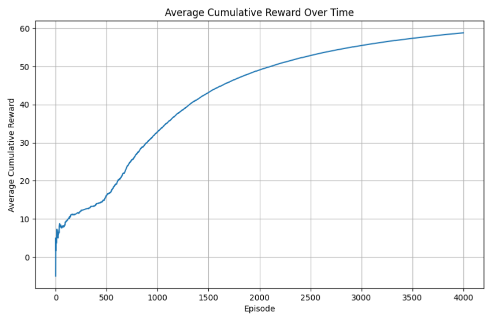


(a) Average reward over time

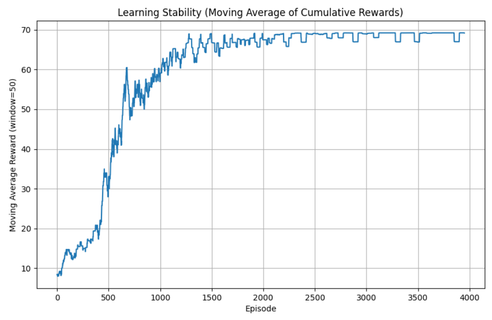


(b) Average stability

Figure 5: Average reward over time and average stability for Q-learning and learning rate of 0.5



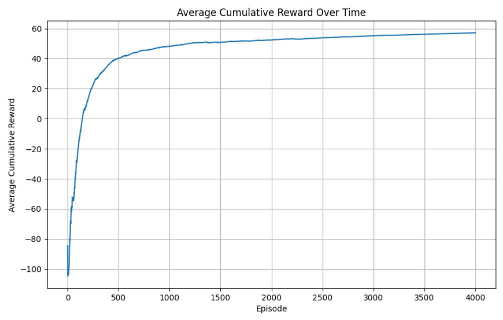
(a) Average reward over time



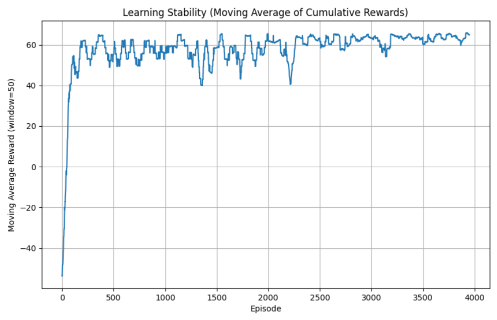
(b) Average stability

Figure 6: Average reward over time and average stability for Q-learning and learning rate of 0.8

For the REINFORCE algorithm, the results were slightly different. For the lowest learning rate of 0.01, the algorithm was able to perform optimally (APPENDIX C). It is crucial to mention that REINFORCE was able to converge on the optimal solution much quicker than Q-Learning. Faster convergence could occur due to our revisit penalty implemented in REINFORCE, which encouraged the agent to find optimal solutions more quickly. This further explains why, in the beginning, the agent is performing with a negative reward as it revisits the states often. Furthermore, the heatmap in Figure 8 suggests that even after the agent finds the optimal path around the 500th episode, it still explores further to try to optimize.



(a) Average reward over time



(b) Average stability

Figure 7: Average reward over time and average stability for REINFORCE with learning rate of 0.01

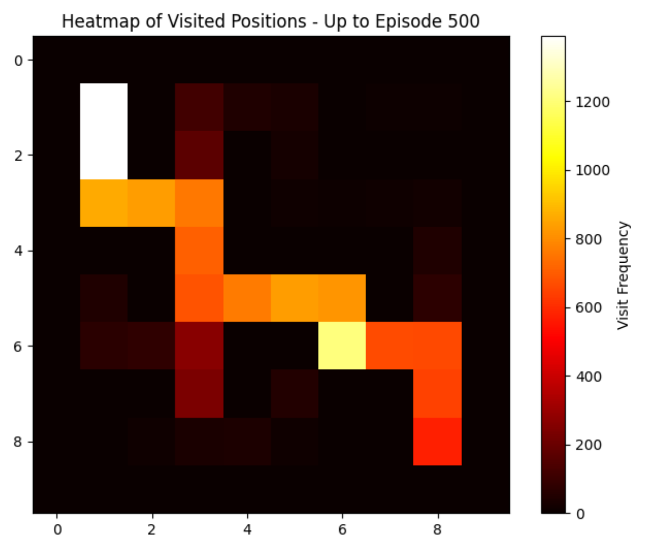
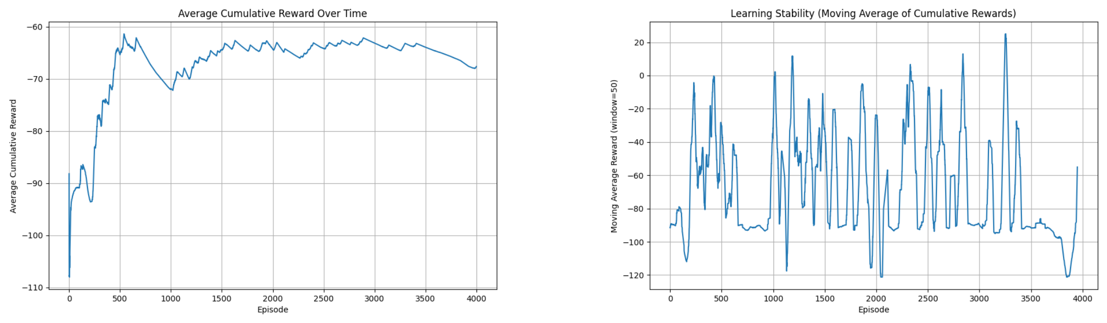


Figure 8: Heatmap for REINFORCE after 500 episodes.

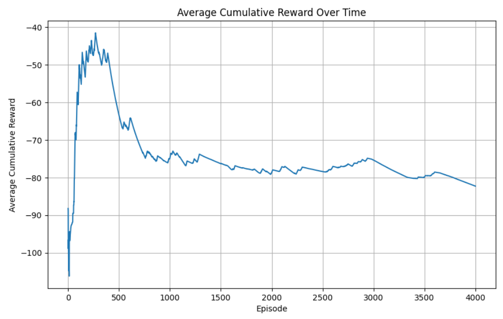
For the learning rates of 0.1 (Figure 9), 0.5 (Figure 10)), and 0.8 (Figure 11), the agent significantly underperforms with respect to the optimal solution. This could happen because the REINFORCE algorithm is already a high-variance algorithm, and increasing the learning rate too high may cause excessive variation. This can be seen in the moving average, where the values have high instability and fluctuate.



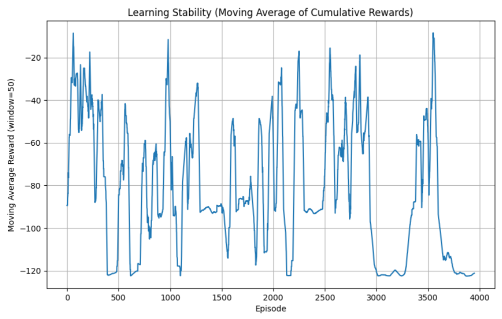
(a) Average reward over time

(b) Average stability

Figure 9: Average reward over time and average stability for REINFORCE with learning rate of 0.1

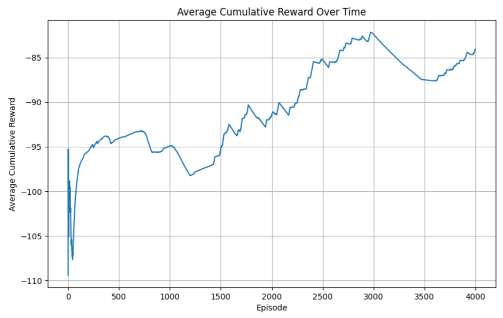


(a) Average reward over time

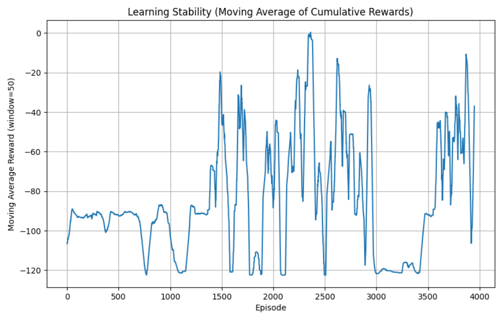


(b) Average stability

Figure 10: Average reward over time and average stability for REINFORCE with learning rate of 0.5



(a) Average reward over time



(b) Average stability

Figure 11: Average reward over time and average stability for REINFORCE with learning rate of 0.8

5 Conclusion

In this study, we explored the effectiveness of two reinforcement learning algorithms, Q-Learning and REINFORCE, in navigating an agent through a maze environment with a sub-goal and a final goal. The objective was to assess each algorithm's ability to learn optimal policies that balance exploration and exploitation while efficiently reaching both goals.

The results show that both algorithms were able to find the optimal solution to this maze environment. However, REINFORCE was able to find the optimal path much quicker than Q-learning. Furthermore, the study found that reinforce performs much better at very low learning rates: < 0.1 whereas Q-learning performs better in higher learning rates: > 0.1. As a further study, we suggest introducing several sub-goals to the environment. Furthermore, the Q-learning could be compared to actor-critic or to algorithms utilizing deep neural networks such as deep Q-learning.

6 Individual Contributions

Everyone contributed equally to the project. Specific contributions are explained below, in alphabetical order.

- Jan Burakowski: Report (Methodology, Background, Conclusion), Programming (Q-Learning, REINFORCE).
- Mateusz Kielan Report (Results, Conclusion, Methodology), Programming (Environment, Q-Learning).
- Kamil Pulchny: Report (Introduction, Results, Methodology), Programming (Q-Learning, REINFORCE).

- Stanisław Wasilewski: Report (Background, Conclusion, LaTex formatting), Programming (Environment, REINFORCE).