# Final Project Report for Basic Reinforcement Learning

Course Name:Reinforcement Learning - Course Code: GAME MAZE

Instructor's Name: Dr VU HOANG DIEU

Student's Name: HA HUY THIEN - Student ID: 22014474
Class: K16 AI RB

Submission Date: October 11, 2024

# Contents

## List of Figures

## Contents

## List of Tables

**Abstract**

This paper investigates the use of Q-Learning and SARSA algorithms for solving maze navigation tasks in simulated environments. Mazes serve as structured yet challenging testbeds that require agents to make sequential decisions to reach a goal efficiently. Q-Learning, an off-policy algorithm, allows agents to estimate optimal actions by maximizing future rewards, offering flexibility in policy adherence. In contrast, SARSA, an on-policy algorithm, updates action-values based on actual transitions observed during learning, resulting in a more adaptable behavior aligned with the agent's policy. Experiments conducted on mazes of various sizes (4x4, 5x5, 10x10) assess each algorithm's performance in terms of pathfinding efficiency, convergence speed, and adaptability. The study highlights the distinct strengths of Q-Learning and SARSA for dynamic navigation problems, offering insights into the suitability of each approach for tasks requiring adaptive decision-making in structured environments.

**Keywords:** Reinforcement Learning, Q-Learning, SARSA, Maze Navigation, Sequential Decision-Making, Path Optimization, Adaptive Learning

## 1 Introduction

Reinforcement learning [1] is an important and complex field of study that combines the science of adaptive behavior and advanced computational methods to help intelligent agents discover optimal behaviors in uncertain environments.The main goal of reinforcement learning is to enable entities to make the best decisions to achieve maximum rewards, even when the environments in which they operate are not fully clear or are continuously changing.Reinforcement learning systems are designed to adapt and learn from experience through trial and error, allowing them to optimize behavior over time. These systems learn how to act in an environment to achieve a specific goal without the need for explicit supervision, as is required in supervised learning methods.As a field of research, reinforcement learning has made tremendous strides in the past decade, particularly in areas such as robotics control, video games, and autonomous vehicles. Modern algorithms, such as Q-learning, SARSA, and others, have opened up new possibilities for optimizing the behavior of intelligent agents in complex and unpredictable environments.

A maze game is a type of logic puzzle where the player or agent is tasked with finding a path from a starting point to a goal within a grid of cells. Each cell in the maze can either be a passable pathway or an obstacle (wall) that blocks movement. This setup makes the maze game an ideal example for applying reinforcement learning, as it provides a complex environment where the agent must learn to make decisions and optimize its path toward the goal, despite potential obstacles. In reinforcement learning, each position within the maze represents a state, and from each state, the agent has a set of possible actions: moving up, down, left, or right. When the agent takes an action, it transitions to a new state and receives a reward based on the nature of the new position. For example, reaching the goal yields a high positive reward, encountering an obstacle incurs a penalty or negative reward, and each movement step may also come with a small negative reward to encourage the agent to find the shortest route. Through trial and error, the agent learns to maximize the cumulative reward by selecting actions that lead to states with higher rewards, helping it navigate the maze more efficiently over time.

The objective of this research is to apply two popular reinforcement learning (RL) algorithms, Q-Learning and SARSA, to train agents capable of solving puzzles in maze games with the minimum number of steps possible. In this context, each algorithm provides a different approach to estimating the value of actions and states within the maze environment, thereby influencing how the agent learns and optimizes its movement strategy.

Q-Learning [2] is an unsupervised learning algorithm that allows the agent to learn through trial and error. This algorithm is based on constructing a Q-Table that stores the Q values for each state-action pair. Each Q value represents the "goodness" of a specific action in a given state. The agent starts by exploring the environment, taking actions randomly, and receiving corresponding rewards or penalties for each action. Over time, as the agent collects enough data from the environment, it updates the Q-Table according to the Q-Learning formula, allowing it to determine which action will yield the highest reward in each state. The ultimate goal of Q-Learning is to optimize the agent's path to the target, enabling it to solve the maze with the minimum number of steps.

On the other hand, the SARSA [3] (State-Action-Reward-State-Action) algorithm also uses a Q-Table but employs a different approach. SARSA updates the Q value based on the actual action the agent takes in each state, rather than solely relying on the optimal action as in Q-Learning. Specifically, when the agent moves from the current state to a new state, it receives a reward and proceeds to take another action, thereby updating the Q value based on the real experiences it has encountered. This method creates a more practical learning strategy, as it accurately reflects the actions the agent has taken, allowing it to adjust its behavior based on what it has learned in the actual environment.

Through the comparison and analysis of the effectiveness of both Q-Learning and SARSA algorithms [4], the project aims not only to find the optimal method for solving maze puzzles but also to provide deeper insights into how reinforcement learning algorithms function in tackling complex problems in environments characterized by uncertainty. The differences in how the two algorithms approach this problem will enhance our understanding of the advantages and limitations of each method, enabling better strategy selection for agents in similar situations in the future.

## 2 Environment.

A maze[5] is a geometric structure organized into a grid made up of numerous square cells, forming a complex and varied network. Within the maze, each cell can be an open pathway or an obstacle, typically a wall that blocks the agent's route. Mazes come in a wide range of sizes, from small mazes with just a few cells, where players or agents can quickly find the shortest path, to larger mazes with many cells, creating complex pathfinding challenges that require optimal search strategies.

The difficulty of a maze is determined not only by the number of cells or the placement of obstacles but also by how the pathways and barriers are arranged. Some mazes may include many dead ends, creating difficult turns or open areas, providing the agent with a variety of paths to explore. This diversity increases the challenge of finding the most efficient route to the goal, especially when the agent needs to distinguish between dead ends and pathways that lead to success.

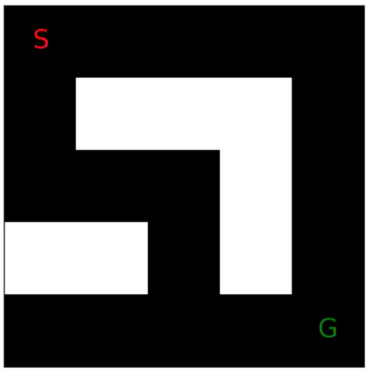

Figure 1: MAP 1

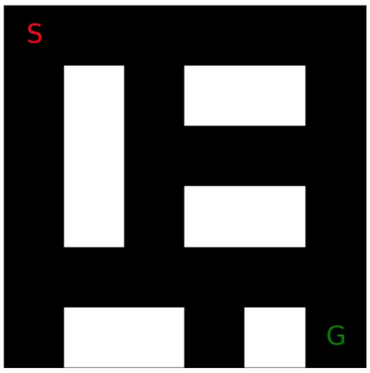

Figure 2: MAP 2

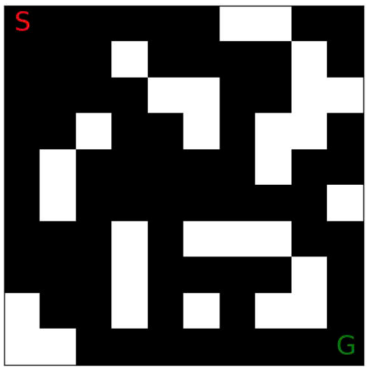

Figure 3: MAP 3

The state space in a maze game is an abstract model that represents all possible positions an agent can reach while navigating from the starting point to the goal. Each state represents a cell in the maze where the agent can either stop or find a way to move through. Within this state space, each component plays a crucial role in determining how the agent navigates and optimizes its path:

**Pathways:** These are cells the agent can move through. The pathways connect with one another to form routes within the maze, allowing the agent to progress toward the goal.

**Walls:** These are impassable cells that restrict the agent's movement. Walls create barriers, forcing the agent to find alternative routes or avoid dead ends.

**Starting Position:** This is the agent's initial state in the maze. From here, the agent begins its exploration and learning to find a path to the goal.

**Goal Position:** This is the target state the agent needs to reach. When the agent arrives at the goal position, the pathfinding process ends, and the agent receives a high reward to reinforce reaching the objective.

**Agent's Current Position:** This marks the agent's current state at each movement step. This position continually changes as the agent navigates through the maze.

This state space is often represented as a two-dimensional grid,where each cell corresponds to a state. For easier calculations and state transitions, the two-dimensional grid can be encoded into a one-dimensional array, simplifying the process of calculating state changes and optimizing the agent's pathfinding algorithm.

The agent can perform a set of actions at each state. These actions typically include:

- Up (move up one cell)
- Down (move down one cell)
- Left (move left one cell)
- Right (move right one cell)

However, the agent cannot move into cells that contain obstacles (walls), creating constraints on its action choices. The agent must also carefully consider its available actions to ensure that it not only avoids obstacles but also finds the most optimal path to the goal.

The reward system is a crucial element in reinforcement learning, as it provides feedback to the agent about the performance of the actions it takes. Rewards can be classified as follows:

**High Reward:** Granted when the agent reaches the goal position, encouraging it to seek the path leading to the target. This reward can be larger than other rewards to emphasize the importance of completing the task.

**Negative Reward:** May be given when the agent collides with an obstacle or takes ineffective steps (such as hitting a dead end). This helps the agent learn and avoid unproductive actions in the future. Negative rewards can also help limit the number of times the agent enters ineffective areas.

**Neutral Reward:** Can be awarded for each normal step taken in the maze. The purpose of this reward is to encourage the agent to seek the shortest path, as it not only receives a reward upon reaching the destination but also must consider its performance at each step.

Table 1: The reward value

| Condition | Action/State | Reward |
|---|---|---|
| Taking a step | Player moves to an adjacent empty space | -1 |
| Hitting a wall | Player attempts to move into a wall | -10 |
| Reaching the goal | Player moves to the goal | +100 |

The table provides an overview of the reward system designed to guide the agent in the maze environment, helping it find the optimal path to the goal. This system includes three specific reward conditions: when the agent moves to an adjacent empty cell, when it hits a wall, and when it reaches the goal.

First, when the agent moves to an adjacent empty cell, it receives a reward of -1. This slight negative reward helps the agent understand that every step has a "cost," encouraging it to find the shortest and quickest route to the goal, avoiding unnecessary steps. This aspect is crucial in motivating the agent to choose the most efficient actions without getting sidetracked in the maze.

Second, when the agent attempts to move into a cell containing a wall (an obstacle), it receives a reward of -10, significantly higher than the penalty for a regular step. This penalty aims to discourage the agent from repeating ineffective actions, as hitting a wall does not bring it closer to the goal. This compels the agent to learn to distinguish between valid paths and dead ends, avoiding wasted time and resources on impassable routes.

Finally, when the agent reaches the goal, it receives a large reward of +100. This positive reward is much higher than other rewards, acting as a strong motivation for the agent to prioritize finding and reaching the goal. The substantial reward reinforces the importance of completing the task, encouraging the agent to adjust its movement strategy to achieve the objective quickly and with minimal cost.

In summary, the maze game environment is a complex system with multiple interacting elements, providing a challenging platform for the agent to learn and refine its behavior in order to find the most efficient path to the goal. This environment's diversity, including various paths, dead ends, and obstacles, adds layers of complexity that make the game both engaging and educational for intelligent agents. As the agent navigates the maze, it must balance exploration and exploitation to optimize its path, learning from rewards and penalties associated with each action. This process of learning within a structured yet dynamic environment offers researchers valuable insights into reinforcement learning algorithms and allows them to observe how agents adapt their strategies over time. By experimenting with and improving agent behavior in the maze, researchers can better understand and enhance the capability of these algorithms, ultimately equipping agents with the skills needed to tackle similarly complex challenges in real-world applications.

## 3 Methods

### 3.1 Q - learning

Q-Learning is a powerful reinforcement learning algorithm used to train agents to navigate environments like maze games by learning an optimal policy through estimating the Q-function, or action-value function. As a model-free method, Q-Learning does not require knowledge of the environment's transition dynamics, making it ideal for complex or unpredictable environments. In the context of maze games, Q-Learning enables an agent to find the shortest path to reach the goal by maximizing cumulative rewards through strategic actions.

In a maze game, the Q-Learning algorithm initializes the Q-function $Q(s, a)$ arbitrarily for all state-action pairs. The maze is represented as a grid, where each cell corresponds to a state, and possible actions (moving up, down, left, or right) lead the agent to neighboring states. At each step, the agent observes its current position (state $s$), selects an action $a$ based on an exploration strategy (e.g., an $\epsilon$-greedy approach), and receives feedback from the environment—a reward for moving towards or reaching the goal and a penalty for invalid moves (such as hitting walls). The Q-value for the state-action pair $(s, a)$ is updated using the following rule:

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

where $\alpha$ is the learning rate that controls the extent of the update, $\gamma$ is the discount factor that balances immediate versus future rewards, and $r$ is the immediate reward received.

This process of updating the Q-value is known as the temporal-difference (TD) learning step, as it estimates the difference between the predicted reward and the observed reward from taking an action.

An important feature of Q-Learning is its off-policy nature, allowing the agent to improve its policy based on an $\epsilon$-greedy exploration strategy. With probability $1 - \epsilon$, the agent selects the action that has the highest Q-value, while with probability $\epsilon$, it chooses a random action to encourage exploration. Over time, $\epsilon$ is reduced, guiding the agent towards exploiting its learned values more frequently as it becomes more confident in its Q-value estimates. This exploration-exploitation balance is critical in maze games, where the agent must explore to avoid dead ends and eventually learn the most efficient path to the goal.

The Q-Learning algorithm is well-suited for maze games, as it learns optimal moves by repeatedly interacting with the environment, updating the Q-values based on the immediate feedback and anticipated future rewards. In practice, this enables the agent to navigate through increasingly complex mazes by selecting actions that maximize the cumulative reward. Through consistent training, the Q-values for each state-action pair converge, allowing the agent to follow a policy that reaches the goal state with minimal steps.

To implement Q-learning in a maze game, we start by creating a grid to represent the maze, defining valid actions based on wall and path locations. The Q-table is initialized with random values, and an exploration strategy guides the agent's movement. At each step, the agent updates the Q-values based on received rewards, gradually improving its policy. Over many episodes, the agent learns the optimal path to the goal, enabling it to solve new maze layouts effectively.

**Pseudocode**

```
Initialize the Q-table for all state-action pairs.
For each episode:
    Initialize the starting state.
    For each step in the episode:
        Choose an action using an -greedy strategy.
        Perform the action and observe the next state and reward.
        Update the Q-value for the current state-action pair using the
        Q-learning update rule.
        Set the next state as the current state.
        End the episode if the goal state is reached.
    Decay the exploration rate  gradually.
Return the Q-table and track performance metrics.
```

After training, the agent's progress is measured by metrics such as total rewards and steps taken to reach the goal. This iterative learning process enables the agent to find optimal paths and adapt to various maze structures, making Q-Learning a suitable choice for training agents to solve maze games efficiently.

## 3.2 Sarsa.

SARSA is an on-policy reinforcement learning algorithm used to train agents to navigate environments such as maze games by learning an optimal policy through action-value estimates. Unlike Q-Learning, which is off-policy, SARSA updates its Q-values based on the actual actions taken, which ensures a more stable and reliable learning process. SARSA is particularly suitable for uncertain or dynamic environments like maze games, where agents benefit from a learning approach that emphasizes the current policy rather than always aiming to optimize future actions independently of the policy followed.

In maze games, SARSA initializes the Q-function $Q(s, a)$ arbitrarily for all state-action pairs. The maze is represented as a grid, where each cell corresponds to a state, and actions (such as moving up, down, left, or right) allow the agent to transition between states. At each time step, the agent observes its current position (state $s$), selects an action $a$ based on an exploration strategy like $\epsilon$-greedy, and receives feedback from the environment—a reward for making progress toward the goal or a penalty for invalid actions (like hitting walls). The Q-value for the state-action pair $(s, a)$ is updated using the following update rule:

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

where $\alpha$ is the learning rate that controls the update step size, $\gamma$ is the discount factor that balances immediate and future rewards, and $r$ is the immediate reward received. Unlike Q-Learning, SARSA's update rule incorporates the next action $a'$ taken according to the agent's current policy, which ensures that SARSA remains on-policy by basing updates on actual agent actions.

A key feature of SARSA is its on-policy nature, where the agent improves its policy based on the current strategy it follows. With a probability of $1 - \epsilon$, the agent chooses the action with the highest Q-value, and with a probability of $\epsilon$, it selects a random action to encourage exploration. Over time, $\epsilon$ is gradually reduced, allowing the agent to shift towards exploitation as it becomes more confident in its Q-value estimates. This balance between exploration and exploitation is especially crucial in maze games, where the agent must explore to avoid dead-ends and eventually learn the most efficient path to the goal.

The SARSA algorithm is well-suited for maze games as it learns optimal actions by iteratively interacting with the environment, updating Q-values based on immediate feedback and expected future rewards. In practice, this enables the agent to navigate increasingly complex mazes by selecting actions that maximize cumulative rewards based on its current policy. Through steady training, the Q-values for each state-action pair converge, enabling the agent to follow a policy that reaches the target state with minimal steps.

**Pseudocode**

```
Initialize the Q-table for all state-action pairs.
For each training episode:
    Initialize the starting state.
    Choose an initial action using the -greedy strategy.
    For each step within the episode:
        Take the action, observe the next state and reward.
        Choose the next action based on the -greedy policy.
        Update the Q-value for the current state-action pair using the
        SARSA update rule.
        Set the current state and action to the next state and action.
        End the episode if the goal state is reached.
    Decay the exploration rate .
Return the Q-table and track performance metrics.
```

Following training, the agent's progress is measured through metrics such as the total reward obtained and the number of steps required to reach the goal. This repeated learning process enables the agent to discover optimal paths and adapt to varying maze structures, making SARSA a strong choice for training agents to solve maze games effectively.

Q-Learning and SARSA have been selected as the primary algorithms for this task due to their unique advantages. Q-Learning is favored for its ability to explore the environment rapidly, as it can estimate optimal actions without requiring a specific model, making it well-suited for complex and dynamic scenarios. This algorithm also focuses on maximizing long-term rewards, helping the agent make strategic decisions that account for future outcomes. Additionally, Q-Learning scales effectively to large state spaces, enabling the learning of optimal strategies even in complex environments with numerous actions.

Meanwhile, SARSA provides a more stable learning process as it updates action-value estimates based on the actual actions taken, leading to efficient learning in uncertain environments. SARSA also helps reduce risk in decision-making, as it follows the current policy, limiting uncertain action choices and encouraging safer decisions. Moreover, SARSA adapts well to stochastic conditions, allowing the agent to handle unforeseen changes and maintain robust performance in fluctuating environments.

## 4 Experimental Setup

### 4.1 Environment Setup

Environment Used: In this experiment, a custom maze environment is created using a 2D numpy array. The maze consists of paths (0) and walls (1). The agent starts at a designated position and aims to reach the goal position while navigating through the maze.
**Learning Rate** ($\alpha$): Set to 0.1, this parameter controls how much the agent updates its Q-values based on new experiences. A higher learning rate allows the agent to learn more quickly but may lead to instability.

**Discount Factor** ($\gamma$): Set to 0.9, this factor determines the importance of future rewards compared to immediate rewards. A value close to 1 prioritizes long-term rewards.

**Exploration Rate** ($\epsilon$): This value starts at 1.0 and decays to 0.01 over the course of training. It controls the balance between exploration (trying new actions) and exploitation (choosing actions based on known Q-values).

**Number of Episodes**: The agent is trained over 100 episodes, allowing it to learn the optimal path to the goal through repeated interactions with the environment.

### 4.2 Hyperparameter Tuning

**Q-learning Experiment**

In the Q-learning experiment, we will focus on testing different values of the learning rate ($\alpha$) hyperparameter to assess its impact on the agent's learning performance and decision-making in the maze environment. By keeping other parameters constant, we aim to analyze the effect of the learning rate alone on the algorithm's convergence and stability. The learning rate values tested include:
**Learning Rate ($\alpha$):**Values of 0.1, 0.5, and 0.9 will be tested. The learning rate directly affects the degree to which the agent adjusts its Q-values after each action. A smaller $\alpha$ value (like 0.1) leads to slower, more cautious updates, promoting stability but possibly delaying convergence. In contrast, larger $\alpha$ values (such as 0.5 and 0.9) allow the agent to adjust Q-values more rapidly, potentially accelerating convergence but also increasing the risk of instability.
**Discount Factor ($\gamma$):** Fixed at 0.9 in this experiment to maintain a high importance for future rewards compared to immediate rewards. This helps the agent evaluate actions with a long-term perspective and make more strategic decisions in the maze environment.
**Exploration Start ($\epsilon$):** Fixed at 1.0, allowing the agent to start with a high exploration tendency to explore various directions in the maze.
**Exploration End ($\epsilon$):** Set at a fixed value of 0.01, ensuring that after several training episodes, the agent gradually reduces its exploration to focus on exploiting the learned policy.

Through these different learning rate values, we will analyze variations in the agent's performance in finding optimal solutions in the maze and the algorithm's convergence speed.

**SARSA Experiment**

The SARSA experiment will also focus on adjusting the learning rate with the same values as in the Q-learning experiment to compare the agent's learning performance in

the maze environment and evaluate the impact of the learning rate on decision-making.
**Learning Rate ($\alpha$):** Values of 0.1, 0.5, and 0.9 will be used. This will allow us to observe how the rate of adjusting Q-values affects the agent's convergence process. In SARSA, the Q-values are updated based on the agent's actual actions, so changes in $\alpha$ may significantly impact the systematic and careful learning approach compared to Q-learning.
**Discount Factor ($\gamma$):** Fixed at 0.9 for consistency in evaluating the importance of future rewards, allowing us to isolate the effect of the learning rate.
**Exploration Start ($\epsilon$):** Fixed at 1 to ensure high initial exploration,creating opportunities for the agent to explore many directions in the early learning stages.
**Exploration End ($\epsilon$):** Fixed at 0.01, allowing the agent to gradually reduce exploration to optimize based on accumulated experience.

The goal of this experiment is to evaluate the performance of the SARSA algorithm when learning from the maze environment, compare it with Q-learning, and analyze the impact of different learning rate values on the agent's convergence and decision-making process. Performance metrics such as total reward and the number of steps taken to reach the goal will be recorded for each algorithm, providing an overview of the differences between these two reinforcement learning methods.

## 5 Result

In this section, we present a thorough analysis of the results obtained from implementing Q-learning and SARSA in a meticulously structured 6x6 maze environment. Our primary objective was to investigate how different learning rates ($\alpha$) impact the agent's ability to learn optimally and navigate the maze efficiently. We experimented with three distinct values for the learning rate: 0.1, 0.5, and 0.9, keeping the discount factor ($\gamma$) fixed at 0.9 throughout to provide a consistent foundation for assessing learning strategies. Exploration parameters were held constant across trials to facilitate a fair comparison of performance under different learning rates.

The 6x6 maze layout was specifically designed to offer a balanced combination of simple paths and more complex navigational challenges. This intentional complexity served as an effective context for evaluating the adaptability and learning efficiency of both algorithms across varying learning rates. Key metrics, including cumulative rewards and the number of steps required to reach the goal, were carefully tracked to identify the optimal configurations for Q-learning and SARSA within the constraints of this maze environment.

Our analysis aims to provide in-depth insights into the effects of the learning rate parameter ($\alpha$) on each algorithm's convergence speed, stability, and overall effectiveness in navigating the maze. In the following sections, we will detail our findings, offering a comparison of each algorithm's performance at different learning rates. Prominent trends, variations in performance, and the broader implications of these findings for future applications of reinforcement learning will be discussed. Through this exploration, we seek to deepen our understanding of the learning dynamics at play and to contribute to the design of more robust and effective reinforcement learning algorithms for navigation-based tasks.

## 5.1 Result Q-learning

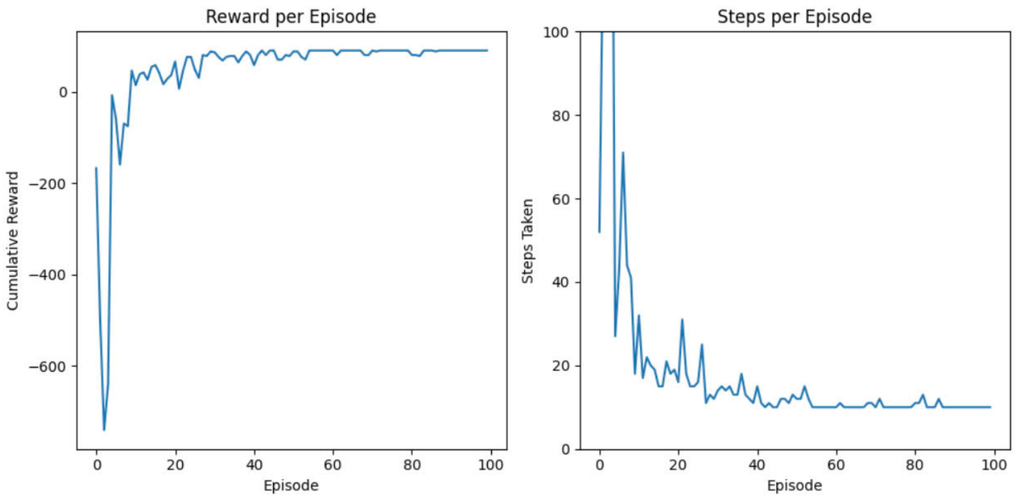

Figure 4: Q-learning with $\alpha$ =0.1

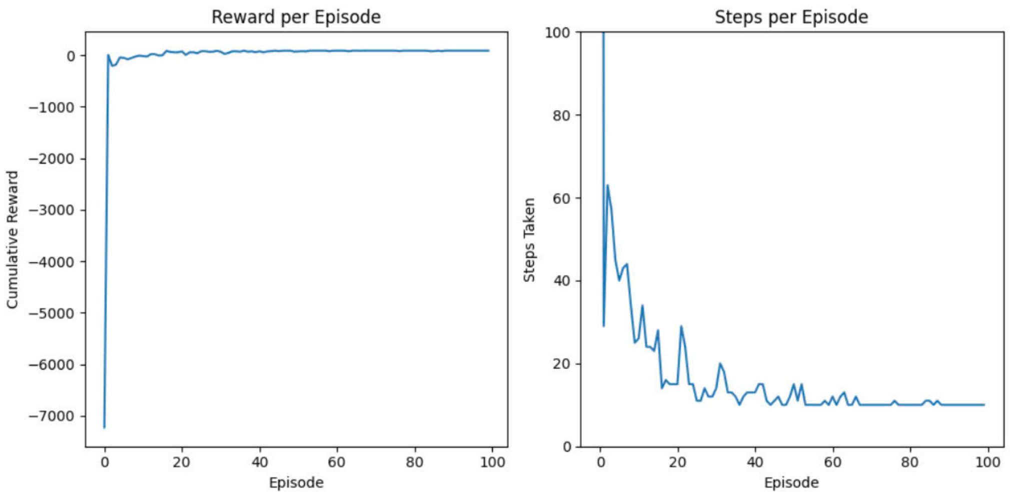

Figure 5: Q-learning with $\alpha$ =0.5

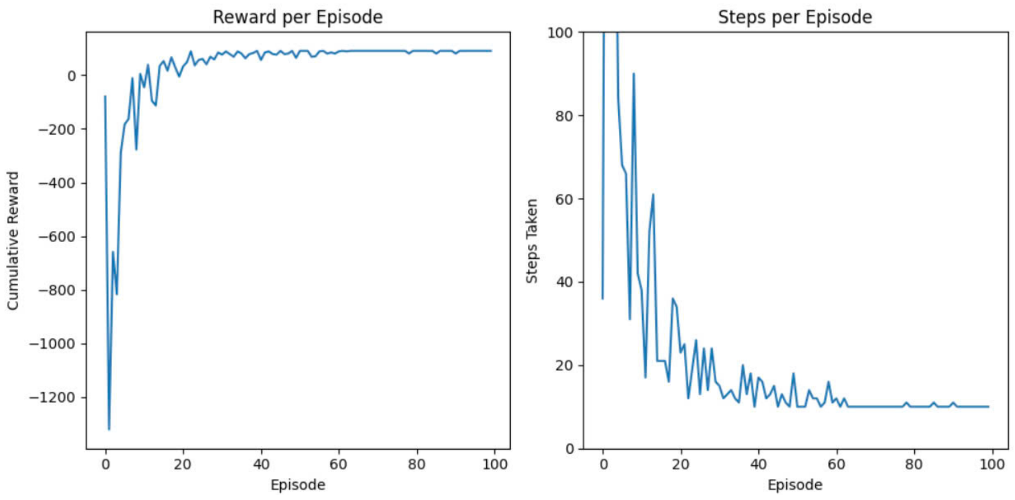

Figure 6: Q-learning with $\alpha$ =0.9

Table 2: Result

| $\alpha$ | Average Steps | Average Reward |
|---|---|---|
| 0.1 | 19.38 | 46.16 |
| 0.5 | 29.4 | -10.12 |
| 0.9 | 22.97 | 28.08 |

When $\alpha = 0.1$, the agent learns very slowly and takes more time to converge. In the "Reward per Episode" graph, we can see that the cumulative reward gradually increases and stabilizes at a high level after about 40 to 50 episodes. This indicates that with a low learning rate, the agent updates the Q-values cautiously, causing the cumulative reward to increase gradually and reach the highest optimal value in the long run.

In the "Steps per Episode" graph, the number of steps required to complete each episode steadily decreases and converges after about 80 episodes. The agent gradually learns to navigate the maze with the fewest steps, although the convergence speed is slower than with other$\alpha$ values. With $\alpha$= 0.1, the agent's average steps are the lowest (19.38), while the average reward is the highest (46.16). This shows that a low $\alpha$ enables the agent to achieve the highest long-term performance, with both high cumulative reward and low steps, even though more episodes are needed to reach an optimal state.

When $\alpha$ increases to 0.5, the agent converges faster than with $\alpha = 0.1$. In the "Reward per Episode" graph, the cumulative reward stabilizes more quickly, after only about 20 episodes. However, the final reward is lower than when $\alpha = 0.1$, indicating that a moderate learning rate allows the agent to find optimal actions faster but does not achieve the highest reward.

In the "Steps per Episode" graph, the number of steps the agent needs to complete each episode decreases rapidly and converges after around 30 episodes. This suggests that a moderate learning rate enables the agent to learn how to navigate the maze more efficiently in a shorter time, though it doesn't reach the optimal number of steps as with $\alpha$ = 0.1. According to the results table, the agent's average steps are 29.4, and the average reward is -10.12, indicating that a moderate $\alpha$ helps the agent converge quickly but does not result in a high cumulative reward.

With $\alpha = 0.9$, the agent achieves the fastest convergence among the tested $\alpha$ values. In the "Reward per Episode" graph, the cumulative reward stabilizes after only around 10 to 20 episodes. However, the final reward is significantly lower than with $\alpha = 0.1$, suggesting that a high learning rate allows the agent to find optimal short-term actions but tends to overlook options that could yield higher rewards in the long term.

In the "Steps per Episode" graph, the number of steps the agent requires to complete each episode quickly decreases and converges after only about 10 to 20 episodes. This indicates that with a high $\alpha$, the agent can quickly find the shortest route through the maze; however, this comes with a lower average reward. According to the results table, the agent's average steps are 22.97, and the average reward is 28.08, showing that a high learning rate helps the agent achieve fewer steps but not the highest reward in the long run.

Therefore, $\alpha$ tuning is crucial for balancing convergence speed, steps, and reward for the agent during learning, depending on the priority goals of the problem.

## 5.2 Result SARSA

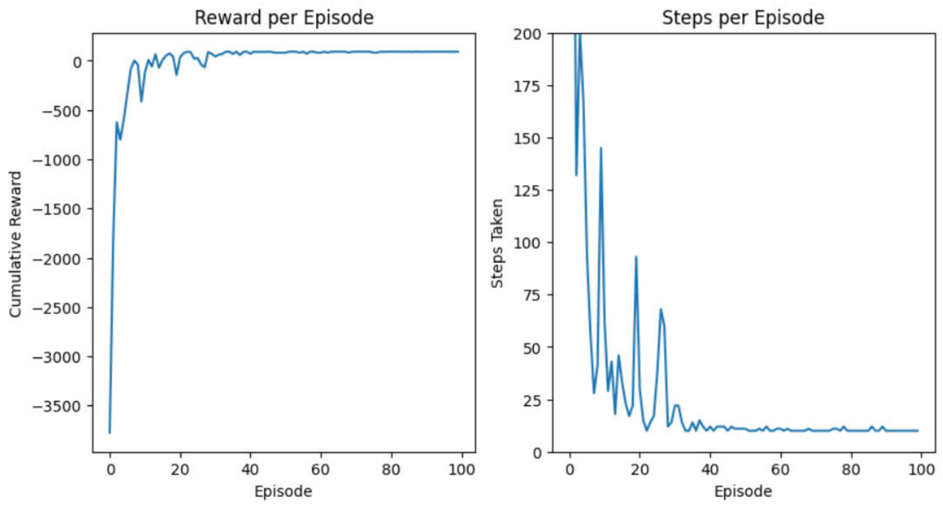

Figure 7: SARSA with $\alpha$ =0.1

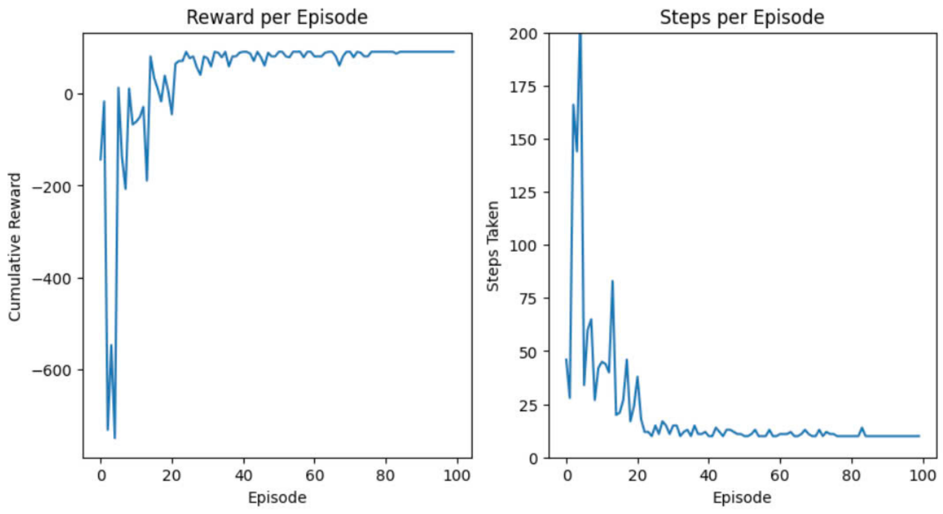

Figure 8: SARSA with $\alpha$ =0.5

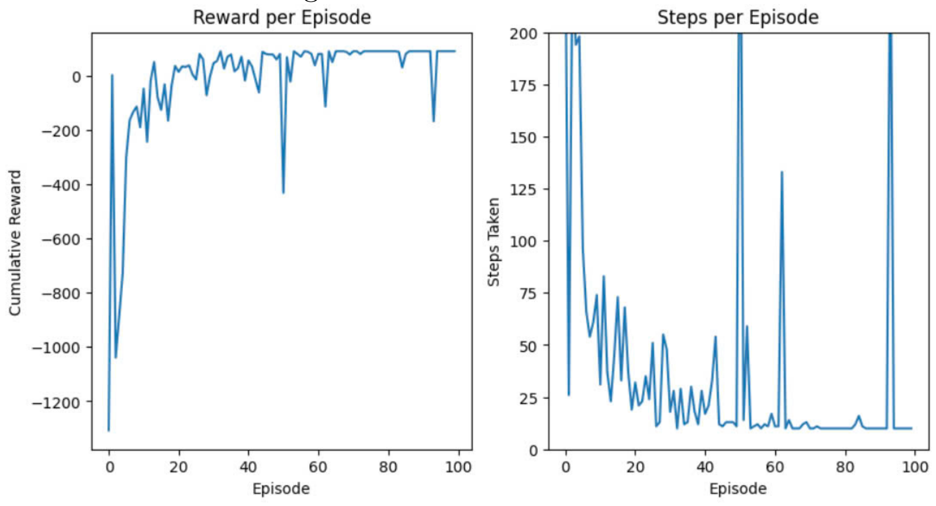

Figure 9: SARSA with $\alpha$ =0.9

Table 3: Result

| $\alpha$ | Average Steps | Average Reward |
|---|---|---|
| 0.1 | 33.74 | -21.84 |
| 0.5 | 21.17 | 38.52 |
| 0.9 | 36.61 | -11.84 |

When $\alpha = 0.1$, the agent using the SARSA algorithm learns slowly but steadily. In the "Reward per Episode" graph, we observe that the cumulative reward gradually increases and stabilizes at a relatively high value after around 30 to 40 episodes. This shows that with a low learning rate, SARSA's updates are conservative, allowing the agent to accumulate rewards gradually and reach a high reward level over time.

In the "Steps per Episode" graph, the number of steps decreases steadily and converges after about 50 episodes. Although the convergence rate is slower, the agent gradually learns to minimize steps to complete each episode. According to Table 3, with $\alpha = 0.1$, the average steps are 33.47, and the average reward is -21.84. This indicates that a lower learning rate helps achieve a relatively high reward, although it requires more steps to reach the optimal path compared to higher values of $\alpha$.

With $\alpha = 0.5$, the agent converges more quickly than with $\alpha = 0.1$. In the "Reward per Episode" graph, the cumulative reward stabilizes faster, after about 20 episodes. However, the final reward is lower than with $\alpha = 0.1$, indicating that a moderate learning rate allows the agent to learn faster but sacrifices some long-term reward potential.

In the "Steps per Episode" graph, the steps decrease rapidly and converge after about 20 to 30 episodes. This suggests that the agent quickly finds a reasonable path through the maze, though it may not be the most optimal path. According to Table 3, with $\alpha = 0.5$, the agent's average steps are 21.17, and the average reward is -38.52, showing that a moderate learning rate results in a faster convergence and a more efficient path but at the expense of a lower cumulative reward.
With $\alpha = 0.9$, the SARSA agent converges the fastest among the tested values. In the "Reward per Episode" graph, we see that the reward curve stabilizes quickly but fluctuates, indicating that with a high learning rate, the agent's reward varies significantly, possibly due to overestimating short-term rewards.

In the "Steps per Episode" graph, the steps required to complete each episode decrease and stabilize within the first 10 to 20 episodes, which is much faster than the other $\alpha$ values. However, the quick convergence is accompanied by instability in the cumulative reward, as seen in both the reward curve and the average reward value. According to Table 3, with $\alpha = 0.9$, the agent's average steps are 36.61, and the average reward is -11.84, showing that a high learning rate achieves fewer steps in the shortest time but results in unstable and lower cumulative rewards.

In summary, the choice of $\alpha$ in SARSA affects the trade-off between convergence speed, path efficiency, and cumulative reward. A lower learning rate favors higher long-term rewards, while a higher learning rate prioritizes speed but may lead to suboptimal performance in terms of reward.

## 5.3 Evaluation.

In this section, we evaluate and compare the performance of the Q-learning and SARSA algorithms within the 6x6 maze environment. By applying the same set of hyperparameters across both algorithms—learning rate ($\alpha$) of 0.1, discount factor ($\gamma$) of 0.9, and initial exploration rate ($\epsilon$) starting at 1.0 with a decay rate until reaching 0.01—we aim to assess their respective capabilities in learning effective navigation strategies under identical conditions.

Using consistent metrics, such as total accumulated rewards and the number of steps required to reach the goal,we plotted performance over time on the same graph to facilitate a direct comparison.This approach provides insights into each algorithm's convergence rate, stability, and overall efficiency in adapting to the environment.By analyzing these trends, we seek to identify key differences and relative strengths between Q-learning's off-policy and SARSA's on-policy approach,thereby deepening our understanding of their application in reinforcement learning tasks with similar constraints and challenges.

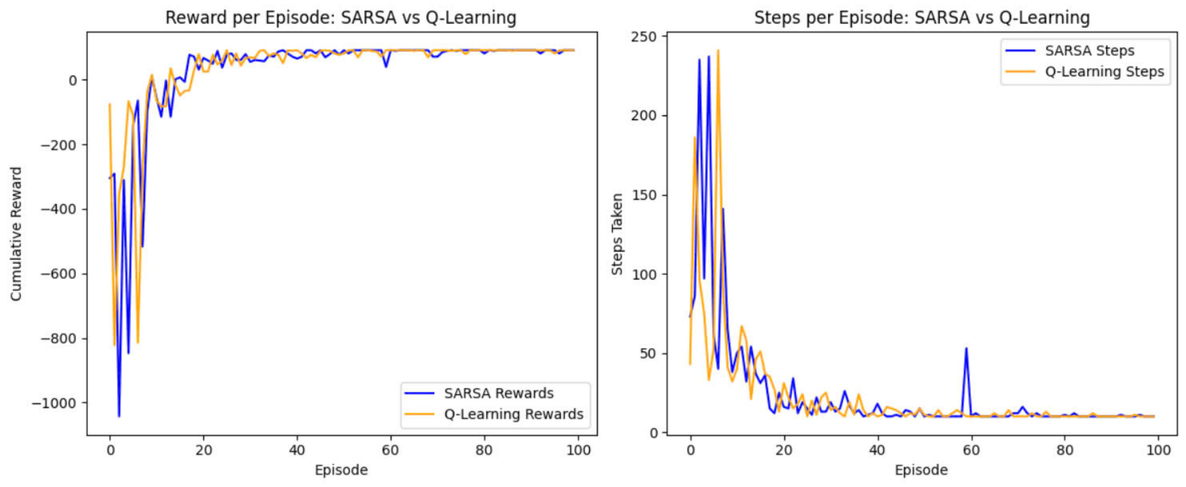

Figure 10: Result Q-learning and Sarsa

Table 4: Average results of reward and steps.

| Average | Q-Learning | Sarsa |
|---|---|---|
| Rewards | 35.46 | 28.58 |
| Steps | 22.79 | 24.27 |

In the "Reward per Episode" graph, both Q-learning and SARSA show a tendency to converge to a high cumulative reward, but there are some slight differences in the learning process. Q-learning (blue line) achieves a higher cumulative reward than SARSA (yellow line) in the long run, indicating that Q-learning tends to find optimal actions with higher rewards over the course of learning.

However, SARSA appears to be more stable, as its cumulative reward increases steadily with fewer fluctuations. This is due to SARSA being an "on-policy" algorithm, meaning it updates based on the agent's actual actions, which helps to create a more stable learning process with less influence from random choices compared to Q-learning, an "off-policy" algorithm. In the "Steps per Episode" graph, Q-learning also converges faster than SARSA in reducing the number of steps needed to reach the goal in each episode.

However, in some initial stages, SARSA shows more fluctuations in the number of steps before stabilizing. This could be due to SARSA's "on-policy" nature, making it more sensitive to current actions, while Q-learning can "explore" other actions through its "off-policy" method, leading to faster convergence in terms of steps.

Results from Table 4 show that Q-learning has a lower average number of steps (22.79 compared to 24.27 for SARSA), indicating that Q-learning is more efficient in finding shorter paths through the maze after convergence

The results indicate that each algorithm has its strengths: Q-learning tends to achieve higher rewards and requires fewer steps but shows greater fluctuation, while SARSA demonstrates greater stability but achieves a lower average reward. This highlights that Q-learning may be better suited for tasks that prioritize maximizing reward and convergence speed, while SARSA is a better choice if stability in learning is a priority.

# 6 Conclusion

## 6.1 Summary

Performance and Convergence: When comparing the two algorithms, Q-Learning and SARSA, we arrive at the following conclusions: Q-Learning generally converges faster than SARSA, mainly due to its approach of optimizing for future rewards. This allows Q-Learning to quickly find optimal paths, particularly in simpler environments. However, Q-Learning can also experience fluctuations when the environment becomes complex and dynamic, as its strategy tends to exploit the highest rewards without necessarily relying on the actual actions taken by the agent. Conversely, SARSA, being an on-policy method, allows for a more stable learning process by relying on the actual actions the agent chooses during the learning process. This characteristic can cause SARSA to take more time to find optimal paths in complex environments, due to the on-policy approach's limitations in seeking out the highest values.

Strengths and Weaknesses: As an off-policy algorithm, Q-Learning can explore higher-value actions, enabling faster convergence, but it may also encounter fluctuations in environments that change frequently. The flexibility in seeking out optimal actions gives Q-Learning an advantage in environments requiring rapid convergence. However, SARSA offers a safer learning process due to its on-policy nature, as it is based on the actual actions taken by the agent and the policy being used. This makes SARSA more stable but can slow down the process of reaching the optimal policy, especially in environments that demand high adaptability.

Implementation Challenges: Both algorithms heavily rely on tuning hyperparameters like the learning rate ($\alpha$) and discount factor ($\gamma$), which directly affect performance and stability. Balancing exploration and exploitation is also essential; excessive exploitation may miss high-reward actions, while too much exploration can hinder efficient task-solving

## 6.2 Lessons Learned

Through the development of a maze game using Q-Learning and SARSA algorithms, I gained valuable insights into Reinforcement Learning (RL) techniques and their potential applications to real-world problems. Building and implementing Q-Learning helped me deepen my understanding of how an agent can learn from experience and adjust its behavior to maximize the rewards it receives in an environment.

The project also highlighted the crucial role of hyperparameter tuning and establishing an effective exploration-exploitation policy to achieve optimal performance. RL techniques, such as Q-Learning and SARSA, are particularly useful for solving problems that lack specific rules for reaching optimal outcomes, making them applicable in many real-world domains. From this project, I realized that RL holds significant potential in fields like supply chain optimization, robotic planning, and the development of automated recommendation systems.

## 6.3 Future Work

To further enhance this project, several research directions and potential improvements include:

**Experimenting with Deep Reinforcement Learning (Deep RL) Methods:** By using deep RL methods such as Deep Q-Network (DQN), the agent's capacity to learn from more complex environments could be expanded, particularly in environments with continuous or large state spaces. This would enable the agent to perform well in games with higher complexity.

**Exploring More Complex Environments:** Applying RL algorithms to environments with high randomness or more intricate interactions, such as 3D environments or multi-agent scenarios. This will help test the scalability of traditional RL algorithms and identify their limitations.

**Using Direct Policy Learning Algorithms:** Direct policy learning methods like Policy Gradient or Actor-Critic could help the agent optimize policies better in environments where calculating Q-values is impractical.

**Integrating RL Techniques into Real-World Applications:** Experimenting with applying RL algorithms to practical fields such as energy system optimization, intelligent traffic control, or drone planning. This would help explore the real-world benefits of RL and the challenges of implementation in actual scenarios.

## References

[1] Marco A Wiering **and** Martijn Van Otterlo. "Reinforcement learning". **in***Adaptation, learning, and optimization*: 12 (2012), **page** 729.

[2] Christopher JCH Watkins **and** Peter Dayan. "Q-learning". **in***Machine learning*: 8 (1992), **pages** 279–292.

[3] Hao Jiang **andothers**. "An Improved Sarsa( $\lambda$ ) Reinforcement Learning Algorithm for Wireless Communication Systems". **in***IEEE Access*: 7 (2019), **pages** 115418–115427. DOI: 10.1109/ACCESS.2019.2935255.

[4] Wang Qiang **and** Zhan Zhongli. "Reinforcement learning model, algorithms and its application". **in***2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*: 2011, **pages** 1143–1146. DOI: 10.1109/MEC.2011.6025669.

[5] Haifeng Zhang **andothers**. *Learning to Design Games: Strategic Environments in Reinforcement Learning*. 2019. arXiv: 1707.01310 [cs.AI]. URL: https://arxiv.org/abs/1707.01310.