Comparative Analysis of Value Iteration and Q-learning for Solving the Maze Problem

Group 18: Joana Petkova, Mahbod Tajdini, Selman Gül, Ionut Moise

1. Introduction

This report details our solution to a version of the maze problem in which the agent must visit a sub-goal before reaching the final goal in a 10x10 maze. We employed two reinforcement learning algorithms to solve this problem: Value Iteration and Tabular Q-learning. Value Iteration goes through all state-action pairs to find an optimal policy. Since it checks every possible combination, it resembles a brute-force approach. Thus, we use it as a baseline for comparison with Tabular Q-learning, which uses a smarter exploitation vs. exploration strategy. Tabular Q-learning incrementally improves its policy by interacting with the environment and focusing only on relevant state-action pairs.

We begin by detailing the methodology for implementing both the environment and the algorithms. Then we present the results obtained for each algorithm and compare the two algorithms. Finally, we discuss the broader implications of our findings to more complex real-world scenarios.

2. Methodology

2.1. Environment

In this project, the environment is modeled as a 10x10 grid maze where the agent interacts with every distinct cell. Each cell in the maze is either a wall (not passable) or an open path (passable). The maze has key elements, such as:

- **Start point:** The agent's starting position.
- **Sub-goal:** An intermediate objective the agent must reach before the final goal.
- **End goal:** The agent's destination and final goal.

The main maze layout used for this project is shown in Figure 1, with the walls colored in black, the start point in yellow, the sub-goal in blue, and the end goal in green. This layout is discrete, fixed, and fully observable, enabling the agent to have complete knowledge of the environment.

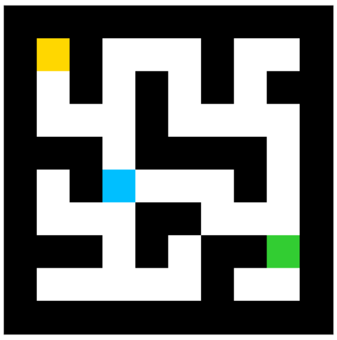

Figure 1. The Maze Layout.

An alternative maze layout with the same characteristics but a different sub-goal position is also used to verify the correctness of the reinforcement learning algorithms. It can be seen in Appendix A.

2.1.1. Encoding

Each position in the maze is encoded to a unique state. Without considering the sub-goal, the grid consists of 100 states, one for each cell. The agent's state is encoded as:

$$State = \frac{x \times 10}{y}$$

where x is the row index and y is the column index of the agent's current location. This one-dimensional encoding allows the agent to navigate through the environment using a structured state space.

The state space becomes 200 states when the sub-goal is introduced, taking into consideration whether the sub-goal has been visited or not. This added complexity allows the agent to track both its physical location and whether it has accomplished the sub-goal. The agent's ability to distinguish between these expanded states improves its handling of multi-step objectives by enabling it to modify its strategy based on its progress.

2.1.2. States & Actions

In the environment, each square corresponds to a unique state, and the maze itself is structured as a 2D list where each element represents either a wall or an open path. The agent has four possible actions: up, down, left, right.

State transitions are deterministic, meaning the agent's next state is entirely predictable based on its current state and selected action. If an action leads the agent into a wall or outside the maze boundaries, it remains in its current state. This reinforces the notion that the environment is fully observable and foreseeable, which guarantees that the agent has complete control over its movements. For instance, if the agent is at the start position (1, 1) and chooses to move right, it will transition to position (1, 2) unless obstructed by a wall. Similarly, moving down from (1, 1) will lead to position (2, 1). These predictable state transitions are implemented to ensure consistent outcomes for every action.

2.1.3. Rewards

The rewards are essential for shaping agent learning in the case of reinforcement learning algorithms. In our environment, each action generates a distinct reward, which is subsequently used to adjust the agent's behavior through methods like Value Iteration and Tabular Q-learning. The following rewards and penalties exist in our environment:

- **Step penalty** (-0.1): This functions as a time penalty, encouraging the agent to minimize the number of steps and driving it toward a more efficient policy.
- **Wall penalty** (-1); This provides strong negative reinforcement, teaching the agent to avoid invalid actions.
- **Sub-goal reward** (+1): This serves as positive reinforcement for achieving intermediate success, helping the agent decompose the task into smaller objectives.
- **End-goal reward** (+10): As the largest positive reinforcement, this reward is only attainable if the agent first reaches the sub-goal before arriving at the end goal. This ensures that the agent follows a step-by-step approach to complete the tasks.
- **Premature end-goal penalty** (-10): This implements negative reinforcement for failing to follow the proper task sequence, ensuring that the agent does not bypass critical intermediate objectives.

Penalties offer negative feedback to deter undesirable behavior, while positive reinforcements promote goal-seeking behavior. By balancing immediate and future benefits, the agent can develop the most effective policy, leading to strategic behavior and gradual maximization of the cumulative rewards.

The only modification we made to incorporate the sub-goal in the learning process of the agent was in the environment's rewards. By adding a reward for reaching the sub-goal and a penalty for reaching the end goal without first visiting the sub-goal, we ensure the agent learns the desired behavior without altering the reinforcement learning algorithms themselves. This behavior mirrors the requirements of more complicated real-world problems, where multiple goals or benchmarks need to be achieved in a specific order in order to complete a task. To ensure robustness, we tested the algorithm with various sub-goal positions, confirming the effectiveness of the rewards and penalties across different scenarios.

2.2. Reinforcement Learning Algorithms

2.2.1. Value Iteration

Value iteration is a dynamic programming technique used to solve Markov Decision Processes (MDPs). It turns the Bellman optimality equation for the state-value function, shown below, into an update rule to obtain the optimal value function.

$$V^*(s) = max_{a\in A} \left[R(s,a,s') + \gamma \sum_{s'\in S} P(s'|s,a)V^*(s') \right]$$

In this formula:

- $V^*(s)$ is the value of state s;
- $R(s, a, s')$ is the reward for transitioning from s to s' given action a;
- γ is the discount factor, which signifies the importance of future rewards;
- $P(s'|s, a)$ is the transition probability from state s to s' given action a;
- $V^*(s')$ is the value of state s'.

Since our environment is deterministic, the transition probability $P(s'|s, a)$ becomes 1 for a single specific next state s' and 0 for all others. This means that, for any action a, the agent deterministically moves to a particular neighboring state. For instance, if the agent takes the action for moving left, it will always move to the block directly to the left of its current position. Therefore, in our case, the Bellman optimality equation reduces to:

$$V^*(s) = max_{a \in A} [R(s, a, s') + \gamma V^*(s')]$$

Based on the formula, the Value Iteration algorithm determines the optimal value function through iteratively updating the value of each state by selecting the action that maximizes the immediate reward plus the value of the next state, assuming the best possible actions will be taken in all future states.

The process begins by initializing the value function for each state to zero. Then, the algorithm applies the update rule derived from the Bellman optimality equation to refine the value function. To be more specific, at each iteration, the algorithm computes Q-values for each possible action in each state, which correspond to the expected reward for taking that action in the given state. The value of each state is then updated based on the maximum Q-value, which represents the best expected outcome. The computation of the Q-values of all state-action pairs and the updates of the state values is carried out repeatedly until convergence.

Convergence is reached when further iterations do not significantly change the value function, indicating that the values are stable and represent the best possible outcome. In our implementation, the parameter θ signifies the smallest change in the value function that is considered significant. When the maximum value change in an iteration falls below this threshold, the algorithm stops as further improvements are negligible. In our case, we set θ to the very small value of 1×10^{-4} to ensure high precision in the value function at the cost of longer computation time. The stabilized value function obtained after convergence is used to derive the optimal policy.

The Value Iteration algorithm is chosen as the baseline method because it resembles a brute-force approach. In each iteration, all possible state-action pairs have to be computed, which means the algorithm is slower than other methods, especially for large state spaces. However, for the moderate 200-state space of our problem, Value Iteration is certain to converge within a reasonable time, although likely not as fast as Tabular Q-learning. Moreover, our maze is a discrete environment with known reward and state-transition functions, which makes the problem suitable for application of a model-based approach like Value Iteration. Value Iteration is also guaranteed to reach an optimal policy, making it a reliable benchmark for comparison against other more efficient methods, such as Tabular Q-learning. By using Value Iteration as a baseline, we can compare the performance of Tabular Q-learning not only in terms of policy quality but also in terms of convergence speed.

2.2.2. Tabular Q-learning

Tabular Q-learning is a model-free reinforcement learning approach, which uses a balance of exploration and exploitation to solve decision-making problems. The agent learns from its interactions with the environment, utilizing each action taken in a state, the resulting new state, and the received reward to improve its behavior. The goal is to maximize cumulative rewards by updating a Q-table, which stores the expected future rewards (Q-values) for each state-action pair. Initially, the Q-table is filled with zeros for all state-action pairs. Then, for a set amount of episodes, the Tabular Q-learning algorithm updates the Q-values in the table using the following update rule, based on the Bellman optimality equation for the action-value function:

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

In this formula:

- $Q(s, a)$ is the current estimate of the expected cumulative reward for taking action a in state s;
- α is the learning rate, determining the speed at which the agent learns from new experiences;
- r is the immediate reward received after taking action a in state s;
- γ is the discount factor, which signifies the importance of future rewards;

- $Q(s', a')$ is the future reward the agent can expect from the next state s' when taking the best action a';
- $[r + \gamma\, max_{a'} Q(s', a') - Q(s, a)]$ represents the Temporal Difference error, which measures how much the agent's prediction of the reward (the old Q-value) differs from what it actually observes.

In our implementation, the learning rate α is fixed at 0.1. This value ensures that the agent makes moderate incremental updates to its Q-values, allowing for stable learning and preventing overreaction to noisy or one-off rewards. A very high learning rate would cause rapid changes in the Q-values in response to each new reward, which leads to instability in the learning process. On the other hand, a very low learning rate would cause very small changes in the Q-values in response to each new reward, which leads to a high amount of episodes needed for convergence. A moderate learning rate such as 0.1 is suitable in many reinforcement learning settings where the reward signals might contain short-term noise but reflect long-term trends.

A key challenge in Q-learning is managing the trade-off between exploration and exploitation. Exploration entails trying new actions to discover their rewards, whereas exploitation involves choosing actions that are known to yield high rewards. To handle the trade-off, we used the ϵ-greedy strategy, which introduces a parameter ϵ to regulate the agent's tendency to explore. The agent takes a random action with probability ϵ (exploration), while with probability $1 - \epsilon$, it chooses the action with the highest Q-value for the current state (exploitation).

Given the significance of the exploration rate ϵ in determining the agent's learning behavior, we used the Optuna library for hyperparameter optimization to automatically tune ϵ. An Optuna study was created to search for the optimal ϵ within the defined range of $[0.01, 0.99]$ by proposing candidate values and evaluating the learning of the agent. The objective was to minimize the number of episodes required for the agent to converge to an optimal policy. The episode of convergence was estimated by taking the mean of the cumulative rewards obtained by the agent over each 50 episodes after a 100 episodes had passed and comparing it to the mean of the cumulative rewards obtained by the agent over the previous 50 episodes. If the difference between the two means was lower than the threshold of 0.01, the algorithm was considered to have converged. Running the Optuna study with 1000 trials of 1000 episodes resulted in an optimal ϵ value of 0.05. The study also helped us determine an appropriate number of episodes for running the Q-learning algorithm, ensuring that the obtained plots are manageable in size and clearly show that convergence is achieved.

Tabular Q-learning was chosen for solving the maze problem for several reasons. As a model-free algorithm, it does not require prior knowledge of the environment's model. This makes Tabular Q-learning highly suitable for a wide variety of environments, including our maze scenario, where the agent can learn directly from trial and error without access to transition probabilities or reward structures.

In addition, given sufficient exploration and learning time, Tabular Q-learning converges to an optimal policy, even if the initial policy is far from optimal. Its exploration-exploitation strategy enables the agent to focus on the most relevant parts of the state space, making the algorithm more efficient even in environments with larger state spaces. As a result, Tabular Q-learning is expected to find an optimal solution for the moderate state space of the maze faster than Value Iteration, which requires exhaustively calculating the values of all possible state-action combinations. This makes Q-learning more practical for real-world applications, where exhaustive search methods are often infeasible. It is precisely these characteristics that make examining Tabular Q-learning's performance in the maze problem both relevant and important.

However, Tabular Q-learning also has some limitations. Storing all state-action values in a Q-table can become memory-intensive in environments with large state or action spaces, although this is not a concern in our maze problem. Additionally, Tabular Q-learning is restricted to discrete environments (such as the maze), since each Q-value must correspond to a specific state-action pair. Finally, the performance of Q-learning is sensitive to the selection of its hyperparameters—in our case, the learning rate α and the exploration rate ϵ. If these parameters are poorly chosen, it can lead to suboptimal policies and slower convergence. Despite this, we believe that the benefits of Tabular Q-learning outweigh its limitations in the case of our problem.

2.2.3. Common Parameters for the Value Iteration and Tabular Q-learning Algorithms

The discount factor γ is set to 0.95 for both the Value Iteration and Tabular Q-learning algorithms to ensure an unbiased comparison between them. A discount factor of 0.95 encourages the agent to focus on exploring more longer-term strategies, rather than prioritizing immediate rewards. This is crucial for our problem, where a sub-goal must be visited before reaching the final goal. Setting the discount factor too low would make the agent too short-sighted, while a value really close to 1 could hinder learning by over-emphasizing distant rewards. Thus, $\gamma = 0.95$ provides a good compromise, which accounts for the need to prioritize visiting the sub-goal before the end goal in our agent's decision-making.

Additionally, a random seed was set in the Jupyter notebook used to organize the project code to ensure that the experiments are reproducible. The seed influences the sampling of the actions from the environment, the ϵ-greedy strategy of the Q-learning algorithm, and the Optuna study's search for the optimal ϵ value. The policies learned by the Value Iteration and the Tabular Q-learning algorithm are deterministic; hence, they are not affected by the seed. The random seed value used to create the plots and visualizations in the Results section of this report is 25.

3. Results & Analysis

In this section, we present the results of applying both the Value Iteration and Tabular Q-learning algorithms to the maze problem. We tested each algorithm on two versions of the environments: one with the position of the sub-goal given in the instructions (visualized in Figure 1) and one with an alternative position of the sub-goal (visualized in Appendix A). This approach was used to validate that the two algorithms work correctly and visit the sub-goal regardless of its position. However, in this report, we only go into detail for the environment with the position of the sub-goal given in the instructions, as it represents the main task for the project.

We begin by analyzing the performance of each algorithm individually, focusing on convergence plots, visualizations of the policy execution, and the overall learned policy. Then we compare the two algorithms, evaluating their learned policies and the time required to converge to an optimal solution. By examining these aspects, we aim to highlight the strengths and limitations of each approach in the context of our problem.

3.1. Value Iteration

The results of applying the Value Iteration algorithm to the maze problem are summarized by examining its convergence behavior, execution of the learned policy, and visualization of the overall learned policy.

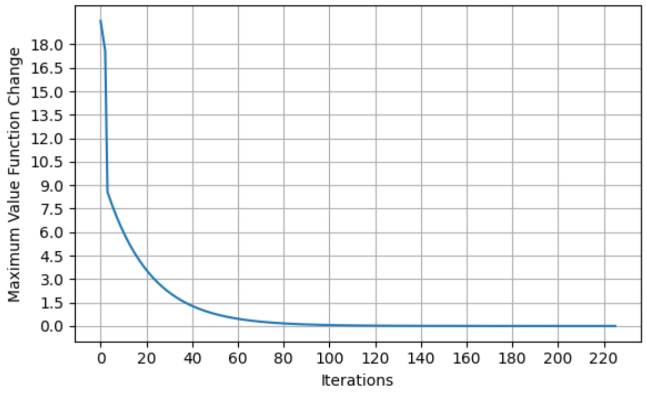

Figure 2. Maximum Change in the Value Function per Iteration.

Figure 2 shows the maximum change in the value function across iterations for the Value Iteration algorithm. The algorithm exhibits rapid convergence, with the maximum value function change decreasing sharply in the first few iterations and then gradually approaching zero. The value function stabilizes after approximately 100 iterations. However, the algorithm continues improving the value function until the maximum changes fall below the threshold set by the θ parameter and finally reaches convergence around iteration 225. Such results are observed independently of the value of the random seed set in the Jupyter notebook. The absence of fluctuations in the maximum value function change demonstrates stable learning through all iterations. Given the steep drop in Figure 2, the convergence speed of the algorithm is relatively fast, which is expected given the moderate size of the state space in our maze problem. Although these results are promising, convergence speed is challenging to assess without comparison; thus, we compare Value Iteration with Tabular Q-learning at the end of the Results section.

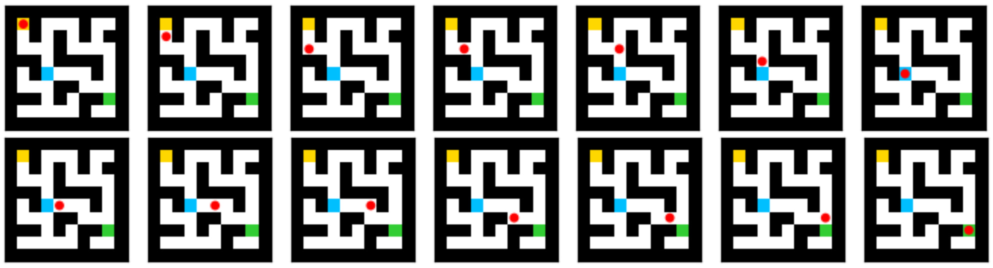
Figure 3. Execution of the Policy Learned by the Value Iteration Algorithm.

Figure 3 illustrates the agent's trajectory in the maze, as it follows the policy to which the Value Iteration algorithm converged. The red dot represents the agent, black squares denote walls, the yellow square indicates the starting position, the blue square marks the sub-goal, and the green square designates the end goal. The agent follows the shortest path that passes through the sub-goal before reaching the final goal. This behavior is observed, regardless of the value of the random seed. Hence, the agent always succeeds at finding the optimal solution, which is to be expected, as the Value Iteration algorithm does not entail any stochasticity.

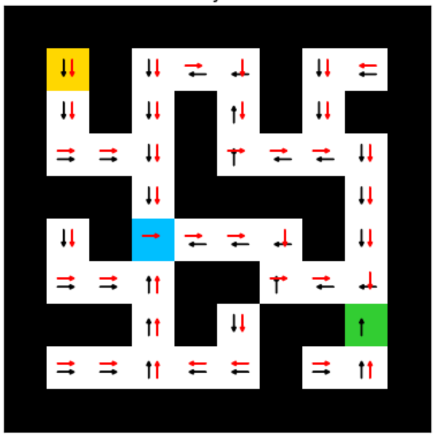
Figure 4. Visualization of the Overall Policy Learned by the Value Iteration Algorithm.

The overall policy learned by the Value Iteration algorithm is visualized in Figure 4, with black arrows indicating optimal actions for states before the sub-goal has been reached and red arrows indicating actions for states after the sub-goal has been visited. The policy clearly guides the agent first toward the sub-goal (blue square) and subsequently toward the final goal (green square), demonstrating effective learning by the algorithm. In fact, the visualization confirms that Value Iteration has calculated the optimal actions for all positions in the maze, a result of its exhaustive exploration of all state-action pairs to derive the optimal value function. This overall policy remains consistent across different runs, regardless of the random seed value set at the beginning of the Jupyter notebook.

For further validation of the algorithm, we also tested it in an alternative maze layout where the sub-goal is located at a different position. The results, presented in Appendix B, illustrate that the Value Iteration algorithm still converges to an optimal policy, which guides the agent from the start position to the goal via the new sub-goal location. The algorithm's convergence, learned optimal path, and overall policy remain consistent across different seeds. Hence, Value Iteration successfully solves both versions of the maze environment, demonstrating its robustness.

3.2. Tabular Q-Learning

For the Tabular Q-learning algorithm, we experimented with two different parameter settings. Initially, we ran Tabular Q-learning with a random exploration rate (ϵ) of 0.45. After that, we optimized the exploration rate using an Optuna study, aiming to enhance the convergence speed, and ran Tabular Q-learning again with the optimized exploration rate value of 0.05. Both configurations were tested for 250 episodes, which was sufficient for convergence in trials conducted across multiple random seeds. The results for each case are detailed below.

3.2.1. Tabular Q-learning with Random Exploration Rate

The results of applying the Tabular Q-learning algorithm with a random exploration rate ($\epsilon = 0.45$) are summarized by examining its convergence behavior, execution of the learned policy, and visualization of the overall learned policy.

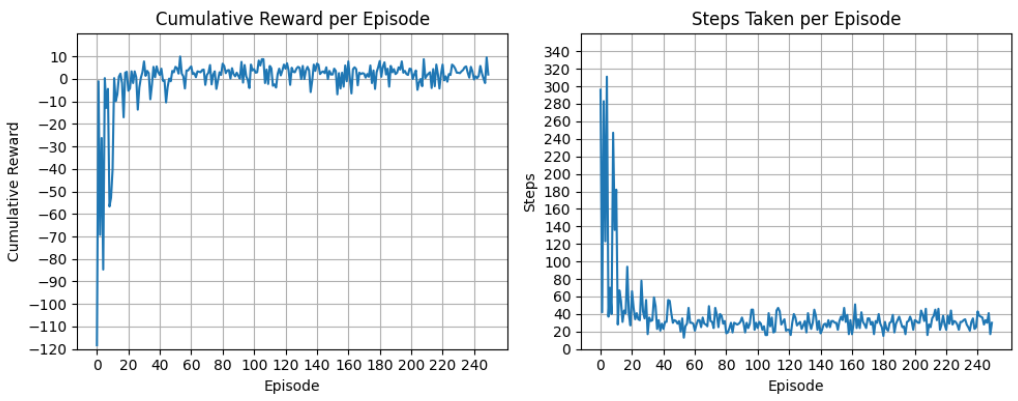

Figure 5. Cumulative Reward and Steps Taken per Episode by the Q-Learning Algorithm With Random Exploration Rate.

Figure 5 illustrates the cumulative reward and steps taken per episode by the agent. In the first 10 to 20 episodes, the cumulative reward rises sharply, while the steps taken decrease significantly, indicating that the agent quickly learns more efficient paths. Stabilization for both plots occurs after roughly 80 episodes, with convergence to an optimal policy achieved around episode 208 according to our convergence episode metric. This fast convergence speed is in line with expectations given the moderate size of the state space. However, notable fluctuations in both plots suggest learning instability, likely caused by the random exploration rate of the algorithm's ϵ-greedy component being rather high. Similar plots were observed across different random seeds, which indicated the need to fine-tune the exploration rate for more stable learning.

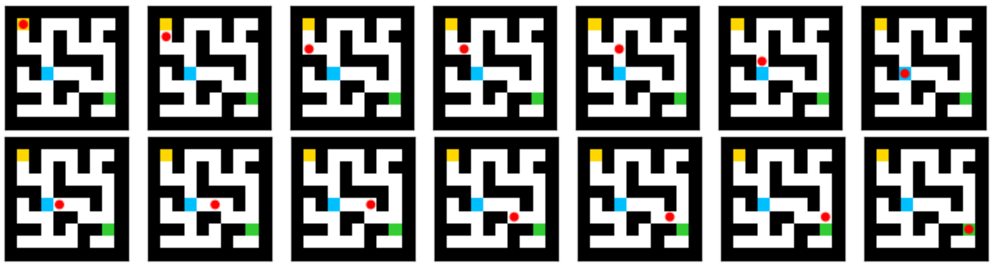

Figure 6. Execution of the Policy Learned by the Q-Learning Algorithm With Random Exploration Rate.

In Figure 6, we observe the agent navigating the maze using the learned policy. Once again, the red dot represents the agent, black squares denote walls, the yellow square marks the starting position, the blue square identifies the sub-goal, and the green square indicates the end goal. The agent follows the shortest path, beginning from the starting point, passing through the sub-goal and reaching the final goal, as expected from the optimal policy. Regardless of the value of the random seed set in the Jupyter notebook, the trajectory remains consistent, which is not surprising given that the learned policy is deterministic. Thus, Tabular Q-learning with a random exploration rate always succeeds at finding the optimal policy within 250 episodes.

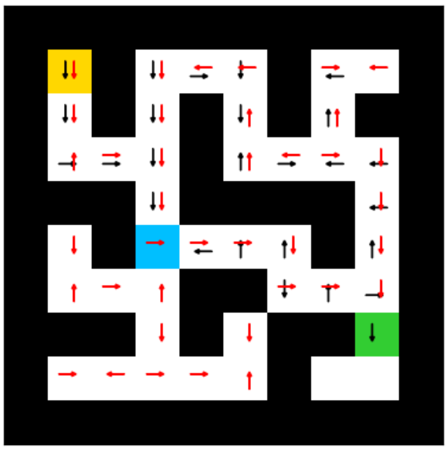

Figure 7. Visualization of the Overall Policy Learned by the Q-learning Algorithm with Random Exploration Rate.

The overall policy learned by the algorithm is depicted in Figure 7, where black arrows represent optimal actions for states before reaching the sub-goal, and red arrows represent actions for states after passing the sub-goal. From the starting point to the final goal, the arrows guide the agent along the shortest path through the sub-goal, indicating successful learning of the optimal path. However, other areas of the maze reveal gaps: if the agent starts in squares not along the optimal path and prior to the subgoal, it may struggle to complete the task.

This incomplete overall policy occurs due to Tabular Q-learning's exploration-exploitation trade-off. Since the algorithm emphasizes states associated with higher cumulative rewards through exploitation, the states relevant to the optimal path are reinforced through repeated visits, allowing the agent to learn optimal actions for them. Meanwhile, states outside the optimal path, visited primarily during exploration, lack sufficient visits for the algorithm to learn the optimal actions for most of them. The environment terminates once the end goal is reached, so any squares beyond the end goal remain unexplored by the agent. In addition, some squares below the sub-goal lack black arrows, as they concern states before the sub-goal is visited, which are unreachable from the current starting position. Although the actions necessary for the shortest path are learned consistently across different random seeds, the extent of optimal action learning in less frequently visited states is dependent on the seed due to the stochastic nature of Tabular Q-learning's exploration-exploitation strategy.

3.2.2. Tabular Q-learning with Optimized Exploration Rate

To enhance the convergence speed, we applied an Optuna study as detailed in the Methodology section, which identified an optimal exploration rate of 0.05 for the Tabular Q-learning algorithm. The results with this optimized exploration rate are analyzed in terms of convergence behavior, execution of the learned policy, and visualization of the overall learned policy.

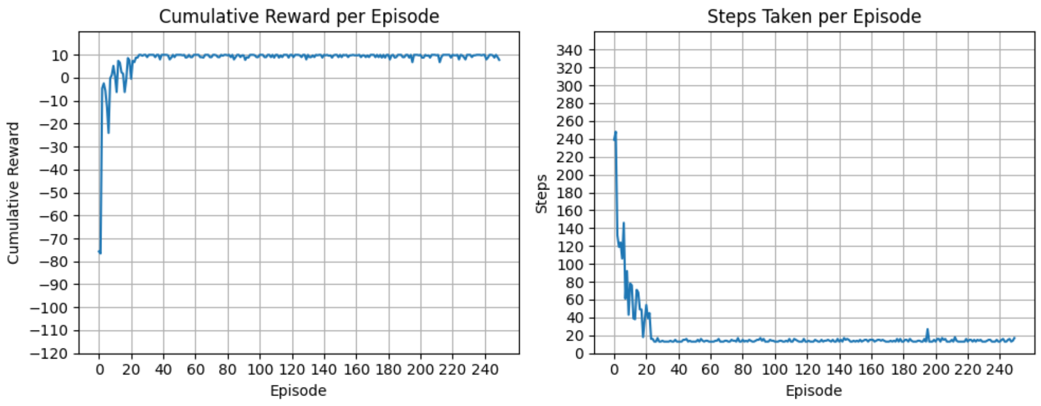

Figure 8. Cumulative Reward and Steps Taken per Episode by the Q-Learning Algorithm With Optimized Exploration Rate.

In Figure 8, we can see the cumulative reward and steps taken by the agent over the 250 episodes. The cumulative reward increases rapidly, whereas the number of steps decreases sharply within the first 20 episodes.

Stabilization occurs around episode 50, and convergence to the optimal policy is reached at approximately episode 122 based on our convergence metric. Compared to the previous version of Tabular Q-learning with a random exploration rate, this optimized version achieves a higher convergence speed. The large fluctuations observed in the previous plots (Figure 5) are no longer present, indicating a notable improvement in learning stability due to the fine-tuned exploration rate. These observations are consistent across different random seed values set in the Jupyter notebook.

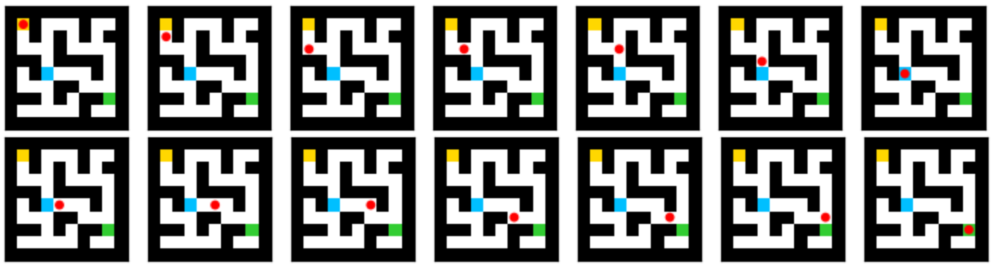

Figure 9. Execution of the Policy Learned by the Q-Learning Algorithm With Optimized Exploration Rate.

Figure 9 depicts the agent following the policy learned by the Tabular Q-learning algorithm through the maze. The representations for agent, walls, start position, sub-goal, and final goal are the same as in previous images showcasing the execution of the policy (Figures 3 and 6). Similarly to the version with random exploration rate, the algorithm finds the shortest path from the start point to the final goal via the sub-goal. Regardless of the random seed, the execution of the policy remains identical, as the optimal path is always found within 250 episodes and the learned policy is deterministic.

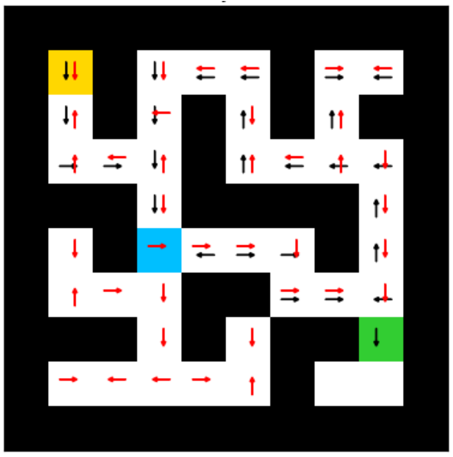

Figure 10. Visualization of the Overall Policy Learned by the Q-learning Algorithm with Optimized Exploration Rate.

Figure 10 presents a visualization of the overall policy learned by the Tabular Q-learning algorithm. Once again, black arrows indicate optimal actions for states before the sub-goal has been reached and red arrows represent actions for states after the sub-goal has been visited. The observations that can be made are similar to those for the random exploration rate version of Tabular Q-learning. The arrows from the starting point to the final goal effectively direct the agent along the shortest path that passes through the sub-goal. However, most arrows in non-adjacent areas of the optimal path show random actions, indicating that these squares were not sufficiently explored to identify optimal actions. In fact, the lower exploration rate in the optimized version compared to the random version has resulted in even more random arrows in non-optimal areas, which aligns with the algorithm's increased emphasis on exploitation over exploration. The absence of certain colored arrows in some squares is explained by the same factors noted for the other version of Tabular Q-learning. Additionally, different random seeds once more influence for which states away from the shortest path optimal actions are learned, though the agent consistently learns the optimal actions for the shortest path itself.

To further validate the Tabular Q-learning algorithm, we tested the optimized version in a maze layout with a different sub-goal position. As shown in Appendix C, the algorithm successfully identifies the optimal solution to the maze problem even with the new layout. Regardless of sub-goal location, the agent finds the optimal path from the start position to the final goal via the sub-goal across multiple seeds. These results confirm that Tabular Q-learning successfully completes the task in both maze configurations, which shows the validity of the algorithm.

3.3. Comparison of Value Iteration and Tabular Q-learning with Optimized Exploration Rate

As previously discussed, both algorithms successfully learn the optimal policy, which entails following the shortest path beginning from the start point, passing through the sub-goal, and reaching the end goal. Since the agent receives a penalty for each additional square visited, any longer path that still visits the sub-goal and then the final goal yields a lower cumulative reward and is thus suboptimal. The optimal policy is learned by both Value Iteration and Tabular Q-learning even when the sub-goal position is different from the one specified in the instructions, as demonstrated in Appendices B and C.

Despite the two algorithms converging on the same optimal path, they learn different overall policies for the maze, as shown in Figures 4 and 10. Value Iteration calculates the best action for every square before and after the sub-goal is visited, whether or not the square lies along the optimal path, resulting in a policy that covers the entire maze. In contrast, Tabular Q-learning, with its focus on balancing exploration and exploitation, primarily learns the best actions for squares directly relevant to the optimal path. These behaviors of both algorithms persist even when the sub-goal position is different, as shown in Appendices B and C. Thus, the overall policy learned by Value Iteration is significantly more comprehensive than that of Tabular Q-learning, especially when its exploration rate is lower as in the optimized version. However, the completeness in the policy of the baseline algorithm does not come without a cost.

To quantify this cost, we decided to compare the time each algorithm requires to converge. Although convergence time can vary depending on the machine used, it offers the common unit of seconds for assessing the algorithms' convergence speeds. Value Iteration and Tabular Q-learning fundamentally use different measures of convergence—iterations for Value Iteration and episodes for Tabular Q-learning—yet time provides a consistent basis for comparison. To ensure fairness, both algorithms were run sequentially on the same machine. The results illustrate that Value Iteration requires 0.1639 seconds to converge, more than seven times longer than Tabular Q-learning, which converges within 0.0234 seconds with an optimized exploration rate. While direct comparison of iterations to episodes is not possible, a relationship appears to exist between the estimated number of steps and the time to convergence for each algorithm: Value Iteration converges in 225 iterations, whereas Tabular Q-learning achieves convergence in 122 episodes. Despite this, since Value Iteration lacks any stochasticity, it shows greater learning stability than Tabular Q-learning—even when compared to the version with optimized exploration rate—as can be seen in Figures 2 and 8.

However, since the primary task was to find the optimal path from the starting position to the end goal via the sub-goal—rather than finding optimal paths from every possible position or achieving greater learning stability—Tabular Q-learning with its focus only on relevant parts of the state space proves more effective for this problem. Value Iteration's exhaustive, brute-force approach is excessive for this task, as Tabular Q-learning's targeted exploration enables more efficient convergence to the optimal solution.

4. Conclusion

In this project, we applied two algorithms—Value Iteration and Tabular Q-learning—to solve a maze where the agent must first visit a sub-goal before reaching the final goal from a specified starting point. Value Iteration, used as a baseline due to its brute-force approach, generated a comprehensive policy across all maze states by exploring all possible state-action pairs, including the optimal path from the starting position. In contrast, Tabular Q-learning learned an incomplete policy that includes the optimal solution to the task, but does not involve the best actions for squares distant from the optimal path. Fine-tuning parameters, particularly the exploration rate, proved crucial to Tabular Q-learning's stability of learning and convergence speed, as demonstrated by the trials of the algorithm with different exploration rate values. A decaying exploration rate, starting from a higher value and gradually decreasing, might have resulted in a more complete policy for Q-learning if paired with runs consisting of more episodes. Nevertheless, given that the main objective was to determine the optimal path from the starting point to the end goal via the sub-goal, we believe that our current implementation effectively meets these requirements. While both algorithms successfully accomplished the task, their efficiency differed significantly, with Value Iteration requiring more time to converge than Tabular Q-learning.

Our findings highlight important considerations for applying the two reinforcement learning algorithms to real-world problems. Value Iteration is best suited for tasks in fully observable, discrete, and static environments where a complete optimal policy covering all states and high learning stability are required, even if it demands longer computational time. This approach is ideal for simpler applications, such as board games with limited state-action pairs or robotic pathfinding in small, structured environments like factory floors, where layouts remain fixed. However, Value Iteration's exhaustive search method becomes impractical for real-life tasks involving large state or action spaces. Conversely, Tabular Q-learning is well-suited for discrete environments where only partial knowledge is available and a complete optimal policy is not necessary. Instead, the algorithm focuses on deriving an optimal path from a given starting position with moderate learning stability. Its exploration-exploitation strategy, driven by interaction with the environment, enables efficient problem-solving even in large state or action spaces. These characteristics make Tabular Q-learning a compelling choice for dynamic, real-time applications, such as inventory management for small retail settings or small-scale stock portfolio management. To conclude, while Value Iteration offers high reliability in small, predictable settings, Tabular Q-learning provides a more efficient solution for large, evolving environments.

5. Contributions

Member	Contribution
Joana Petkova	In terms of coding, I implemented the functions for visualizing the policy, added comments to several functions, and helped improve the overall code quality. I also organized the cells in the Jupyter notebook and created headers for each section. For the report, I wrote the Introduction section, the Methodology subsection covering the common parameters between the two algorithms, the Results subsection comparing the two algorithms, and the Conclusion section. Additionally, I edited the entire report and added captions and figure numbers in both the main text and the Appendices.
Mahbod Tajdini	For this project, I was responsible for writing and coding several key components. I authored the entire section 2.1, covering the environment setup. In terms of coding, I worked on key parts of the maze environment and created the base code for both the Q-learning and Value Iteration algorithms. After others made some edits, I added the time component to improve how these algorithms were evaluated. I also made a few adjustments to the *plot_metrics* function and helped a bit with organizing and polishing the Jupyter notebook overall.
Selman Gül	In the implementation phase, I was responsible for developing the exploration rate optimization component for Q-learning, with the primary objective of enhancing the algorithm's convergence speed. This required experimenting with several different metrics and parameter setups. I also added some comments to the Jupyter notebook. Regarding the report, I wrote the Methodology and Results sections for Tabular Q-learning. Additionally, I provided general suggestions about the paper's organization and content.
Ionut Moise	For the coding part of the project, I assisted with debugging and adjusting the functions that create the graphs for the convergence of the algorithms, as well as the functions that generate visualizations for the optimal policy learned. I also adjusted the Value Iteration code used in our notebook for us to be able to make the convergence plot. For the project's report, I was responsible for writing the Methodology and Results subsections about Value Iteration. Additionally, I contributed slightly to the Conclusion section and corrected grammatical mistakes in the text.

6. References

The work for this project was mostly done based on the materials provided for the course: lecture slides and two textbooks. Some external coding walkthroughs were also utilized. The textbooks and the coding walkthroughs are cited below.

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, Massachusetts: MIT Press, 2018. Accessed: Oct. 25, 2024. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[2] R. Borhani, S. Borhani, and A. K. Katsaggelos, "Reinforcement Learning," in *Fundamentals of Machine Learning and Deep Learning in Medicine*, Springer, Cham, 2022, pp. 165–189. doi: https://doi.org/10.1007/978-3-031-19502-0_8.

[3] T. Miller, "Value Iteration," *gibberblot.github.io*, 2023. https://gibberblot.github.io/rl-notes/single-agent/value-iteration.html (accessed Oct. 25, 2024).

[4] A. Amine, "Q-Learning Algorithm: From Explanation to Implementation," *Medium*, Dec. 19, 2020. https://towardsdatascience.com/q-learning-algorithm-from-explanation-to-implementation-cdbeda2ea187 (accessed Oct. 25, 2024).

7. Appendices

A. Visualization of Alternative Maze Layout

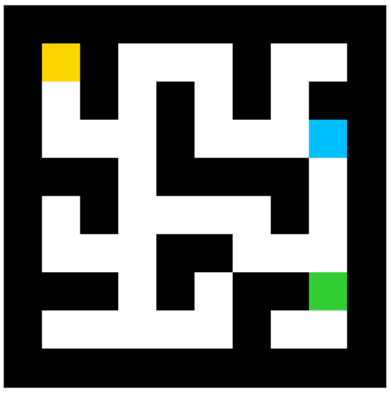

Figure 11. Alternative Maze Layout.

B. Results for Value Iteration on the Alternative Maze Layout

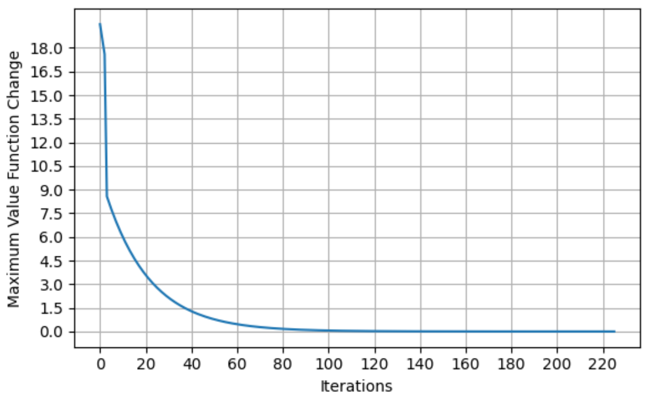

Figure 12. Maximum Change in the Value Function per Iteration.

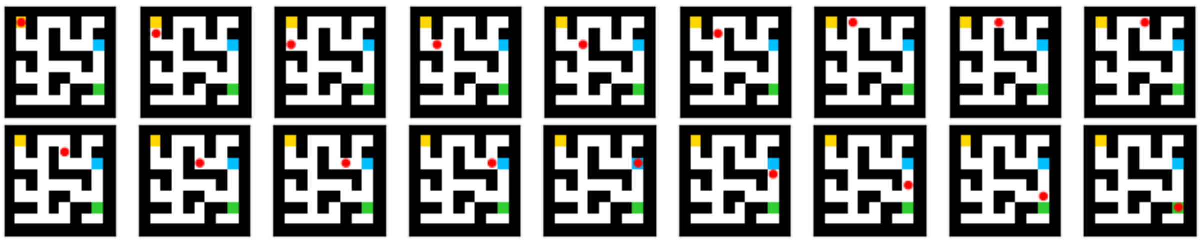

Figure 13. Execution of the Policy Learned by the Value Iteration Algorithm.

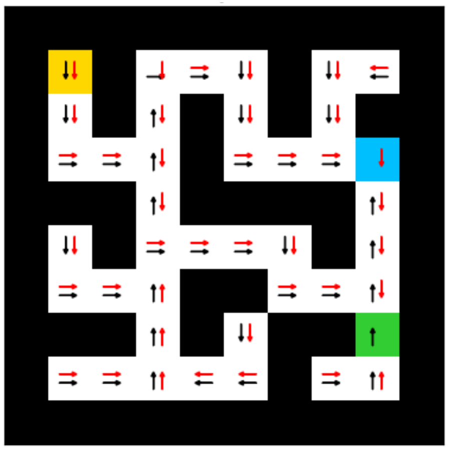

Figure 14. Visualization of the Overall Policy Learned by the Value Iteration Algorithm.

C. Results for Q-learning with Optimized Exploration Rate on the Alternative Maze Layout

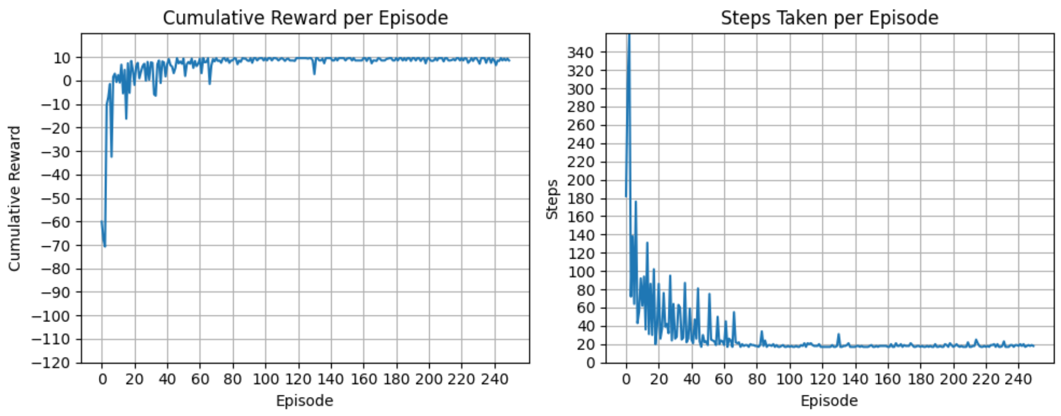

Figure 15. Cumulative Reward and Steps Taken per Episode by the Q-Learning Algorithm With Optimized Exploration Rate.

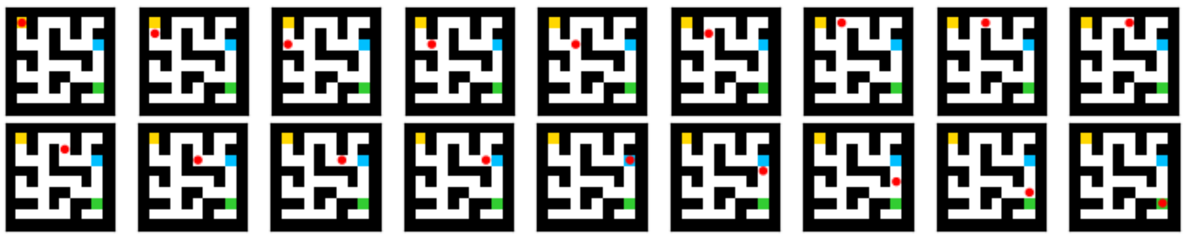

Figure 16. Execution of the Policy Learned by the Q-learning Algorithm with Optimized Exploration Rate.

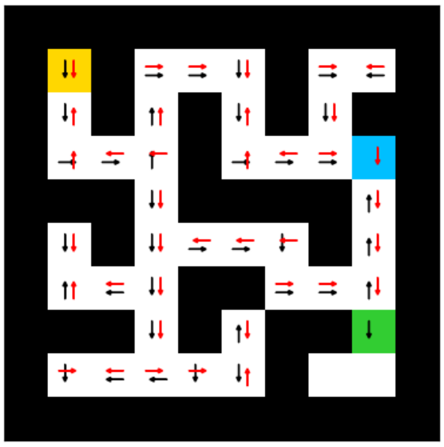

Figure 17. Visualization of the Overall Policy Learned by the Q-learning Algorithm with Optimized Exploration Rate.