参赛学生姓名：孙浩宸

中学：北京师范大学附属实验中学

省份：北京

国家/地区：中国/北方赛区

指导老师姓名：常菱芸

指导老师单位：北京师范大学附属实验中学

论文题目：Generalization and Optimization of the Nash-Q Algorithm in Multi-Agent Reinforcement Learning

Research Paper for the Mathematics Award
of the S.-T. Yau High School Science Award

Generalization and Optimization of the Nash-Q Algorithm in Multi-Agent Reinforcement Learning

Author: Haochen Sun

Academic Supervisor: Lingyun Chang

November 5, 2025

Abstract

Reinforcement Learning (RL), a core machine-learning paradigm, enables agents to learn reward-maximizing policies via environmental interaction; game theory offers strategic models for such contexts.

This paper begins by introducing classical reinforcement learning algorithms and then extends the single-agent setting to a multi-agent environment. Traditional RL algorithms often perform poorly in such scenarios because they typically adopt greedy policies focused only on maximizing the agent's own reward function, ignoring the influence of others' actions. To address this, we introduce the Markov game framework. We then present two strategies from game theory for two-player zero-sum games: the safety strategy and the mixed-strategy Nash equilibrium. Both strategies take into account the value functions of the agent and its opponent. By embedding these into multi-agent reinforcement learning algorithms, we aim to develop more efficient and accurate methods.

After incorporating game-theoretic concepts, we introduce two modified Q-learning algorithms: the Minmax-Q algorithm, which uses the max-min strategy, and the Nash-Q algorithm, based on the Nash equilibrium strategy. Both improve upon the TD(0) method. We provide a comprehensive summary of when to use each strategy. However, the Nash-Q algorithm has notable limitations in scenarios with multiple Nash equilibria, as agents may fail to converge to the same one or get stuck locally. To address this, we propose a penalty mechanism that encourages agents to select the same Nash equilibrium, improving accuracy. We also prove that this penalty mechanism does not alter the optimal policy of the original model nor disrupt convergence.

Finally, we adopt a community parking resource allocation scenario as a simulation testbed, interpreting a grid-based game as a simplified parking environment. In this system, each driver is an independent agent, parking space occupancy is the state, parking choices are actions, walking distance and convenience determine the reward, while conflicts and congestion act as penalties. Comparing the Nash-Q algorithm with and without the penalty mechanism, we find the proposed mechanism effectively guides agents from inefficient equilibria—where multiple vehicles compete for the same spot—to more efficient ones with better distribution. This significantly reduces conflict frequency and deadlocks, improves convergence speed and policy consistency, demonstrating practical value for mitigating urban parking conflicts and optimizing resource allocation.

Keywords: Reinforcement Learning, Game Theory, Nash Equilibrium Strategy, Nash-Q Algorithm, Penalty Mechanism

Contents

1 Introduction

1.1 Background and Significance

Game theory, as a discipline, has become increasingly important with the development of the times and is widely applied in fields such as economics, mathematics, and military strategy, holding an extremely significant position. The structural components of mathematical models primarily include players, actions, strategies, information, and payoff functions. Among these, players, strategies, and payoff functions are the most fundamental structural elements, primarily focusing on formalizing the interactions among these structural components, predicting players' actions, and calculating equilibrium strategies. Machine learning and game theory have always been closely related. In 1928, the father of the computer, John von Neumann, published a paper titled "On the General Theory of Choice Games," which laid the foundation for his game theory ideas and introduced the principle of the maximum and minimum values [1]. Alan Turing, a pioneer in artificial intelligence [2], utilized game theory knowledge to provide theoretical support for computer programs in chess. Various phenomena indicate that machine learning and game theory are closely related. At times, game theory requires the computational power and training of machine learning, while at other times, machine learning relies on game theory as a theoretical foundation. As John von Neumann once said: "Real life is full of bluffing and deception; I cannot predict my opponent's next move. If I want to use a theory to explore the underlying patterns of life, then game theory is the most appropriate choice [3]."

Reinforcement learning (RL) in machine learning has recently sparked a wave of enthusiasm because it can achieve heights unattainable by humans in many fields. Strategy-based reinforcement learning can adapt to environments better and faster, finding optimal paths. Reinforcement learning is widely applied in our daily lives, such as robots in human-machine competition modes in games. The artificial intelligence system Alphago, which defeated Lee Sedol in a Go match with a score of 4:1 in 2016; and the chess-playing AI [4] can predict the next 13 moves in a game of chess. These systems all demonstrate strong competitive abilities and achieve extremely high win rates in human-machine competitions.

Reinforcement learning is an effective tool for intelligent agents to learn through interaction with their environment. However, in multi-agent environments, classical single-agent reinforcement learning algorithms become inefficient because multiple agents coexist and influence each other. The original method of considering only the agent's own value function and strategy becomes irrational. Game theory, on the other hand, was developed for multi-agent competition and provides methods for finding strategies in multi-agent environments. Markov games provides a framework for multi-agent reinforcement learning [5]. Incorporating game theory strategy concepts into multi-agent reinforcement learning algorithms can achieve maximum efficiency. Therefore, using game theory concepts to extend multi-agent reinforcement learning is both feasible and necessary.

In 1994, regarding two-player zero-sum games, Littman proposed an improvement to

the Q-learning algorithm by replacing the greedy strategy with the maximum-minimum strategy, resulting in the Minimax-Q algorithm [6]. This approach not only considers the agent's own strategy and value function but also accounts for the opponent's actions, achieving significant success in comparative experiments. However, the paper of Littman only considered the algorithm and improved the Q-learning algorithm using the maximum-minimum strategy. Subsequently, many scholars proposed using Nash equilibrium strategies to improve reinforcement learning algorithms, resulting in the Nash-Q algorithms [7, 8], which also achieved good results. Both of these algorithms use game theory to improve multi-agent reinforcement learning. This paper will introduce the two algorithms, summarize their advantages and disadvantages, and analyze when certain game theory strategies are most effective, providing a new perspective for those who wish to improve multi-agent reinforcement learning using game theory. Additionally, the Nash-Q algorithm suffers from a choice problem, which agents may not choose the same Nash equilibrium, in models with multiple Nash equilibria, making algorithm accuracy reduced. It may also get stuck in a local Nash equilibriuma locally optimal solution after iteration. To address this, we propose a penalty mechanism to force agents to choose the same Nash equilibrium, thereby improving algorithm efficiency.

1.2 Related Work

1.2.1 Reinforcement Learning

Even in the early days of computer science and artificial intelligence, game-playing agents captivated researchers with their ability to learn and think like humans. Shortly after the invention of the first computer in 1950, scholars began to focus on enabling computers to learn chess and checkers games [9]. These methods of enabling computers to learn strategies on their own were the early forms of reinforcement learning. Research on machine games has continued for more than half a century, and Go and chess remain important research topics in the field of artificial intelligence.

Until the 1970s, computer agents were still unable to match human masters in chess. However, in the 1980s and 1990s, new technologies, modern computer hardware, and innovative academic achievements emerged, leading to rapid advancements in the gaming capabilities of computer agents. Among these, complete information machine games achieved several milestone accomplishments. In 1980, the "Berliner" at Carnegie Mellon University developed an artificial intelligence computer program based on reinforcement learning that defeated the then-champion of backgammon Luigi Villa with a score of 7:1, marking the first time a computer program had defeated a human top player in an intellectual game [10]. In 1994, the computer program Cinook defeated the reigning champion in checkers, Marion Tinsley. By 2007, Chinook had completely mastered the game of checkers, with even the top checkers players only able to draw against it; In 1997, the supercomputer "Deep Blue" defeated international chess grandmaster Garry Kasparov with a score of 3.5 to 2.5, shocking the world. Afterward, Kasparov recalled that the second game was crucial, as the machine's performance exceeded his expectations, often sacrificing short-term gains and exhibiting a very human sense of danger, showing a very human sense of danger.

In the decades that followed, the "battle" between computers and humans in games never ceased. Research into two-player zero-sum games has been ongoing for many years, and some complex reinforcement learning algorithms have achieved a certain level of general intelligence capable of solving complex problems, reaching human-level performance

in Go and video games. The learning capabilities of computer agents have reached new heights with the advancement of theoretical knowledge in reinforcement learning and game theory. The focus of research has gradually shifted from sequential games, where players act in a specific order, to simultaneous games [11], where players act simultaneously. In sequential games, the opponent's actions can be regarded as part of the environment, so the reinforcement learning process only needs to consider maximizing its own rewards. However, in simultaneous games, due to the uncertainty of the opponent's actions, treating the opponent as part of the environment would reduce the game-playing ability. Therefore, we need to extend single-agent reinforcement learning to multi-agent reinforcement learning.

This paper is based on this background and uses a two-player zero-sum grid game as an example to study efficient multi-agent reinforcement learning algorithms.

1.2.2 Multi-Agent Reinforcement Learning and Game Theory

Reinforcement learning has been a hot topic in artificial intelligence research since the 1990s. With the refinement of single-agent reinforcement learning and the development of game theory and information theory, researchers have shifted their focus to multi-agent reinforcement learning. In 1992, Thuijsman proposed the idea and framework of extending single-agent reinforcement learning to multi-agent reinforcement learning [12], which subsequently sparked a wave of research on multi-agent reinforcement learning. In 1993, Tan's "Multi-agent reinforcement learning" [13] refined the theoretical knowledge of multi-agent reinforcement learning and demonstrated the feasibility of incorporating game theory strategies into reinforcement learning. In 1994, Littman first proposed incorporating the maximum-minimum strategy from game theory into multi-agent reinforcement learning to improve the Q-learning algorithm [6]. This method significantly enhanced the algorithm's convergence speed and accuracy. Based on the work of Littman, reinforcement learning researchers at the University of Michigan expanded their research to multi-agent general games and introduced Nash equilibrium strategies from game theory into multi-agent reinforcement learning to improve the Q-learning algorithm [8], also achieving good results. However, there is currently no experimental evidence or consensus on which of these two algorithms is superior. Multi-agent reinforcement learning describes objects that closely resemble real-world scenarios, making its applications extremely diverse. In 2000, Wiering improved traffic light self-regulation using multi-agent reinforcement learning [14], and many regions still use his improved traffic light system today. In the recently popular field of autonomous driving, we can also see the application of multi-agent reinforcement learning, with the resulting intelligent agents demonstrating extremely high safety [15]. In 2019, Vinyals and his team created AlphaStar [16], which competed online against human players in the game "StarCraft II," achieving a win rate of 99.8%. The AI's game play was dubbed "master-level" by the gaming community. AlphaStar's victory was a milestone because "StarCraft II" holds a iconic status in the most challenging professional eSports. AlphaStar's extremely high win rate signifies that the development and application of multi-agent reinforcement learning have reached a new height.

China has made many breakthroughs in multi-agent reinforcement learning research in the 21st century. In 2002, Luo Qing, Li Zhijun, and others from Shanghai Jiao Tong University discussed the methods and problems of using reinforcement learning in complex, dynamic environments and used multi-agent reinforcement learning to solve problems such as delayed rewards, exploration and exploitation, and incomplete information in soccer robots [17]. In 2003, Gu Guochang proposed the application of multi-robot systems in

collaborative tasks, reducing the learning space dimension by predicting the probability of each robot's actions to accelerate algorithm convergence [18]. In 2021, Hu Haoran achieved significant progress in single-agent artificial intelligence decision-making control by combining deep learning and reinforcement learning in deep reinforcement learning technology [19]. There are countless applications of multi-agent reinforcement learning in coordination, scheduling, and control [20, 21, 22, 23], among which Professor Gu Guochang, a council member of the Chinese Association for Artificial Intelligence and the Intelligent Robot Society, has made outstanding contributions [24]. Following breakthroughs in the mathematical foundations of reinforcement learning, research and applications in this field have proliferated [25]. Currently, reinforcement learning has been widely applied in various fields in China, including handicraft manufacturing, robot control, optimization and scheduling, simulation modeling, and game theory [26]. Multi-agent reinforcement learning, which integrates multi-agent systems and reinforcement learning methods, is gradually emerging as one of the research hotspots in the field of reinforcement learning and has been widely applied in various fields [27].

1.3 Modeling the Community Parking Resource Allocation Problem

Community parking resource allocation presents a quintessential urban management challenge, centered on efficiently and equitably satisfying substantial vehicle parking demands within constrained spaces. Conventional approaches typically rely on static rules, such as first-come-first-served or centralized dispatching, yet these struggle to address real-time dynamic variations and the complexity of individual driver decision-making. Consequently, this paper aims to abstract this practical problem and rigorously model it through a multi-agent reinforcement learning framework.

The practical significance of this modeling approach lies in treating each driver as an agent with independent decision-making capabilities. This better aligns with the behavioral model where drivers make choices based on personal preferences rather than uniform directives. Furthermore, it conceives the car park as a dynamically evolving environment, capable of reflecting complex states such as occupied spaces and traffic congestion in real time. his enables the model to handle uncertainty and dynamic interactions. Finally, through reinforcement learning's 'reward and punishment' mechanism, we can design an incentive and constraint system guiding drivers' individually optimal decisions to spontaneously converge towards the globally optimal outcome—namely, reducing conflicts and enhancing parking efficiency. This approach offers novel theoretical and technical pathways for developing intelligent parking systems and alleviating urban traffic congestion.

The proposed methodology aligns with this challenge as the multi-agent reinforcement learning framework inherently suits describing scenarios where multiple agents interact and engage in strategic competition within a shared environment. By integrating game theory principles into reinforcement learning, agents learn not only to maximize their own gains but also to account for the behavior of other agents, thereby achieving more robust coordination and competition.

The core elements of this model are defined as follows:

1. **Agents:** Each driver (or vehicle) within the parking system is treated as an independent agent $i \in \{1, 2, \cdots, N\}$. Each agent's objective is to autonomously make decisions that minimize the time required for parking and the walking distance.

2. **Environment:** The environment encompasses all elements beyond the agents, primarily comprising all parking spaces $j \in \{1, 2, \cdots, M\}$ within the car park and their states, the traffic road structure, and all agents within the system.

3. **State:** At time t, the system state S_t is a complete vector containing the real-time occupancy status of all parking spaces and the current location information of all agents. For example, the state may be represented as a vector with M elements, where $S_t(j) \in \{0, 1\}$ denotes whether parking space j is vacant, alongside the coordinate information of all agents.

4. **Action:** Given a state, agent i's action $a_{i,t}$ involves selecting an available parking space $j \in \{1, 2, \cdots, M\}$ and moving there. Action conflicts occur when multiple agents simultaneously select the same space.

5. **Rewards and Penalties:** The reward $R_{i,t}$ is the immediate feedback received by agent i upon executing action $a_{i,t}$. In our model, reward design is directly linked to parking efficiency and convenience. Should agent i successfully park in target space j, the reward received will be inversely proportional to the walking distance. Conversely, should an action conflict occur (i.e. multiple agents simultaneously selecting the same space), a penalty mechanism will be triggered, substantially reducing the rewards for all conflicting agents to incentivize them to select alternative parking spaces.

1.4 Main Research Content and Organizational Structure

The significance of this research lies in the creative application of multi-agent reinforcement learning and game theory frameworks to a highly relevant urban management problem: the allocation of community parking resources. This issue involves not only dynamic multi-agent interactions but fundamentally concerns the efficient allocation of finite resources and the enhancement of overall system efficiency – a critical challenge in smart city development. This paper first introduces the single-agent reinforcement learning framework (Markov Decision Processes, MDP), the reinforcement learning algorithm process and the Monte Carlo (MC) method, TD(0). It then introduces the basic theoretical knowledge of two-player zero-sum games and two game strategies, extends single-agent reinforcement learning to multi-agent reinforcement learning through the Markov game framework, embeds the two game strategies from game theory into multi-agent reinforcement learning to improve algorithm performance, and proposes improvements to existing algorithms. The main research content is as follows:

1. Introduce the fundamentals of reinforcement learning and game theory, and present two existing multi-agent reinforcement learning algorithms improved using game theory concepts. Summarize the advantages and disadvantages of both algorithms and analyze under what conditions each algorithm is more effective.

2. Address the shortcomings of the "Nash-Q" algorithm in models with multiple Nash equilibria by proposing a penalty mechanism to force agents to choose the same Nash equilibrium. It is proven that adding the penalty mechanism does not alter the optimal strategy of the original "Nash-Q" algorithm and does not compromise the convergence of the iteration process. Crucially, rigorous theoretical proofs validate that this penalty mechanism significantly enhances convergence speed and stability without modifying the original algorithm's optimal strategy.

3. At the practical level, this paper concretizes the 3×3 grid simulation game into a simplified community parking environment. By comparing the performance of the Nash-Q algorithm in this simulated environment before and after introducing the penalty mechanism, the paper validates the effectiveness of this mechanism. Experimental results clearly demonstrate that the novel penalty mechanism effectively reduces parking conflicts and system deadlocks between vehicles, markedly improves parking space utilization, and substantially shortens average vehicle waiting times. This highlights the method's considerable application potential in resolving real-world resource contention issues.

Based on the above research content, the organizational structure of this paper is as follows:

- **Section 2:** This chapter explains the fundamentals and elements of reinforcement learning, introduces the single-agent reinforcement learning model (MDP), introduces the concept of value functions and the definition of optimal strategies, and introduces the algorithm flow (decision evaluation, decision improvement) to lay the foundation for multi-agent reinforcement learning.

- **Section 3:** This chapter introduces two methods for updating the value function (MC method and TD(0) method), introduces the Q-learning algorithm under the TD(0) method, and lays the groundwork for introducing game theory concepts to improve the Q-learning algorithm in subsequent sections.

- **Section 4:** This chapter introduces the basic theoretical knowledge of game theory, then introduces two strategies (maximum-minimum strategy and Nash equilibrium strategy), and introduces the Minmax-Q algorithm (an improved version of the Q-learning algorithm using the maximum-minimum strategy) and the Nash-Q algorithm (an improved version of the Q-learning algorithm using the Nash equilibrium strategy). A summary and analysis of these two algorithms are provided to offer guidance on which algorithm to use in different scenarios.

- **Section 5:** In games with multiple Nash equilibria, an issue of the Nash-Q Algorithm arises where agents may not choose the same Nash equilibrium, thereby reducing algorithm accuracy. They may also get stuck in a local Nash equilibrium (a locally optimal solution). To address this issue, this paper proposes a penalty mechanism to force agents to choose the same Nash equilibrium and sets the target Nash equilibrium based on the value function, thereby selecting the globally optimal solution. This section proves that the optimal strategy of the original model remains unchanged under this penalty mechanism and that the convergence of the iteration is not compromised.

- **Section 6:** This chapter uses a two-player zero-sum grid game as an experimental background to compare the accuracy of the "Nash-Q" algorithm before and after the addition of a penalty mechanism. In this experiment, the number of times the agents with the penalty mechanism reached the end point as expected increased significantly, while the number of times they fell into a deadlock (in this game, if both parties do not choose the same Nash equilibrium path, there is a certain probability of a deadlock occurring, which will be explained in detail later) decreased significantly. The experiment demonstrates that the Nash-Q algorithm with a penalty mechanism is more advantageous when multiple Nash equilibria exist.

- **Section 7:** Summary of the paper, analysis of the limitations of the penalty mechanism, and proposals for improvements and future directions.

This paper is motivated by the poor performance of classic reinforcement learning algorithms in multi-agent environments. It incorporates game theory strategies into multi-agent reinforcement learning and introduces the Q-learning algorithm improved with the maximum-minimum strategy, i.e., the Minmax-Q algorithm, and the Q-learning algorithm improved with the Nash equilibrium strategy, i.e., the Nash-Q algorithm. We summarize these two algorithms, analyze, and explain under what circumstances using which game theory strategy would be more beneficial for multi-agent learning, providing a perspective for those seeking to improve reinforcement learning algorithms using game theory strategies. However, strategies in game theory are not necessarily unique; for example, multiple Nash equilibria may exist in a single game. Therefore, we designed a penalty mechanism to address this uncertainty, providing a tool for improving reinforcement learning algorithms.

2 Theoretical Frameworks in Reinforcement Learning

2.1 Reinforcement Learning

Reinforcement learning is a type of learning problem in the field of machine learning. Its biggest difference from common supervised learning and unsupervised learning is that it learns through interaction and feedback with the environment. Just like a baby, it explores the environment through crying, sucking, crawling, etc., and gradually accumulates perception of the environment, thereby learning the characteristics of the environment step by step so that its actions can quickly achieve its desires. Another example is Go, where players improve their skills by playing multiple games against different opponents, gradually gaining insights into each move and enhancing their overall Go proficiency. The AlphaGo Go program developed by DeepMind utilized reinforcement learning techniques during its training process.

Let us now formally define the problem of reinforcement learning. The basic model of reinforcement learning is the interaction between an individual and the environment. The individual/agent is the part that can take a series of actions and expects to obtain higher benefits or achieve a certain goal, such as the baby in the previous example or the player learning to play Go. The other related parts are collectively referred to as the environment, such as the baby's environment (including the surrounding room and the baby's parents) in the previous example, or the chessboard in front of you and your opponent. The entire process is discretized into different time steps. At each time step, the environment and the individual interact. The individual can take certain actions, which are applied to the environment. After receiving the individual's actions, the environment feeds back to the individual the current state of the environment and the reward generated by the previous action. It is worth noting that the division between individuals and the environment is not necessarily based on the proximity of entities. For example, in animal behavior, the rewards obtained by animals may actually come from the secretion of chemicals in their own brains. In this case, the part of the animal's brain that implements this reward mechanism should also be classified as the environment, while the individual only includes the part that receives signals and makes decisions. The goal of reinforcement learning is to maximize the total reward obtained by the individual from the environment. That is, our goal is not to obtain the maximum single-period reward after a short-term action, but to obtain more rewards in the long term. For example, a baby may steal a snack and obtain physical pleasure (i.e., a large short-term reward), but this behavior may lead to criticism from parents after a period of time, thereby reducing the total reward in the long term. In many common tasks, such as playing Go, the reward is often zero before the game ends, and only when the game ends is a reward generated based on the individual's win or loss. In other tasks, the environment may give rewards at each moment. For tasks like playing Go, which have a termination state and all rewards are settled before that state, we call them episodic tasks. There is another type of task that does not have a

termination state, meaning that in principle, they can run indefinitely. The rewards for these tasks are distributed over a series of consecutive moments, and we call this type of task a continuing task. Since our goal is to maximize the total reward, we aim to quantitatively define this total reward, which we refer to as the return. For episodic tasks, we can directly define the return as

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T = \sum_{k=t}^{T} R_{k+1}, \tag{2.1}$$

where T is the time when the turn-based task ends, and R_t is the reward at time t (with a more explicit definition in the next subsection). For continuing tasks, there is no such termination state, so this definition may diverge in continuing tasks. Therefore, we introduce another way to calculate the return, called the discounted return.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.2}$$

where the discount rate γ satisfies $0 \leqslant \gamma \leqslant 1$. This definition is also easy to understand. Compared to more distant rewards, we prefer closer rewards, so rewards that are closer have higher weights. When γ is small, we focus more on short-term rewards; when γ is large, we focus more on long-term rewards.

The result of reinforcement learning is a series of action decisions, which we call a policy. A policy specifies the action to take in each state, and we can denote this policy as π.

In summary, the task of reinforcement learning is to enable an "entity" to acquire the ability to independently complete a specific task. This "entity" is referred to as an agent, and the environment in which the agent learns and operates is referred to as the environment. To gain a deeper understanding of the interaction between the agent and the environment, we must first understand the Markov Decision Process (MDP) model.

2.2 Markov Decision Processes (MDPs)

A Markov decision process is an abstract model of the real world and is widely applied in various fields, including the reinforcement learning domain we are discussing. All the algorithms mentioned in this paper are based on the Markov decision process model. We must understand the problem before we can solve it. The MDPs discussed in this text are all finite Markov decision processes, both the state space and action space. Below is the definition of a Markov decision process:

A Markov decision process is an abstract model of an intelligent agent interacting with its environment, consisting of a quintuple: $\langle S, A, P, r, \gamma \rangle$.

- S represents the state space. In a finite Markov decision process (finite MDP) [28], the number of elements in this set is finite, and S_t denotes the state at time t;
- $A(s)$ represents the action space available when the system is in state s. In a finite Markov decision process, the number of elements in $A(s)$ is finite, and A_t represents the action at time t;
- $P(s' \mid s, a)$ represents the probability of transitioning to state s' when taking action $a \in A(s)$ in state s. $P : S \times A \to \Delta(S)$, where $\Delta(S)$ is the set of all probability distributions in the state space S;

- $r(s, a, s')$ represents the expected reward obtained when taking action $a \in A(s)$ and transitioning to state s' from state s:

$$r(s, a, s') = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s'\right]. \tag{2.3}$$

  Similarly, we can define the expected reward obtained by taking action $a \in A(s)$ in state s:

$$r(s, a) = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right] = \sum_{s' \in \mathcal{S}} P(s' \mid s, a)\, r(s, a, s'), \tag{2.4}$$

  where R_t represents the reward at time t.

- $\gamma \in [0, 1]$ represents the discount rate, which indicates the objective of the MDP:

$$\max_{\pi} \mathbb{E}_{\pi}\left[G_t\right], \tag{2.5}$$

  where G_t is the Equation 2.2, and $\pi : S \to A(s)$ denotes the set of policies, whose elements $\pi(a \mid s)$ represent the probabilities of taking possible actions a for a given state s in the process, expressed as:

$$\pi(a \mid s) = P\left(A_t = a \mid S_t = s\right). \tag{2.6}$$

  A strategy π is classified into pure strategies and mixed strategies. When, under any state, the action determined by the strategy π is unique, such a strategy is called a pure strategy. Otherwise, it is called a mixed strategy, which is defined below.

Definition 2.1 (Mixed Strategy) *Given a set of pure strategies (action set) $A(s)$, $s \in S$, a mixed strategy π is a probability distribution over $A(s)$. That is, $\pi : A(s) \to [0, 1]$ is a mapping that assigns a probability $\pi(a \mid s)$ to each pure strategy (action) $a \in A(s)$ such that*

$$\sum_{a \in A(s)} \pi(a \mid s) = 1, \quad \forall s \in S. \tag{2.7}$$

The pure strategy $a \in A(s)$ can be regarded as the action a with probability 1 in state s and probability 0 for all other actions.

Regarding strategies, note the following:

- The strategy π fully defines an individual's behavior pattern, i.e., it includes all behaviors and probabilities of the individual in each state.
- The strategy π is static and independent of time.
- The strategy π is only dependent on the current state and is independent of historical information.

Markov Decision Processes (MDPs) satisfy the Markov property by definition [28], meaning that the next state of an intelligent agent is determined solely by the current state s and the action taken (policy π). Take chess as an example. When we are in a certain position (state s) and take a move (action a) according to the strategy π, we cannot determine the opponent's choice, which leads to the next state s' , i.e., the state transition probability $P(s' \mid s, a)$. However, the opponent's choice only depends on s and

a, and does not consider earlier states or actions. That is, s' is randomly generated based on s and a. The formula is defined as follows:

$$P\left(S_{t+1} \mid S_t\right) = P\left(S_{t+1} \mid S_1, S_2, \cdots, S_t\right). \tag{2.8}$$

The future state (S_{t+1}) is independent of past states and is only related to the current state (S_t) and actions (a_t).

The interaction flowchart between the intelligent agent and the environment in MDP is as follows:

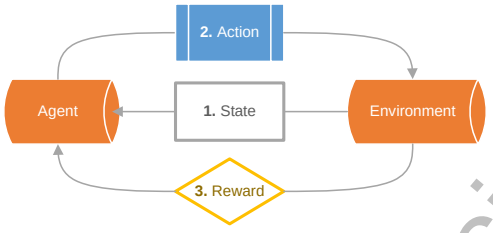

Figure 2.1: MDP Interaction Diagram

The interaction between an individual and the environment can be represented using Figure 2.1. At time $t = 1, 2, 3, \cdots$, all individuals receive a state signal $S_t \in S$ from the environment. At each step, an individual selects an action $A_t \in A\left(S_t\right)$ from the set of actions $A\left(S_t\right)$ allowed by the current state. After receiving this action signal, the environment provides feedback to the individual in the next time step, including the corresponding state signal $S_{t+1} \in S$ and immediate reward $r\left(S_t, A(S_t)\right)$. This process is repeated until the termination state is reached or the process continues indefinitely.

In a random MDP, each moment t experiences two uncertainties: the first is the uncertainty of the action itself, determined by the strategy π, and the second is the uncertainty of state transition (i.e., environmental uncertainty). Let's illustrate this with an example. In the Monopoly game, players can choose to roll the dice once or twice (the sum of the dice determines the number of steps taken), but we want to ultimately move 5 steps. In this case, there are two uncertainties: the first is whether to roll the dice once or twice, which is the player's choice; the second is the number of points rolled each time, which is the randomness of the environment and beyond the player's control. In multi-player zero-sum games, multiple agents interact with each other, significantly increasing the randomness and complexity of the environment, which will be discussed in Section 4. Under these two types of uncertainty, the MDP flowchart is presented as follows (Figure 2.2):

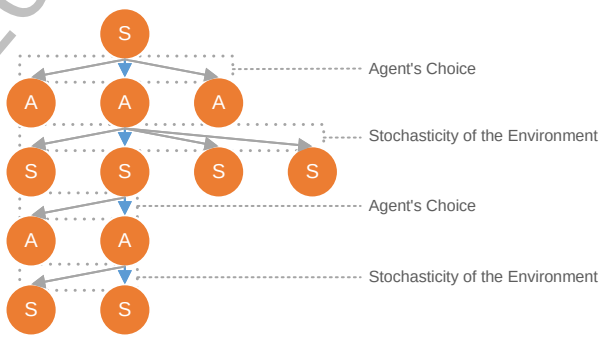

Figure 2.2: MDP Flowchart

The goal of reinforcement learning is to learn a strategy (π) that maximizes the reward within the MDP model.

2.3 Value Function and Optimal Policy

2.3.1 Significance of the Value Function

In the previous section, it is mentioned that reinforcement learning learns a mapping from environment states to actions (i.e., a strategy). However, reinforcement learning also has the characteristic of delayed rewards. In chess, if you lose the game at step n , only the state s_n and action a_n receive the current reward $r(s_n, a_n) = -1$, while all previous states immediately receive a reward of 0. Therefore, for any previous state s and action a , the current reward function $r(s, a)$ cannot indicate the quality of the strategy. Hence, we need to define a value function to indicate the long-term impact of the strategy π in the current state.

After understanding the MDP dynamic process, we can see from Figure 2.2 that actions a and state s are nodes in the graph.

Definition 2.2 (Value Function) *The value of a state is called the state value function (state value function) $V_\pi(s)$, which represents the expected reward obtained by starting from state s and following policy π:*

$$V_\pi(s) = \mathbb{E}_\pi[G_0 \mid S_0 = s]. \tag{2.9}$$

We refer to the reward of an action as the action value function ($Q_\pi(s,a)$), which represents the expected reward obtained by executing a specific action (a) in the current state (s) and then executing subsequent actions according to the policy (π):

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]. \tag{2.10}$$

We combine the action value function ($Q_\pi(s,a)$) with the state value function ($V_\pi(s)$) to describe the MDP. In the diagram, circles represent states, squares represent actions, and wedges represent the reward values after the action is completed.

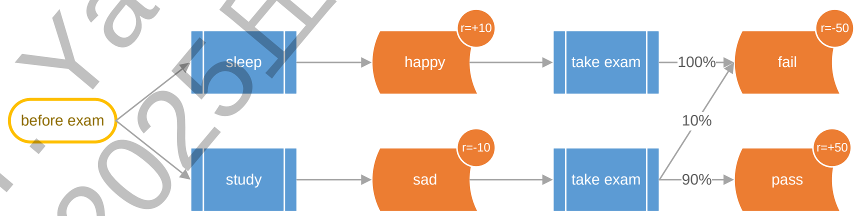

Figure 2.3: $Q_\pi(s,a)$ and $V_\pi(s)$ Diagram

Observe Figure 2.3. Assuming a discount rate of $\gamma = 1$, failing and passing are the terminal states of the MDP, where failing results in a reward of -50 and passing results in a reward of $+50$. The expected value of the strategy of sleep before the exam, $Q_\pi(\text{before exam}, \text{sleep})$, is $10 + (-50) = -40$, and the expected value of the strategy of reviewing before the exam, $Q_\pi(\text{before exam}, \text{study})$, is $-10+50\times 90\%+(-50)\times 10\% = 30$. Therefore, when making decisions, we must take a long-term perspective and choose the decision that maximizes the total future rewards. This is precisely the principle of maximizing expected rewards in MDPs.

Currently, there is an issue regarding the value of the initial "before exam" state $V_\pi(s)$. If we choose the action of reviewing, then the value of the "before exam" state $V_\pi(s)$ is 30; if we choose the action of sleep, then the value of the "before exam" state $V_\pi(s)$ is -40. Therefore, the value of the state $V_\pi(s)$ is related to our strategy π. Similarly, if we choose "study" in the "before exam" state and then participate in the "take exam," then $Q_\pi(s,a) = 30$; if we choose "study" in the "before exam" state and then do not participate in the "exam," then $Q_\pi(s,a) = -10 + (-50) = -60$. Therefore, the $Q_\pi(s,a)$ of a state-action pair is also related to our subsequent strategy π.

2.3.2 Bellman Equations for Value Functions and Optimal Policy

From the above description, action value functions and state value functions have similar meanings: firstly, they are both nodes in an MDP; secondly, the value evaluation method is also consistent: starting from the current node, following a certain strategy until reaching the final node, and taking the discounted expected sum of all rewards. Therefore, $V_\pi(s)$ and $Q_\pi(s,a)$ can be derived from each other.

First, we will use the $Q_\pi(s,a)$ to calculate the $V_\pi(s)$ based on the following diagram (the diagram below is a portion of Figure 2.2)

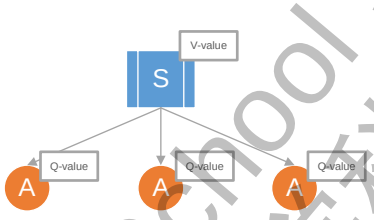

Figure 2.4: Calculating $V_\pi(s)$ Using $Q_\pi(s,a)$

Based on Figure 2.4, assuming we have already calculated $Q_\pi(s,a)$, $a \in A(s)$, then according to the strategy π, the expected value of $Q_\pi(s,a)$ is $V_\pi(s)$. The formula is as follows:

$$V_\pi(s) = \sum_{a \in A} \pi(a \mid s)\, Q_\pi(s,a). \tag{2.11}$$

Similarly, we can use $V_\pi(s)$ to calculate $Q_\pi(s,a)$:

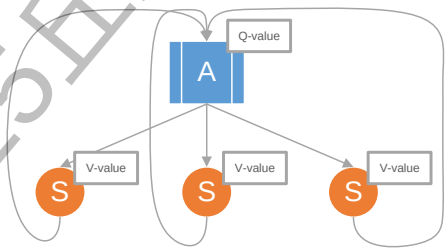

Figure 2.5: Calculating $Q_\pi(s_t,a_t)$ Using $V_\pi(s_{t+1})$

Based on Figure 2.5, assuming we have already calculated the next moment's $V_\pi(s_{t+1})$, $s_{t+1} \in S$, we can calculate the expected value of $V_\pi(s_{t+1})$ using the strategy π, which is $Q_\pi(s_t,a_t)$. Unlike the calculation of $V_\pi(s)$ using $Q_\pi(s,a)$, this process involves a time jump of t, so we need to add the discount rate γ and the stage reward r. The formula is as follows:

$$Q_\pi(s_t,a_t) = r(s_t,a_t,s_{t+1}) + \sum_{s_{t+1}} P(s_{t+1} \mid s_t,a_t)\, \gamma V_\pi(s_{t+1}). \tag{2.12}$$

Substituting formula Equation 2.11 into Equation 2.12 yields the updated formula for $V_\pi(s_t)$ and the updated formula for $Q_\pi(s_t, a)$, which are the Bellman equations:

Bellman equation for state value function is as follows:

$$V_\pi(s_t) = \sum_{a_t \in A(s_t)} \pi(a_t \mid s_t)\, r(s_t, a_t) + \sum_{a_t \in A(s_t)} \pi(a_t \mid s_t) \sum_{s_{t+1} \in S} P(S_{t+1} \mid s_t, a_t)\, \gamma V_\pi(s_{t+1}). \tag{2.13}$$

Bellman equation for the action value function is as follows:

$$Q_\pi(s_t, a_t) = r(s_t, a_t) + \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t)\, \gamma \sum_{a_{t+1} \in A(s_{t+1})} \pi(a_{t+1} \mid s_{t+1})\, Q_\pi(s_{t+1}, a_{t+1}). \tag{2.14}$$

The Bellman equation establishes the relationship between the current state value function and the next state value function. As introduced in MDP, the objective of the MDP is to maximize the Equation 2.5, i.e., to find a strategy that maximizes the reward. This is mapped to the Bellman equation as finding a strategy that maximizes the value function.

For a finite Markov decision process, two strategies are ranked based on their value functions:

$$\pi' \geqslant \pi \Longleftrightarrow V_{\pi'}(s) \geqslant V_\pi(s), \quad \forall s \in S. \tag{2.15}$$

The strategy with the maximum state value function, π^*, is called the optimal strategy:

Definition 2.3 (Optimal Strategy) *The optimal strategy has the optimal state value function, as shown below:*

$$V_{\pi^*}(s) = \max_\pi V_\pi(s), \quad \forall s \in S. \tag{2.16}$$

Similarly, the optimal strategy also has the same optimal action value function, as shown below:

$$Q_{\pi^*}(s, a) = \max_\pi Q_\pi(s, a), \quad \forall s \in S,\ \forall a \in A(s). \tag{2.17}$$

Having understood the optimal policy, we observe Equation 2.14, which holds for any policy and, of course, for the optimal policy. Therefore, we have:

$$Q^*(s, a) = r(s, a) + \sum_{s'} P(s' \mid s, a)\, \gamma V^*(s'), \tag{2.18}$$

where $V^*(s)$ and $Q^*(s, a)$ are abbreviations for $V_\pi^*(s)$ and $Q_{\pi^*}(s, a)$, respectively.

On one hand, $V^*(s)$ follows the state value function of the optimal strategy π^* and must also satisfy the Bellman equation for state value functions expressed by Equation 2.13, i.e.:

$$V_{\pi^*}(s_t) = \sum_{a_t \in A(s_t)} \pi(a_t \mid s_t)\, r(s_t, a_t) + \sum_{a_t \in A(s_t)} \pi(a_t \mid s_t) \sum_{s_{t+1} \in S} P(S_{t+1} \mid s_t, a_t)\, \gamma V_{\pi^*}(s_{t+1}). \tag{2.19}$$

On the other hand, since it is the optimal state value function, by definition it must equal the largest action value function obtained by taking the optimal strategy in state s. Thus, we obtain:

$$V^*(s) = \max_{a \in A(s)} Q^*(s, a)$$

$$= \max_{a \in A(s)} \left[r(s,a) + \sum_{s' \in S} P\left(s' \mid s,a\right) \gamma V_{\pi^*}\left(s'\right) \right], \tag{2.20}$$

which is known as the Bellman optimality equation for the state value function.

Similarly, we can also derive the Bellman optimality equation for the action value function:

$$Q^*\left(s,a\right) = \sum_{s' \in S} r\left(s,a\right) + P(s' \mid s,a)\gamma \max_{a' \in A(s')} Q^*\left(s',a'\right). \tag{2.21}$$

For any MDP, the following holds:

1. An optimal strategy is better than or at least equal to any other strategy;
2. All optimal policies have the same state value function;
3. All optimal strategies have the same action value function.

In an MDP, the optimal strategy maximizes the expected total discounted reward. In "Markov Decision Processes," Puterman has proven that for a finite-state Markov decision process, there must exist at least one optimal strategy π^*, and this strategy is both stationary and deterministic. It is called stationary because π^* does not change over time; it is called deterministic because when the agent is in state s, it always chooses the same action, i.e., pure strategy (this is only applicable in the MDP model; in the multi-agent game MDP model, pure strategy may not always yield optimal results, as discussed in). In this paper, we will focus on stable strategies. Non-stable strategies, which allow adjusting actions based on the game history, are more complex [29] and have been studied relatively less in two-player zero-sum games, which is not conducive to incorporating game theory strategies into multi-agent reinforcement learning in the future.

2.4 Game-Theoretic Analysis in Parking Scenarios

Within multi-agent reinforcement learning, the decisions of agents (drivers) are not isolated but interdependent and mutually influential. This complex dynamic interaction renders traditional single-agent frameworks inadequate. Consequently, introducing game-theoretic analytical tools offers a novel perspective for understanding and resolving community parking resource allocation problems. We conceptualize the parking selection process as a dynamic game, centered on the strategic choices of multiple rational drivers competing for a finite number of parking spaces.

2.4.1 Safe Strategy and Conflict Equilibrium

In game theory, a "safety strategy" represents an extremely conservative approach aimed at minimizing worst-case losses. Within the community parking context, this corresponds to a "greedy" and myopic behavioral pattern: each driver adheres to a simple decision rule–selecting only the closest, seemingly vacant parking space.

While seemingly optimal for individuals, this behavior often leads to systemic failure in multi-agent environments, forming an unstable "conflict equilibrium". When multiple vehicles enter a car park simultaneously, all drivers may fixate on the same or a few geographically optimal spaces. This results in the following consequences:

1. **Multi-agent conflict:** Multiple agents simultaneously execute the same "parking" action, competing for the same physical space.

2. **Declining returns:** Due to conflicts, all parties must re-select, increasing detour time and fuel consumption, resulting in a significant reduction in individual benefits.

3. **System inefficiency:** A large number of vehicles circling and waiting in confined areas causes localized congestion or even deadlocks, while other vacant spaces further away remain unused, resulting in extremely low overall system utilization.

The equilibrium in this state is fragile and inefficient, as it cannot self-correct without external intervention–such as introducing penalties to break the suboptimal cycle caused by collective "greed".

2.4.2 Mixed Strategy Nash Equilibrium and Efficiency Enhancement

Unlike the safety strategy, the "mixed-strategy Nash equilibrium" represents a more intelligent decision-making paradigm. Under this approach, each agent does not select a single optimal action but instead chooses from all possible actions according to a probability distribution. This achieves a stable state where no individual can unilaterally alter their behavior to gain an advantage.

In the community parking game, the mixed-strategy Nash equilibrium corresponds to a dispersed, collaborative parking behavior. Drivers no longer blindly rush to the nearest space but, with a certain probability, allocate part of their choice to slightly more distant spaces with lower conflict risks. This "dispersed parking" behavior pattern can bring about significant improvements in system efficiency:

1. **Reduced conflict rate:** As agents select different parking spaces, the probability of competing for the same space is significantly lowered, minimizing unnecessary detours and waiting times.

2. **Enhancing resource utilization:** Overall parking space utilization is balanced across the car park, preventing excessive congestion in popular areas and idle resources in less frequented zones.

3. **Achieving global optimality:** While each driver's decision incorporates randomness, this randomness is grounded in rational game-theoretic calculations. Ultimately, this maximizes the aggregate benefit for all agents, realizing the optimal allocation of social welfare.

Consequently, the core research objective of this paper is to devise a mechanism guiding agents from inefficient conflict equilibria towards efficient mixed-strategy Nash equilibria. This demonstrates, both theoretically and practically, that well-designed mechanisms can effectively mitigate real-world resource conflicts.

2.5 Strategy Evaluation and Improvement

For any strategy π, how do we calculate its value function $V_\pi(s)$? This problem is referred to policy evaluation.

In reinforcement learning, iterative methods are generally used to update the value function. First, all state-action pairs ($V_\pi(s)$) are assigned initial values (typically 0), and then the value function is updated using the following equation ($k+1$ iteration):

$$V_\pi^{k+1}(s_t) = \sum_{a \in A(s_t)} \pi(a \mid s_t)\, r(s_t, a) + \sum_{a \in A(s_t)} \pi(a \mid s_t) \sum_{s_{t+1} \in S} P(s_{t+1} \mid s_t, a)\, \gamma V_\pi^k(s_{t+1}). \tag{2.22}$$

The value function is stored in an array, with each new value overwriting the previous one.

The policy estimation algorithm is shown in Algorithm 1:

Algorithm 1 Policy Evaluation Algorithm

Input: Strategy π
Initialization: $V(s) = 0, \ \forall s \in S$

Iterative Experimentation
$\Delta \leftarrow 0$
repeat
  for each $s \in S$ **do**
    temp $\leftarrow V(s)$
    $V(s) \leftarrow \sum_{a \in A(s)} \pi(a|s_t) r(s,a) + \sum_a \pi(a|s) \sum_{s'} P(s'|s,a) \gamma V(s')$
    $\Delta \leftarrow \max(\Delta, |\text{temp} - V(s)|)$
  end for
until $\Delta < \theta$ ▷ θ is a very small positive number

Output: $V(s) \approx V_\pi(s)$

We have understood the principles and process of policy evaluation, and the purpose of policy evaluation is to find better policies. This process is called policy improvement.

Suppose we have a policy π and have determined its value function $V_\pi(s)$ through policy evaluation. For a state s, there is an action $a = \pi(s)$. Then, if we do not adopt action a according to policy π in state s, but instead adopt another policy π', would that be better?

Theorem 2.1 (Policy Improvement Theorem) *If π and π' are two deterministic policies, and $\forall s \in S$, $Q_\pi(s, \pi'(s)) \geqslant V_\pi(s)$ holds, then policy π' is necessarily better than policy π, or at least as good, i.e., $\forall s \in S$, $V_{\pi'}(s) \geqslant V_\pi(s)$ holds.*

The meaning of the theorem is that as long as the actions selected according to the new strategy have an action value function that is no less than the state value function under the old strategy, then such a strategy is a better strategy.

Proof 2.1 (Policy Improvement Theorem)

$$\begin{aligned}
V_\pi(s) &\leqslant Q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[G_t \mid S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}[R_{t+1} + \gamma V_\pi(s_{t+1}) \mid S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma V_\pi(s_{t+1}) \mid S_t = s] \\
&\leqslant \mathbb{E}_{\pi'}[R_{t+1} + \gamma Q_\pi(s_{t+1}, \pi'(s_{t+1})) \mid S_t = s] \\
&\cdots \\
&\leqslant \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s\right] \\
&= V_{\pi'}(s). \qquad (2.23)
\end{aligned}$$

Based on the above theorem, how can we construct a better strategy? In other words, how can we ensure that the action value function $Q_\pi(s, \pi'(s))$ of the new strategy $a' = \pi'(s)$ is greater than the state value function $V_\pi(s)$? The value function is the expected value of the action value function under different actions (for deterministic strategies, the two are equal because there is only one action). Therefore, we simply need to choose the action that maximizes $Q(s, a)$. Because

$$\max_a Q(s, a) \geqslant \mathbb{E}_\pi [Q(s, a)] = V_\pi(s), \tag{2.24}$$

the new policy is defined as

$$\begin{aligned} \pi'(s) &= \arg\max_{a \in A(s)} Q_\pi(s, a) \\ &= \arg\max_{a \in A(s)} \left[r(s, a) + \sum_{s'} P(s' \mid s, a) \gamma V_\pi(s') \right]. \end{aligned} \tag{2.25}$$

This deterministic strategy is called the greedy strategy because it always aims to maximize the value function.

Finally, you may wonder whether the greedy strategy converges to the optimal strategy.

Here, we assume that the policy improvement process has converged, i.e., for all s, $V_{\pi'}(s)$ equals $V_\pi(s)$. Then, according to Equation 2.13 and Equation 2.25, we can see that for all $s \in S$, the following holds:

$$V_{\pi'}(s) = \max_a \left[r(s, a) + \sum_{s'} P(s' \mid s, a) \gamma V_\pi(s') \right]. \tag{2.26}$$

This is precisely the optimal Bellman equation for the state value function in Equation 2.25. Therefore, π and π' are both optimal strategies.

The algorithmic structure of reinforcement learning consists of two parts: policy evaluation and policy improvement. The specific algorithm flow is shown in Algorithm 2:

Algorithm 2 Reinforcement Learning Algorithm Flowchart

Initialization:
for all $s \in S$ **do**
    Assign arbitrary values to $V(s) \in \mathbb{R}$, $\pi(s) \in A(s)$
end for

Policy Evaluation:
Iterate Experiment
$\Delta \leftarrow 0$
repeat
    for all $s \in S$ **do**
        temp $\leftarrow V(s)$
        $V(s) \leftarrow r(s,a) + \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)\,\gamma V(s')$
        $\Delta \leftarrow \max(\Delta, |\text{temp} - V(s)|)$
    end for
until $\Delta < \theta$ ▷ θ is a very small positive number

Policy Improvement:
Policy_stable $\leftarrow$ true
for all $s \in S$ **do**
    temp $\leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg\max_a [r(s,a) + \sum_{s'} P(s'|s,a)\,\gamma V_\pi(s')]$
    if temp $\neq \pi(s)$ **then**
        Policy_stable $\leftarrow$ false
    end if
end for

if Policy_stable = true **then**
    Output: $V(s)$ and $\pi(s)$
else
    Return to "Policy Evaluation"
end if

The classic single-agent reinforcement learning algorithm has spawned various algorithms (MC, TD(0), TD(lambda), etc.) due to differences in policy evaluation, but in the policy improvement phase, all use a greedy strategy.

2.6 Summary of This Chapter

This chapter introduced the fundamental concepts of reinforcement learning, covering key definitions such as Markov Decision Processes (MDPs), value functions, optimal policies, Bellman equations, policy evaluation, and policy improvement. The MDP framework underpins single-agent reinforcement learning, with its algorithmic workflow divided into policy evaluation and policy improvement. The distinction in policy evaluation methods leads to the two fundamental reinforcement learning algorithms introduced in subsequent sections.

However, we recognize that when problems extend from single agents to multi agent interactions, the traditional MDP framework proves inadequate as it fails to capture the

mutual influences between agents. Therefore, in the latter part of Section 2, we shift our focus from abstract theoretical models to the concrete real-world application of community parking resource allocation. We introduce game theory concepts to deeply analyze the decision-making behavior of agents, specifically drivers, within the parking scenario.

Through this integration of theory and application, we discover that drivers' greedy decisions–specifically, safe strategies–may lead to inefficient conflict equilibria, causing congestion and resource wastage. Conversely, optimal decisions, embodied in mixed strategies, can guide collective behavior towards efficient Nash equilibria, thereby enhancing overall system efficiency. This approach of incorporating game-theoretic strategies into multi-agent reinforcement learning constitutes the core theoretical and practical contribution of this paper.

This chapter establishes a robust theoretical foundation for subsequent algorithmic analysis and refinement. The ultimate objective of this work is to introduce Nash equilibrium strategies from game theory within the multi-agent reinforcement learning framework, proposing targeted enhancements to effectively address real-world community parking challenges while simultaneously improving algorithmic convergence accuracy and speed.

3 Algorithms of Reinforcement Learning

This chapter will elucidate the principles of two classical reinforcement learning algorithms through mathematical modeling, detailing how various algorithms implement value function updates. As noted earlier, the distinction between the two algorithms lies in their differing approaches to value update during policy evaluation, whilst both employ greedy strategies for policy improvement. Section 4 summarizes how game theory-based strategy improvements can be applied to the TD(0) algorithm.

3.1 Monte Carlo Update Method (MC Method)

3.1.1 Monte Carlo Algorithm (MC Algorithm)

The Monte Carlo algorithm (abbreviated as MC algorithm) is introduced first.

The MC algorithmic steps for solving the $V_\pi(s)$ value function comprise five stages, as shown in Figure 3.1:

1. Place the agent in any state of the environment;
2. From this state, select an action according to the policy π and transition to a new state.
3. Repeat step 2 until the final state is reached;
4. Trace back from the final state: compute the $G_t(s)$ value (Equation 2.2) for each state s.
5. Repeat multiple times, then average the $G_t(s)$ value for each state s, this yields the required $V_\pi(s_t)$ value.

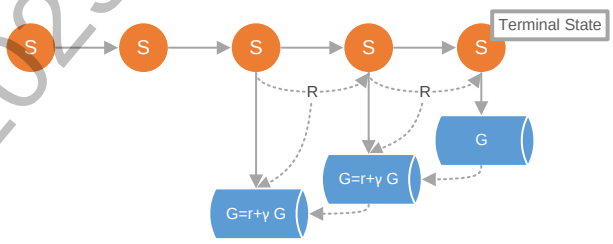

Figure 3.1: Conceptual Diagram of the MC Algorithm

Agent operations comprise three phases:

1. The agent moves forward according to the strategy π until reaching the end. During this phase, the agent performs no calculations; it merely records each state transition and the corresponding reward r.

2. The agent moves backwards from the endpoint, calculating $G_t(s)$ during the process. The $G_t(s)$ value equals the $G_t(s')$ value of the subsequent state s' multiplied by the discount rate γ, plus the reward r of the current state.

3. The agent repeats the experiment multiple times according to the policy, taking the average of the $G_t(s)$ values corresponding to each state. This average becomes the $V_\pi(s)$ value for state s.

For ease of understanding, assume a discount rate of 1. The G_t value represents the total reward sum from a given state to the final state during a single game (Equation 2.2). $V_\pi(s)$, therefore, denotes the average of $G_t(s)$ values across multiple games.

3.1.2 The Core Principle of the MC Algorithm

The core concept of the Monte Carlo algorithm, as outlined above, involves calculating the $G_t(s)$ value for each state in a single game. By averaging the $G_t(s)$ values across multiple games, this average represents the $V_\pi(s)$ of the s values for all state s.

However, this approach necessitates creating a new space for each state to record changes in the $G_t(s)$ value, significantly increasing both the algorithm's complexity and computational load. Therefore, the aim is to bypass the $G_t(s)$ value and directly compute the $V_\pi(s)$ for the state s_t.

At this point, we can transcend the problem by applying mathematical reasoning to transform it. Since the state $V_\pi(s)$ represents the average of G_t values, the problem can be reframed as determining how to compute the mean.

Suppose one rope has a length of x_1 and another has a length of x_2. Their average length should be

$$X_1 = \frac{x_1 + x_2}{2} = \frac{1}{2}(x_2 - x_1) + x_1. \tag{3.1}$$

If a third rope is added, the average becomes

$$X_2 = \frac{1}{3}(x_3 - X_1) + X_1. \tag{3.2}$$

By extension, when a third rope of length is added, the average becomes

$$X_n = \frac{1}{n}(x_n - X_{n-1}) + X_{n-1}. \tag{3.3}$$

This method constitutes incremental updating. One need only retain the previous average X_{n-1}, the newly added length x_n, and the count to calculate the latest average X_n.

In reinforcement learning, $0 \leqslant \alpha_t \leqslant 1$ is employed, replacing $\frac{1}{n}$. α_t is termed the learning rate. Even if $\alpha_t \neq \frac{1}{n}$, provided the number of steps is sufficiently large, X_n will still converge to the expected value.

Applying the method of averaging incremental updates to the MC Algorithm yields

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha_t \left[G_t - V_\pi(s_t)\right], \tag{3.4}$$

where α_t is the learning rate, and $G_t = G_{s_t}$. When the game is played a sufficient number of times, $V_\pi(s_t)$ will gradually approach the expectation of G_t. To ensure that $V_\pi(s_t)$ eventually converges to a point and the amplitude of fluctuation due to new G_t values becomes smaller, we require that α_t be sufficiently small. However, we also hope that α_t is not too small at the beginning, so that convergence can be achieved more quickly.

3.2 Temporal Difference Method (TD(0)-based Update Approach)

3.2.1 Conceptual Framework of the Temporal Difference Method

Although the MC algorithm can estimate $V_\pi(s)$ through sampling when environmental information P and r are unknown, this approach has limitations. As discussed previously, the MC algorithm must run until the final state is reached before backtracking to calculate $V_\pi(s)$ for each state s. Furthermore, multiple experiments are required for calibration to ensure the stability and accuracy of $V_\pi(s)$. When the environment's state space is large, the time required for the MC algorithm to reach the final state and compute $V_\pi(s)$ becomes substantial, rendering the MC algorithm inefficient.

If we liken the algorithm's operation to mountain climbing, where the reward represents the distance from the current node to the summit, the MC algorithm must first ascend to the peak (final state) before descending to update each intermediate node with its distance from the summit. When the mountain is vast, the ascent takes an excessive amount of time, resulting in a slow update rate for each node. A novel update approach may be considered: treating each node as a signpost. When traversing from one signpost to the next, one observes the distance to the summit displayed on the subsequent signpost (future reward). The value of the preceding signpost is then updated by adding the distance between the two signposts (the reward for the current action) to the sum of the distance to the summit displayed on the next signpost.

This method of updating the previous state using the next state and the intervening reward ($V_\pi(s)$) is termed the temporal difference algorithm (TD(0)). The MC algorithm requires traversing the entire path, whereas the TD(0) algorithm treats the next state as the final state after each action, then updates the previous state's value through backpropagation. Compare the update formulas for the MC and TD(0) algorithms.

MC Update formula:

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha_t \left[G_t - V_\pi(s_t)\right]. \tag{3.5}$$

TD(0) Update formula:

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha_t \left[R_{t+1} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)\right]. \tag{3.6}$$

Having undergone a transition at time t, the discount rate γ and reward R_{t+1} must be applied.

Comparing the two formulas reveals that the MC formula calculates G_t by tracing back from the final state after the agent completes the entire path to update $V_\pi(s_t)$. The TD(0) formula, however, uses the next state's $V_\pi(s_{t+1})$ combined with the state transition reward R_{t+1} as the update target. This method does not require completing the entire path; the agent can update the previous state's $V_\pi(s_t)$ with each action. Consequently, this approach enables faster updates and significantly reduces computational space requirements.

The $V_\pi(s_t)$ update diagram for the TD(0) method is illustrated below in Figure 3.2:

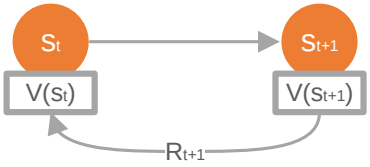

Figure 3.2: TD(0) $V_t(s_t)$ Update Diagram

3.2.2 Q-learning Algorithm

The previous section outlined how to update $V_\pi(s)$ values using the TD(0) method. Since both $V_\pi(s)$ and $Q_\pi(s,a)$ are nodes within the "Markov tree," $Q_\pi(s,a)$ can similarly be updated using the TD(0) approach (Figure 3.3).

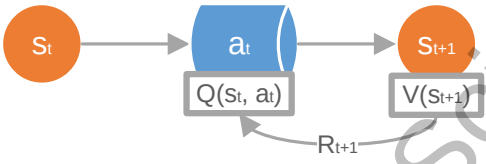

Figure 3.3: TD(0) $Q_\pi(s,a)$ Update Diagram

The significance of $V_\pi(s_{t+1})$ is that it represents the expected reward obtainable by consistently acting according to policy π from state s_{t+1}. The meaning of $Q_\pi(s_1,a_t)$ is the expected reward obtainable by consistently acting according to policy π from state s_t, selecting action a_t. Thus, $V_\pi(s_{t+1})$ can be viewed as a signpost along the mountain path, indicating the remaining distance to the summit. By adding the segment R_t, one determines the distance $Q_\pi(s_t,a_t)$ from the path to the summit.

However, a complication arises: the update method shown above requires estimating not only $Q_\pi(s_t,a_t)$ but also $V_\pi(s_{t+1})$, entailing excessive storage and computational overhead. Therefore, we may substitute $V_\pi(s_{t+1})$, the next-time-step state, with $Q_\pi(s_{t+1},a_{t+1})$, the next-time-step action. This substitution is illustrated below (Figure 3.4):

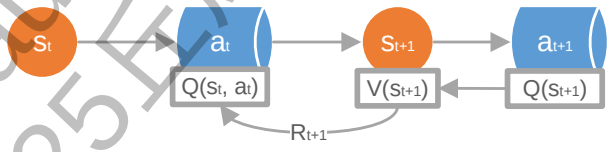

Figure 3.4: TD(0) $V_t(s_t)$ Update Diagram 2

The above analysis stems from the absence of at transition between state s_{t+1} and action a_{t+1}. Crucially, the Markov model is a tree rather than a chain (refer to Figure 2.3). Under state s_{t+1}, multiple actions a_{t+1} may exist, each yielding distinct $Q_\pi(s_{t+1},a_{t+1})$ values. Consequently, $Q_\pi(s_{t+1},a_{t+1})$ is not equivalent to $V_\pi(s_{t+1})$.

Although not entirely equivalent, according to Equation 2.11, one $Q_\pi(s_{t+1},a_{t+1})$ can substitute for $V_\pi(s_{t+1})$, as there must exist a $Q_\pi(s_{t+1},a_{t+1})$ that partially represents $V_\pi(s_{t+1})$. Numerous approaches exist for selecting which $Q_\pi(s_{t+1},a_{t+1})$ to substitute for $V_\pi(s_{t+1})$, with two most prominent ideas:

Adopting the same policy π, and selecting the action a_{t+1} generated under that policy π to replace $V_\pi(s_{t+1})$. This constitutes the SARSA algorithm.

Select the action with the highest $Q(s_t, a_t)$, and choose the maximum $Q(s_{t+1}, a_{t+1})$ to substitute $V(s_{t+1})$. This constitutes the Q-learning algorithm.

The formula for updating the Q-value in the SARSA algorithm is:

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha_t \left[R_t + \gamma Q_\pi(s_{t+1}, \pi(a_{t+1})) - Q_\pi(s_t, a_t)\right], \tag{3.7}$$

where $Q_\pi(s_t, a_t)$ corresponds to π in $Q_\pi(s_{t+1}, \pi(a_{t+1}))$.

Why is it reasonable for SARSA to use the action value $Q_\pi(s_{t+1}, a_{t+1})$ generated under the same policy, substituting the state value $V_\pi(s_{t+1})$? The answer is simple: it works. In truth, reinforcement learning, though involving considerable mathematics, is not a rigorous science; it is more akin to a technique. As long as it proves effective in practice, that suffices.

Q-learning algorithm's $Q_\pi(s_t, a_t)$ update formula is:

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha_t \left[R_t + \gamma \max_{a_{t+1} \in A(s_{t+1})} Q_\pi(s_{t+1}, a_{t+1}) - Q_\pi(s_t, a_t)\right]. \tag{3.8}$$

This is the renowned Q-learning algorithm in reinforcement learning. It substitutes the Q-learning algorithm's approach by replacing the maximum action value function ($\max_{a_{t+1} \in A(s_{t+1})} Q_\pi(s_{t+1}, a_{t+1})$) with the maximum expected reward function ($V_\pi(s_{t+1})$).

The underlying principle for substitution in Q-learning is fundamentally straightforward. Since the objective is to identify the action yielding the highest reward, the $Q_\pi(s_t, a_t)$ represents the expected value of future rewards. Consequently, only the action with the maximum $Q_\pi(s_{t+1}, a_{t+1})$ is selected for updating.

Here is the summary of this section:

1. Both the Q-learning and SARSA algorithms are based on the concept of TD(0). However, in the preceding discussion, TD(0) was used to estimate state values, whereas the Q-learning and SARSA algorithms estimate action values.

2. The core principle of both the Q-learning and SARSA algorithms is to estimate $Q_\pi(s_t, a_t)$ using $V_\pi(s_{t+1})$ of the next-time-step state s_{t+1}.

3. Estimating both state values and action values simultaneously proves cumbersome, entailing excessive computational and storage demands. Therefore, the $Q_\pi(s_{t+1}, a_{t+1})$ for a specific action within the next state can substitute for the $V_\pi(s_{t+1})$ of the next state itself.

4. The sole distinction between the Q-learning and SARSA algorithms lies in which $Q_\pi(s_{t+1}, a_{t+1})$ is substituted for $V_\pi(s_{t+1})$. The SARSA algorithm selects the $Q_\pi(s_{t+1}, \pi(a_{t+1}))$ generated by the same policy at s_t. The Q-learning algorithm chooses the action yielding the maximum value, $\max_{a_{t+1} \in A(s_{t+1})} Q_\pi(s_{t+1}, a_{t+1})$.

In Section 5, the Q-learning algorithm is refined by substituting game theoretic policy replacement with greedy policy improvement. For value function updates, game-theoretic operators are also employed to replace the maximum Q value.

3.3 Summary of This Chapter

This chapter elucidates the significance and computational methods of value functions, introducing the principles and origins of two value function update approaches in reinforcement learning: the Monte Carlo (MC) method and the Time Discounted (TD(0))

method. Both methods possess highly intuitive implications within our model. The MC method updates the value function only after completing the entire parking process, based on the final outcome. Its advantage lies in accurate evaluation, though it is less efficient. The TD(0) method, conversely, enables real-time refinement of decisions at each step by incorporating value estimates for subsequent states, thereby enhancing learning efficiency. Both methods employ greedy strategies in single-agent scenarios with commendable results. However, in multi-agent contexts, the complexity for each agent increases exponentially due to mutual interactions. Simply maximizing individual payoffs no longer yields satisfactory outcomes. Consequently, the Markov game framework is employed, integrating game theory with reinforcement learning to devise strategies better suited to multi-agent environments and pursue algorithms of higher precision and efficiency.

4 Embedding Nash Equilibrium Strategies into Reinforcement Learning

As discussed previously, single-agent algorithms are categorized based on their policy evaluation methods, with the model represented as MDP. When the environment transitions to multi-agent scenarios, classical reinforcement learning algorithms fail to deliver satisfactory outcomes. Taking the Q-learning algorithm as a straightforward example, its update formula is Equation 3.8.

In multi-agent scenarios, the update objective cannot solely maximize the Q-value, as the Q-value simultaneously depends on the actions of other agents. This renders traditional reinforcement learning methods both imprecise and inefficient within multi-agent environments.

In game theory contexts, replacing a single agent with multiple agents substantially increases environmental randomness. This necessitates considering not only one's own payoffs and strategies but also those of opponents. Understanding the multi-agent game framework is therefore essential.

In two-player random games, the mixed-strategy Nash equilibrium corresponds to zero-sum games and maximin strategies. Nash equilibrium strategies are flexible and adaptable; their significance lies in the fact that no player can deviate from this strategy to gain greater rewards while the other player's strategy remains unchanged. Maximin strategies represent a conservative, safe approach, assuming the opponent will act in the most disadvantageous manner and maximizing one's own payoff based on this premise. Games are categorized as static or dynamic based on the simultaneity or sequence of actions. This paper exclusively addresses simultaneous-move static games, where players are unaware of their opponent's actions at any given moment.

4.1 Two-Player Zero-Sum Games

A two-player zero-sum game is a form of game theory where, under strict competition, one party's gain necessarily implies the other's loss. Whatever one party gains, the other loses; players may win or lose, but the total payoff of the game is always zero. Thus, the sum of rewards for both players is always 0. In such circumstances, cooperation is impossible [30]. Both parties will endeavor to act in a manner that "harms others for personal gain". Two-player zero-sum games describe situations of strict competition between two individuals. Matrix games are two-player zero-sum games with finite action sets.

4.1.1 Matrix Games

A matrix game is constituted by the tuple $\langle \{1,2\}, A^1, A^2, r^1 \rangle$. Here, 1 denotes Player One and 2 denotes Player Two; A^i represents Player i's finite action space, $i = 1, 2$;

$$A^1 = \left\{ a_1^1, a_2^1, \cdots, a_m^1 \right\}, \tag{4.1}$$

a_i^1 denotes Player 1's i-th action, and m represents the total number of actions available to Player 1.

$$A^2 = \left\{a_1^2, a_2^2, \cdots, a_n^2\right\}, \tag{4.2}$$

a_i^2 represents Player 2's i-th action, where n denotes Player 2's total number of actions.

r^1 represents Player One's payoff, $-r^1$ represents Player Two's payoff. $r^1(i,j)$ denotes Player One's payoff when Player One chooses action i and Player Two chooses action j. Here, $i = a_i^1$, where $j = a_j^2$.

Defining $b_{ij} = r^1(i,j)$, the matrix $B = [b_{ij}]_{m\times n}$ constitutes Player 1's payoff matrix.

As one player's payoff is precisely the inverse of the other's, a finite-action, two-person, zero-sum static game can be represented by a matrix B with m rows and n columns, where $b_{ij} = r^1(i,j)$, $\forall i \in A^1$ and $\forall j \in A^2$.

b_{ij} represents Player One (the row of the matrix) choosing the i-th action and Player Two (the column of the matrix) choosing the j-th action, yielding a payoff of $-b_{ij}$ for Player One.

4.1.2 Min-Max Strategy

In the aforementioned matrix game, a strategy known as the minimax strategy (maxmin strategy) or safe strategy (security strategy) possesses several favorable properties. Littman proposed the maxmin-Q algorithm in 1994, substituting the "maximize" operator Q-learning in the algorithm with a "minimax" operator, achieving favorable results in experimental comparisons.

In matrix games, maximizing one player's payoff equates to minimizing the other's. When Player 1 (row of the matrix) chooses pure strategy i, Player 2 (column) may select the strategy that minimizes Player 1's payoff:

$$\min_{j\in A^2} b_{ij}. \tag{4.3}$$

Simultaneously, Player One seeks to select the pure strategy i that maximizes the aforementioned payoff. That is, they aim to choose the pure strategy i such that:

$$\max_{i\in A^1} \min_{j\in A^2} b_{ij}. \tag{4.4}$$

Player 1's optimal pure strategy is maximin strategy. Regardless of Player 1's choice, Player Two will select the strategy that minimizes Player 1's payoff. Within this context, Player One selects the pure strategy i that yields the maximum payoff. This strategy is termed Player 1's maximin strategy (safe strategy). Similarly, Player 2's safe strategy can be determined as:

$$\min_{j\in A^2} \max_{i\in A^1} b_{ij} \tag{4.5}$$

This also constitutes Player Two's optimal strategy.

Definition 4.1 (Maximin) *Given a matrix game B, the maximin value is defined as:*

$$r = \max_{i\in A^1} \min_{j\in A^2} b_{ij}. \tag{4.6}$$

Definition 4.2 (Minimax) *Given a matrix game B, the minimax is defined as:*

$$R = \min_{j\in A^2} \max_{i\in A^1} b_{ij}. \tag{4.7}$$

Definition 4.3 (Pure Strategy Value) *Given a matrix game B, if $r = R$, then $r' = r = R$ is termed the pure strategy value of the matrix game.*

Demonstrating how to find the maximin strategy using a matrix game as an example. The payoffs for the matrix game B_1 are as follows:

$$B_1 = \begin{bmatrix} 2 & 1 \\ 3 & 0 \end{bmatrix}. \tag{4.8}$$

The maximin value of the matrix is

$$\max_{i \in A^1} \min_{j \in A^2} b_{ij} = \max\{1, 0\} = 1. \tag{4.9}$$

The minimax value is

$$\min_{j \in A^2} \max_{i \in A^1} b_{ij} = \min\{3, 1\} = 1. \tag{4.10}$$

Therefore, $r' = r = R$ is the pure strategy value for the matrix game B_1. At this point, the pure strategy set corresponds to Player 1 taking action 1 and Player 2 taking action 2. For Player 1, its maximin strategy is action 1.

However, not all matrix games possess pure strategy values. Consider the rock-paper-scissors game: Let B_2 denote the payoff matrix for the rock-paper-scissors game.

$$B_2 = \begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix}. \tag{4.11}$$

where the first row (first column) represents the pure strategy "rock"; the second row (second column) represents the pure strategy "scissors"; and the third row (third column) represents the pure strategy "paper".

The maximin value for this game is

$$\max_{i \in A^1} \min_{j \in A^2} b_{ij} = \max\{-1, -1, -1\} = -1. \tag{4.12}$$

The minimax value is

$$\min_{j \in A^2} \max_{i \in A^1} b_{ij} = \min\{1, 1, 1\} = 1. \tag{4.13}$$

Clearly, the game does not possess a pure strategy value. Moreover, all three pure strategies—rock, scissors, and paper—constitute Player 1's minimax strategy.

The maximin strategy maximizes one's minimum payoff, constituting a conservative, safe approach that considers the worst-case scenario resulting from the opponent's actions. In the article Littman, the maximin strategy is employed to enhance the Q-learning algorithm. To prevent opponents from predicting actions, when multiple maximin strategies exist within a game, a mixed approach is used, assigning probabilities to each action corresponding to a maximin strategy. For instance, Player One's maximin strategy is $\pi = \arg\max_{\pi \in \Delta(A^1)} \min_{j \in A^2} b_{ij}$. This means assigning probabilities to one's own actions against the opponent's worst possible move to maximize one's own payoff. Here, $\Delta(A^1)$ represents Player One's mixed strategy action space.

4.2 Two-Person Normal-Form Games

Stochastic games are designed for competition among multiple agents, sharing this characteristic with matrix games as non-cooperative games where each agent strives to maximize their own objectives. Here, stochastic games are restricted to two players.

4.2.1 Two-Player Randomized Game

A two-person randomized game is constituted by the tuple $\langle \{1,2\}, A^1, A^2, r^1, r^2 \rangle$. Here, 1 denotes Player One and 2 denotes Player Two; A^i represents Player i's finite action space, where $i = 1, 2$:

$$A^1 = \left\{ a_1^1, a_2^1, \cdots, a_m^1 \right\}. \tag{4.14}$$

a_i^1 denotes Player 1's i-th action, where m is the total number of actions for Player 1.

$$A^2 = \left\{ a_1^2, a_2^2, \cdots, a_n^2 \right\}. \tag{4.15}$$

a_i^2 denotes Player 2's i-th action, where n denotes the total number of actions for Player 2.

r^1 represents Player 1's payoff, r^2 denotes Player 2's payoff. $r^1(\pi^1, \pi^2)$ indicates Player 1's payoff when both players employ the mixed strategy set (π^1, π^2).

Definition 4.4 (Mixed Strategy Set) *For a given player i and their pure strategy set A^i, π^i denotes player i's mixed strategy. π^i is a probability distribution over A^i, meaning $\pi^i : A^i \to [0,1]$ is a mapping that assigns a probability $\pi^i(a_j^i)$ to each pure strategy $a_j^i \in A^i$ such that:*

$$\sum_{a_j^i \in A^i} \pi^i \left(a_j^i \right) = 1. \tag{4.16}$$

4.2.2 Mixed Strategy Nash Equilibrium

Definition 4.4 bears a close resemblance to Definition 2.1 for mixed strategies, but Definition 4.1 is multi-player and does not require consideration of state s.

Definition 4.5 (Nash Equilibrium for Mixed Strategies) *Given a two-person randomized game $\langle \{1,2\}, A^1, A^2, r^1, r^2 \rangle$ and a mixed strategy set (π^{1*}, π^{2*}) satisfying*

$$\begin{aligned} r^1 \left(\pi^{1*}, \pi^{2*} \right) &\geqslant r^1 \left(\pi^1, \pi^{2*} \right), \quad \forall \pi^1 \in \Delta \left(A^1 \right); \\ r^2 \left(\pi^{1*}, \pi^{2*} \right) &\geqslant r^2 \left(\pi^{1*}, \pi^2 \right), \quad \forall \pi^2 \in \Delta \left(A^2 \right); \end{aligned} \tag{4.17}$$

then (π^{1*}, π^{2*}) constitutes a mixed strategy Nash equilibrium. Here, $\Delta(A^i)$ denotes the set of all mixed strategies for player i.

The optimal response function (ORF) $b^i(\pi^{-i})$ for player i is defined as follows:

$$b^i(\pi^{-i}) = \left\{ \pi^i \in \Delta \left(A^i \right) : r^i \left(\pi^i, \pi^{-i} \right) \geqslant r^i \left(\pi^{i'}, \pi^{-i} \right), \forall \pi^{i'} \in \Delta \left(A^i \right) \right\}, \quad i = 1, 2, \tag{4.18}$$

where π^{-i} denotes the strategy of the other player, and $b^i(\pi^{-i})$ represents the optimal response of player i to the other player's action π^{-i}.

Clearly, the mixed strategy set (π^{1*}, π^{2*}) constitutes a Nash equilibrium if and only if

$$\pi^{i*} \in b^i \left(\pi^{-i*} \right), \quad i = 1, 2. \tag{4.19}$$

From the preceding, it is evident that a deterministic optimal strategy exists for a single agent under an MDP model. However, in a two-player zero-sum game, a deterministic Nash equilibrium strategy set may not necessarily exist. Instead, mixed strategies are more common (deterministic strategies can also be regarded as a special type of mixed

strategy). Consider the example of rock-paper-scissors. Here, the Nash equilibrium strategy set is (π^1, π^2), where $\pi^i = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$, $i = 1, 2$, meaning each action (rock, paper, scissors) is assigned a one-third probability. Any unilateral deviation from this strategy set by either party yields no higher payoff. However, employing a pure strategy such as consistently choosing rock would prompt the opponent to consistently choose paper upon observation.

How does one find the mixed strategy Nash equilibrium?

Consider the following two-player normal-form (Table 4.1):

Table 4.1: Two-Player Normal-Form Game

Player 1 \ Player 2	**A**	**B**
A	$(2, 1)$	$(0, 0)$
B	$(0, 0)$	$(1, 2)$

The matrix indicates that when Player One and Player Two employ the pure strategy set (A, A), Player One's payoff is 2 and Player Two's payoff is 1. When both adopt the pure strategy set (B, B), Player One's payoff is 1 and Player Two's payoff is 2.

Let Player One's strategy be π^{1*} and Player Two's strategy be π^{2*}. The mixed strategy set (π^{1*}, π^{2*}) constitutes a Nash equilibrium. By Definition 4.4, we have

$$\pi^{i*}(A) + \pi^{i*}(B) = 1, \quad i = 1, 2. \tag{4.20}$$

From the payoff matrix, Player 1's payoff is:

$$\begin{aligned} r^1\left(\pi^{1*}, \pi^{2*}\right) &= 2\pi^{1*}(A)\,\pi^{2*}(A) + 0 \times \pi^{1*}(A)\,\pi^{2*}(B) \\ &\quad + 0 \times \pi^{1*}(B)\,\pi^{2*}(A) + \pi^{1*}(B)\,\pi^{2*}(B) \\ &= 3\pi^{1*}(A)\,\pi^{2*}(A) - \pi^{1*}(A) - \pi^{2*}(A) + 1. \end{aligned} \tag{4.21}$$

Similarly, Player 2's payoff is calculated as:

$$r^2\left(\pi^{1*}, \pi^{2*}\right) = 3\pi^1(A)\,\pi^{2*}(A) - 2\pi^{1*}(A) - 2\pi^{2*}(A) + 2. \tag{4.22}$$

By Definition 4.5, we have

$$\begin{aligned} &3\pi^{1*}(A)\,\pi^{2*}(A) - \pi^{1*}(A) - \pi^{2*}(A) + 1 \\ &\geqslant 3\pi^1(A)\,\pi^{2*}(A) - \pi^1(A) - \pi^{2*}(A) + 1, \quad \forall \pi^1(A) \in \Delta\left(A^1\right). \end{aligned} \tag{4.23}$$

$$\begin{aligned} &3\pi^{1*}(A)\,\pi^{2*}(A) - 2\pi^{1*}(A) - 2\pi^{2*}(A) + 2 \\ &\geqslant 3\pi^{1*}(A)\,\pi^2(A) - 2\pi^{1*}(A) - \pi^2(A) + 1, \quad \forall \pi^2(A) \in \Delta\left(A^2\right). \end{aligned} \tag{4.24}$$

Rearranging yields

$$\begin{aligned} \pi^{1*}(A)\left(3\pi^{2*}(A) - 1\right) &\geqslant \pi^1(A)\left(3\pi^{2*}(A) - 1\right), \quad \forall \pi^1(A) \in \Delta\left(A^1\right); \\ \pi^{2*}(A)\left(3\pi^{1*}(A) - 2\right) &\geqslant \pi^2(A)\left(3\pi^{1*}(A) - 2\right), \quad \forall \pi^2(A) \in \Delta\left(A^2\right). \end{aligned} \tag{4.25}$$

Here, we consider three distinct scenarios:

1. When $[3\pi^{2*}(A) - 1] > 0$, $\pi^{1*}(A) = 1$, yielding the pure strategy set (A, A). Clearly, both Player 1 and Player 2 are optimally responding to each other. Thus, the pure strategy set (A, A) constitutes a pure strategy Nash equilibrium.

2. When $[3\pi^{2*}(A)-1]<0$, $\pi^{1*}(A)=0$, yielding the pure strategy set (B,B). Clearly, both Player 1 and Player 2 are each other's best responses. Thus, the pure strategy set (B,B) constitutes a pure strategy Nash equilibrium.

3. When $[3\pi^{2*}(A)-1]=0$, i.e. $\pi^{2*}(A)=\frac{1}{3}$, we obtain $\pi^{1*}(A)=\frac{2}{3}$, $\pi^{2*}(A)=\frac{1}{3}$. Thus the mixed-strategy Nash equilibrium is (π^{1*},π^{2*}):

$$\begin{aligned}\pi^{1*}(A)&=\frac{2}{3},\quad \pi^{1*}(B)=\frac{1}{3};\\ \pi^{2*}(A)&=\frac{1}{3},\quad \pi^{2*}(B)=\frac{2}{3}.\end{aligned}\tag{4.26}$$

It is straightforward to verify that $\pi^{1*}\in b^1(\pi^{2*})$, $\pi^{2*}\in b^2(\pi^{1*})$.

In single-agent scenarios, the strategy for algorithmic improvement is deterministic; however, in multi-agent systems, mixed strategies are employed. It may seem counterintuitive that optimal strategies can be probabilistic, given that within the framework of MDP, there always exists a deterministic strategy that is no worse than any probabilistic one. Yet in multi-agent environments, the uncertainty of opponents' actions and the need to prevent opponents from exploiting strategies necessitate that each action be specified probabilistically.

Having examined both game types and their corresponding strategy approaches, we shall now address the question of strategy existence.

For a matrix game, a minimax value $\max_{i\in\Delta(A_1)}\min_{j\in A_2}b_{ij}$ can always be found for each player, and thus a minimax strategy necessarily exists. Simultaneously, the minimax theorem indicates that every finite game possesses a mixed Nash equilibrium strategy.

4.3 Markov Game Framework (Markov games)

Within the MDP environment, we understand how an agent interacts with the environment and updates its value function. However, to observe opponents and make optimal decisions, we must transform the original MDP interaction between a single agent and the environment into a multi-agent interaction with the environment (i.e., whereas the original MDP framework treats opponents as part of the environment, we now treat opponents as agents and consider their strategies and actions). The framework of Markov games extends this perspective by generalizing the Markov decision process to scenarios involving two or more agents.

The Markov game framework constitutes an extension of game theory to MDP environments, essentially representing a multi-agent Markov decision process with significant parallels to MDP. A two-player zero-sum Markov game is formally defined as follows:

The complete framework for a two-player Markov game is formally defined by a specific septuple: $\langle S,A^1,A^2,r^1,r^2,P,\gamma\rangle$.

- S denotes the state space; in finite Markov games, this set has a finite number of elements.
- $A^i(s)$ denotes the action space available to agent i when in state s. In finite Markov games, each set element has a finite number of elements.
- $P(s'\mid s,a^1,a^2)$ denotes the probability that, in state s, when Agent 1 takes action $a^1\in A^1(s)$, Agent 2 takes action $a^2\in A^2(s)$, the transition occurs to state s'.

$P : S \times A^1 \times A^2 \to \Delta(S)$, where $\Delta(S)$ represents the entire set of probability distributions over the discrete state space S.

- $r^i(s, a^1, a^2)$ denotes the reward obtained by Agent i when Agent 1 takes action $a^1 \in A^1(s)$ and Agent 2 takes action $a^2 \in A^2(s)$ in state s.

$$r^i\left(s, a^1, a^2\right) = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t^1 = a^1, A_t^2 = a^2\right]. \tag{4.27}$$

- $\gamma \in [0, 1]$ denotes the discount rate, indicating the objective of each agent in the Markov game.

$$\max_{\pi^1, \pi^2} \mathbb{E}_{\pi^1, \pi^2}\left[\sum_{t=1}^{T} \gamma^t r^i\left(s_t, a_t^1, a_t^2\right)\right], \quad i = 1, 2, \tag{4.28}$$

where T denotes the time of reaching the terminal state. π^1, π^2 represents the policy set for both agents.

The definition of a Markov game closely resembles that of MDP, yet extends from single-agent to multi-agent environments. Like MDP, each Markov game possesses a set of non-empty optimal strategies, at least one of which is stationary. Unlike MDP, the optimal strategy in a Markov game is not necessarily deterministic. Instead, the optimal strategy is predominantly a mixed strategy.

In multi-agent settings, agents must consider opponents' strategies alongside their own. Deterministic policies are vulnerable to exploitation or "second-guessing" by opponents [31]. Consider the rock-paper-scissors game: consistently choosing rock would prompt opponents to counter with paper in subsequent rounds, necessitating a mixed strategy where rock, paper, and scissors are selected with probabilities defined by $\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$.

Within the reinforcement learning framework of Markov Decision Processes (MDPs), a single agent interacts with an environment defined by a transition probability function. In this solipsistic perspective, another agent is treated as part of the environment, leading to scant consideration of opponents. The framework of Markov Games addresses this shortcoming by introducing opponents as distinct agents within the game, each possessing their own action space and reward function. Consequently, careful consideration of an opponent's strategy becomes essential before formulating one's own.

4.4 Multi-Agent Reinforcement Learning

Q-learning The algorithm can be extended to multi-agent environments based on the Markov game framework. First, the value function must be redefined for multi-agent scenarios. Subsequently, an algorithm for agents to learn these value functions is presented. Finally, an analysis of the computational complexity and convergence of this algorithm is conducted.

To accommodate multi-agent settings, the initial step involves recognizing the necessity of considering strategy combinations rather than individual actions. Consequently, each agent's action value function should relate to another agent's actions, transforming the single-agent $Q_\pi(s, a)$ into a multi-agent $Q_\pi^i(s, a^1, a^2)$. Analogous to Definition 2.1, the multi-agent value function can be defined as follows:

Definition 4.6 (Multi-agent Value Function) *The state value function for an agent i is*

$$V_{\pi^1, \pi^2}^i(s) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{\pi^1, \pi^2}\left[r^i\left(s_t, a_t^1, a_t^2\right)\right]. \tag{4.29}$$

This represents the expected total reward obtained by agent i when adopting the policy combination (π^1, π^2) following state $s \in S$. Here, s_t denotes the state at time t, a_t^i denotes the action chosen by agent i at time t according to policy $\pi^i,\ i = 1, 2$.

The action value function for agent i is as follows:

$$Q^i_{\pi^1,\pi^2}\left(s, a^1, a^2\right) = r^i\left(s, a^1, a^2\right) + \gamma \sum_{s' \in S} P\left(s' \mid s, a^1, a^2\right) V^i_{\pi^1,\pi^2}\left(s'\right). \tag{4.30}$$

It denotes the expected total reward obtained by agent i when, in state $s \in S$, agent 1 selects action $a^1 \in A^1(s)$ and agent 2 selects action $a^2 \in A^2(s)$, subsequently adopting the policy combination (π^1, π^2).

Regarding the policy combination (π^1, π^2), note that unlike MDPs, it denotes a policy combination encompassing both agents' policies; unlike matrix games, it fully defines both agents' behavioral patterns—specifically, all actions and their probabilities across every state (In matrix games, a strategy profile is defined as the set of actions and their probability distributions for all players in a specific state.).

Based on the above description, when updating the action-value function, let $|S|$ denote the total number of states and $|A^i|$ denote the total number of actions for agent i. Then, the number of entries Q^i needs to maintain is $|S| \times |A^1| \times |A^2|$. Simultaneously, agents must create two Q-tables to record changes in their own and their opponent's value functions. Therefore, the total number of entries each agent needs to maintain is $2 \times |S| \times |A^1| \times |A^2|$. Should one wish to extend single-agent reinforcement learning to n-agent reinforcement learning in future, the number of entries each player must maintain is $n \times |S| \times |A|^n$ (assuming that the total number of actions for each agent is $|A|$). It can be observed that, in terms of space complexity, the change based on the number of states is linear, the change based on the number of actions is polynomial, and the change based on the number of agents is exponential.

We shall now examine reinforcement learning under different scenarios.

1. When there is only one agent, a greedy strategy is employed to maximise self-interest.

$$\pi = \arg\max_{a \in A(s)} Q\left(s, a\right). \tag{4.31}$$

$$Q_{t+1}\left(s, a\right) = Q_t\left(s, a\right) + a_t\left[R_t + \gamma \max_{a \in A(s)} Q_t\left(s, a\right) - Q_t\left(s, a\right)\right]. \tag{4.32}$$

2. When two agents exist and the game is zero-sum, employ the Minmax-Q algorithm from Littman [6]:

$$\pi = \arg\max_{a^1 \in \Delta(A^1)} \min_{a^2 \in A^2} Q\left(s, a^1, a^2\right). \tag{4.33}$$

$$\begin{aligned} &Q_{t+1}\left(s, a^1, a^2\right) \\ &= Q_t\left(s, a^1, a^2\right) + a_t\left[R_t + \gamma \max_{a^1 \in \Delta(A^1)} \min_{a^2 \in A^2} Q_t\left(s, a^1, a^2\right) - Q_t\left(s, a^1, a^2\right)\right]. \end{aligned} \tag{4.34}$$

The search for maximum and minimum values in each state resembles the general demonstration in Section 4.1.2, employing linear programming methods. Note that in the presence of non-unique maxmin strategies, employing a probability distribution over these strategies maximizes the player's expected payoff.

3. When two agents exist and the game is general and two-player, the Nash-Q algorithm from Wellman is employed [7]:

$$\pi^i = \arg \mathrm{Nash}Q^i\left(s, a^1, a^2\right). \tag{4.35}$$

$$Q_{t+1}^i\left(s, a^1, a^2\right) = Q_t^i\left(s, a^1, a^2\right) + a_t\left[R_t + \gamma \mathrm{Nash}Q^i\left(s, a^1, a^2\right) - Q_t^i\left(s, a^1, a^2\right)\right]. \tag{4.36}$$

The procedure for finding mixed strategy Nash equilibria in each state is as described in Section 4.2.2, employing methods for solving systems of linear equations. Zero-sum games constitute a special case of general-sum games, rendering the Nash-Q algorithm more universally applicable than the Minmax-Q approach. However, computations in Section 4.2.2 reveal that a game may harbor multiple Nash equilibria. When one agent selects one Nash equilibrium as strategy, another agent may opt for a different equilibrium. This scenario reduces the algorithm's convergence speed and precision, potentially trapping it in local Nash equilibria–precisely the issue addressed by Wellman when proposing the Nash-Q algorithm.

5 Penalty Mechanism and Convergence Analysis

5.1 Insufficient Convergence of the Nash-Q Algorithm in Multi-Nash Equilibrium and Policy Oscillation in Parking Scenarios

This paper uses a real-world parking scenario to illustrate the shortcomings of the Nash-Q algorithm when confronting multiple Nash equilibria (Figure 5.1).

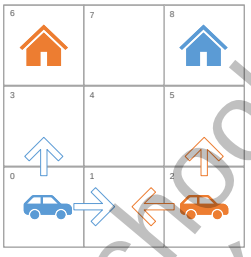

Figure 5.1: Schematic Diagram of the Parking Grid Game

Vehicle 1's parking space is in the top-right corner, while Vehicle 2's is in the top-left corner. Both parties aim to park in their designated spaces. This scenario encompasses all elements of multi-agent reinforcement learning (multiple agents, initial state positions, actions, state transitions, immediate rewards, and long-term rewards). Each vehicle may move only one cell at a time, in any of the four cardinal directions: left, right, up, or down. Should either vehicle attempt to occupy the same cell, excluding their own designated space, it will be bounced back to its original position. The parking sequence concludes when either agent reaches their target space, at which point that driver receives a positive reward.

Thus, the objective is for both parties to reach their respective parking spaces in the fewest possible steps. One driver's "victory" does not preclude the other's success, incentivizing coordination.

The actions of driver i are: $A^i = \{\text{up}, \text{down}, \text{left}, \text{right}\}$ where $i = 1, 2$. The state space is: $S = \{(0,1), (0,2), \cdots, (8,7)\}$.

$s_0 = (0,2)$ denotes the initial position state that Driver 1 at position 0, Driver 2 at position 2. Upon reaching the target position, a reward of 100 points is granted. Should a collision occur, a reward of -1 point is awarded, and both parties are bounced back to their initial positions. In this scenario, the state transitions for both parties are deterministic. To reflect the drivers' desire to reach parking spaces as swiftly as possible, a decay rate of $\gamma = 0.99$ is implemented.

It should be noted that when a driver's strategy depends solely on their position, assuming this strategy is a pure strategy sequence. this strategy effectively defines a path—a sequence of positional points from the starting point to the final destination.

Two mutually non-interfering shortest paths together constitute a Nash equilibrium, as each path (strategy) represents the optimal response to the other. That is, the Nash equilibrium of each driver's game at every stage forms the Nash equilibrium path for the entire dynamic stochastic process. Evidently, there are five Nash equilibrium paths for both drivers at this point, as shown in Figure 5.2.

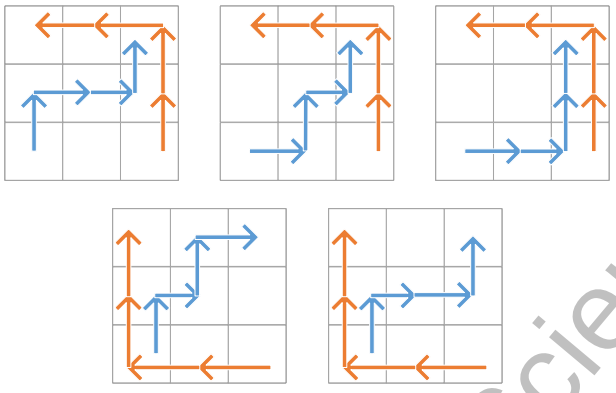

Figure 5.2: Equilibrium Paths

According to Equation 4.29, the state value function for Driver 1 on any Nash equilibrium path is $v^1(s_0) = 0 + 0.99 \times 0 + 0.99^2 \times 0 + 0.99^3 \times 100 = 97$. At this point, Equation 4.30 indicates that Driver 1's action value function $Q_\pi^i(s_0)$ is also 97. If both parties act left and right, $Q^1(s_0, \text{right}, \text{left}) = r^1(s, \text{right}, \text{left}) + \gamma \sum_{s' \in S} P(s' \mid s, \text{right}, \text{left})\, v^1(s) = -1 + 0.99 v^1(s_0) = 95.1$.

In this scenario, every Nash equilibrium path is clearly a globally optimal solution, with the value function being $(97, 97)$, as shown in Table 5.1.

Table 5.1: Theoretical Value Nash-Q under s_0

Driver 1 \ Driver 2	**Left**	**Up**
Right	$(95.1, 95.1)$	$(97, 97)$
Top	$(97, 97)$	$(97, 97)$

Through experimental simulation of this scenario, the agents simultaneously select actions at their initial positions ($s_0 = (0, 2)$). Subsequently, they concurrently acquire new states, their own rewards, and information about the opponent's actions. The learning agent updates its action value function according to Equation 4.30. In the new state, both agents repeat the aforementioned process. When at least one agent moves to the target position, the game restarts. The learning agent retains the action-value functions acquired in previous rounds. Training concludes after 5,000 rounds, with each round averaging approximately 8 steps; thus, a single experiment typically requires 40,000 steps. The grid game comprises 424 state-action pairs, each visited an average of 95 times. The learning rate is defined as the reciprocal of the visit count. In the later stages of learning, newly visited pairs scarcely alter the already learned action value function. After simulating 5000 training iterations, the action value functions of the drivers stabilized, as shown in the Table 5.2.

Table 5.2: Final Action-Value Functions for s_0 after Iterative Training

Driver 1 \ Driver 2	**Left**	**Up**
Right	$(86, 87)$	$(83, 85)$
Top	$(94, 91)$	$(95, 95)$

It can be observed that Table 5.2 yields results remarkably close to those theoretically derived in Table 5.1. Consequently, when every Nash equilibrium path constitutes a global optimum, the Nash-Q algorithm continues to maintain considerable advanced capability.

We can now introduce a more complex scenario. During peak hours, assuming positions 0 and 2 represent an intersection, upward movement may encounter congestion. Specifically, when moving from position 0 to 3 or from position 1 to 5, there is a 1/2 probability of bouncing back to the initial position. In this scenario, state transitions become non-deterministic. For instance, if Driver 1 and Driver 2 both move upwards, the next state has four possible outcomes: $(0, 2)$, $(0, 5)$, $(3, 2)$, and $(3, 5)$, each with a probability of 0.25. If Driver 1 moves upwards while Driver 2 moves leftwards, the probabilities of reaching the next state are $p((0, 1) \mid (0, 2), \text{up}, \text{left}) = 0.5$, $p((3, 1) \mid (0, 2), \text{up}, \text{left}) = 0.5$. Similarly, if Driver 1 moves right and Driver 2 moves up, the probability of reaching the next state is $p((1, 2) \mid (0, 2), \text{up}, \text{left}) = 0.5$, $p((1, 5) \mid (0, 2), \text{right}, \text{up}) = 0.5$. Furthermore, due to peak hour congestion, assume only one parking space is available at position 7. Both drivers compete for this space, with the successful driver receiving a payoff of 100 and the unsuccessful driver receiving 0.

Although the scenario has become more complex, it is evident that the Nash equilibrium paths for the drivers are these two.

Figure 5.3: Equilibrium Paths

According to Equation 4.29

$$\begin{aligned} v^1(0,1) &= 0 + 0.99 + 0.99^2 \times 0 = 0.99; \\ v^1(0,x) &= 0, \quad \text{for } x = 3, \cdots, 8; \\ v^1(1,2) &= 0 + 0.99 \times 100 = 99; \\ v^1(1,x) &= 99, \quad \text{for } x = 3, 5; \\ v^1(1,x) &= 0, \quad \text{for } x = 4, 6, 8. \end{aligned} \tag{5.1}$$

According to Equation 4.30, we have

$$\begin{aligned} Q^1(s_0, \text{right}, \text{left}) &= -1 + 0.99 v^1(s_0), \\ Q^1(s_0, \text{right}, \text{up}) &= 0 + 0.99 \left(\frac{1}{2} v^1(1,2) + \frac{1}{2} v^1(1,3) \right) = 98, \\ Q^1(s_0, \text{up}, \text{left}) &= 0 + 0.99 \left(\frac{1}{2} v^1(0,1) + \frac{1}{2} v^1(3,1) \right) = 49, \end{aligned}$$

$$Q^1(s_0, \text{up}, \text{up}) = 0 + 0.99\left(\frac{1}{4}v^1(0,2) + \frac{1}{4}v^1(0,5) + \frac{1}{4}v^1(3,2) + \frac{1}{4}v^1(3,5)\right)$$
$$= 0.99v^1(s_0) + 49. \tag{5.2}$$

At the initial state s_0, two pure strategy Nash equilibria exist: (right, up) and (up, left). Adopting the Nash equilibrium strategy (right, up) yields $v_\pi^1(s_0) = 98$; adopting (up, left) yields $v_\pi^1(s_0) = 49$. The action-value functions for these strategies s_0 are as follows (Table 5.3 and Table 5.4):

Table 5.3: Theoretical Nash-Q Values under $(s_0, \text{up}, \text{left})$

Driver 1 \ Driver 2	**Left**	**Up**
Right	(47, 96)	(98, 49)
Top	(49, 98)	(61, 73)

Table 5.4: $(s_0, \text{right}, \text{up})$ Lower Theoretical Nash-Q

Driver 1 \ Driver 2	**Left**	**Upper**
Right	(96, 47)	(98, 49)
Up	(49, 98)	(93, 61)

There also exists a mixed strategy Nash equilibrium $(\pi^1(s_0), \pi^2(s_0))$, where $\pi^1(s_0) = \{[p(\text{right}) = 0.97], [p(\text{up}) = 0.03]\}$ and $\pi^2(s_0) = \{[p(\text{left}) = 0.97], [p(\text{up}) = 0.03]\}$. The action-value function for this strategy group s_0 is as follows (Table 5.5):

Table 5.5: Theoretical Value of Nash-Q under $(s_0, \pi^1(s_0), \pi^2(s_0))$

Driver 1 \ Driver 2	**Left**	**Up**
Right	(47.87, 47.87)	(98, 49)
Top	(49, 98)	(61.2, 61.2)

It can be observed that the second parking space scenario contains three distinct sets of Nash-Q values. When multiple Nash equilibria exist, the convergence of learning becomes problematic. Furthermore, in the initial state phase of the game, none of the Nash equilibria derived from the above three tables constitutes a globally optimal solution or a saddle point.

This scenario was similarly simulated through 5000 training iterations. For each state s, the Nash equilibrium $\pi^1(s), \pi^2(s)$ was derived from the stage game formed by the learned action-value function $(Q^1(s), Q^2(s))$. Subsequently, this solution was compared with the theoretically derived Nash equilibrium, revealing that the learning process did not consistently converge towards the Nash equilibrium of the joint strategy (Table 5.6).

Table 5.6: Final Nash-Q Values after Iterative Training of s_0

Driver 1 \ Driver 2	**Left**	**Top**
Right	(39, 84)	(97, 51)
Top	(46, 93)	(59, 74)

This outcome appears inevitable, as none of the three Nash equilibria constitutes a globally optimal solution. Consequently, Driver 1 would prefer to adopt the Nash equilibrium strategy of '(right, up)' to maximize personal gain, while Driver 2 would favor '(up, left)'. Given simultaneous action, Driver 2's behavior ceases to be the optimal response to Driver 1's move. Thus, the probability of mutual left-side collisions causing both vehicles to rebound to their original positions increases significantly.

Suppose a road sign could be erected to guide both parties' actions, or a set of rules established for them to follow, with non-compliance incurring fines or penalties. This would inevitably reduce the probability of mutual conflict. Therefore, a penalty mechanism can be designed: when two agents fail to select the same Nash equilibrium strategy, penalties are imposed to ensure they choose identical Nash equilibria, thereby enhancing convergence speed and accuracy.

5.2 Penalty Mechanism

This section proposes a new penalty mechanism that guides agents towards a unique, optimal Nash equilibrium by penalizing parking conflicts. In multi-agent games where multiple Nash equilibria exist, each equilibrium strategy combination corresponds to distinct rewards. During learning, it cannot be guaranteed that two agents will select the same set of Nash equilibria. Hu, Wellman, Littman and Bowling have proposed distinct conditions aimed at guaranteeing algorithm convergence. Currently, no method exists to autonomously guide agents towards converging on the same Nash equilibrium. To address this shortcoming, this paper incorporates a penalty mechanism that compels agents to select the identical set of Nash equilibrium strategies.

First, we define the multi-agent reinforcement learning framework incorporating the penalty mechanism.

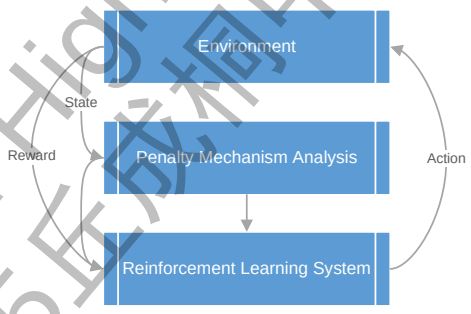

Figure 5.4: Generalized Markov Game Framework

Figure 5.4 depicts the defined generalized Markov game framework, composed of the tuple $\langle S, A^1, A^2, r^{1'}, r^{2'}, P, \gamma \rangle$, where

$$r^{i\prime}\left(s, a^1, a^2, s'\right) = r^i + F. \tag{5.3}$$

r^i denotes the original reward function, and F represents the penalty mechanism function. The above diagram bears some resemblance to Figure 2.2, differing in that figure depicts an interaction between a single agent and the environment, whereas this reinforcement learning system involves two agents. Furthermore, the state is transmitted back to the agents while also being fed into the penalty mechanism analysis. This analysis determines whether the agents have selected the same Nash equilibrium strategy; if not, a penalty is imposed. The selection of which Nash equilibrium and the specific penalty are as follows:

Under the penalty mechanism, the two agents must engage in mutual negotiation. First, all Nash equilibria are solved. This paper employs the Lemmke-Hauser method to compute Nash equilibria, which generates fixed-order equilibrium solutions [32]. Let agents 1 and 2 respectively compute all Nash equilibria as negotiation strategies $\pi^{im}, i = 1, 2, m = 1, \ldots, M$, where M denotes the number of Nash equilibria.

After computing all Nash equilibria, a target Nash equilibrium must be identified according to specific rules. These rules may be flexible and adaptable, tailored to the actual scenario.

To maximize the overall payoff for both parties, the target Nash equilibrium π^{ig} may be defined as:

$$\pi^{ig} = \arg\max_{\pi} \left[Q^1_{\pi^{1j}}\left(s, a^1, a^2\right) + Q^2_{\pi^{2j}}\left(s, a^1, a^2\right) \right] \pi^{ij} \in \pi^{im}, \tag{5.4}$$

namely, selecting the strategy that maximizes the sum of the Q values for agents 1 and 2.

If the objective is to minimize the disparity in payoffs between the two parties, the target Nash equilibrium π^{ig} can be defined as:

$$\pi^{ig} = \arg\min_{\pi} \left| {Q^1}_{\pi^{1j}}\left(s, a^1, a^2\right) - {Q^2}_{\pi^{2j}}\left(s, a^1, a^2\right) \right| \pi^{ij} \in \pi^{im}, \tag{5.5}$$

that is, selecting the strategy that minimizes the absolute difference between the Q values of agents 1 and 2.

Consider again the car-parking scenario from the previous section. The value functions corresponding to the two pure Nash equilibrium paths are perfectly symmetrical. In this case, to ensure fairness for both drivers, a time-series approach can be adopted for the target Nash equilibrium. Thus, when the hour hand is on an odd hour, the first Nash equilibrium is employed, and when on an even hour, the second is used. In practical terms, this functions like traffic lights: running a red light incurs a fine for rule-breaking. The penalty reduces both parties' willingness to break the rules in subsequent rounds—that is, their inclination to deviate from the target Nash equilibrium—thereby steering behavior towards the desired equilibrium.

Here, the penalty function F imposes sanctions based on state. After calculating the target Nash equilibrium strategy, the state transition function is used to compute the probabilities of each state in the next moment under this strategy. These probabilities are recorded in to establish the penalty mechanism for this game phase. Should a state in the next moment not appear in this table—meaning its probability derived from the target Nash equilibrium transition is zero—a penalty is incurred.

Note that due to the state transition function, an agent may deviate from the target Nash equilibrium. However, if the state after the action still satisfies the post-Nash equilibrium states, no penalty is incurred. Yet, with sufficiently numerous trials, the Law of Large Numbers dictates that agents deviating from the target Nash equilibrium will invariably incur penalties of varying severity.

$$F\left(s, a^1, a^2, s'\right) = \gamma\varphi\left(s'\right) - \varphi(s). \tag{5.6}$$

The transition from state s to s' reveals the strategies of both parties. If this strategy is the target policy, set $\varphi\left(s'\right) = 0$; otherwise, set $\varphi\left(s'\right) = -1$. It can be observed that the penalty mechanism function, like the reward function, undergoes temporal jumps, necessitating the inclusion of γ.

5.3 Proof of Convergence Invariance

5.3.1 Proof of Optimal Strategy Invariance

Theorem 5.1 (Optimal-Policy Invariance) *The Nash equilibrium strategy remains unchanged under the penalty mechanism:*

$$\pi_{M'}^{i*}(s) \;=\; \pi_{M}^{i*}(s), \quad \forall s \in S.$$

Proof The optimal strategy within the Markov game framework M will also be the optimal strategy within the generalized Markov game framework M' incorporating the penalty mechanism.

Regarding the model M, Hu and Wellman have proven that in two-player general-sum games, the Nash-Q F value function ${Q_M^i}^*$ satisfies the bellman equation.

$$Q_M^{i^*}\left(s, a^1, a^2\right) = \mathbb{E}\left[r^i\left(s, a^1, a^2, s'\right) + \gamma \operatorname{Nash} Q_M^{i^*}\left(s', a'', a^{2'}\right)\right]. \tag{5.7}$$

Subtracting $\varphi(s)$ from both sides and applying the identity transformation $\gamma\varphi(s')$ to the right-hand side,

$$\begin{aligned} & Q_M^{i*}\left(s, a^1, a^2\right) - \varphi(s) \\ &= \mathbb{E}\left[r^i\left(s, a^1, a^2, s'\right) + \gamma\varphi(s') - \varphi(s) + \gamma\left(\operatorname{Nash} Q_M^{i^*}\left(s', a^{1'}, a^{2'}\right) - \varphi(s')\right)\right]. \end{aligned} \tag{5.8}$$

Definition 5.1

$$Q_{M'}^i\left(s, a^1, a^2\right) \triangleq Q_M^{i*}\left(s, a^1, a^2\right) - \varphi(s). \tag{5.9}$$

Substituting Equation 5.6 and Equation 5.9 into Equation 5.8 yields

$$\begin{aligned} & Q_{M'}^i\left(s, a^1, a^2\right) \\ &= \mathbb{E}\left[r^i\left(s, a^1, a^2\right) + F\left(s, a^1, a^2, s'\right) + \gamma\left(\operatorname{Nash} Q_M^i\left(s', a^{1\prime}, a^{2\prime}\right)\right)\right] \\ &= \mathbb{E}\left[r^{i\prime}\left(s, a^1, a^2, s'\right) + \gamma\left(\operatorname{Nash} Q_M^i\left(s', a^{1\prime}, a^{2\prime}\right)\right)\right]. \end{aligned} \tag{5.10}$$

Equation 5.10 precisely corresponds to the bellman equation of the M' model.

The Nash equilibrium strategy π_M^{i*} for the agent i in the model M' must satisfy the following equation:

$$\pi_{M'}^{i*} = \operatorname{Nash} Q_{M'}^{i*}\left(s', a^1, a^2\right), \tag{5.11}$$

where $Q_{M'}^{i*}$ is the value function Q of the Nash equilibrium strategy in the model M'. Substituting into Definition 5.1,

$$\begin{aligned} \pi_{M'}^{i*} &= \operatorname{Nash} Q_{M'}^{i*}\left(s, a^1, a^2\right) \\ &= \operatorname{Nash}\left[Q_M^{i*}\left(s, a^1, a^2\right) - \varphi(s)\right] \\ &= \operatorname{Nash} Q_M^{i*}\left(s', a^1, a^2\right) \\ &= \pi_M^{i*}. \end{aligned} \tag{5.12}$$

Therefore, the Nash equilibrium strategy in the M' model will also be the Nash equilibrium strategy in the M model. Introducing a penalty mechanism does not alter the original strategy selection.

5.3.2 Proof of Iterative Convergence for Action Value Functions

The convergence of the Nash-Q algorithm prior to incorporating the penalty mechanism has been demonstrated by Hu and Wellman [7]. We now prove that the Nash-Q algorithm remains convergent after adding the penalty mechanism function.

Theorem 5.2 (Penalty-Mechanism Convergence Invariance) *$\forall L \geqslant 0$ and $\forall i$, there $\exists t \geqslant 0$ such that the iterates produced by the penalized Nash–Q update satisfy*

$$Q_{M'}^{i*}(s,a^1,a^2) - L \leqslant Q_{M'_{t+1}}^{i}(s,a^1,a^2) \leqslant Q_{M'}^{i*}(s,a^1,a^2) + L.$$

Proof For the Markov game framework model M, assuming the existence of a value iteration operator $B(Q)$ for Q that is a convergent operator, prove that under the generalized Markov game framework model M', the value iteration operator Q is also convergent.

Definition 5.2 *In the algorithm Nash-Q, the state value function of the Nash equilibrium strategy for player i is given by*

$$V_M^{i*}(s) = \max_{a^1,a^2} Q_M^{i*}\left(s,a^1,a^2\right). \tag{5.13}$$

It is readily apparent that

$$V_{M'}^{i^*}(s) = V_M^{i^*}(s) - \varphi(s). \tag{5.14}$$

For the Markov game framework model M, if $Q_{t+1}^i(s,a^1,a^2) = BQ_t^i(s,a^1,a^2)$ holds, then

$$\max_{i,s,a^1,a^2} \left|Q_{t+1}^i\left(s,a^1,a^2\right) - Q_M^{i*}\left(s,a^1,a^2\right)\right| \leqslant a \max_{i,s,a^1,a^2} \left|Q_t^i\left(s,a^1,a^2\right) - Q_M^{i*}\left(s,a^1,a^2\right)\right|, \tag{5.15}$$

where $0 < a < 1$

Let $L = \max_{i,s,a^1,a^2} |Q_t^i(s,a^1,a^2) - Q_M^{i*}(s,a^1,a^2)|$, then $\forall i$, $\forall (s,a^1,a^2)$ we have

$$Q_M^{i*}\left(s,a^1,a^2\right) - L \leqslant Q_t^i\left(s,a^1,a^2\right) \leqslant Q_M^{i*}\left(s,a^1,a^2\right) + L. \tag{5.16}$$

For the model M

$$Q_{t+1}^i\left(s,a^1,a^2\right) = r^i\left(s,a^1,a^2\right) + \gamma \sum_{s' \in S} P\left(s' \mid s,a^1,a^2\right) V_t^i\left(s'\right). \tag{5.17}$$

Subtract $\varphi(s)$ from both sides of the above equation, and perform an identical transformation on the right-hand side

$$\begin{aligned}
&Q_{t+1}^i\left(s,a^1,a^2\right) - \varphi(s) \\
&= r^i\left(s,a^1,a^2\right) + \gamma\varphi\left(s'\right) - \varphi(s) + \gamma \sum_{s' \in S} P\left(s' \mid s,a^1,a^2\right) V_t^i\left(s'\right) - \gamma\varphi\left(s'\right) \\
&\leqslant r^{i\prime}\left(s,a^1,a^2,s'\right) + \gamma \sum_{s' \in S} P\left(s' \mid s,a^1,a^2\right) \left[V_M^{i^*}\left(s'\right) + L\right] - \gamma\varphi\left(s'\right) \\
&\leqslant r^{i\prime}\left(s,a^1,a^2,s'\right) + \gamma \sum_{s' \in S} P\left(s' \mid s,a^1,a^2\right) \left[V_M^{i^*}\left(s'\right) - \varphi\left(s'\right)\right] + \gamma L.
\end{aligned} \tag{5.18}$$

Equation 5.18 may be rewritten as

$$\begin{aligned} & Q_{t+1}^{i}\left(s, a^{1}, a^{2}\right)-\varphi(s) \\ & =r^{i}\left(s, a^{1}, a^{2}\right)+F\left(s, a^{1}, a^{2}, s^{\prime}\right)+\gamma \sum_{s^{\prime} \in S} P\left(s^{\prime} \mid s, a^{1}, a^{2}\right)\left[V_{t}^{i}\left(s^{\prime}\right)-\varphi\left(s^{\prime}\right)\right] \\ & =r^{i^{\prime}}\left(s, a^{1}, a^{2}, s^{\prime}\right)+\gamma \sum_{s^{\prime} \in S} P\left(s^{\prime} \mid s, a^{1}, a^{2}\right)\left[V_{t}^{i}\left(s^{\prime}\right)-\varphi\left(s^{\prime}\right)\right] . \end{aligned} \tag{5.19}$$

Definition 5.3

$$Q_{M_{t+1}^{\prime}}^{i}\left(s, a^{1}, a^{2}\right) \triangleq Q_{t+1}^{i}\left(s, a^{1}, a^{2}\right)-\varphi(s), \tag{5.20}$$

$$V_{M_{t+1}^{\prime}}^{i}(s) \triangleq V_{t+1}^{i}(s)-\varphi(s). \tag{5.21}$$

Substituting Equation 5.19 and Equation 5.21 into Equation 5.20 yields

$$Q_{M_{t+1}^{\prime}}^{i}\left(s, a^{1}, a^{2}\right)=r^{i^{\prime}}\left(s, a^{1}, a^{2}, s^{\prime}\right)+\gamma \sum_{s^{\prime} \in S} P\left(s^{\prime} \mid s, a^{1}, a^{2}\right)\left[V_{M^{\prime} t}^{i}\left(s^{\prime}\right)\right] . \tag{5.22}$$

This precisely satisfies the equation update rule Bellman within the generalized Markov game framework model M'.

Substituting Equation 5.20 and Equation 5.14 into Equation 5.18 yields

$$Q_{M_{t+1}^{\prime}}^{i}\left(s, a^{1}, a^{2}\right) \leqslant r^{i^{\prime}}\left(s, a^{1}, a^{2}, s^{\prime}\right)+\gamma \sum_{s^{\prime} \in S} P\left(s^{\prime} \mid s, a^{1}, a^{2}\right)\left[V_{M^{\prime}}^{i *}(s)\right]+\gamma L. \tag{5.23}$$

Thus, $Q_{M_{t+1}^{\prime}}^{i}\left(s, a^{1}, a^{2}\right) \leqslant Q_{M^{\prime}}^{i *}\left(s, a^{1}, a^{2}\right)+L$ holds. Similarly, $Q_{M_{t+1}^{\prime}}^{i}\left(s, a^{1}, a^{2}\right) \geqslant Q_{M^{\prime}}^{i *}\left(s, a^{1}, a^{2}\right)-L$ can be proven to hold. Therefore, employing a penalty mechanism function within the M' model still satisfies the iterative update rule of the Bellman equation, whilst ensuring this iterative approach is also convergent.

5.4 Summary of this Chapter

This chapter employs a real-world parking scenario to explore the Nash-Q algorithm's convergence limitations and strategy oscillation issues within multiple Nash equilibria. Firstly, in a simplified scenario featuring a single globally optimal Nash equilibrium, experimental results demonstrate the Nash-Q algorithm's stable convergence to the theoretically optimal solution, validating its effectiveness under ideal conditions. However, when scenarios become more complex—featuring multiple Nash equilibria without a single globally optimal solution—the limitations of the Nash-Q algorithm become apparent. As each agent seeks to maximize its own interests, they fail to coordinate autonomously and reach consensus. Consequently, they become trapped in policy oscillations during the learning process, unable to converge to any joint optimal strategy. To address this shortcoming, this chapter proposes a novel penalty mechanism. This mechanism compels agents to achieve consensus and act collaboratively during learning by penalizing behavior deviating from the "target Nash equilibrium". Theoretical analysis demonstrates that this penalty mechanism not only effectively guides agent behavior but also preserves the convergence and optimal strategy selection of the original Nash-Q algorithm. The introduction of this penalty mechanism offers new insights for applying the Nash-Q algorithm in complex multi-agent environments. In the next chapter, we shall conduct specific experiments to further validate the significant improvements in convergence speed and stability achieved by incorporating the penalty mechanism.

6 Game Experiments and Data Analysis

This chapter designs a grid-based randomized two-player game simulating the scenario where two motorists seek to park their vehicles in their respective garages. Employing a multi-agent reinforcement learning approach based on Markov games for training, the Nash equilibrium strategy is utilized. Experimental comparisons are conducted using both the standard Nash-Q algorithm and the Nash-Q algorithm with the incorporated penalty mechanism. This chapter first introduces the experimental environment, followed by a detailed explanation of the game specifics and experimental results.

6.1 Experimental Environment

All code development and execution for the experiments in this chapter were conducted on the Ubuntu 20.04 operating system. This Linux-based system, primarily designed for desktop applications, has become the preferred choice for machine learning programming due to its ease and efficiency in fields such as deep learning. The server hardware configuration comprises an AMD Ryzen 7 4800H processor with 512GB RAM, meeting the requirements for experimental operations.

6.2 Parking Grid Game

To validate the convergence properties of the Nash-Q algorithm proposed in Chapter 5 under multiple Nash equilibria, this section designs a two-player grid parking game. This game simulates the scenario of two motorists parking during peak hours. Its core lies in creating multiple Nash equilibria to observe the algorithm's convergence behavior and verify the effectiveness of the penalty mechanism. It can adapt to training multi-agent reinforcement learning based on Markov games without loss of generality. The game diagram is shown in Figure 5.1. The game unfolds on a 3×3 grid with two players: Player One and Player Two. Player One may move either upwards or to the right, while Player Two may move either upwards or to the left. Moving upwards incurs resistance; whenever a player selects an upward move, there is a 0.5 probability of failure, causing them to remain stationary.

Should a player attempt to move to a position already occupied by another player, they shall remain stationary and refrain from executing the action. A player is deemed to have reached the end point when Player One moves to the upper-right parking point or Player Two moves to the upper-left parking point; that player shall cease movement. The game concludes when both players reach their respective treasure points, after which player positions are reset as depicted in Figure 5.1.

This game can be described using a Markov game. A Markov Decision Process (MDP) is defined as a quintuple comprising a state space S, an action space A, a transition probability matrix P, a reward function r, and a discount rate γ. This quintuple describes

the interaction between a single agent and an environment defined by the state transition probability function, wherein other agents are treated solely as components of the environment. The Markov game extends this limitation to enable multi-agent interaction or competition. Its general form comprises a state space S, a set of action spaces $A^1, \cdots, A^k$, representing the action spaces available to each agent, a state transition probability matrix P, a reward function r^i , denoting the reward for the i-th agent, and a decay coefficient γ. In the Markov game constituted by this game, the state set S comprises $A_9^2 = 72$ states, representing the positions of both agents within the grid. The action set $A^1 = \{r, u\}$, $A^2 = \{l, u\}$ denotes Player One's rightward and upward actions, and Player Two's leftward and upward actions respectively. Should an agent's intended destination be occupied by another agent, the action is not executed, with the state transition probability $P = 0$. If an agent is already at a grid edge position and executing an action would cause it to move off the grid, the action is also not executed, with a state transition probability of $P = 0$. When either agent selects the upward action, its state transition probability is $P = 0.5$. The transition probability for all other states is $P = 1$. The reward when an agent finds the treasure is $r = 1$; the reward when an agent collides with another agent is $r = -1$; and the reward in all other cases is $R = 0$. Let the decay coefficient be $\gamma = 0.9$, causing agents to ignore rewards after too many rounds, thereby incentivizing them to find the treasure as quickly as possible. The two-agent random game effectively characterizes the features of Markov games without loss of generality. Though simple, it possesses characteristics such as multi-agent information exchange, state transitions, and action selection typical of Markov games. All experiments conducted in the remainder of this chapter are based on the aforementioned game.

6.3 Nash Equilibrium Strategy

Let the value function for state $s \in S$ be denoted as $V(s)$, representing the expected future reward when the agent is in state s. Introducing the action set A^1, A^2, let $Q\left(s, a^1, a^2\right)$ denote the expected reward when both agents are in state s and choose actions a^1, a^2 respectively, where $a^1 \in A^1(s), a^2 \in A^2(s), s \in S$. For an agent, the probability distribution of its action selection is termed a strategy π. Different strategies yield distinct Q values, expressed as:

$$Q_\pi(s, a^1, a^2) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t^1 = a^1, A_t^2 = a^2\right]. \tag{6.1}$$

The process of estimating the Q_π value corresponding to a predicted policy π is termed policy evaluation. The policy evaluation method is illustrated in Table 6.1. When $n = 1$, only the immediate reward for this action is considered, while subsequent rewards are predicted using the $Q(s_{t+1})$ value of the next state. This policy evaluation method is known as the TD(0) (time-difference). This prediction method enables the value function to be updated in real-time, thereby avoiding actions that may degrade the value function. Consequently, it is widely employed. This experimental study will utilize this policy evaluation method.

For reinforcement learning algorithms, the ultimate objective of training an agent is to discover the optimal policy (π^*). Action selection based on this optimal policy further yields the optimal value function ($Q^*(s, a^1, a^2)$). Consequently, selecting the appropriate policy iteration method is paramount. This chapter introduces the Nash equilibrium policy to guide the agent's action selection. First, we introduce the concepts of pure and mixed strategies. For an agent, if its action selection is guided by a strategy to choose a specific action, this strategy is termed a pure strategy. For instance, both Q-learning and

Minimax algorithms belong to pure strategy approaches. Conversely, if strategy π does not select a single action but rather represents a probability distribution over a sequence of actions:

$$\pi = \{p_1, \cdots, p_k\}, \quad \sum_{n=1}^{k} p_i = 1, \tag{6.2}$$

where $p_1, \cdots, p_k$ represents the probability of selecting action $a_1, \cdots, a_k$ respectively.

For this game, when in state s, Player One and Player Two's Q values Q^1, Q^2 are represented by a 2×2 matrix as shown in Table 6.1, where "u" denotes "up", "l" denotes "left", and "r" denotes "right".

Table 6.1: Q-values Representing Intentions

Driver 1 \ Driver 2	**u**	**l**
u	(Q_1^1, Q_1^2)	(Q_2^1, Q_2^2)
r	(Q_3^1, Q_3^2)	(Q_4^1, Q_4^2)

Let the policy set be $\pi = \{\pi_1, \pi_2\}$, where π_1, π_2 denote the policies of Player One and Player Two respectively. The rewards for Player One and Player Two are expressed as:

$$\begin{aligned} r^1(\pi_1, \pi_2) &= \pi_1(\mathrm{u})\pi_2(\mathrm{u})Q_1^1 + \pi_1(\mathrm{u})\pi_2(\mathrm{l})Q_2^1 + \pi_1(\mathrm{r})\pi_2(\mathrm{u})Q_3^1 + \pi_1(\mathrm{r})\pi_2(\mathrm{l})Q_4^1 \\ r^2(\pi_1, \pi_2) &= \pi_1(\mathrm{u})\pi_2(\mathrm{u})Q_1^2 + \pi_1(\mathrm{u})\pi_2(\mathrm{l})Q_2^2 + \pi_1(\mathrm{r})\pi_2(\mathrm{u})Q_3^2 + \pi_1(\mathrm{r})\pi_2(\mathrm{l})Q_4^2. \end{aligned} \tag{6.3}$$

According to the definition of Nash equilibrium strategies, there exists a strategy set $\pi^* = \{\pi_1^*, \pi_2^*\}$ such that the payoff function attains its maximum value, namely, $\forall \pi_1, \pi_2$, we have $r^1(\pi_1^*, \pi_2^*) \geq r^1(\pi_1, \pi_2^*), r^2(\pi_1^*, \pi_2^*) \geq r^1(\pi_1^*, \pi_2)$. Substituting into Equation 6.3 yields:

$$\begin{aligned} &\pi_1^*(\mathrm{u})\left(Q_2^1 - Q_4^1\right) + \pi_2^*(\mathrm{u})\left(Q_3^1 - Q_4^1\right) + \left(Q_1^1 + Q_4^1 - Q_2^1 - Q_3^1\right)\pi_1^*(\mathrm{u})\pi_2^*(\mathrm{u}) \\ &\geqslant \pi_1(\mathrm{u})\left(Q_2^1 - Q_4^1\right) + \pi_2^*(\mathrm{u})\left(Q_3^1 - Q_4^1\right) + \left(Q_1^1 + Q_4^1 - Q_2^1 - Q_3^1\right)\pi_1(\mathrm{u})\pi_2^*(\mathrm{u}). \end{aligned} \tag{6.4}$$

$$\begin{aligned} &\pi_1^*(\mathrm{u})\left(Q_2^2 - Q_4^2\right) + \pi_2^*(\mathrm{u})\left(Q_3^2 - Q_4^2\right) + \left(Q_1^2 + Q_4^2 - Q_2^2 - Q_3^2\right)\pi_1^*(\mathrm{u})\pi_2^*(\mathrm{u}) \\ &\geqslant \pi_1^*(\mathrm{u})\left(Q_2^2 - Q_4^2\right) + \pi_2(\mathrm{u})\left(Q_3^2 - Q_4^2\right) + \left(Q_1^2 + Q_4^2 - Q_2^2 - Q_3^2\right)\pi_1^*(\mathrm{u})\pi_2(\mathrm{u}). \end{aligned} \tag{6.5}$$

Rearranging yields:

$$\begin{aligned} &\pi_1^*(\mathrm{u})\left[\left(Q_2^1 - Q_4^1\right) + \left(Q_1^1 + Q_4^1 - Q_2^1 - Q_3^1\right)\pi_2^*(\mathrm{u})\right] \\ &\geqslant \pi_1(\mathrm{u})\left[\left(Q_2^1 - Q_4^1\right) + \left(Q_1^1 + Q_4^1 - Q_2^1 - Q_3^1\right)\pi_2^*(\mathrm{u})\right]. \end{aligned} \tag{6.6}$$

$$\begin{aligned} &\pi_2^*(\mathrm{u})\left[\left(Q_3^2 - Q_4^2\right) + \left(Q_1^2 + Q_4^2 - Q_2^2 - Q_3^2\right)\pi_1^*(\mathrm{u})\right] \\ &\geqslant \pi_2(\mathrm{u})\left[\left(Q_3^2 - Q_4^2\right) + \left(Q_1^2 + Q_4^2 - Q_2^2 - Q_3^2\right)\pi_1^*(\mathrm{u})\right]. \end{aligned} \tag{6.7}$$

Therefore, only when the strategy set $\{\pi_1^*, \pi_2^*\}$ simultaneously satisfies both inequalities is it termed a Nash equilibrium strategy. Evidently, multiple Nash equilibrium strategies may exist, potentially including mixed-strategy Nash equilibria and pure-strategy Nash equilibria. If $0 < \frac{Q_4^1 - Q_2^1}{Q_1^1 + Q_4^1 - Q_2^1 - Q_3^1} < 1$ and $0 < \frac{Q_4^2 - Q_3^2}{Q_1^2 + Q_4^2 - Q_2^2 - Q_3^2} < 1$ hold, and we set:

$$\pi_2^*(\mathrm{u}) = \frac{Q_4^1 - Q_2^1}{Q_1^1 + Q_4^1 - Q_2^1 - Q_3^1}, \quad \pi_2^*(\mathrm{l}) = 1 - \pi_2^*(\mathrm{u});$$

$$\pi_2^*(\mathrm{u}) = \frac{Q_4^2 - Q_3^2}{Q_1^2 + Q_4^2 - Q_2^2 - Q_3^2}, \quad \pi_1^*(\mathrm{r}) = 1 - \pi_1^*(\mathrm{u}). \tag{6.8}$$

then both inequalities are satisfied simultaneously, yielding a mixed-strategy Nash equilibrium. If $\pi_2^*(u) = 0$, consider the value of $Q_2^1 - Q_4^1$. If this value is greater than 0, set $\pi_1^*(u) = 1$ and substitute into the second inequality for verification. If both are satisfied, a pure-strategy Nash equilibrium is obtained; if not, attempt other possible solutions. In summary, for different values of Q, one or more Nash equilibrium strategies may exist. During actual experimentation, the agent must evaluate the current Q value to determine and execute the Nash equilibrium strategy. The subsequent section will present the specific experimental process and results analysis.

6.4 Experimental Results

Although the Nash-Q algorithm proves effective in simple scenarios, in complex scenarios with multiple Nash equilibria, the still exhibits issues such as step count fluctuations and deadlocks, consistent with the theoretical analysis in Section 5. To validate the practical effectiveness of the Nash equilibrium strategy, this paper conducts empirical verification. Both players adopt Nash equilibrium strategies. To prevent them from becoming trapped in local optima and ceasing exploration of other viable solutions, a greedy coefficient $\varepsilon = 0.1$ is set. The game undergoes 1000 training iterations, with observation of scores and moves per game.

Experiments revealed that when both players reach the positions depicted in Figure 6.1, their conflicting actions cause the game to enter an infinite loop. Consequently, if the game persists beyond 30 rounds without resolution, it is deemed a deadlock. A 'death' variable is introduced to record the frequency of such deadlocks.

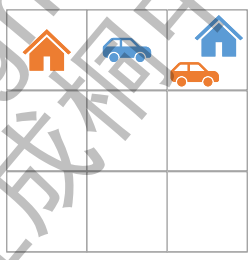

Figure 6.1: Deadlock Scenario

Upon program execution, if no collision occurs between the two agents, no points are deducted; each collision results in a 1-point deduction. The deduction pattern over the first 500 iterations is illustrated in Figure 6.2. As iterations increase, the frequency of point deductions markedly decreases, ultimately converging towards zero, demonstrating the algorithm's efficacy. Discontinuous points indicate deadlock states, whose number also significantly diminishes with increasing iterations.

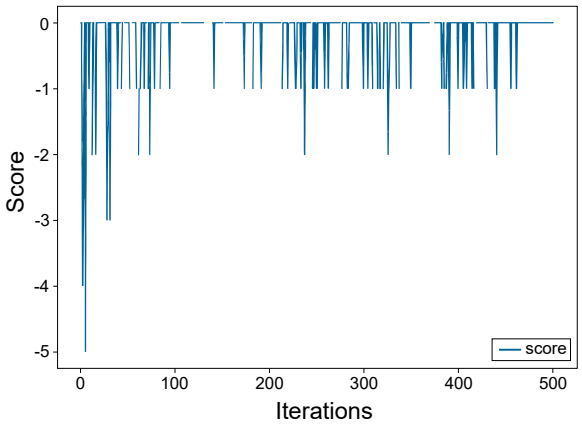

Figure 6.2: Score Diagram

To observe post-training outcomes, two agents adopting Nash equilibrium strategies underwent 1000 iterations of training. Subsequently, these trained agents were tested across 100 games to examine scoring performance and steps per game. The results are shown in Figure 6.3. It can be seen that the scores have converged, and the number of steps taken per game has stabilized within 10 steps. The algorithm is effective and performs well in this game. During the 1000 iterations, there were 96 deadlock situations, accounting for less than 10% of the total games. The agent's performance meets expectations.

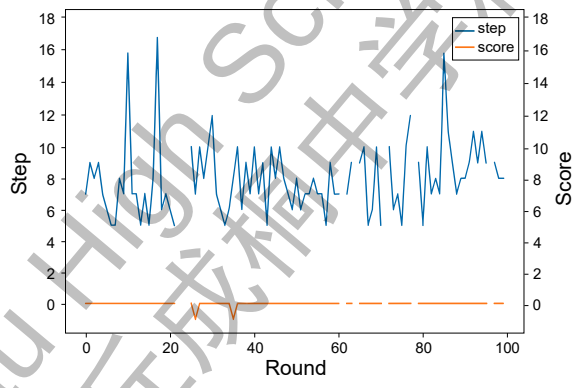

Figure 6.3: Agent Test Results

Although the number of steps has largely converged, occasional "peaks" still occur. Some games concluded after 17 steps, and isolated instances of score deductions were observed. The following analysis examines these outcomes.

Firstly, this stems from the game mechanics. Since collisions between agents yield a reward of $r = -1$, while discovering treasure grants $r = 1$, both affected by decay rates, agents tend to adopt conservative collision-avoidance strategies rather than aggressive treasure-seeking approaches. This explains why score convergence outperforms move-count convergence.

Secondly, to enhance game difficulty and demonstrate Nash equilibrium advantages, upward movement traps were introduced. This forces agents to balance avoiding upward actions with preventing lateral collision collisions, increasing randomness in per-game move counts. As the upward transition probability $P = 0.5$, move counts only exhibit approximate convergence without stabilizing at a fixed value. Attempting to remove this

constraint by setting the upward transition probability to $P = 1$ yields test results as shown in Figure 6.4, where the number of steps clearly converges.

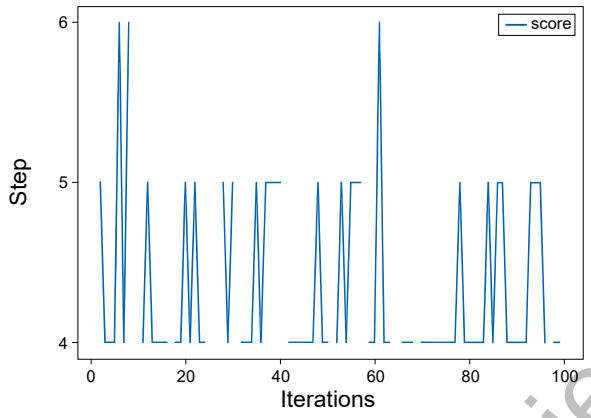

Figure 6.4: Step Count Test

Finally, inherent flaws within Nash equilibrium strategies may occasionally cause scoring deductions and step count fluctuations. As noted earlier, multiple Nash equilibrium strategies may exist under identical conditions. When both agents obtain Nash equilibrium strategies and multiple options are available, they will randomly select one to execute. Consequently, the strategy sets employed by each agent may differ, leading to mismatches. This situation not only leads to point deductions and high move counts but also increases the deadlock rate. For instance, when the game reaches the scenario depicted in Figure 6.5, two Nash equilibrium pure strategy sets may exist: $\pi = \{0, 1\}, \{1, 0\}$. This occurs when Player 1 chooses to move right while Player 2 moves up, or when Player 2 chooses to move left while Player 1 moves up. Should both players select different Nash equilibrium pure strategy sets, this may result in reduced reward values or a deadlock scenario. For instance, if Player One adopts the first Nash equilibrium strategy set while Player Two adopts the second, both agents will move upwards. If this movement succeeds, the game progresses to the state depicted in Figure 6.5, where neither agent can reach their respective treasure locations, thus entering a deadlock. Conversely, should Player One select the second Nash equilibrium strategy while Player Two chooses the first, the two agents will collide, incurring unnecessary penalty points. This demonstrates the critical importance of facilitating information exchange between the two agents to ensure they select the same Nash equilibrium strategy set.

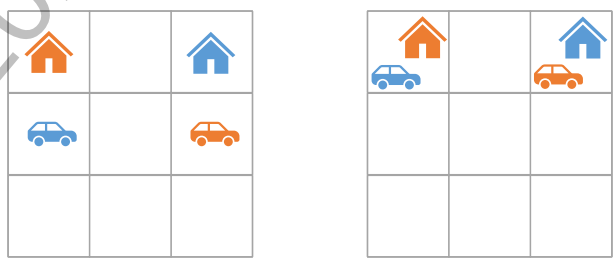

Figure 6.5: Analysis of Game Scenario

To address this mismatch in strategy selection, a penalty mechanism has been introduced. In addition to the reward function used during value function updates, a penalty

function is applied during Nash equilibrium strategy group selection. Should two agents select the same Nash equilibrium strategy group (where the target Nash equilibrium is set according to the principle of maximizing the sum of value functions), then $r^{'} = 0$. Otherwise, $r^{'} = -1$. The operational results following the introduction of this heuristic reward mechanism are illustrated in Figure 6.6.

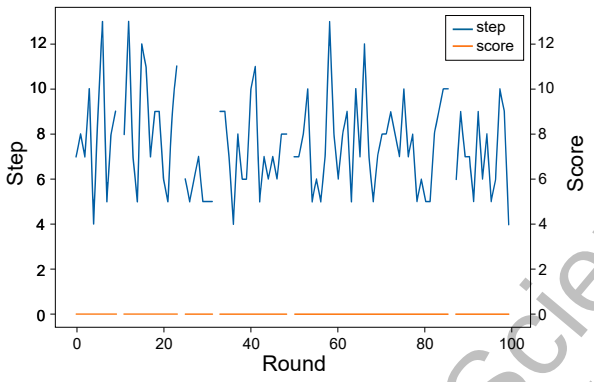

Figure 6.6: Test Results with Heuristic Reward Introduction

It is evident that the algorithm significantly outperforms that in Figure 6.3. Scoring has fully converged without deductions, with smaller fluctuations in steps per game and no instances exceeding 13 steps. Moreover, across 1000 game iterations, death = 44, representing a reduction of over half in deadlock rates compared to the unmodified algorithm. This marked improvement demonstrates the effectiveness of the reward mechanism.

In real-world parking scenarios, deadlock signifies mutual obstruction between vehicles. The average move count represents the time cost incurred by both drivers. This experiment demonstrates that even with a relatively low penalty function value $F\left(s, a^1, a^2, s'\right)$, the probability of deadlock can still be halved. Furthermore, the average time spent by both drivers is reduced $20\% - 30\%$, thereby enhancing the parking experience for both parties. Although unproven, one might conceive that setting a larger penalty function value $F\left(s, a^1, a^2, s'\right)$ would reduce the deadlock probability to zero, with the number of steps converging to four in the later learning phase. This would strictly enforce both parties to follow the target Nash equilibrium path, yet partially contradicts the original intent of allowing agents to learn and explore autonomously.

In summary, this chapter's parking game experiments introduced multi-agent reinforcement learning based on Nash equilibrium strategies into a two-player randomized game, achieving the anticipated results that validate the effectiveness of Nash equilibrium strategies. Furthermore, a Nash equilibrium strategy incorporating a penalty mechanism was proposed. Compared to the original results, this approach demonstrated improved convergence in the players' total number of steps, proving the rationality of introducing a reward mechanism and verifying the theoretical component.

7 Summary and Outlook

7.1 Innovations, Contributions, and Limitations of This Research

This research addresses the instability and slow convergence of the Nash-Q algorithm when confronting multiple Nash equilibria within a multi-agent reinforcement learning Markov game framework. It proposes a generalized Markov game framework. This framework incorporates additional steps for computing the target Nash equilibrium and penalizing the value function (where the equilibrium itself serves as the target Nash equilibrium if only one exists). It demonstrates that penalizing the value function does not alter the convergence properties of the original Nash-Q algorithm's value function and policy. It is conceivable that even when faced with multiple Nash equilibria where the Nash-Q algorithm fails to converge, the penalty function can incentivize agents to preferentially select the designated target Nash equilibrium. Although unproven, in such scenarios, provided all agents employ the Nash-Q algorithm with penalty mechanisms and appropriately calibrated penalty parameters, convergence to the target Nash equilibrium is assured.

To validate the algorithm's practical efficacy, this paper designed a two-player parking simulation experiment. Experimental results demonstrate that in complex multi-Nash equilibrium scenarios, while the original Nash-Q algorithm achieves respectable outcomes, performance improves significantly when agents employ the improved algorithm with penalty mechanisms. Specifically, traffic congestion probability for both parties decreases by over half, with time costs reduced by $20 - 30\%$. This effectively resolves strategy oscillation issues, rendering agent behavior more coordinated and efficient.

Limitations of this work: For this scenario, the Nash-Q algorithm requires maintaining two Q tables to track value function changes for both agents and opponents. The number of entries to manage is $|S| \times |A^1| \times |A^2|$. Additionally, calculating Nash equilibria based on stage-based games imposes computational demands far exceeding those of single-agent reinforcement learning Q-learning. Generalized Markov games further require calculating the target Nash equilibrium from multiple Nash equilibria based on specified conditions, whilst establishing entries to record all possible transitions to the next state using the target Nash equilibrium within this phase, incurring penalties if the next state is not within these entries. Consequently, this algorithm demands substantial computational power and proves difficult to generalize to n agents, where both space and computational complexity grow exponentially with the number of agents.

7.2 Outlook

Future research should explore the following two directions:

1. The selection of the target Nash equilibrium is customizable based on scenario and requirements. For instance, in a parking game, to prevent either party's parking experience from being significantly worsened, the target Nash equilibrium could be

set as:

$$\pi^{ig} = \arg\max_{\pi} \min \left| Q^{i}_{\pi^{1j}} \left(s, a^1, s^2 \right) \right| \tag{7.1}$$

If maximizing the combined experience of both parties is desired, the target Nash equilibrium could be set as:

$$\pi^{ig} = \arg\max_{\pi} \left[Q^{1}_{\pi^{1j}} \left(s, a^1, a^2 \right) + Q^{2}_{\pi^{2j}} \left(s, a^1, a^2 \right) \right] \tag{7.2}$$

However, such target Nash equilibrium settings rely on subjective judgments without quantified theoretical criteria. Subsequent research should therefore engage in more rigorous discussion, ideally developing adaptive configuration methods for different game scenarios.

2. Similarly, the parameterization of the penalty function is also subjective. It is hoped that, upon obtaining the Q tables for both parties, penalty values can be adaptively determined for this specific game. When single-agent reinforcement learning encounters multi-agent scenarios, the Q-learning algorithm treats other agents as part of the environment. While computationally efficient, this approach clearly fails to meet the strategic demands of complex situations, leading to the emergence of multi-agent reinforcement learning within Markov games. The Nash-Q algorithm within multi-agent Markov games records other players' information and actions while seeking Nash equilibrium solutions at each stage. However, the presence of multiple Nash equilibria introduces uncertainty into agents' choices, diminishing both convergence speed and accuracy. The generalized Markov game incorporating penalty mechanisms addresses this bottleneck, though it introduces new challenges in computational demands and processing speed. Subjectivity remains unavoidable in both target Nash equilibrium selection and penalty function design. Nevertheless, this algorithm delivers stable strategies for more complex games while enhancing convergence speed.

References

[1] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior: 60th Anniversary Commemorative Edition*, 60th ed. Princeton: Princeton University Press, 2007, with introduction by Harold W. Kuhn and afterword by Ariel Rubinstein.

[2] B. V. Bowden, *Faster than thought*. Pitman, 1953.

[3] J. Bronowski, *The Ascent of Man*. BBC Books, 2011, originally published 1973.

[4] M. Campbell, A. J. Hoane, and F.-h. Hsu, "Deep blue," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[5] J. Van der Wal, "Discounted markov games: Generalized policy iteration method," *Journal of Optimization Theory and Applications*, vol. 25, no. 1, pp. 125–138, 1978.

[6] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine learning proceedings 1994*. Elsevier, 1994, pp. 157–163.

[7] J. Hu and M. P. Wellman, "Nash q-learning for general-sum stochastic games," *Journal of Machine Learning Research*, vol. 4, pp. 1039–1069, 2003, submitted 11/01; Revised 10/02; Published 11/03.

[8] J. Hu, M. P. Wellman *et al.*, "Multiagent reinforcement learning: theoretical framework and an algorithm." in *ICML*, vol. 98, 1998, pp. 242–250.

[9] J. Schaeffer and J. van den Herik, "Games, computers, and artificial intelligence," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 1–7, 2002.

[10] H. J. Berliner, "Backgammon computer program beats world champion," *Artificial Intelligence*, vol. 14, pp. 205–220, 1980, reprinted in: D.N.L. Levy (ed.), Computer Games I, Springer-Verlag, 1988.

[11] D. Fudenberg and J. Tirole, *Game Theory*. Cambridge, MA: MIT Press, 1991.

[12] F. Thuijsman, "Optimality and equilibria in stochastic games," Ph.D. dissertation, Maastricht University, jan 1989.

[13] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the Tenth International Conference on Machine Learning*, P. E. Utgoff, Ed. Amherst, MA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 330–337.

[14] M. A. Wiering *et al.*, "Multi-agent reinforcement learning for traffic light control," in *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*, 2000, pp. 1151–1158.

[15] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016.

[16] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[17] L. Qing, L. Zhijun, and L. Tiansheng, "Multi-agent reinforcement learning in complex environments," *Journal of Shanghai Jiaotong University*, vol. 36, no. 3, pp. 302–305, 2002.

[18] G. Gu, Y. Zhong, and R. Zhang, "A new multi-agent reinforcement learning algorithm and its application to multi-robot cooperation tasks," *Robot*, vol. 25, no. 4, pp. 344–348, 2003, in Chinese.

[19] H. Hu, "Research and implementation on multi-agent reinforcement learning," Master's Thesis, Beijing University of Posts and Telecommunications, 2021, in Chinese.

[20] H. Wang, "Research on train dispatching method based on multi-agent reinforcement learning," Master's Thesis, Heilongjiang University, 2021, in Chinese.

[21] Q. Sun, "Research of multi-agent cooperation mechanism based on reinforcement learning," Master's Thesis, Zhejiang University of Technology, 2015, in Chinese.

[22] X. Bai, "Research and application of reinforcement learning in multi-agent collaboration," Master's Thesis, University of Electronic Science and Technology of China, 2020, in Chinese.

[23] P. Ye, X. Jia, X. Yang, and C. Niu, "Multi-agent reinforcement learning for edge-cloud collaborative offloading in vehicular networks," *Computer Engineering*, vol. 47, no. 4, pp. 13–20, 2021, in Chinese.

[24] R. Zhang, G. Gu, Z. Liu, and X. Wang, "Reinforcement learning theory, algorithms and its application," *Control Theory and Applications*, vol. 17, no. 5, pp. 637–642, 2000, in Chinese.

[25] Z.-H. Zhao, Y. Gao, B. Luo, and S.-F. Chen, "Reinforcement learning technology in multi-agent system," *Computer Science*, vol. 31, no. 3, 2004, in Chinese.

[26] L. Quan, Z. Jianwei, Z. Zongchang, Z. Shan, Z. Qian, Z. Peng, and X. Jin, "A survey of deep reinforcement learning," *Chinese Journal of Computers*, vol. 41, no. 1, pp. 1–27, 2018.

[27] W. Yang, L. Zhang, and F. Zhu, "Multi-agent reinforcement learning based traffic signal control for integrated urban network: survey of state of art," *Application Research of Computers*, vol. 35, no. 6, pp. 1613–1618, 2018, in Chinese.

[28] M. L. Puterman, "Markov decision processes," *Handbooks in operations research and management science*, vol. 2, pp. 331–434, 1990.

[29] D. Fudenberg and D. K. Levine, *The Theory of Learning in Games*, ser. Economic Learning and Social Evolution. The MIT Press, 1998, paperback edition.

[30] D. F. Labaree, *Someone Has to Fail: The Zero-Sum Game of Public Schooling.* Harvard University Press, April 2012, first edition.

[31] P. Murphy, "The limits of symmetry: A game theory approach to symmetric and asymmetric public relations," *Public Relations Research Annual*, vol. 3, no. 1-4, pp. 115–131, 1991.

[32] R. W. Cottle, J.-S. Pang, and R. E. Stone, *The Linear Complementarity Problem.* Society for Industrial and Applied Mathematics, 2009.

Acknowledgments

My first encounter with strategic thinking came from mathematics and informatics competitions in junior high school. Problems about "winning strategies" in math contests and greedy algorithms in informatics training showed me how local choices can shape global outcomes. These experiences planted the seed for my later study of game theory.

In the second semester of Grade 10, inspired by these early experiences, I decided to explore this field more systematically. It was the classic scenarios in game theory—such as the individual rationality leading to collective loss in the Prisoner's Dilemma, the coordination challenge in the Battle of the Sexes, and the free-rider problem in the Boxed Pigs Game—that deepened my fascination with strategic decision-making. Coincidentally, I observed a real-world instance of such interactions in my own community: multiple drivers competing for limited parking spaces, resulting in wasted time and widespread dissatisfaction. This everyday scene mirrored those classic models and solidified my interest in studying multi-agent decision-making systematically.

Through extended reading, I became fascinated by the intersection of game theory and multi-agent systems. A course project on reinforcement learning further revealed the limitations of classical algorithms in multi-agent settings, where individual optimization often leads to collective inefficiency. This gap between single-agent methods and multi-agent realities sparked the idea of using game-theoretic equilibria to improve coordination. I became convinced that this was not merely contest problem-solving but meaningful scientific research, and I gradually narrowed my focus to multi-agent reinforcement learning—which eventually evolved into the topic of this paper.

During the Ross Mathematics Program, I attended several lectures on mathematics and game theory given by Professor Jim Fowler and counselor Blaze Okonogi-Neth. In private conversations, both were generous with their time and encouraged me to continue my investigation, offering valuable perspectives at the early stage. They also helped me brainstorm directions when my initial plan was too broad. For example, Professor Fowler pointed out that my original design might be too large to carry out and suggested that I narrow it down. This advice was important when I restructured my research and focused on the Nash-Q penalty mechanism.

I am especially grateful to Ms. Lingyun Chang, my high-school mathematics teacher. Her guidance was essential at key moments. She discussed topic selection with me, helping me think about scope and feasibility. She suggested testing my ideas through computer simulation and gave insights that strengthened the convergence analysis. In the process, she helped me identify the Nash-Q penalty as a useful framework, but I built the environment, set up the model, proved the results, and connected the pieces myself. She also read my drafts, gave comments on how to highlight the main contributions, and encouraged me to refine the analysis in later sections. She advised me on how to explain why my adjustments from the original, overly ambitious idea still kept the project meaningful. These suggestions helped me present the work more clearly and to academic standards. Her steady encouragement was a great source of motivation throughout this journey.

The source of the topic came from my own daily observation, while most parts of

this paper—including the idea, theory development, algorithm design, simulations, and writing—were completed independently. The only exception was that my teacher guided me in recognizing the Nash-Q penalty module, though its refinement and final use were completed independently. During the course of this research, I ran into some tough challenges. My first plan was to solve the convergence issue of Nash-Q in a very general Markov game framework, but the scope was too broad to complete. Under Ms. Chang's advice, I scaled down the project but kept the key theoretical contributions. At times, I also struggled to connect the theory with applications. Ms. Chang suggested that the "wasteful parking" example could serve as a good scenario, and this made my abstract ideas easier to understand.

I thank all teachers and friends who helped me along the way. The following list is complete to the best of my knowledge and is ordered alphabetically by last name:

Lingyun Chang, Jim Fowler, Yisai Gao, Caiying Huang, Blaze Okonogi-Neth, Ying Shang, Wenrui Xie.

I apologize for any unintentional omissions.

The guidance and encouragement I received from everyone named above were offered freely and without any form of compensation.

Finally, I am grateful to the S.-T. Yau High School Science Award for providing a platform for this work, and to everyone who supported me during its preparation.