Reinforcement Learning for Multi-Agent Cooperation with Internal Reward Mechanism in Maze Environment

Tuan-Anh Pham
Student ID: 2201107, Class: K16 AIRB
Course : Multi-Agent Systems and Reinforcement Learning
Instructor: Hoang-DIeu Vu
11/10/2024

***Abstract*—Reinforcement Learning (RL) is a branch of machine learning where an agent learns how to interact with an environment by taking actions and observing the results. Unlike supervised learning, where labeled data is provided, the agent in RL optimizes its behavior by receiving rewards or penalties from the environment. This makes RL a powerful tool in decision-making systems such as robotics, game AI, and autonomous systems. In the multi-agent maze-solving problem I am working on, multiple agents collaborate to find a way out of the maze. These agents not only optimize their individual actions but also need to coordinate with each other to efficiently achieve a common goal. Each agent receives internal rewards based on the progress of both itself and other agents in the group, improving their cooperation and allowing them to adjust their strategies dynamically. RL in this context helps the agents make intelligent decisions in a complex environment, requiring continuous learning and adaptation through trial and error.**

***Index Terms*—Reinforcement Learning, Q-Learning, SARSA, Monte Carlo, Value iteration, Policy iteration, Maze, Artificial Intelligence.**

I. Introduction

In this project, I focus on solving a classic problem in Reinforcement Learning (RL): the multi-agent maze-solving problem. Two agents are placed in a maze and must find their way out by moving through square cells step by step. The objective of the two agents is to jointly reach the destination with the shortest possible number of steps. The maze environment contains walls, and the agents must learn how to navigate through different states to achieve their objectives as efficiently as possible. .

The goal is for both agents to reach the destination with the fewest steps possible.

Using Multi-AgentReinforcement Learning (MARL) offers numerous benefits, including decentralization and the ability to learn from one another, which enhance performance and flexibility in complex cooperative tasks. However, effectively coordinating agents in a non-communicative environment remains a significant challenge. Traditional RL algorithms often rely on communication between agents to share information and coordinate actions, but this is not always feasible in real-world scenarios where communication may be limited or unreliable[1].

The primary objective of this project is to implement and compare various traditional RL algorithms, such as Q-Learning, SARSA, Policy Iteration, Value Iteration, and Monte Carlo, to address the multi-agent maze-solving problem. I will evaluate these algorithms based on their performance in solving the problem, convergence speed, and the quality of the optimal path that both agents can achieve[2].

This report is structured as follows:

- Theoretical background, introducing core RL concepts and algorithms.
- Methodology, where we describe the implementation details of the RL problem.
- Experimental setup, detailing the environment and parameters.
- Results, presenting the performance of the algorithms.
- Discussion, analyzing the results and challenges encountered.
- Conclusion, summarizing key findings and suggestions for future work.

II. Theoretical Background

A. Multi-Agent Reinforcement Learning

Reinforcement Learning (RL) is built on the interaction between an **agent** and an **environment**. The agent takes **actions** based on its current **state**, aiming to maximize the cumulative **reward** it receives from the environment over time. The **policy** defines the agent's behavior, mapping states to actions. RL operates under the framework of a **Markov Decision Process (MDP)**, which assumes that the future state depends only on the current state and action, not on previous states or actions[1],[2] .

Agent:
Instead of having a single agent, your problem involves multiple agents that interact with each other and with the environment. Each agent tries to optimize its actions based on information received from the environment and information shared by other agents.

Environment:
The system in which the agents operate. This is where states, rewards, and state transitions are defined. In your problem, the environment consists of a maze and the

agents' goals.

State:

A representation of the current situation in the environment from the perspective of each agent. In your case, the state might include the current position of each agent and other factors within the maze environment.

Action:

The choices available to the agents at each state, such as moving up, down, left, or right within the maze.

Reward:

Each agent receives a reward after performing an action, which indicates how good or bad that action was. In your problem, the reward could include internal rewards, which are calculated based on information about the time taken to reach goals by other agents.

Internal Reward:

Internal rewards play an important role in promoting cooperation among multiple reinforcement learning (RL) agents. Unlike conventional rewards, internal rewards are designed to avoid conflicts in dilemma situations. With well-designed internal rewards, agents can cooperate without the need for extensive communication. This method reduces the need for information sharing, thereby increasing efficiency in multi-agent systems.

Policy:

The strategy used by the agents to decide which action to take at each state. In your problem, agents need to balance between exploration, to discover better strategies, and exploitation, to use the knowledge they have already gained to maximize rewards.

In this multi-agent problem, sharing information between agents helps to optimize the learning process, such as sharing the best times to reach goals. This helps agents improve their decision-making as they move through the maze.

A critical concept in RL is the trade-off between **exploration** and **exploitation**. The agent needs to explore the environment to discover better actions, but also exploit the knowledge it has gained to maximize rewards. This balance is key to learning an optimal policy [2],[3].

B. Cooperation in Maze Problem

[1]

	Goal X	Goal Y
Agent A	t_{AX}	t_{AY}
Agent B	t_{BX}	t_{BY}

Fig. 1: Distance

- t_{AX}: The minimum number of steps agent A needs to reach goal X (for example, goal S).
- t_{AY}: The minimum number of steps agent A needs to reach goal Y (for example, goal L).
- t_{BX}: The minimum number of steps agent B needs to reach goal X (for example, goal S).
- t_{BY}: The minimum number of steps agent B needs to reach goal Y (for example, goal L).

These values help determine how agents can cooperate to achieve the most efficient outcome. From this, agents can adjust their strategies to avoid conflicts and achieve optimal cooperation[1].

C. Goal selection

[1]

1) Each agent i stores the information about the minimum steps from its start position to the goals t_{iX}, t_{iY} (for example: t_{AS}, t_{AL} for agent A, and t_{BS}, t_{BL} for agent B).
2) When an agent i reaches a goal with a new number of steps that is smaller than the current steps, $t_{iX}^{new} < t_{iX}$, it sends this information to the other agents j.
3) Each agent j updates its own minimum steps based on the information received from agent i and selects the goal with the smallest minimum steps. The update rule for the goal is:

$$\min(t_{iX}, t_{jX}) \quad \text{and} \quad \min(t_{iY}, t_{jY})$$

4) After receiving the information, the agent selects the goal with the smallest number of steps. For instance, if $t_{AS} < t_{AL}$, agent A will choose goal S.
5) The information-sharing and updating process happens continuously, ensuring that the agents always have the most up-to-date information about the minimum steps.

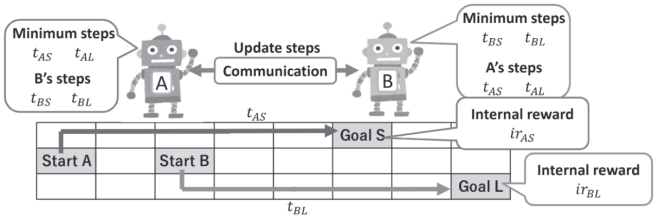

Fig. 2: Information-sharing

We can generalize the inequality $ir_L^Y > ir_S^Y$ described in the previous section in the maze problem. To acquire cooperative behaviors, the internal rewards ir should be designed as the following equations. The internal rewards ir_{AS} and ir_{AL} are the internal rewards of the goals S and L for agent A, while ir_{BS} and ir_{BL} are the internal rewards of the goals S and L for agent B.

Considering that the minimum steps $t_{AS}, t_{AL}, t_{BS}, t_{BL}$, the Q-values of the action toward the goal S at the starts A and B are described as follows:

$$\gamma^{t_{AS}} ir_{AS} > \gamma^{t_{BS}} ir_{BS}. \tag{1}$$

Similarly, the Q-values of the action toward the goal L are described as follows:

$$\gamma^{t_{AL}} ir_{AL} > \gamma^{t_{BL}} ir_{BL}. \tag{2}$$

Since the agent A must reach the goal S, the following inequality should be satisfied in any states:

$$ir_{AS} > \frac{ir_{AL}}{\gamma^{t_{AL}-t_{AS}}}. \tag{3}$$

In the same manner, the following inequality for agent B should also be satisfied in any states:

$$ir_{BL} > \frac{ir_{BS}}{\gamma^{t_{AL}-t_{BL}}}. \tag{4}$$

Finally, the following equations are employed instead of the above inequalities:

$$ir_{AS} = \gamma^{t_{AL}-t_{AS}} ir_{AL} + \delta, \tag{5}$$

$$ir_{BL} = \frac{ir_{BS}}{\gamma^{t_{AL}-t_{BL}}} + \delta. \tag{6}$$

These equations ensure that the agents can optimize their steps and internal rewards to achieve the best cooperative behavior in the maze environment[1].

Require: Agents take starting positions
$t^i_{jx} = MaxStep\,(i, j = [0, AgentNumber), x = [0, GoalNumber))$
$T_{ix} = MaxStep\,(i = [0, AgentNumber), x = [0, GoalNumber))$
$g_i \in G$

for *iteration* = 0 to *MaxIteration* **do**
  for All agents reach the goals or *step* = 0 to *MaxStep* **do**
    Agents observe their states
    Agents choose actions
    The agents which don't reach the goal update Q-value
    if Agent i has reached goal x **then**
      $T_{ix} = step$
    end if
  end for
  for i = 0 to *AgentNumber* **do**
    if $T_{ix} < t^i_{ix}$ **then**
      $t^i_{ix} = T_{ix}$
      Agent i send t^i_{ix} to other agents
    end if
  end for
  Agents estimate internal reward
  Agents update Q-value by the internal reward
end for

Fig. 3: Goal

III. Methodology

A. Problem Definition

The problem we are solving is a **multi-agent cooperation problem in a maze environment**. The task for the agents is to navigate through the maze to achieve their individual goals while sharing information with each other to improve cooperation efficiency. Each agent must learn to find the optimal path using Reinforcement Learning (RL) and by interacting with other agents. The maze is represented as a grid, where each cell is either passable space or a wall [4],[5].

- **State space**: Each state represents the specific position of each agent in the maze. This position is determined by the agent's coordinates within the grid. The initial state of each agent is the starting position, while the goal state is defined by the agent's target location. Agents can share information about their states to optimize the learning process.
- **Action space**: Each agent can perform actions that involve moving from its current position to neighboring cells within the maze (up, down, left, right). Valid actions are determined by whether there is a wall between neighboring cells. If there is no wall, the agent can move to the next cell; if not, the action is invalid and will result in a penalty.
- **Reward structure**: Each agent receives a reward after performing an action, which includes:
  - A negative reward (-0.1) for each movement, encouraging the agent to find the shortest path to the goal.
  - A negative reward (-1) for invalid actions, such as moving into a wall.
  - A large positive reward (+10) when the agent reaches its goal.
  - Internal rewards are shared between agents, allowing them to learn from each other's experiences, especially regarding the time taken to complete the goal.

The objective of the agents is to learn an optimal policy through cooperation and information sharing, which helps minimize the number of steps needed to reach their respective goals within the maze environment.
The goal of the agent is to find an optimal policy that minimizes the number of steps needed to reach the goal[1],[4].

B. Algorithm Selection

Several RL algorithms were considered for solving this problem, including Q-Learning, SARSA, Policy Iteration, Value Iteration, and Monte Carlo methods. The reasons for choosing these algorithms are:

- **Q-Learning**: It is a simple and efficient model-free, off-policy method suitable for environments like mazes, where the optimal policy may differ from the agent's current exploration policy[6].
- **SARSA**: This algorithm, being on-policy, is useful for environments where we want the agent to follow a consistent strategy during learning[6].
- **Policy Iteration and Value Iteration**: These model-based algorithms are chosen for their ability to explicitly compute and refine policies and value functions in structured environments[6].

Algorithm 1 Q-Learning Algorithm

Initialize $Q(s, a)$ arbitrarily for all states s and actions a
for each episode **do**
  Initialize state s
  repeat
    Choose action a using epsilon-greedy policy based on $Q(s, a)$
    Take action a, observe reward r and next state s'
    Update Q-value:

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

    $s \leftarrow s'$
  until s is terminal
end for

Fig. 4: Algorithm Q-learning

Algorithm 2 SARSA Algorithm

Initialize $Q(s, a)$ arbitrarily for all states s and actions a
for each episode **do**
  Initialize state s
  Choose action a using epsilon-greedy policy based on $Q(s, a)$
  repeat
    Take action a, observe reward r and next state s'
    Choose next action a' using epsilon-greedy policy based on $Q(s', a')$
    Update Q-value:

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

    $s \leftarrow s'$
    $a \leftarrow a'$
  until s is terminal
end for

Fig. 5: Algorithm Sarsa

Algorithm 3 Policy Iteration Algorithm

Initialize policy π arbitrarily
repeat
  Policy Evaluation: For each state s, calculate $V^{\pi}(s)$ by solving:

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V^{\pi}(s')]$$

  Policy Improvement: Update policy π to maximize expected value:

$$\pi(s) \leftarrow \arg\max_{a} \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V^{\pi}(s')]$$

until policy π converges to optimal policy

Fig. 6: Policy Iteration

Algorithm 4 Value Iteration Algorithm

Initialize $V(s)$ arbitrarily for all states s
repeat
  for each state s **do**
    Update value for state s:

$$V(s) \leftarrow \max_{a} \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V(s')]$$

  end for

- **Monte Carlo methods**: Useful for estimating values based on complete episodes, they are effective when the environment is deterministic, as is the case with the maze problem[6].

Algorithm 5 Monte Carlo Algorithm

Initialize $Q(s, a)$ arbitrarily for all states s and actions a
for each episode **do**
  Initialize empty list to store episode history
  Initialize state s
  repeat
    Choose action a using epsilon-greedy policy based on $Q(s, a)$
    Take action a, observe reward r and next state s'
    Append (s, a, r) to episode history
    $s \leftarrow s'$
  until s is terminal
  for each state-action pair (s, a) in the episode **do**
    Calculate return G_t from time step t onward
    Update Q-value:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[G_t - Q(s,a) \right]$$

  end for
end for

Fig. 8: Algorithm MonteCarlo

IV. Experimental Setup

A. Environment Setup

In this multi-agent maze-solving problem, each agent is placed in a grid-based environment where they must navigate to reach their respective goals. The environment is structured as follows:

Agents: There are multiple agents, each with a defined start position and a goal position. These agents must cooperate to optimize their paths in reaching the goals while sharing their progress (minimum steps to goals).

Actions: Each agent can move in four possible directions: up, down, left, and right. The set of actions is defined for each agent, and their selection of actions follows a policy that balances exploration and exploitation.

Shared Q-table: The agents share a common Q-table for learning. This allows them to cooperate by sharing their Q-values during the learning process, helping them make informed decisions based on not only their own experience but also the shared knowledge.

1) Key Hyperparameters: **Learning Rate** (): Set to 0.1, this parameter defines how quickly each agent updates its knowledge of the environment and adjusts its Q-values based on rewards received from actions.

Discount Factor (): Set to 0.9, this parameter controls the importance of future rewards compared to immediate rewards, allowing agents to plan ahead in their paths.

Exploration Rate (): Starting at 1.0, the exploration rate decreases over time (decay rate = 0.995). This controls the trade-off between exploring new paths (random actions) and exploiting known optimal paths (actions with the highest Q-values).

Delta (): A constant factor (0.1) used to calculate internal rewards, helping agents cooperate by sharing information and adjusting their strategies based on the progress of other agents.
Episodes: Each agent undergoes 1000 training episodes. During each episode, agents take multiple steps (up to a predefined maximum number) to navigate the maze and update their Q-values.
This cooperative environment encourages agents to work together by sharing the steps they take to reach goals, ensuring that they learn not only from their individual experience but also from the collective knowledge of other agents. This cooperation helps solve the dilemma problem in multi-agent settings where agents must balance their own goals with the overall optimal solution for the group.

- **Learning rate** (α): 0.1. This defines how quickly the agent updates its knowledge of the environment.
- **Discount factor** (γ): 0.99. This determines the importance of future rewards compared to immediate rewards.
- **Exploration rate** (ϵ): Initially set to 0.1 and decayed over time. This controls the balance between exploration (choosing random actions) and exploitation (choosing the best-known action).
- **Number of episodes**: 1000 episodes were run to allow sufficient training time for the agent to learn the optimal policy.

These hyperparameters were selected after several trials, with the aim of achieving fast convergence while maintaining the balance between exploration and exploitation.

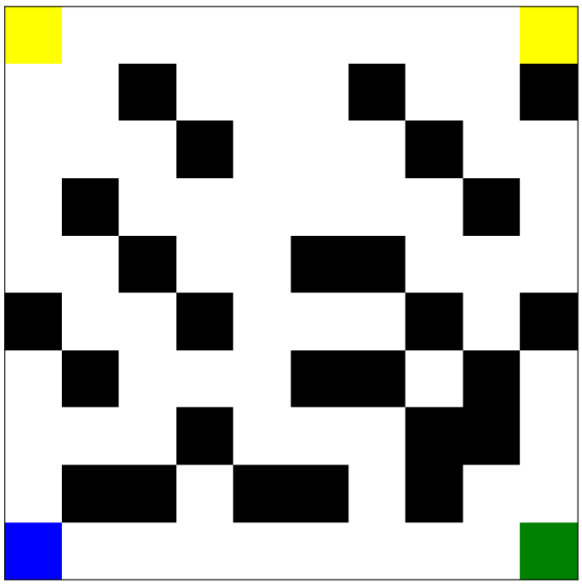

Fig. 9: Maze

B. Data

The maze environment used in this experiment was custom-designed. Each maze is represented as a grid of size 10x10 and contains fixed walls to increase complexity and create narrow pathways and corridors. The maze layout includes start and goal positions for two agents, as well as walls that restrict movement. The walls were defined as a fixed set of obstacles, with one specific wall removed to modify the challenge level.
Maze Structure:

- **Size:** The maze has a fixed size of 10x10.
- **Start Positions:** The agents start at opposite corners: agent A starts at (0, 0), and agent B starts at (0, 9).
- **Goal Positions:** The goal for each agent is located at the bottom corners: agent A's goal is at (9, 0), and agent B's goal is at (9, 9).
- **Fixed Walls:** A set of fixed walls creates challenging pathways for agents to navigate through, blocking direct access to the goal.
- **Wall Layout:** Walls are strategically placed in the maze at specific coordinates, such as (1, 1), (1, 2), (2, 3), etc., to increase difficulty and create narrow paths. However, certain walls can be removed, such as the wall at (6, 4), to adjust the challenge level.

The agents are required to navigate through this environment using their pre-defined movement actions (up, down, left, right) and cooperate to reach their respective goals. The visualization of the maze can be generated using the `plot_maze` function, which highlights the positions of the agents, their goal positions, and the walls.

V. Results

In this section, I present the results for each algorithm, including Q-Learning, SARSA, Monte Carlo, Policy Iteration, and Value Iteration. I utilize tables, charts, and graphs to display performance metrics such as cumulative rewards, the number of episodes required to converge, and the success rate. The parameters used for the algorithms include: gamma = 0.9, learning rate = 0.1, exploration decay = 0.995, and delta = 0.1. Additionally, I gradually adjusted the exploration parameter over time to observe its impact on the convergence process and the overall efficiency of each algorithm.

A. Q-Learning

The Q-Learning algorithm demonstrated a gradual improvement in performance as the Q-values updated over time. The following graph illustrates the cumulative reward per episode, showing how the agent converged to an optimal policy. The values of ϵ (0.1, 0.5, and 0.9) affected the exploration-exploitation balance, with lower values favoring quicker convergence but slower exploration of the environment.
float

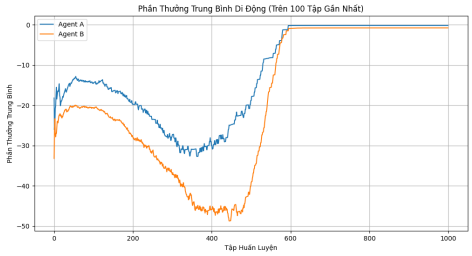

Fig. 10: ϵ = 0.1

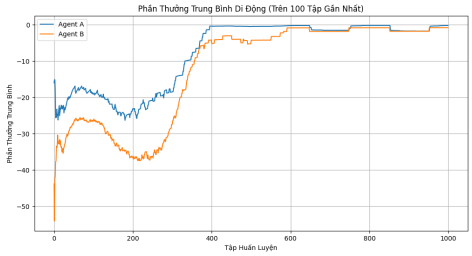

Fig. 11: ϵ = 0.5

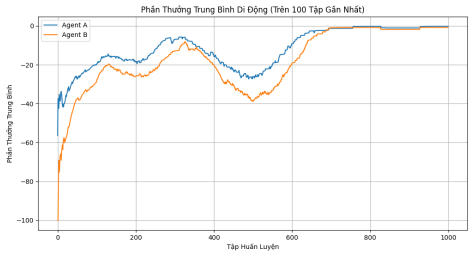

Fig. 12: ϵ = 0.9

Exploration Rate	Running Time (seconds)
0.1	4.2357
0.5	4.5312
0.9	5.76

TABLE I: Runtime for different exploration rates Q-learning

B. SARSA

SARSA, being an on-policy algorithm, showed more stable performance compared to Q-Learning. This is attributed to its reliance on the current policy for updates, leading to oscillations in performance, exploration decay=0.995.

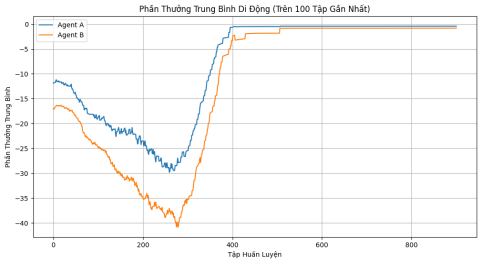

Fig. 13: ϵ = 0.1

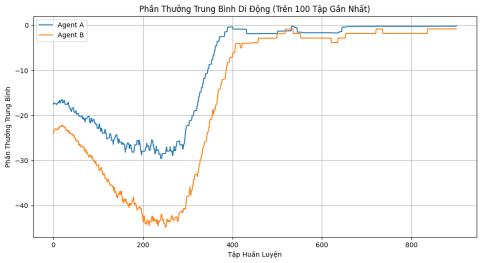

Fig. 14: ϵ = 0.5

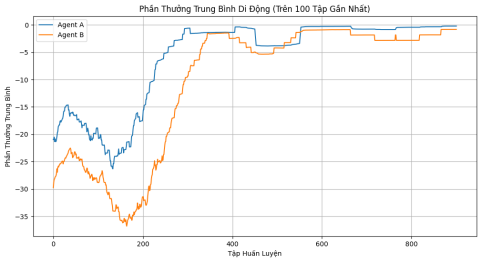

Fig. 15: ϵ = 0.9

Exploration Rate	Running Time (seconds)
0.1	4.375
0.5	3.97
0.9	4.58

TABLE II: Runtime for different exploration rates SARSA

C. Monte Carlo

The Monte Carlo algorithm, using a fixed $\epsilon = 0.5$ and varying γ, showed a slower convergence rate compared to Q-Learning and SARSA. However, it performed well in environments where episodes could be simulated until completion. The results highlight how different discount factors influenced the learning process. However, the performance of this algorithm is quite poor when applied to large-sized maze environments.

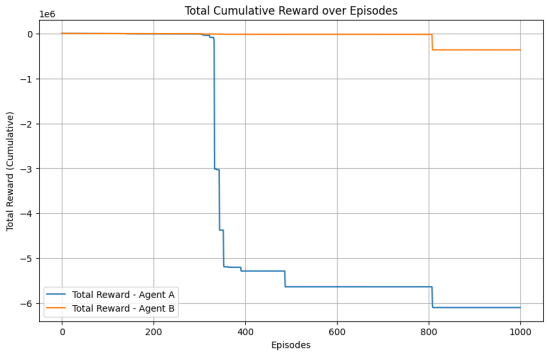

Fig. 16: Monte Carlo

Gamma	Running Time (seconds)
0.1	2462.125
0.9	761.678

TABLE III: Algorithm Running Time by Gamma Value

D. Policy Iteration

Policy Iteration showed fast convergence compared to the other methods due to its iterative approach, alternating between policy evaluation and policy improvement. It successfully found the optimal policy in fewer iterations than Q-Learning or SARSA. The result is only γ = 0.99 which gives the optimal policy.

E. Value Iteration

Value Iteration, like Policy Iteration, converged quickly by iteratively updating the value function until convergence. It provided a more straightforward implementation for finding optimal policies but required careful consideration of the stopping criteria. The result is only γ = 0.99 which gives the optimal policy.

F. Performance Comparison

The following table summarizes the performance of each algorithm in terms of stability, speed of convergence, and the optimal policy found.
In testing various algorithms for solving the multi-agent maze problem, we evaluated performance based on two main criteria: **training time** and **average steps per episode**. After conducting experiments, we found that the SARSA algorithm was the most effective, followed by Q-Learning, while Monte Carlo and policy iteration methods such as Value Iteration and Policy Iteration proved inefficient in this environment.

1. SARSA - The Most Effective Algorithm

SARSA was shown to be the most effective algorithm, with the lowest training time among all trials. As an on-policy reinforcement learning algorithm, SARSA optimizes learning in complex environments without consuming excessive resources. Key points about SARSA include:

- **Training Time**: SARSA had the shortest training time, saving resources and minimizing computational time.
- **Average Steps**: SARSA also achieved a relatively low average number of steps per episode, indicating its ability to optimize the path in the maze.

Due to its ability to update values based on the current policy, SARSA demonstrated high stability and suitability for uncertain environments, making it the top choice for real-world applications.

2. Q-Learning - Second Most Effective

Q-Learning, although an off-policy algorithm, also demonstrated high performance in pathfinding. It can optimize independently of the current policy, gradually improving its ability to find the optimal path. Notable points include:

- **Training Time**: Q-Learning required more training time than SARSA but remained acceptable, ranking second after SARSA.

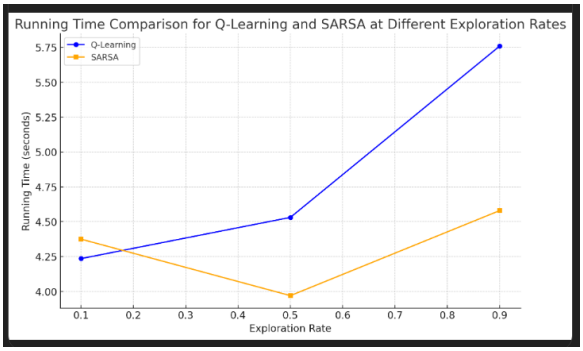

Fig. 17: Time

- **Average Steps**: Q-Learning also showed strong path optimization capability, helping the agent find efficient routes within the maze.

While not as fast as SARSA, Q-Learning remains a viable option when optimal step calculations are needed and can perform well across various environments.

3. Monte Carlo and Value, Policy Iteration - Ineffective

Monte Carlo and policy iteration methods such as Value Iteration and Policy Iteration were not suitable for this maze environment. Their limitations include:

- **High Training Time**: Monte Carlo had long training times, making it impractical for problems requiring fast response times.
- **Poor Optimization Capability**: Value Iteration and Policy Iteration required numerous iterations and could not achieve optimal efficiency like SARSA and Q-Learning in this environment.
- **Limited Real-World Application**: Due to the complexity and lengthy training time, these methods are not ideal for applications that require real-time processing.

Conclusion

Among the tested algorithms, **SARSA** proved to be the best choice, with the **shortest training time** and high performance in finding the optimal path. **Q-Learning** ranked second, offering an effective solution but requiring more time than SARSA. **Monte Carlo** and **Value Iteration** and **Policy Iteration** were not effective in this maze environment due to high time and resource requirements.
These results confirm that, for the multi-agent maze problem, SARSA is the best choice, especially when time and resource constraints are critical factors.

VI. Conclusion

This project implemented and applied multi-agent reinforcement learning (MARL) methods to tackle the challenge of agent cooperation under limited or no communication. Through the design and optimization

Algorithm	Stability	Speed	Optimal Policy
Q-Learning	Medium	Medium	Yes
SARSA	High	Fast	Yes
Monte Carlo	Low	Slow	No
Policy Iteration	Low	High	No
Value Iteration	Low	High	No

TABLE IV: Performance Comparison

of internal reward structures, agents were able to effectively coordinate and achieve shared goals in complex environments without requiring direct interaction.

The proposed methods emphasized the importance of updating goal values dynamically and adjusting reward structures to encourage agents to prioritize cooperative actions. These techniques ensured that even in scenarios where communication was restricted, the agents could still converge on efficient solutions to multi-agent tasks such as maze solving.

Several key insights were gained throughout this project. First, balancing exploration and exploitation was critical, especially in environments with sparse communication. Effective reward tuning helped improve convergence rates and facilitated cooperation. Additionally, larger state spaces, as observed in complex scenarios, highlighted the need for more scalable learning methods to handle high-dimensional environments.

The project also demonstrated the flexibility of MARL approaches, which can be adapted to a wide range of real-world tasks, including traffic coordination, resource management, and automated systems. The custom environments created for this study provided a robust platform for testing agent interactions, offering valuable insights into how MARL can be applied to real-world multi-agent systems.

Looking forward, several avenues for future research and improvement can be explored. One direction is to investigate how deep reinforcement learning (DRL) methods, such as Deep Q-Networks (DQN), can handle more complex environments with larger state spaces and richer reward structures. DRL may provide better scalability and learning efficiency in high-dimensional tasks where traditional RL methods face limitations[1],[7].

Another area of improvement is to extend the current MARL framework to more dynamic environments, incorporating stochastic transitions and time-varying rewards. This could enhance the agents' ability to adapt to rapidly changing environments and improve overall system performance. Finally, hybrid methods that combine aspects of both traditional RL and DRL techniques may offer new ways to optimize agent behavior and cooperation in multi-agent systems[1],[7],[8].

ACKNOWLEDGMENT

I would like to thank Dr.Vu Hoang Dieu for the guidance and support throughout this project.

REFERENCES

[1] Fumito Uwano et al. "Multi-agent cooperation based on reinforcement learning with internal reward in maze problem". In: *SICE Journal of Control, Measurement, and System Integration* 11.4 (2018), pp. 321–330.

[2] Richard S Sutton. "Reinforcement learning: An introduction". In: *A Bradford Book* (2018).

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

[4] Dwi Astuti Wahyu Nurhayati. "Improving Students' English Pronunciation Ability through Go Fish Game and Maze Game." In: *Dinamika ilmu* 15.2 (2015), pp. 215–233.

[5] Ahmad EL Sallab et al. "Deep reinforcement learning framework for autonomous driving". In: *arXiv preprint arXiv:1704.02532* (2017).

[6] Richard Bellman. "A Markovian decision process". In: *Journal of mathematics and mechanics* (1957), pp. 679–684.

[7] Nicola Dalla Pozza et al. "Quantum reinforcement learning: the maze problem". In: *Quantum Machine Intelligence* 4.1 (2022), p. 11.

[8] Stuart J Russell and Andrew Zimdars. "Q-decomposition for reinforcement learning agents". In: *Proceedings of the 20th international conference on machine learning (ICML-03)*. 2003, pp. 656–663.