Multi-Agent Reinforcement Learning in Games

by

Xiaosong Lu, **M.A.Sc.**

A thesis submitted to

the Faculty of Graduate and Postdoctoral Affairs

in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario

March, 2012

The undersigned recommend to

the Faculty of Graduate and Postdoctoral Affairs

acceptance of the thesis

Multi-Agent Reinforcement Learning in Games

Submitted by **Xiaosong Lu**, **M.A.Sc.**

In partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Professor Howard M. Schwartz, Thesis Supervisor

Professor Abdelhamid Tayebi, External Examiner

Professor Howard M. Schwartz, Chair,
Department of Systems and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

March, 2012

Abstract

Agents in a multi-agent system observe the environment and take actions based on their strategies. Without prior knowledge of the environment, agents need to learn to act using learning techniques. Reinforcement learning can be used for agents to learn their desired strategies by interaction with the environment. This thesis focuses on the study of multi-agent reinforcement learning in games. In this thesis, we investigate how reinforcement learning algorithms can be applied to different types of games.

We provide four main contributions in this thesis. First, we convert Isaacs' guarding a territory game to a gird game of guarding a territory under the framework of stochastic games. We apply two reinforcement learning algorithms to the grid game and compare them through simulation results. Second, we design a decentralized learning algorithm called the L_{R-I} lagging anchor algorithm and prove the convergence of this algorithm to Nash equilibria in two-player two-action general-sum matrix games. We then provide empirical results of this algorithm to more general stochastic games. Third, we apply the potential-based shaping method to multi-player general-sum stochastic games and prove the policy invariance under reward transformations in general-sum stochastic games. Fourth, we apply fuzzy reinforcement learning to Isaacs' differential game of guarding a territory. A potential-base shaping function is introduced to help the defenders improve the learning performance in both the two-player and the three-player differential game of guarding a territory.

Acknowledgments

First, I would like to thank my advisor professor Howard Schwartz. He is not only my mentor and teacher, but also my guide, encourager and friend. He is the reason why I came to Carleton to pursue my study seven years ago.

I would also like to thank Professor Sidney Givigi for his guide on running robotic experiments and suggestions on publications. A special thanks to my committee members for their valuable suggestions on the thesis.

Finally, I would like to thank my wife Ying. Without her enormous support and encouragement, I could not finish my thesis successfully.

Table of Contents

List of Tables

List of Figures

List of Acronyms

Acronyms	Definition
DP	Dynamic programming
FACL	fuzzy actor-critic learning
FFQ	friend-or-foe Q-learning
FIS	fuzzy inference system
FQL	fuzzy Q-learning
L_{R-I}	linear reward-inaction
L_{R-P}	linear reward-penalty
MARL	multi-agent reinforcement learning
MDP	Markov decision process
MF	membership function
NE	Nash equilibrium
ODE	ordinary differential equation
PHC	policy hill-climbing

RL	Reinforcement learning
SG	stochastic game
TD	Temporal-difference
TS	Takagi-Sugeno
WoLF-IGA	"Win or Learn Fast" infinitesimal gradient ascent
WoLF-PHC	"Win or Learn Fast" policy hill-climbing

List of Symbols

Symbols	Definition
a_t	action at t
α	learning rate
A	action space
δ_t	temporal-difference error at t
$dist$	Manhattan distance
η	step size
F	shaping reward function
γ	discount factor
i	player i in a game
j	player's jth action
M	a stochastic game
M'	a transformed stochastic game with reward shaping
N	an MDP

N'	a transformed MDP with reward shaping
$P(\cdot)$	payoff function
$\Phi(s)$	shaping potential
π	policy
π^*	optimal policy
$Q^\pi(s,a)$	action-value function under policy π
$Q^*(s,a)$	action-value function under optimal policy
r_t	immediate reward at t
R	reward function
s_t	state at t
s_T	terminal state
S	state space
t	discrete time step
t_f	terminal time
Tr	transition function
$V^\pi(s)$	state-value function under policy π
$V^*(s)$	state-value function under optimal policy
ε	greedy parameter

Chapter 1

Introduction

A multi-agent system consists of a number of intelligent agents that interact with other agents in a multi-agent environment [1–3]. An agent is an autonomous entity that observes the environment and takes an action to satisfy its own objective based on its knowledge. The agents in a multi-agent system can be software agents or physical agents such as robots [4]. Unlike a stationary single-agent environment, the multi-agent environment can be complex and dynamic. The agents in a multi-agent environment may not have a priori knowledge of the correct actions or the desired policies to achieve their goals.

In a multi-agent environment, each agent may have independent goals. The agents need to learn to take actions based on their interaction with other agents. Learning is the essential way of obtaining the desired behavior for an agent in a dynamic environment. Different from supervised learning, there is no external supervisor to guide the agent's learning process. The agents have to acquire the knowledge of their desired actions themselves by interacting with the environment.

Reinforcement learning (RL) can be used for an agent to discover the good actions through interaction with the environment. In a reinforcement learning problem, rewards are given to the agent for the selection of good actions. Reinforcement learning has been studied extensively in a single-agent environment [5]. Recent studies have

extended reinforcement learning from the single-agent environment to the multi-agent environment [6]. In this dissertation, we focus on the study of multi-agent reinforcement learning (MARL) in different types of games.

1.1 Motivation

The motivation of this dissertation starts from Isaacs' differential game of guarding a territory. This game is played by a defender and an invader in a continuous domain. The defender tries to intercept the invader before it enters the territory. Differential games can be studied under a discrete domain by discretizing the state space and the players' action space. One type of discretization is to map the differential game into a grid world. Examples of grid games can be found in the predator-prey game [7] and the soccer game [8]. These grid games have been studied as reinforcement learning problems in [8–10]. Therefore, our first motivation is to study Isaacs' guarding a territory game as a reinforcement learning problem in a discrete domain. We want to create a grid game of guarding a territory as a test bed for reinforcement learning algorithms.

Agents in a multi-agent environment may have independent goals and do not share the information with other agents. Each agent has to learn to act on its own based on its observation and received information from the environment. Therefore, we want to find a decentralized reinforcement learning algorithm that can help agents learn their desired strategies. The proposed decentralized reinforcement learning algorithm needs to have the convergence property, which can guarantee the convergence to the agent's equilibrium strategy.

Based on the characteristics of the game of guarding a territory, the reward is only received when the game reaches the terminal states where the defender intercepts the invader or the invader enters the territory. No immediate rewards are given to

the players until the end of the game. This problem is called the temporal credit assignment problem where a reward is received after a sequence of actions. Another example of this problem can be found in the soccer game where the reward is only received after a goal is scored. If the game includes a large number of states, the delayed rewards will slow down the player's learning process and even cause the player to fail to learn its equilibrium strategy. Therefore, our third motivation is to design artificial rewards as supplements to the delayed rewards to speed up the player's learning process.

Reinforcement learning can also be applied to differential games. In [11–13], fuzzy reinforcement learning has been applied to the pursuit-evasion differential game. In [12], experimental results showed that the pursuer successfully learned to capture the invader. For Isaacs' differential game of guarding a territory, there is a lack of investigation on how the players can learn their equilibrium strategies by playing the game. We want to investigate how reinforcement learning algorithms can be applied to Isaacs's differential game of guarding a territory.

1.2 Contributions and Publications

The main contributions of this thesis are:

1. We map Isaacs' guarding a territory game into a grid world and create a grid game of guarding a territory. As a reinforcement learning problem, the game is investigated under the framework of stochastic games (SGs). We apply two reinforcement learning algorithms to the grid game of guarding a territory. The performance of the two reinforcement learning algorithms is illustrated through simulation results.

2. We introduce a decentralized learning algorithm called the L_{R-I} lagging anchor

algorithm. We prove that the L_{R-I} lagging anchor algorithm can guarantee the convergence to Nash equilibria in two-player two-action general-sum matrix games. We also extend the algorithm to a practical L_{R-I} lagging anchor algorithm for stochastic games. Three examples of matrix games and Hu and Wellman's [14] grid game are simulated to show the convergence of the proposed L_{R-I} lagging anchor algorithm and the practical L_{R-I} lagging anchor algorithm.

3. We apply the potential-based shaping method to multi-player general-sum stochastic games. We prove that the integration of the potential-based shaping reward into the original reward function does not change the Nash equilibria in multi-player general-sum stochastic games. The modified Hu and Wellman's grid game and the grid game of guarding a territory with two defenders and one invader are simulated to test the players' learning performance with different shaping rewards.

4. We apply fuzzy reinforcement learning algorithms to Isaacs' differential game of guarding a territory. A potential-base shaping function is introduced to solve the temporal credit assignment problem caused by the delayed reward. We then extend the game to a three-player differential game by adding one more defender to the game. Simulation results are provided to show how the designed potential-base shaping function can help the defenders improve their learning performance in both the two-player and the three-player differential game of guarding a territory.

The related publications are listed as follows:

1. X. Lu and H. M. Schwartz, "Decentralized Learning in General-Sum Matrix Games: An L_{R-I} Lagging Anchor Algorithm," *International Journal of Innovative Computing, Information and Control*, vol. 8, 2012. to be published.

2. X. Lu, H. M. Schwartz, and S. N. Givigi, "Policy invariance under reward transformations for general-sum stochastic games," *Journal of Artificial Intelligence Research*, vol. 41, pp. 397-406, 2011.

3. X. Lu and H. M. Schwartz, "Decentralized learning in two-player zero-sum games: A LR-I lagging anchor algorithm," in *American Control Conference (ACC), 2011*, (San Francisco, CA), pp. 107-112, 2011

4. X. Lu and H. M. Schwartz, "An investigation of guarding a territory problem in a grid world," in *American Control Conference (ACC), 2010*, (Baltimore, MD), pp. 3204-3210, Jun. 2010.

5. S. N. Givigi, H. M. Schwartz, and X. Lu, "A reinforcement learning adaptive fuzzy controller for differential games," *Journal of Intelligent and Robotic Systems*, vol. 59, pp. 3-30, 2010.

6. S. N. Givigi, H. M. Schwartz, and X. Lu, "An experimental adaptive fuzzy controller for differential games," in *Proc. IEEE Systems, Man and Cybernetics'09*, (San Antonio, United States), Oct. 2009.

1.3 Organization of the Thesis

The outline of this thesis is as follows:

Chapter 2-A Framework for Reinforcement Learning. Under the framework of reinforcement learning, we review Markov decision processes (MDPs), matrix games and stochastic games. This chapter provides the fundamental background for the work in the subsequent chapters.

Chapter 3-Reinforcement Learning in Stochastic Games. We present and compare four multi-agent reinforcement learning algorithms in stochastic games.

Then we introduce a grid game of guarding a territory as a two-player zero-sum stochastic game (SG). We apply two multi-agent reinforcement learning algorithms to the game.

Chapter 4-Decentralized Learning in Matrix Games. We present and compare four existing learning algorithms for matrix games. We propose an L_{R-I} lagging anchor algorithm as a completely decentralized learning algorithm. We prove the convergence of the L_{R-I} lagging anchor algorithm to Nash equilibria in two-player two-action general-sum matrix games. Simulations are provided to show the convergence of the proposed L_{R-I} lagging anchor algorithm in three matrix games and the practical L_{R-I} lagging anchor algorithm in a general-sum stochastic game.

Chapter 5-Potential-Based Shaping in Stochastic Games. We present the application of the potential-based shaping method to general-sum stochastic games. We prove the policy invariance under reward transformations in general-sum stochastic games. Potential-based shaping rewards are applied to two grid games to show how shaping rewards can affect the players' learning performance.

Chapter 6-Reinforcement Learning in Differential Games. We present the application of fuzzy reinforcement learning to the differential game of guarding a territory. Fuzzy Q-learning (FQL) and fuzzy actor-critic learning (FACL) algorithms are presented in this chapter. To compensate for the delayed rewards during learning, shaping functions are designed to increase the speed of the player's learning process. In this chapter, we first apply FQL and FACL algorithms to the two-player differential game of guarding a territory. We then extend the game to a three-player differential game of guarding a territory with two defenders and one invader. Simulation results are provided to show the overall performance of the defenders in both the two-player differential game of

guarding a territory game and the three-player differential game of guarding a territory game.

Chapter 7-Conclusion. We conclude this thesis by reviewing the main contributions along with new future research directions for multi-agent reinforcement learning in games.

Chapter 2

A Framework for Reinforcement Learning

2.1 Introduction

Reinforcement learning is learning to map situations to actions so as to maximize a numerical reward [5, 15]. Without knowing which actions to take, the learner must discover which actions yield the most reward by trying them. Actions may affect not only the immediate reward but also the next situation and all subsequent rewards [5]. Different from supervised learning, which is learning from examples provided by a knowledgable external supervisor, reinforcement learning is adequate for learning from interaction [5]. Since it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations, the learner must be able to learn from its own experience [5]. Therefore, the reinforcement learning problem is a problem of learning from interaction to achieve a goal.

The learner is called the agent or the player and the outside which the agent interacts with is called the environment. The agent chooses actions to maximize the rewards presented by the environment. Suppose we have a sequence of discrete time steps $t = 0, 1, 2, 3, \cdots$. At each time step t, the agent observes the current state s_t from the environment. We define a_t as the action the agent takes at t. At the next time step, as a consequence of its action a_t, the agent receives a numerical reward

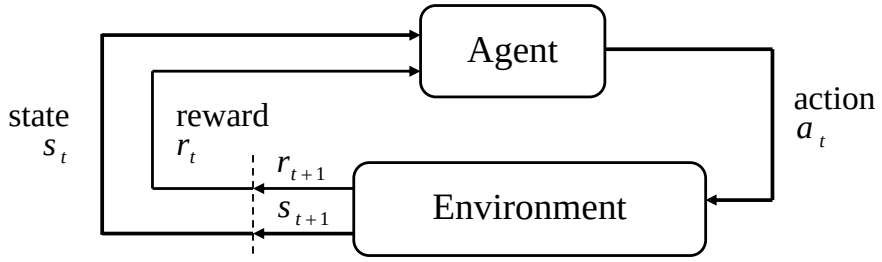


Figure 2.1: The agent-environment interaction in reinforcement learning

$r_{t+1} \in \Re$ and moves to a new state s_{t+1} as shown in Fig. 2.1. At each time step, the agent implements a mapping from states to probabilities of selecting each possible action [5]. This mapping is called the agent's policy and is denoted as $\pi_t(s, a)$ which is the probability of taking action a at the current state s. Reinforcement learning methods specify how the agent can learn its policy to maximize the total amount of reward it receives over the long run [5].

A reinforcement learning problem can be studied under the framework of stochastic games [10]. The framework of stochastic games contains two simpler frameworks: Markov decision processes and matrix games [10]. Markov decision processes involve a single agent and multiple states, while matrix games include multiple agents and a single state. Combining Markov decision processes and matrix games, stochastic games are considered as reinforcement learning problems with multiple agents and multiple states.

In the following sections, we present Markov decision processes in Section 2.2, matrix games in Section 2.3 and stochastic games in Section 2.4. Examples are provided for different types of games under the framework of stochastic games.

2.2 Markov Decision Processes

A Markov decision process (MDP) [16] is a tuple (S, A, Tr, γ, R) where S is the state space, A is the action space, $Tr : S \times A \times S \to [0, 1]$ is the transition function, $\gamma \in [0, 1]$ is the discount factor and $R : S \times A \times S \to \mathbb{R}$ is the reward function. The transition function denotes a probability distribution over next states given the current state and action such that

$$\sum_{s' \in S} Tr(s, a, s') = 1 \quad \forall s \in S, \ \forall a \in A \tag{2.1}$$

where s' represents a possible state at the next time step. The reward function denotes the received reward at the next state given the current action and the current state. A Markov decision process has the following Markov property: the conditional probability distribution of the player's next state and reward only depends on the player's current state and action such that

$$Pr\left\{ s_{t+1} = s', r_{t+1} = r' \,\middle|\, s_t, a_t, \ldots, s_0, a_0 \right\} = Pr\left\{ s_{t+1} = s', r_{t+1} = r' \,\middle|\, s_t, a_t \right\}. \tag{2.2}$$

A player's policy $\pi : S \to A$ is defined as a probability distribution over the player's actions from a given state. A player's policy $\pi(s, a)$ satisfies

$$\sum_{a \in A} \pi(s, a) = 1 \quad \forall s \in S. \tag{2.3}$$

For any MDP, there exists a deterministic optimal policy for the player, where $\pi^*(s, a) \in \{0, 1\}$ [17]. The goal of a player in an MDP is to maximize the expected long-term reward. In order to evaluate a player's policy, we have the following concept of the state-value function. The value of a state s (or the state-value function) under a policy π is defined as the expected return when the player starts at state s

and follows a policy π thereafter. Then the state-value function $V^{\pi}(s)$ becomes

$$V^{\pi}(s) = E_{\pi}\left\{ \sum_{k=0}^{t_f-t-1} \gamma^k r_{k+t+1} \,\middle|\, s_t = s \right\} \tag{2.4}$$

where t_f is a final time step, t is the current time step, r_{k+t+1} is the received immediate reward at the time step $k+t+1$, $\gamma \in [0,1]$ is a discount factor. In (2.4), we have $t_f \to \infty$ if the task is an infinite-horizon task such that the task will run over infinite period. If the task is episodic, t_f is defined as the terminal time when each episode is terminated at the time step t_f. Then we call the state where each episode ends as the terminal state s_T. In a terminal state, the state-value function is always zero such that $V(s_T) = 0 \; \forall s_T \in S$. An optimal policy π^* will maximize the player's discounted future reward for all states such that

$$V^*(s) \geq V^{\pi}(s) \;\; \forall \pi, \forall s \in S \tag{2.5}$$

The state-value function under a policy in (2.4) can be rewritten as follows

$$\begin{aligned}
V^{\pi}(s) &= E_{\pi}\left\{ \sum_{k=0}^{t_f} \gamma^k r_{k+t+1} \,\middle|\, s_t = s \right\} \\
&= \sum_{a \in A} \pi(s,a) \sum_{s' \in S} Tr(s,a,s') E_{\pi}\Bigg\{ r_{t+1} + \\
&\qquad\qquad \gamma \sum_{k=0}^{t_f} \gamma^k r_{k+t+2} \,\Big|\, s_t = s, a_t = a, s_{t+1} = s' \Bigg\} \\
&= \sum_{a \in A} \pi(s,a) \sum_{s' \in S} Tr(s,a,s') E_{\pi}\left\{ r_{t+1} \,\middle|\, s_t = s, a_t = a, s_{t+1} = s' \right\} + \\
&\quad \sum_{a \in A} \pi(s,a) \sum_{s' \in S} Tr(s,a,s') E_{\pi}\left\{ \gamma \sum_{k=0}^{t_f} \gamma^k r_{k+t+2} \,\middle|\, s_t = s, a_t = a, s_{t+1} = s' \right\}
\end{aligned} \tag{2.6}$$

where $Tr(s,a,s') = Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$ is the probability of the next state being $s_{t+1} = s'$ given the current state $s_t = s$ and action $a_t = a$ at time step t. Based on the Markov property given in (2.2), we get

$$E_\pi\left\{\gamma \sum_{k=0}^{t_f} \gamma^k r_{k+t+2} \,\middle|\, s_t = s, a_t = a, s_{t+1} = s'\right\} = E_\pi\left\{\gamma \sum_{k=0}^{t_f} \gamma^k r_{k+t+2} \,\middle|\, s_{t+1} = s'\right\}$$

Then equation (2.6) becomes

$$\begin{aligned} V^\pi(s) &= \sum_{a\in A} \pi(s,a) \sum_{s'\in S} Tr(s,a,s') E_\pi\left\{r_{t+1} \,\middle|\, s_t = s, a_t = a, s_{t+1} = s'\right\} + \\ &\qquad \sum_{a\in A} \pi(s,a) \sum_{s'\in S} Tr(s,a,s') E_\pi\left\{\gamma \sum_{k=0}^{t_f} \gamma^k r_{k+t+2} \,\middle|\, s_{t+1} = s'\right\} \\ &= \sum_{a\in A} \pi(s,a) \sum_{s'\in S} Tr(s,a,s')\big(R(s,a,s') + \gamma V^\pi(s')\big) \end{aligned} \tag{2.7}$$

where $R(s,a,s') = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$ is the expected immediate reward received at state s' given the current state s and action a. The above equation (2.7) is called the Bellman equation [18]. If the player starts at state s and follows the optimal policy π^* thereafter, we have the optimal state-value function denoted by $V^*(s)$. The optimal state-value function $V^*(s)$ is also called the Bellman optimality equation where

$$V^*(s) = \max_{a\in A} \sum_{s'\in S} Tr(s,a,s')\big(R(s,a,s') + \gamma V^*(s')\big). \tag{2.8}$$

We can also define the action-value function as the expected return of choosing a particular action a at state s and following a policy π thereafter. The action-value function $Q^\pi(s,a)$ is given as

$$Q^\pi(s,a) = \sum_{s'\in S} Tr(s,a,s')\big(R(s,a,s') + \gamma V^\pi(s')\big) \tag{2.9}$$

Then the state-value function becomes

$$V(s) = \max_{a \in A} \sum_{s' \in S} Q^{\pi}(s, a). \tag{2.10}$$

If the player chooses action a at state s and follows the optimal policy π^* thereafter, the action-value function becomes the optimal action-value function $Q^*(s, a)$ where

$$Q^*(s, a) = \sum_{s' \in S} Tr(s, a, s')\big(R(s, a, s') + \gamma V^*(s')\big) \tag{2.11}$$

The state-value function under the optimal policy becomes

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} Q^*(s, a). \tag{2.12}$$

Similar to the state-value function, in a terminal state s_T, the action-value function is always zero such that $Q(s_T, a) = 0 \ \forall \ s_T \in S$.

2.2.1 Dynamic Programming

Dynamic programming (DP) methods refer to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process [5, 19]. A perfect model of the environment is a model that can perfectly predict or mimic the behavior of the environment [5]. To obtain a perfect model of the environment, one needs to know the agent's reward function and transition function in an MDP.

The key idea behind DP is using value functions to search and find agent's optimal policy. One way to do that is performing backup operation to update the value functions and the agent's policies. The backup operation can be achieved by turning the Bellman optimality equation in (2.6) into an update rule [5]. This method is

called the value iteration algorithm and listed in Algorithm 2.1. Theoretically, the value function will converge to the optimal value function as the iteration goes to infinity. In Algorithm 2.1, we terminate the value iteration when the value function converges within a small range $[-\theta, \theta]$. Then we update the agent's policy based on the updated value function. We provide an example to show how we can use DP to find an agent's optimal policy in an MDP.

Algorithm 2.1 Value iteration algorithm

Initialize $V(s) = 0$ for all $s \in S$ and $\Delta = 0$
repeat
  For each $s \in S$:
  $v \leftarrow V(s)$
  $V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} Tr(s, a, s') \big(R(s, a, s') + \gamma V(s')\big)$
  $\Delta = \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ for all $s \in S$ (θ is a small positive number)
Obtain a deterministic policy $\pi(s)$ such that
$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} Tr(s, a, s') \big(R(s, a, s') + \gamma V(s')\big)$

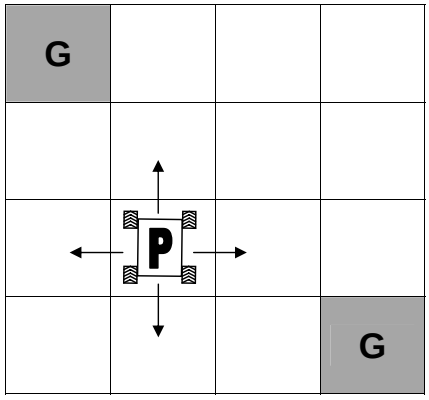
Example 2.1. We consider an example of a Markov decision process introduced in [5]. A player on a 4×4 playing field tries to reach one of the two goals labeled as "G" on the two opposite corners as shown in Fig. 2.2(a). Each cell in the 4×4 grid represents a state numbered from 1 to 16, as shown in Fig. 2.2(b). The player has 4 possible actions in its action set A: moving up, down, left and right. At each time step, the player takes an action a and moves from one cell to another. If the chosen action is taking the player off the grid, the player will stay still. For simplicity, the transition function in this game is set to 1 for each movement. For example, $Tr(2, Up, 1) = 1$ denotes that the probability of moving to the next state $s' = 1$ is 1 given the current state $s = 2$ and the chosen action $a = Up$. The reward function is given as

$$R(s, a, s') = -1, \quad \forall s \in \{2, ..., 15\} \tag{2.13}$$

such that the player receives -1 for each movement until the player reaches the goal or the terminal state. There are two terminal states $s_T \in \{1, 16\}$ located at the upper left corner and the lower right corner.

The player's aim in this example is to reach a terminal state $s_T = \{1, 16\}$ with minimum steps from its initial state $s \in \{2, ...15\}$. In order to do that, the player needs to find the optimal policy among all the possible deterministic policies. We assume we know the player's reward function and transition function. Then we can use the value iteration algorithm in Algorithm 2.1 to find the optimal state-value function and the player's optimal policy accordingly.

To be consistent with the example in [5], we set the discount factor $\gamma = 1$. Fig. 2.3 shows that the state-value function converges to the optimal state-value function after 4 iterations. The value in each cell in Fig. 2.3(d) represents the optimal state-value function for each state. Because the reward function is undiscounted ($\gamma = 1$) and the player receives -1 for each movement, the value in each cell can also indicate the actual steps for the optimal player to reach the terminal state. For example, the value -3 at the bottom left cell in Fig. 2.3(d) represents that the optimal player will take 3 steps to reach the closest terminal state. Based on the optimal state-value function, we can get the player's optimal policy using Algorithm 2.1. Fig. 2.4 shows the player's optimal policy. The arrows in Fig. 2.4 show the moving direction of the optimal player from any initial state $s \in \{2, ...15\}$ to one of the terminal states. Multiple arrows in Fig. 2.4 show that there are more than one optimal action for the player to take at that cell. It also means that the player has multiple optimal deterministic policies in this example.



(a) 4×4 playing field

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	**16**

(b) 16 states with two terminal states ($s_T = 1, 16$)

Figure 2.2: An example of Markov decision processes

2.2.2 Temporal-Difference Learning

Temporal-difference (TD) learning is a prediction technique that can learn how to predict the total rewards received in the future [20]. TD methods learn directly from raw experience without knowing the model of the environment such as the reward function or the transition function [5]. Two main temporal-difference learning algorithms in TD learning are Q-learning [21, 22] and actor-critic learning [5].

Q-Learning

Q-learning was first introduced by Watkins [21]. Using Q-learning, the agent can learn to act optimally without knowing the agent's reward function and transition function. Q-learning is an off-policy TD learning method. Off-policy methods, as opposed to on-policy methods, separate the current policy used to generate the agent's behavior and the long-term policy to be improved. For on-policy methods, the policy to be evaluated and improved is the same policy used to generate the agent's current action.

For the problems in discrete domains, the Q-learning method can estimate an optimal action-value function $Q^*(x, a)$ for all state-action pairs based on the TD

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

(a) The state-value function at iteration $k = 0$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

(b) The state-value function at iteration $k = 1$

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-1
-2	-2	-1	0

(c) The state-value function at iteration $k = 2$

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

(d) The state-value function at iteration $k = 3$

Figure 2.3: State-value function iteration algorithm in Example 2.1

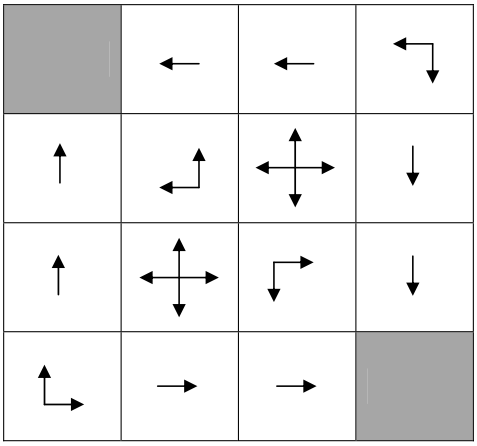


Figure 2.4: The optimal policy in Example 2.1

error [23]. For the control problems in continuous domains, the Q-learning method can discretize the action space and the state space and select the optimal action based on the finite discrete action a and the estimated $Q(x, a)$. However, when a fine discretization is used, the number of state-action pairs becomes large, which results in large memory storage and slow learning procedures [23]. On the contrary, when a coarse discretization is used, the action is not smooth and the resulting performance is poor [23].

We list the Q-learning algorithm in Algorithm 2.2.

Algorithm 2.2 Q-learning algorithm

1: Initialize $Q(s, a) = 0\ \forall s \in S, a \in A$
2: **for** Each iteration **do**
3: Select action a at current state s based on mixed exploration-exploitation strategy.
4: Take action a and observe the reward r and the subsequent state s'.
5: Update $Q(s, a)$
$Q(s, a) \leftarrow Q(s, a) + \alpha\big(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\big)$ where α is the learning rate and γ is the discount factor.
6: Update current policy $\pi(s)$
$\pi(s) = \arg\max_{a \in A} Q(s, a)$
7: **end for**

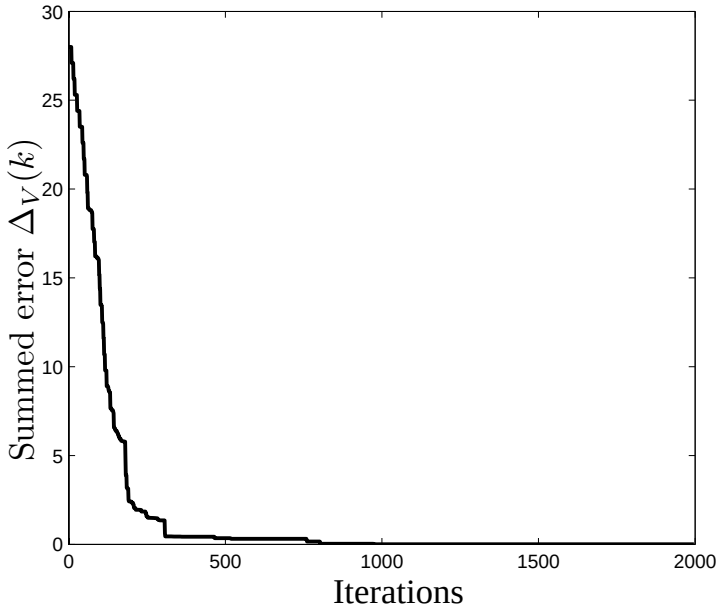


Figure 2.5: The summed error $\Delta_V(k)$

We assume that the player does not know the reward function or the transition function. We use the above Q-learning algorithm to simulate Example 2.1. We choose a mixed exploration-exploitation strategy such that the player selects an action randomly from the action set with probability 0.2 and the greedy action with probability 0.8. The greedy action means that the player chooses an action associated with the maximum Q value. We define the summed error $\Delta_V(k)$ as

$$\Delta_V(k) = \sum_{s=2}^{15} |V^*(s) - V^k(s)|. \tag{2.14}$$

where $V^*(s)$ is the optimal state-value function obtained in Fig. 2.3(d), and $V^k(s) = \max_{a \in A} Q^k(s,a)$ is the state-value function at iteration k. We set the learning rate as $\alpha = 0.9$ and run the simulation for 1000 iterations. Fig. 2.5 shows that the summed error Δ_V converges to zero after 600 iterations.

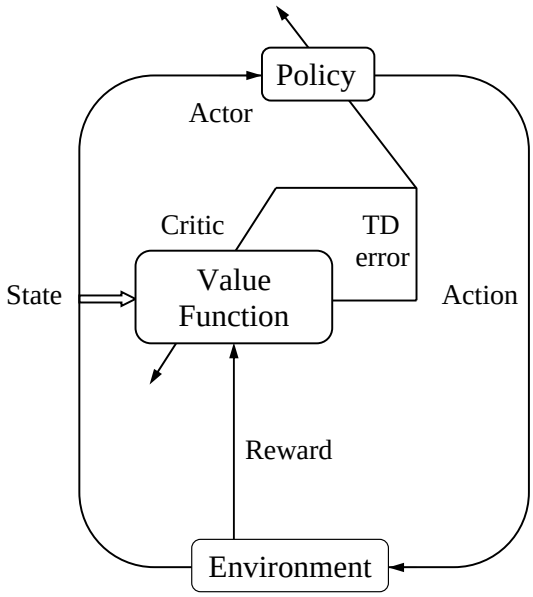


Figure 2.6: The actor-critic architecture

Actor-Critic Methods

Actor-critic methods are the natural extension of the idea of reinforcement comparison methods to TD learning methods [5, 20]. The actor-critic learning system contains two parts: one to estimate the state-value function $V(s)$, and the other to choose the optimal action for each state. The task of the critic is to predict the future system performance. After each action selection, the critic evaluates the new state to determine whether things have gone better or worse than expected [5]. The critic takes the form of a TD error defined as

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2.15}$$

where V is the current state-value function implemented by the critic at time step t. This TD error can be used to evaluate the current selected action. If the TD error is positive, it suggests that the tendency to the current selected action should

be strengthened for the future, whereas if the TD error is negative, it suggests the tendency should be weakened [5].

The state-value function $V(\cdot)$ in (2.15) can be approximated by a nonlinear function approximator such as a neural network or a fuzzy system [24]. We define $\hat{V}(\cdot)$ as the prediction of the value function $V(\cdot)$ and rewrite (2.15) as

$$\Delta = [\ r_{t+1} + \gamma \hat{V}(s_{t+1})\ ] - \hat{V}(s_t) \tag{2.16}$$

where Δ is denoted as the temporal difference that is used to adapt the critic and the actor as shown in Fig. 2.6. Compared with the Q-learning method, the actor-critic learning method is an on-policy learning method where the agent's current policy is adjusted based on the evaluation from the critic.

2.3 Matrix Games

A matrix game [25] is a tuple $(n, A_1, \ldots, A_n, R_1, \ldots, R_n)$ where n is the number of players, $A_i (i = 1, \ldots, n)$ is the action set for player i and $R_i : A_1 \times \cdots \times A_n \to \mathbb{R}$ is the reward function for player i. A matrix game is a game involving multiple players and a single state. Each player $i (i = 1, \ldots, n)$ selects an action from its action set A_i and receives a reward. The player i's reward function R_i is determined by all players' joint action from joint action space $A_1 \times \cdots \times A_n$.

In a matrix game, each player tries to maximize its own reward based on the player's strategy. A player's strategy in a matrix game is a probability distribution over the player's action set. To evaluate a player's strategy, we present the following concept of Nash equilibrium (NE).

Definition 2.1. A **Nash equilibrium** in a matrix game is a collection of all players'

strategies $(\pi_1^*, \cdots, \pi_n^*)$ such that

$$V_i(\pi_1^*, \cdots, \pi_i^*, \cdots, \pi_n^*) \geq V_i(\pi_1^*, \cdots, \pi_i, \cdots, \pi_n^*), \quad \forall \pi_i \in \Pi_i, i = 1, \cdots, n \tag{2.17}$$

where $V_i(\cdot)$ is player i's value function which is the player i's expected reward given all players' strategies, and π_i is any strategy of player i from the strategy space Π_i.

In other words, a Nash equilibrium is a collection of strategies for all players such that no player can do better by changing its own strategy given that other players continue playing their Nash equilibrium strategies [26, 27]. We define $Q_i(a_1, \ldots, a_n)$ as the received reward of the player i given players' joint action $a_1, \ldots, a_n$, and $\pi_i(a_i)$ $(i = 1, \ldots, n)$ as the probability of player i choosing action a_i. Then the Nash equilibrium defined in (2.17) becomes

$$\begin{aligned} &\sum_{a_1,\ldots,a_n \in A_1 \times \cdots \times A_n} Q_i(a_1, \ldots, a_n) \pi_1^*(a_1) \cdots \pi_i^*(a_i) \cdots \pi_n^*(a_n) \geq \\ &\sum_{a_1,\ldots,a_n \in A_1 \times \cdots \times A_n} Q_i(a_1, \ldots, a_n) \pi_1^*(a_1) \cdots \pi_i(a_i) \cdots \pi_n^*(a_n), \quad \forall \pi_i \in \Pi_i, i = 1, \cdots, n \end{aligned} \tag{2.18}$$

where $\pi_i^*(a_i)$ is the probability of player i choosing action a_i under the player i's Nash equilibrium strategy π_i^*.

We provide the following definitions regarding matrix games.

Definition 2.2. A Nash equilibrium is called a **strict** Nash equilibrium if (2.17) is strict [28].

Definition 2.3. If the probability of any action from the action set is greater than 0, then the player's strategy is called a **fully mixed strategy**.

Definition 2.4. If the player selects one action with probability of 1 and other actions with probability of 0, then the player's strategy is called a **pure strategy**.

Definition 2.5. A Nash equilibrium is called a **strict Nash equilibrium in pure strategies** if each player's equilibrium action is better than all its other actions, given the other players' actions [29].

2.3.1 Nash Equilibria in Two-Player Matrix Games

For a two-player matrix game, we can set up a matrix with each element containing a reward for each joint action pair [30]. Then the reward function R_i for player $i(i = 1, 2)$ becomes a matrix.

A two-player matrix game is called a zero-sum game if the two players are fully competitive. In this way, we have $R_1 = -R_2$. A zero-sum game has a unique Nash equilibrium in the sense of the expected reward. It means that, although each player may have multiple Nash equilibrium strategies in a zero-sum game, the value of the expected reward or the value of the state under these Nash equilibrium strategies will be the same. A general-sum matrix game refers to all types of matrix games. In a general-sum matrix game, the Nash equilibrium is no longer unique and the game might have multiple Nash equilibria. Unlike the deterministic optimal policy for a single agent in an MDP, the equilibrium strategies in a multi-player matrix game may be stochastic.

For a two-player matrix game, we define $\pi_i = (\pi_i(a_1), \cdots, \pi_i(a_{m_i}))$ as the set of all probability distributions over player i's action set $A_i(i = 1, 2)$ where m_i denotes the number of actions for player i. Then V_i becomes

$$V_i \;=\; \pi_1 R_i \pi_2^T \tag{2.19}$$

A Nash equilibrium for a two-player matrix game is the strategy pair (π_1^*, π_2^*) for two

players such that, for $i = 1, 2$,

$$V_i(\pi_i^*, \pi_{-i}^*) \geq V_i(\pi_i, \pi_{-i}^*), \forall \pi_i \in PD(A_i) \tag{2.20}$$

where $-i$ denotes any other player than player i, and $PD(A_i)$ is the set of all probability distributions over player i's action set A_i.

Given that each player has two actions in the game, we can define a two-player two-action general-sum game as

$$R_1 = \begin{bmatrix} r_{11} & r_{12} \\ \\ r_{21} & r_{22} \end{bmatrix}, R_2 = \begin{bmatrix} c_{11} & c_{12} \\ \\ c_{21} & c_{22} \end{bmatrix} \tag{2.21}$$

where r_{lf} and c_{lf} denote the reward to the row player (player 1) and the reward to the column player (player 2) respectively. The row player chooses action $l \in \{1, 2\}$ and the column player chooses action $f \in \{1, 2\}$. Based on Definition 2.2 and (2.20), the pure strategies l and f are called a strict Nash equilibrium in pure strategies if

$$r_{lf} > r_{-lf}, c_{lf} > c_{l-f} \qquad \text{for } l, f \in \{1, 2\} \tag{2.22}$$

where $-l$ and $-f$ denote any row other than row l and any column other than column f respectively.

Linear programming in two-player zero-sum matrix games

Finding the Nash equilibrium in a two-player zero-sum matrix game is equal to finding the minimax solution for the following equation [8]

$$\max_{\pi_i \in PD(A_i)} \min_{a_{-i} \in A_{-i}} \sum_{a_i \in A_i} R_i \pi_i(a_i) \tag{2.23}$$

where $\pi_i(a_i)$ denotes the probability distribution over player i's action a_i, and a_{-i} denotes any action from another player than player i. According to (2.23), each player tries to maximize the reward in the worst case scenario against its opponent. To find the solution for (2.23), one can use linear programming.

Assume we have a 2×2 zero-sum matrix game given as

$$R_1 = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix}, R_2 = -R_1 \tag{2.24}$$

where R_1 is player 1's reward matrix and R_2 is player 2's reward matrix. We define p_j $(j = 1, 2)$ as the probability distribution over player 1's jth action and q_j as the probability distribution over player 2's jth action.

Then the linear program for player 1 is:

$$\text{Find } (p_1, p_2) \text{ to maximize } V_1$$

subject to

$$r_{11}p_1 + r_{21}p_2 \geq V_1 \tag{2.25}$$

$$r_{12}p_1 + r_{22}p_2 \geq V_1 \tag{2.26}$$

$$p_1 + p_2 = 1 \tag{2.27}$$

$$p_j \geq 0, \quad j = 1, 2 \tag{2.28}$$

The linear program for player 2 is:

$$\text{Find } (q_1, q_2) \text{ to maximize } V_2$$

subject to

$$-r_{11}q_1 - r_{12}q_2 \geq V_2 \tag{2.29}$$

$$-r_{21}q_1 - r_{22}q_2 \geq V_2 \tag{2.30}$$

$$q_1 + q_2 = 1 \tag{2.31}$$

$$q_j \geq 0, \quad j = 1, 2 \tag{2.32}$$

To solve the above linear programming, one can use the simplex method to find the optimal points geometrically. We provide three 2×2 zero-sum games below.

Example 2.2. We take the matching pennies game for example. The reward matrix for player 1 is

$$R_1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2.33}$$

Since $p_2 = 1 - p_1$, the linear program for player 1 becomes

Player 1: find p_1 to maximize V_1

subject to

$$2p_1 - 1 \geq V_1 \tag{2.34}$$

$$-2p_1 + 1 \geq V_1 \tag{2.35}$$

$$0 \leq p_1 \leq 1 \tag{2.36}$$

We use the simplex method to find the solution geometrically. Fig. 2.7 shows the plot of p_1 over V_1 where the grey area satisfies the constraints in (2.34)-(2.36). From the plot, the maximum value of V_1 within the grey area is 0 when $p_1 = 0.5$.

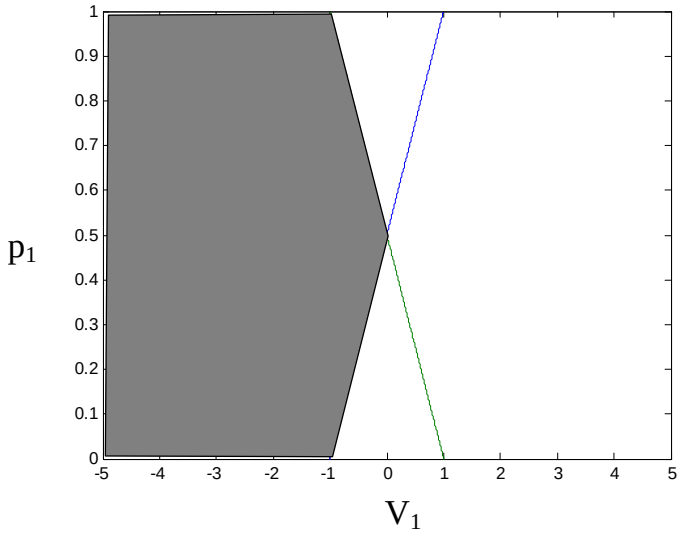


Figure 2.7: Simplex method for player 1 in the matching pennies game

Therefore, $p_1 = 0.5$ is the Nash equilibrium strategy for player 1. Similarly, we can use the simplex method to find the Nash equilibrium strategy for player 2. After solving (2.29) - (2.32), we can find that the maximum value of V_2 is 0 when $q_1 = 0.5$. Then this game has a Nash equilibrium $(p_1^* = 0.5,\ q_1^* = 0.5)$ which is a fully mixed strategy Nash equilibrium.

Example 2.3. We change the reward r_{12} from -1 in (2.33) to 2 and call this game as the revised version of the matching pennies game. The reward matrix for player 1 becomes

$$R_1 = \begin{bmatrix} 1 & 2 \\ -1 & 1 \end{bmatrix} \tag{2.37}$$

The linear program for player 1 is

Player 1: find p_1 to maximize V_1

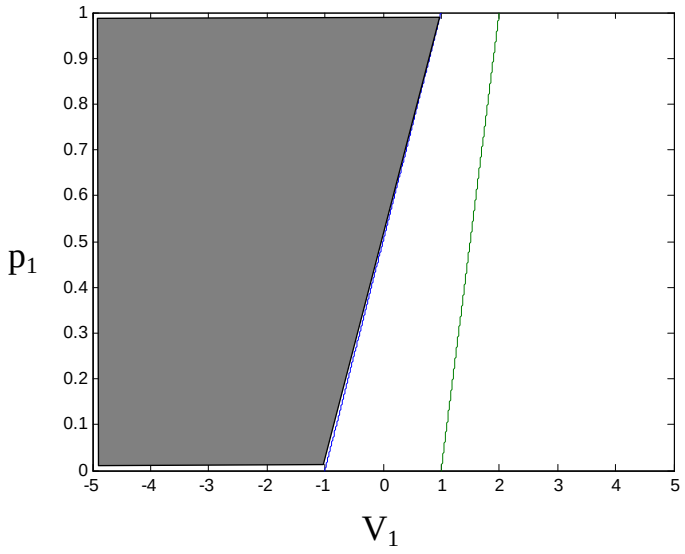


Figure 2.8: Simplex method for player 1 in the revised matching pennies game

subject to

$$2p_1 - 1 \geq V_1 \tag{2.38}$$

$$p_1 + 1 \geq V_1 \tag{2.39}$$

$$0 \leq p_1 \leq 1 \tag{2.40}$$

From the plot in Fig. 2.8, we can find that the maximum value of V_1 in the grey area is 1 when $p_1 = 1$. Similarly, we can find the maximum value of $V_2 = -1$ when $q_1 = 1$. Therefore, this game has a Nash equilibrium $(p_1^* = 1, q_1^* = 1)$ which is a pure strategy Nash equilibrium.

Example 2.4. We now consider the following zero-sum matrix game

$$R_1 = \begin{bmatrix} r_{11} & 2 \\ 3 & -1 \end{bmatrix}, R_2 = -R_1 \tag{2.41}$$

where $r_{11} \in \Re$. Based on different values of r_{11}, we want to find the Nash equilibrium strategies (p_1, q_1). The linear program for each player becomes

Player 1: Find p_1 to maximize V_1

subject to

$$(r_{11} - 3)p_1 + 3 \geq V_1 \tag{2.42}$$

$$3p_1 - 1 \geq V_1 \tag{2.43}$$

$$0 \leq p_1 \leq 1 \tag{2.44}$$

Player 2: Find q_1 to maximize V_2

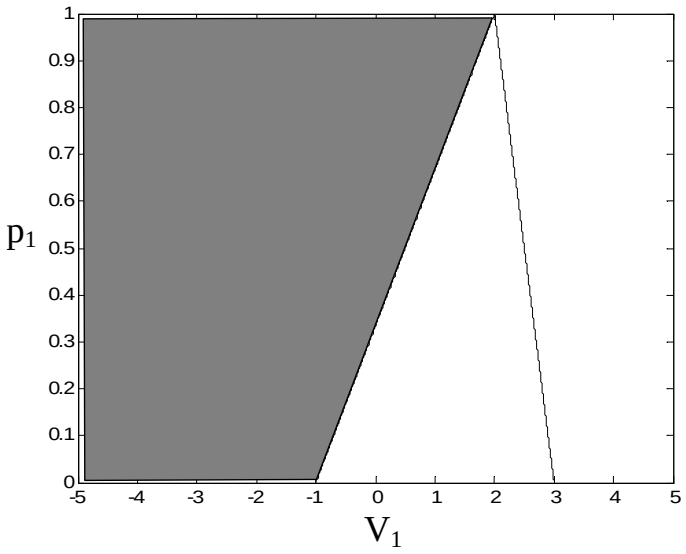
subject to

$$(2 - r_{11})q_1 - 2 \geq V_2 \tag{2.45}$$

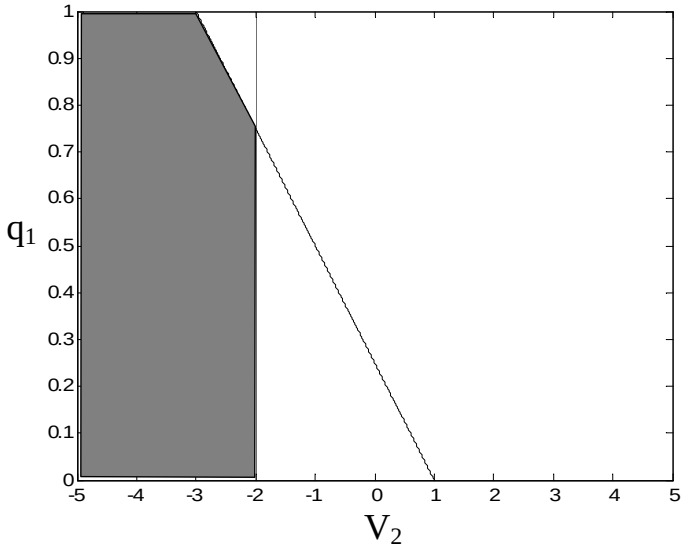
$$-4q_1 + 1 \geq V_2 \tag{2.46}$$

$$0 \leq q_1 \leq 1 \tag{2.47}$$

We use the simplex method to find the Nash equilibria for the players with a varying r_{11}. When $r_{11} > 2$, we found that the Nash equilibrium is in pure strategies $\big(p_1^* = 1, q_1^* = 0\big)$. When $r_{11} < 2$, we found that the Nash equilibrium is in fully mixed strategies $\big(p_1^* = 4/(6 - r_{11}), q_1^* = 3/(6 - r_{11})\big)$. For $r_{11} = 2$, we plot the players' strategies over their value functions in Fig. 2.9. From the plot we found that player 1's Nash equilibrium strategy is $p_1^* = 1$, and player 2's Nash equilibrium strategy is $q_1^* \in [0, 0.75]$ which is a set of strategies. Therefore, at $r_{11} = 2$, we have multiple Nash equilibria which are $p_1^* = 1, q_1^* \in [0, 0.75]$. We also plot the Nash equilibria $(p_1,$



(a) Simplex method for player 1 at $r_{11} = 2$



(b) Simplex method for player 2 at $r_{11} = 2$

Figure 2.9: Simplex method at $r_{11} = 2$ in Example 2.4

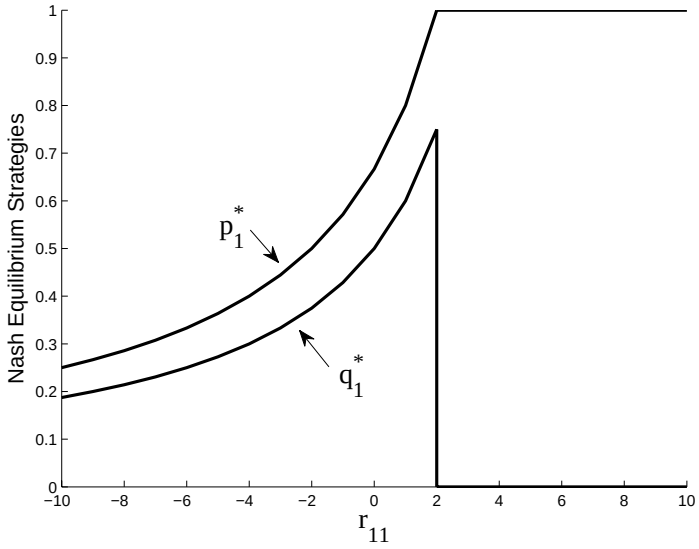


Figure 2.10: Players' NE strategies v.s. r_{11}

q_1) over r_{11} in Fig. 2.10.

2.4 Stochastic Games

A Markov decision process contains a single player and multiple states while a matrix game contains multiple players and a single state. For a game with more than one player and multiple states, it becomes a stochastic game (or Markov game) [31,32] as the combination of Markov decision processes and matrix games. A stochastic game is a tuple $(n, S, A_1, \ldots, A_n, Tr, \gamma, R_1, \ldots, R_n)$ where n is the number of the players, $Tr : S \times A_1 \times \cdots \times A_n \times S \to [0, 1]$ is the transition function, $A_i (i = 1, \ldots, n)$ is the action set for the player i, $\gamma \in [0, 1]$ is the discount factor and $R_i : S \times A_1 \times \cdots \times A_n \times S \to \mathbb{R}$ is the reward function for player i. The transition function in a stochastic game is a probability distribution over next states given the current state and joint action of the players. The reward function $R_i(s, a_1, \ldots, a_n, s')$ denotes the reward received by player i in state s' after taking joint action $(a_1, \ldots, a_n)$ in state s. Similar to Markov

decision processes, stochastic games also have the Markov property. That is, the player's next state and reward only depend on the current state and all the players' current actions.

For a multi-player stochastic game, we want to find the Nash equilibria in the game if we know the reward function and transition function in the game. A Nash equilibrium in a stochastic game can be described as a tuple of n strategies $(\pi_1^*, \ldots, \pi_n^*)$ such that for all $s \in S$ and $i = 1, \cdots, n$,

$$V_i(s, \pi_1^*, \ldots, \pi_i^*, \ldots, \pi_n^*) \geq V_i(s, \pi_1^*, \ldots, \pi_i, \ldots, \pi_n^*) \text{ for all } \pi_i \in \Pi_i \tag{2.48}$$

where Π_i is the set of strategies available to player i and $V_i(s, \pi_1^*, \ldots, \pi_n^*)$ is the expected sum of discounted rewards for player i given the current state and all the players' equilibrium strategies. To simplify notation, we use $V_i^*(s)$ to represent $V_i(s, \pi_1^*, \cdots, \pi_n^*)$ as the state-value function under Nash equilibrium strategies. We can also define the action-value function $Q^*(s, a_1, \cdots, a_n)$ as the expected sum of discounted rewards for player i given the current state and the current joint action of all the players, and following the Nash equilibrium strategies thereafter. Then we can get

$$V_i^*(s) = \sum_{a_1, \cdots, a_n \in A_1 \times \cdots \times A_n} Q_i^*(s, a_1, \cdots, a_n) \pi_1^*(s, a_1) \cdots \pi_n^*(s, a_n) \tag{2.49}$$

$$Q_i^*(s, a_1, \ldots, a_n) = \sum_{s' \in S} Tr(s, a_1, \ldots, a_n, s') \big[R_i(s, a_1, \ldots, a_n, s') + \gamma V_i^*(s') \big] \tag{2.50}$$

where $\pi_i^*(s, a_i) \in \mathrm{PD}(A_i)$ is a probability distribution over action a_i under player i's Nash equilibrium strategy, $Tr(s, a_1, \ldots, a_n, s') = Pr\{s_{k+1} = s' | s_k = s, a_1, \ldots, a_n\}$ is the probability of the next state being s' given the current state s and joint action $(a_1, \ldots, a_n)$, and $R_i(s, a_1, \ldots, a_n, s')$ is the expected reward received in state s' given the current state s and joint action $(a_1, \ldots, a_n)$. Based on (2.49) and (2.50), the Nash

equilibrium in (2.48) can be rewritten as

$$\sum_{a_1,\ldots,a_n \in A_1 \times \cdots \times A_n} Q_i^*(s, a_1, \ldots, a_n)\pi_1^*(s, a_1) \cdots \pi_i^*(s, a_i) \cdots \pi_n^*(s, a_n) \geq$$
$$\sum_{a_1,\ldots,a_n \in A_1 \times \cdots \times A_n} Q_i^*(s, a_1, \ldots, a_n)\pi_1^*(s, a_1) \cdots \pi_i(s, a_i) \cdots \pi_n^*(s, a_n). \quad (2.51)$$

Stochastic games can be classified based on the players' reward functions. If all the players have the same reward function, the game is called a fully cooperative game or a team game. If one player's reward function is always the opposite sign of the other player's, the game is called a two-player fully competitive game or zero-sum game. For the game with all types of reward functions, we call it a general-sum stochastic game.

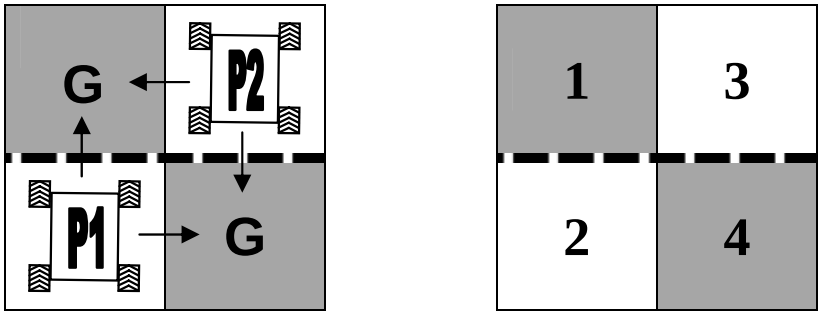
To solve a stochastic game, we need to find a strategy $\pi_i : S \to A_i$ that can maximize player i's discounted future reward with a discount factor γ. Similar to matrix games, the player's strategy in a stochastic game is probabilistic. An example is the soccer game introduced by Littman [8] where an agent on the offensive side must use a probabilistic strategy to pass to an unknown defender. In the literature, a solution to a stochastic game can be described as Nash equilibrium strategies in a set of associated **state-specific matrix games** [8, 33]. A state-specific matrix game is also called a **stage game**. In these state-specific matrix games, we define the action-value function $Q_i^*(s, a_1, \ldots, a_n)$ as the expected reward for player i when all the players take joint action $a_1, \ldots, a_n$ in state s and follow the Nash equilibrium strategies thereafter. If the value of $Q_i^*(s, a_1, \ldots, a_n)$ is known for all the states, we can find player i's Nash equilibrium strategy by solving the associated state-specific matrix game [33]. Therefore, for each state s, we have a matrix game and we can find the Nash equilibrium strategies in this matrix game. Then the Nash equilibrium strategies for the game are the collection of Nash equilibrium strategies in each state-specific matrix game for all the states. We present an example here.

Example 2.5. We define a 2×2 grid game with two players denoted as $P1$ and $P2$. Two players' initial positions are located at the bottom left corner for player 1 and the upper right corner for player 2, as shown in Fig. 2.11(a). Both players try to reach one of the two goals denoted as "G" in minimum steps. Starting from their initial positions, each player has two possible moves which are moving up or right for player 1, and moving left or down for player 2. Figure 2.11(b) shows the numbered cells in this game. Each player takes an action and moves one cell at a time. The game ends when either one of the players reaches the goal and receives a reward 10. The dash line between the upper cells and the bottom cells in Fig. 2.11(a) is the barrier that the player can pass through with a probability 0.5. If both players move to the same cell, both players bounce back to their original positions. Figure 2.11(c) shows the possible transitions in the game. The number of possible states (players' joint positions) is 7 containing the players' initial positions $s_1 = (2, 3)$ and 6 terminal states $(s_2, ..., s_7)$ which are shown in Fig. 2.11.

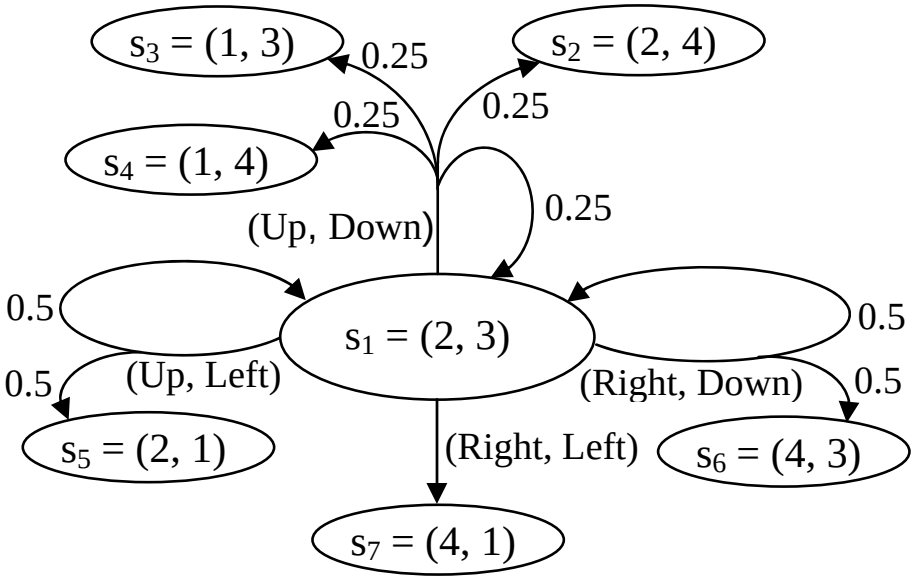
According to the above description of the game, we can find the Nash equilibrium in this example. The NE in this game is to avoid the barrier and move to the goals next to them without crossing the barrier. Therefore, the Nash equilibrium is the players' joint action $(a_1 = \text{Right}, a_2 = \text{Left})$. Based on (2.49) and (2.50), the state-value function $V_i^*(s_1)$ under the Nash equilibrium strategies is

$$\begin{aligned} V_i^*(s_1) &= R_i(s_1, \text{Right}, \text{Left}, s_7) + \gamma V_i^*(s_7) \\ &= 10 + 0.9 \cdot 0 = 10 \end{aligned} \tag{2.52}$$

where $\gamma = 0.9$, $R_i(s_1, \text{Right}, \text{Left}, s_7) = 10$, and $V_i^*(s_7) = 0$ (the state-value functions at terminal states are always zero). We can also find the action-value function $Q_i^*(s_1, a_1, a_2)$. For example, the action-value function $Q_1^*(s_1, \text{Up}, \text{Down})$ for player 1



(a) A 2×2 grid game with two players (b) The numbered cells in the game



(c) Possible state transitions given players' joint action (a_1, a_2)

Figure 2.11: An example of stochastic games

Table 2.1: The action-value function $Q_i^*(s_1, a_1, a_2)$ in Example 2.5

		a_2	
	$Q_1^*(s_1, a_1, a_2)$	Left	Down
a_1	Up	4.5	7.25
	Right	10	9.5

		a_2	
	$Q_2^*(s_1, a_1, a_2)$	Left	Down
a_1	Up	9.5	7.25
	Right	10	4.5

can be written as

$$\begin{aligned} Q_1^*(s_1, \text{Up}, \text{Down}) &= \sum_{s'=s_1 \sim s_4} Tr(s_1, \text{Up}, \text{Down}, s') \big[R_1(s_1, \text{Up}, \text{Down}, s') \\ &\quad + \gamma V_1^*(s') \big] \\ &= 0.25(0 + 0.9V_1^*(s_1)) + 0.25(0 + 0.9V_1^*(s_2)) \\ &\quad + 0.25(10 + 0.9V_1^*(s_3)) + 0.25(10 + 0.9V_1^*(s_4)) \\ &= 0.25 \cdot 0.9 \cdot 10 + 0.25 \cdot 0 + 0.25 \cdot 10 + 0.25 \cdot 10 \\ &= 7.25 \end{aligned} \tag{2.53}$$

Table 2.1 shows the action-value functions under the players' Nash equilibrium strategies.

2.5 Summary

In this chapter, we reviewed reinforcement learning under the framework of stochastic games. We presented Markov decision processes and matrix games as the special cases of stochastic games. In Markove decision processes, we reviewed dynamic programming as the method to find the agent's optimal policy if we have the perfect model of the environment. And we presented temporal-difference learning as the learning algorithm to learn the agent's optimal policy if we do not know the agent's reward function and transition function. In matrix games, we presented the basic concepts

and definitions of matrix games including Nash equilibrium, strict Nash equilibrium, fully mixed strategies, etc. We also presented linear programming as the approach to find Nash equilibria in matrix games. We provided examples using the simplex method to find the Nash equilibrium strategies for three 2×2 zero-sum matrix games. Stochastic games were also presented in this chapter. We denoted that the Nash equilibrium strategies can be found by solving the associated state-specific matrix game for all the possible states in a stochastic game. Then we provided an example to demonstrate how to find a solution in a two-player stochastic game. This chapter provides the fundamental background for the works in the subsequent chapters.

Chapter 3

Reinforcement Learning in Stochastic Games

3.1 Introduction

Learning in stochastic games can be formalized as a multi-agent reinforcement learning problem [10, 34]. Agents select actions simultaneously at the current state and receive rewards at the next state. Different from the algorithm that can solve for a Nash equilibrium in a stochastic game, the goal of a reinforcement learning algorithm is to learn equilibrium strategies through interaction with the environment. Generally, in a multi-agent reinforcement learning problem, agents may not know the transition function or the reward function from the environment. Instead, agents are required to select actions and observe the received immediate reward and the next state in order to gain information of the transition function or the reward function.

Rationality and convergence are two desirable properties for multi-agent learning algorithms in stochastic games [10, 35]. The property of **rationality** means that the learning agent will converge to a stationary strategy that is optimal to the other players' stationary strategies. The property of **convergence** means that the learning

agent will converge to a stationary strategy while the other agents using an algorithm from some class of learning algorithms [10]. In the literature, the property of convergence usually refers to the convergence of self-play. It means that, if all the players use the same learning algorithm, the players' strategies will converge to their equilibrium strategies. Another way to distinguish the learning algorithms is based on the applicability to different types of stochastic games, which can be zero-sum games or general-sum games.

In this chapter, we review some existing reinforcement learning algorithms in stochastic games. We analyze these algorithms based on their applicability, rationality and convergence properties.

Isaacs [36] introduced a differential game of guarding a territory where a defender tries to intercept an invader before the invader reaches the territory. In this chapter, we introduce a grid version of Isaacs' game called the grid game of guarding a territory. It is a two-player zero-sum stochastic game where the defender plays against the invader in a grid world. We then study how the players learn to play the game using multi-agent reinforcement learning algorithms. We apply two reinforcement learning algorithms to this game and test the performance of these learning algorithms based on convergence and rationality properties.

The main **contribution in this chapter** is:

- Establish a grid game of guarding a territory;
- Apply two multi-agent reinforcement learning algorithms to the game.

The above contribution has been published in [37].

In Sect. 3.2, we present four multi-agent reinforcement learning algorithms and compare them. We define the grid game of guarding a territory in Sect. 3.3.1 and provide simulations in Sect. 3.3.2.

3.2 Reinforcement Learning Algorithms in Stochastic Games

In this section we examine four existing reinforcement learning algorithms in stochastic games. We compare these reinforcement learning algorithms based on their applicability, rationality and convergence properties.

3.2.1 Minimax-Q Algorithm

Littman [8] proposed a minimax-Q algorithm specifically designed for two-player zero-sum stochastic games. The minimax-Q algorithm uses the minimax principle to solve for players' Nash equilibrium strategies and values of states for two-player zero-sum stochastic games. Similar to Q-learning, minimax-Q algorithm is a temporal-difference learning method that perform back-propagation on values of states or state-action pairs. We show the minimax-Q algorithm as follows.

In a two-player zero-sum stochastic game, given the current state s, we define the state-value function for player i as

$$V_i^*(s) = \max_{\pi_i(s,\cdot)} \min_{a_{-i} \in A_{-i}} \sum_{a_i \in A_i} Q_i^*(s, a_i, a_{-i}) \pi_i(s, a_i), \ (i = 1, 2) \tag{3.1}$$

where $-i$ denotes player i's opponent, $\pi_i(s, \cdot)$ denotes all the possible strategies of player i at state s, and $Q_i^*(s, a_i, a_{-i})$ is the expected reward when player i and its opponent choose action $a_i \in A_i$ and $a_{-i} \in A_{-i}$ respectively and follow their Nash equilibrium strategies after that. If we know $Q_i^*(s, a_i, a_{-i})$, we can solve the above equation (3.1) and find player i's Nash equilibrium strategy $\pi^*(s, \cdot)$. Similar to finding the minimax solution for (2.23), one can use linear programming to solve equation (3.1). For a multi-agent reinforcement learning problem, $Q_i^*(s, a_i, a_{-i})$ is unknown to the players in the game. Therefore, an updating rule similar to the Q-learning

algorithm in Section 2.2.2 is needed.

The minimax-Q algorithm is listed in Algorithm 3.1.

Algorithm 3.1 Minimax-Q algorithm

1: Initialize $Q_i(s, a_i, a_{-i})$, $V_i(s)$ and π_i
2: **for** Each iteration **do**
3: Player i takes an action a_i from current state s based on an exploration-exploitation strategy
4: At the subsequent state s', player i observes the received reward r_i and the opponent's action taken at the previous state s.
5: Update $Q_i(s, a_i, a_{-i})$:

$$Q_i(s, a_i, a_{-i}) \leftarrow (1-\alpha)Q_i(s, a_i, a_{-i}) + \alpha\big[r_i + \gamma V_i(s')\big] \tag{3.2}$$

where α is the learning rate and γ is the discount factor.
6: Use linear programming to solve equation (3.1) and obtain the updated $\pi_i(s, \cdot)$ and $V_i(s)$
7: **end for**
Note: an exploration-exploitation strategy means that the player selects an action randomly from the action set with a probability of ε and the greedy action with a probability of $1 - \varepsilon$.

The minimax-Q algorithm can guarantee the convergence to a Nash equilibrium if all the possible states and players' possible actions are visited infinitely often [38]. The proof of convergence for the minimax-Q algorithm can be found in [38]. One drawback of this algorithm is that we have to use linear programming to solve for $\pi_i(s, \cdot)$ and $V_i(s)$ at each iteration in step 6 of Algorithm 3.1. This will lead to a slow learning process. Also, in order to perform linear programming, player i has to know the opponent's action space.

Using the minimax-Q algorithm, the player will always play a "safe" strategy in case of the worst scenario caused by the opponent. However, if the opponent is currently playing a stationary strategy which is not its equilibrium strategy, the minimax-Q algorithm cannot make the player adapt its strategy to the change in the

opponent's strategy. The reason is that the minimax-Q algorithm is an opponent-independent algorithm and it will converge to the player's Nash equilibrium strategy no matter what strategy the opponent uses. If the player's opponent is a weak opponent that does not play its equilibrium strategy, then the player's optimal strategy is not the same as its Nash equilibrium strategy. The player's optimal strategy will do better than the player's Nash equilibrium strategy in this case.

Overall, the minimax-Q algorithm, which is applicable to zero-sum stochastic games, does not satisfy the rationality property but it does satisfy the convergence property.

3.2.2 Nash Q-Learning

The Nash Q-learning algorithm, first introduced in [39], extends the minimax-Q algorithm [8] from zero-sum stochastic games to general-sum stochastic games. In the Nash Q-learning algorithm, the Nash Q-values need to be calculated at each state using quadratic programming in order to update the action-value functions and find the equilibrium strategies. Although Nash Q-learning is applied to general-sum stochastic games, the conditions for the convergence to a Nash equilibrium do not cover a correspondingly general class of environments [14]. The corresponding class of environments are actually limited to cases where the game being learned only has coordination or adversarial equilibrium [14, 40, 41].

The Nash Q-Learning algorithm is shown in Algorithm 3.2.

To guarantee the convergence to Nash equilibria in general-sum stochastic games, the Nash Q-learning algorithm needs to hold the following condition during learning, that is, every stage game (or state-specific matrix game) has a global optimal point or a saddle point for all time steps and all the states [14]. Since the above strict condition is defined in terms of the stage games as perceived during learning, it

Algorithm 3.2 Nash Q-learning algorithm

1: Initialize $Q_i(s, a_1, \ldots, a_n) = 0, \ \forall a_i \in A_i \ , i = 1, \ldots, n$
2: **for** Each iteration **do**
3: Player i takes an action a_i from current state s based on an exploration-exploitation strategy
4: At the subsequent state s', player i observes the rewards received from all the players $r_1, \ldots, r_n$, and all the players' actions taken at the previous state s.
5: Update $Q_i(s, a_1, \ldots, a_n)$:

$$Q_i(s, a_1, \ldots, a_n) \leftarrow (1 - \alpha) Q_i(s, a_1, \ldots, a_n) + \alpha \big[r_i + \gamma \text{Nash} Q_i(s') \big] \tag{3.3}$$

where α is the learning rate and γ is the discount factor
6: Update $\text{Nash}Q_i(s)$ and $\pi_i(s, \cdot)$ using quadratic programming
7: **end for**

cannot be evaluated in terms of the actual game being learned [14].

Similar to the minimax-Q learning, the Nash Q-learning algorithm needs to solve the quadratic programming problem at each iteration in order to obtain the Nash Q-values which leads to a slow learning process. Above all, the Nash Q-learning algorithm does satisfy the convergence property, does not satisfy the rationality property, and it can be applied to some general-sum stochastic games with only coordination or adversarial equilibrium.

3.2.3 Friend-or-Foe Q-Learning

For a two-player zero-sum stochastic game, the minimax-Q algorithm [8] is well suited for the players to learn a Nash equilibrium in the game. For general-sum stochastic games, Littman proposed a friend-or-foe Q-learning (FFQ) algorithm such that a learner is told to treat the other players as either a "friend" or "foe" [40]. The friend-or-foe Q-learning algorithm assumes that the players in a general-sum stochastic game can be grouped into two types: player i's friends and player i's foes. Player i's friends are assumed to work together to maximize player i's value, while player i's foes are working together to minimize player i's value [40]. Thus, a n-player general-sum

stochastic game can be treated as a two-player zero-sum game with an extended action set [40].

The friend-or-foe Q-learning algorithm for player i is given in Algorithm 3.3.

Algorithm 3.3 Friend-or-foe Q-learning Algorithm

1: Initialize $V_i(s) = 0$ and $Q_i(s, a_1, ..., a_{n_1}, o_1, ..., o_{n_2}) = 0$ where $(a_1, ..., a_{n_1})$ denotes player i and its friends' actions and $(o_1, ..., o_{n_2})$ denotes its opponents' actions.
2: **for** Each iteration **do**
3: Player i takes an action a_i from current state s based on an exploration-exploitation strategy.
4: At the subsequent state s', player i observe the received reward r_i, its friends' and opponents' actions taken at state s.
5: Update $Q_i(s, a_1, ..., a_{n_1}, o_1, ..., o_{n_2})$:

$$Q_i(s, a_1, ..., a_{n_1}, o_1, ..., o_{n_2}) \leftarrow (1-\alpha)Q_i(s, a_1, ..., a_{n_1}, o_1, ..., o_{n_2}) + \alpha\big[r_i + \gamma V_i(s')\big]$$

where α is the learning rate and γ is the discount factor.
6: Update $V_i(s)$ using linear programming:

$$V_i(s) = \max_{\pi_1(s,\cdot),...,\pi_{n_1}(s,\cdot)} \min_{o_1,...,o_{n_2} \in O_1 \times \cdots \times O_{n_2}} \sum_{a_1,...,a_{n_1} \in A_1 \times \cdots \times A_{n_1}} Q_i(s, a_1, ..., a_{n_1}, o_1, ..., o_{n_2})\pi_1(s, a_1) \cdots \pi_{n_1}(s, a_{n_1}) \quad (3.4)$$

7: **end for**

Note that the friend-or-foe Q-learning algorithm is different from the minimax-Q algorithm for a two-team zero-sum stochastic game. In a two-team zero-sum stochastic game, a team leader controls the team players' actions and maintains the value of the state for the whole team. The received reward is also the whole team's reward. For the friend-or-foe Q-learning algorithm, there is no team leader to send commands to control the team players' actions. The FFQ player chooses its own action and maintains its own state-value function and equilibrium strategy. In order to update the action-value function $Q_i(s, a_1, ..., a_{n_1}, o_1, ..., o_{n_2})$, the FFQ player needs to observe its friends and opponents' actions at each time step.

Littman's friend-or-foe Q-learning algorithm can guarantee the convergence to a

Nash equilibrium if all states and actions are visited infinitely often. The proof of convergence for the friend-or-foe Q-learning algorithm can be found in [40]. Similar to the minimax-Q and Nash Q-learning algorithms, the learning speed is slow due to the execution of linear programming at each iteration in Algorithm 3.3.

3.2.4 WoLF Policy Hill-Climbing Algorithm

The aforementioned reinforcement learning algorithms in Sect. 3.2.1-Sect. 3.2.3 require agents to maintain their Q-functions. Each player's Q-function includes the information of other players' actions. We define agent i's action space as $A_i(i = 1, ..., n)$ and $|S|$ as the number of states. We assume that all the agents have the same size of action space such that $|A_1| = \cdots = |A_n| = |A|$. Then the total space requirement for each agent is $|S| \cdot |A|^n$. In terms of space complexity, the state requirement for these learning algorithms lead to be exponential in the number of agents.

The "Win or Learn Fast" policy hill-climbing (WoLF-PHC) algorithm is a practical algorithm for learning in stochastic games [10]. The WoLF-PHC algorithm only requires each player's own action, which reduces the space requirement from $|S| \cdot |A|^n$ to $|S| \cdot |A|$. The WoLF-PHC algorithm is the combination of two methods: the "Win or Learn Fast" principle and the policy hill-climbing method. The "Win or Learn Fast" principle means that a learner should adapt quickly when it is doing more poorly than expected and be cautious when it is doing better than expected [10].

The policy hill-climbing algorithm is shown in Algorithm 3.4. The PHC method is a rational learning algorithm [10]. With the PHC method, the agent's policy is improved by increasing the probability of selecting the action with the highest value in the associated Q function according to a learning rate [10]. But the PHC method can only guarantee the convergence to the player's optimal policy in a stationary

Algorithm 3.4 Policy hill-climbing algorithm

1: Initialize $Q_i(s, a_i) \leftarrow 0$ and $\pi_i(s, a_i) \leftarrow \frac{1}{|A_i|}$. Choose the learning rate α, δ and the discount factor γ.
2: **for** Each iteration **do**
3: Select action a_c from current state s based on a mixed exploration-exploitation strategy
4: Take action a_c and observe the reward r_i and the subsequent state s'
5: Update $Q_i(s, a_c)$

$$Q_i(s, a_c) = Q_i(s, a_c) + \alpha\big[r_i + \gamma \max_{a'_i} Q(s', a'_i) - Q(s, a_c)\big] \tag{3.5}$$

where a'_i is player i's action at the next state s' and a_c is the action player i has taken at state s.
6: Update $\pi_i(s, a_i)$

$$\pi_i(s, a_i) = \pi_i(s, a_i) + \Delta_{sa_i} \quad (\forall a_i \in A_i) \tag{3.6}$$

where

$$\Delta_{sa_i} = \begin{cases} -\delta_{sa_i} & \text{if } a_c \neq \arg\max_{a_i \in A_i} Q_i(s, a_i) \\ \sum_{a_j \neq a_i} \delta_{sa_j} & \text{otherwise} \end{cases} \tag{3.7}$$

$$\delta_{sa_i} = \min\big(\pi_i(s, a_i), \frac{\delta}{|A_i| - 1}\big) \tag{3.8}$$

7: **end for**

environment for a single agent.

To encourage the convergence in a dynamic environment, Bowling and Veloso [10] modified the policy hill-climbing algorithm by adding a "Win or Learn Fast" (WoLF) learning rate to the PHC algorithm. The WoLF-PHC algorithm for player i is provided in Algorithm 3.5. In the WoLF-PHC algorithm, a varying learning rate δ is introduced to perform "Win or Learn Fast". The learning rate δ_l for the losing situation is larger than the learning rate δ_w for the winning situation. If the player is losing, it should learn quickly to escape from the losing situation. If the player is winning, it should learn cautiously to maintain the convergence of the policy. Different from the aforementioned learning algorithms, the WoLF-PHC algorithm does not

Algorithm 3.5 WoLF-PHC learning algorithm

1: Initialize $Q_i(s, a_i) \leftarrow 0$, $\pi_i(s, a_i) \leftarrow \frac{1}{|A_i|}$, $\bar{\pi}_i(s, a_i) \leftarrow \frac{1}{|A_i|}$ and $C(s) \leftarrow 0$. Choose the learning rate α, δ and the discount factor γ

2: **for** Each iteration **do**

3: Select action a_c from current state s based on a mixed exploration-exploitation strategy

4: Take action a_c and observe the reward r_i and the subsequent state s'

5: Update $Q_i(s, a_c)$

$$Q_i(s, a_c) = Q_i(s, a_c) + \alpha\big[r_i + \gamma \max_{a'_i} Q(s', a'_i) - Q(s, a_c)\big] \tag{3.9}$$

where a'_i is player i's action at the next state s' and a_c is the action player i has taken at state s.

6: Update the estimate of average strategy $\bar{\pi}_i$

$$C(s) = C(s) + 1 \tag{3.10}$$

$$\bar{\pi}_i(s, a_i) = \bar{\pi}_i(s, a_i) + \frac{1}{C(s)}\big(\pi_i(s, a_i) - \bar{\pi}_i(s, a_i)\big) \quad (\forall a_i \in A_i) \tag{3.11}$$

where $C(s)$ denotes how many times the state s has been visited.

7: Update $\pi_i(s, a_i)$

$$\pi_i(s, a_i) = \pi_i(s, a_i) + \Delta_{sa_i} \quad (\forall a_i \in A_i) \tag{3.12}$$

where

$$\Delta_{sa_i} = \begin{cases} -\delta_{sa_i} & \text{if } a_c \neq \arg\max_{a_i \in A_i} Q_i(s, a_i) \\ \sum_{a_j \neq a_i} \delta_{sa_j} & \text{otherwise} \end{cases} \tag{3.13}$$

$$\delta_{sa_i} = \min\big(\pi_i(s, a_i), \frac{\delta}{|A_i| - 1}\big) \tag{3.14}$$

$$\delta = \begin{cases} \delta_w & \text{if } \sum_{a_i \in A_i} \pi_i(s, a_i) Q_i(s, a_i) > \sum_{a_i \in A_i} \bar{\pi}_i(s, a_i) Q_i(s, a_i) \\ \delta_l & \text{otherwise} \end{cases}$$

8: **end for**

need to observe the other players strategies and actions. Therefore, compared with the other three learning algorithms, the WoLF-PHC algorithm needs less information from the environment. Since the WoLF-PHC algorithm is based on the policy hill-climbing method, neither linear programming nor quadratic programming is required in this algorithm. Since the WoLF-PHC algorithm is a practical algorithm, there was no proof of convergence provided in [10]. Instead, simulation results in [10] illustrated the convergence of players' strategies by manually choosing the according learning rate to different matrix games and stochastic games.

3.2.5 Summary

In this section, we reviewed four multi-agent reinforcement learning algorithms in stochastic games. The analysis of these algorithms was conducted based on three properties: applicability, rationality and convergence. Table 3.1 shows the comparison of these algorithms based on these properties. As a practical algorithm, the WoLF-PHC algorithm does not provide the convergence property, but showed the potential to converge to Nash equilibria by empirical examples. Furthermore, the WoLF-PHC algorithm is a rational learning algorithm such that it can converge to the player's optimal strategy when playing against an opponent with any arbitrary stationary strategy.

In the next section, we create a grid game of guarding a territory as a test bed for reinforcement learning algorithms. We apply the minimax-Q and WoLF-PHC algorithms to the game and test the performance of theses learning algorithms through simulations.

Table 3.1: Comparison of multi-agent reinforcement learning algorithms

Algorithms	Applicability	Rationality	Convergence
Minimax-Q	Zero-sum SGs	No	Yes
Nash Q-learning	Specific general-sum SGs	No	Yes
Friend-or-foe Q-learning	Specific general-sum SGs	No	Yes
WoLF-PHC	General-sum SGs	Yes	No

3.3 Guarding a Territory Problem in a Grid World

The game of guarding a territory was first introduced by Isaacs [36]. In the game, the invader tries to move as close as possible to the territory while the defender tries to intercept and keep the invader as far as possible from the territory. The practical application of this game can be found in surveillance and security missions for autonomous mobile robots [42]. There are few published works in this field since the game was introduced [43, 44]. In these published works, the defender tries to use a fuzzy controller to locate the invader's position [43] or applies a fuzzy reasoning strategy to capture the invader [44]. However, in these works, the defender is assumed to know its optimal policy and the invader's policy. There is no learning technique applied to the players in their works. In our research, we assume the defender or the invader has no prior knowledge of its optimal policy and the opponent's policy. We apply learning algorithms to the players and let the defender or the invader obtain its own optimal behavior after learning.

The problem of guarding a territory in [36] is a differential game problem where the dynamic equations of the players are typically differential equations. In our research, we will investigate how the players learn to behave with no knowledge of the optimal strategies. Therefore, the above problem becomes a multi-agent learning problem in a multi-agent system. In the literature, there is a wealth of published papers on multi-agent systems [6, 45]. Among the multi-agent learning applications,

the predator-prey or the pursuit problem in a grid world has been well studied [11,45]. To better understand the learning process of the two players in the game, we create a grid game of guarding a territory which has never been studied so far.

Most of multi-agent learning algorithms are based on multi-agent reinforcement learning methods [45]. According to the definition of the game in [36], the grid game we established is a two-player zero-sum stochastic game. The minimax-Q algorithm [8] is well suited to solving our problem. However, if the player does not always take the action that is most damaging the opponent, the opponent might have better performance using a rational learning algorithm than the minimax-Q [6]. The rational learning algorithm we used here is the WoLF-PHC learning algorithm. In this section, we run simulations and compare the learning performance of the minimax-Q and WoLF-PHC algorithms.

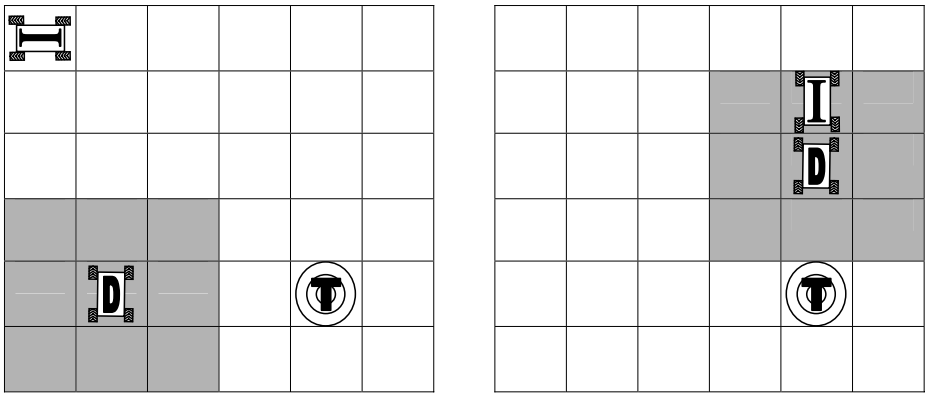
3.3.1 A Grid Game of Guarding a Territory

The problem of guarding a territory in this section is the grid version of the guarding a territory game in [36]. The game is defined as follows:

- We take a 6×6 grid as the playing field shown in Fig. 3.1. The invader starts from the upper-left corner and tries to reach the territory before the capture. The territory is represented by a cell named T in Fig. 3.1. The defender starts from the bottom and tries to intercept the invader. The initial positions of the players are not fixed and can be chosen randomly.

- Both of the players can move up, down, left or right. At each time step, both player take their actions simultaneously and move to their adjacent cells. If the chosen action will take the player off the playing field, the player will stay at the current position.

- The nine gray cells centered around the defender, shown in Fig. 3.1(b), is the



(a) One possible initial positions of the players when the game starts

(b) One possible terminal positions of the players when the game ends

Figure 3.1: Guarding a territory in a grid world

region where the invader will be captured. A successful invasion by the invader is defined in the situation where the invader reaches the territory before the capture or the capture happens at the territory. The game ends when the defender captures the invader or a successful invasion by the invader happens. Then the game restarts with random initial positions of the players.

- The goal of the invader is to reach the territory without interception or move to the territory as close as possible if the capture must happen. On the contrary, the aim of the defender is to intercept the invader at a location as far as possible to the territory.

The terminal time is defined as the time when the invader reaches the territory or is intercepted by the defender. We define the payoff as the distance between the invader and the territory at the terminal time:

$$\text{Payoff} = |x_I(t_f) - x_T| + |y_I(t_f) - y_T| \tag{3.15}$$

Table 3.2: Comparison of pursuit-evasion game and guarding a territory game

	Pursuit-evasion game	Guarding a territory game
Payoff	$Dist_{PE}$	$Dist_{IT}$
Rewards	Immediate rewards	Only terminal rewards

where $\big(x_I(t_f), y_I(t_f)\big)$ is the invader's position at the terminal time t_f and (x_T, y_T) is the territory's position. Based on the definition of the game, the invader tries to minimize the payoff while the defender tries to maximize the payoff.

The difference between this game and the pursuit-evasion game is illustrated in Table 3.2. In a pursuit-evasion game, a pursuer tries to capture an evader. The payoff in the pursuit-evasion game is $Dist_{PE}$ which is the distance between the pursuer and the evader. The players in the pursuit-evasion game are receiving immediate rewards at each time step. For the guarding a territory game, the payoff is $Dist_{IT}$ which is the distance between the invader and the territory at the terminal time. The players are only receiving terminal rewards when the game ends.

3.3.2 Simulation and Results

We use the minimax-Q and WoLF-PHC algorithms introduced in Section 3.2.1 and 3.2.4 to simulate the grid game of guarding a territory. We first present a simple 2×2 grid game to analyze the NE of the game, the property of rationality and the property of convergence. Next, the playing field is enlarged to a 6×6 grid and we examine the performance of the learning algorithms based on this large grid.

We set up two simulations for each grid game. In the first simulation, the players in the game use the same learning algorithm to play against each other. We examine if the algorithm satisfies the convergence property. In the second simulation, we will freeze one player's strategy and let the other player learn its optimal strategy against

its opponent. We use the minimax-Q and WoLF-PHC algorithms to train the learner individually and compare the performance of the minimax-Q trained player and the WoLF-PHC trained player. According to the rationality property shown in Tab. 3.1, we expect the WoLF-PHC trained the defender has better performance than the minimax-Q trained defender in the second simulation.

2×2 Grid Game

The playing field of the 2×2 grid game is shown in Fig. 3.2. The territory to be guarded is located at the bottom-right corner. Initially, the invader starts at the top-left corner while the defender starts at the same cell as the territory. To better illustrate the guarding a territory problem, we simplify the possible actions of each player from 4 actions defined in Sect. 3.3.1 to 2 actions. The invader can only move down or right while the defender can only move up or left. The capture of the invader happens when the defender and the invader move into the same cell excluding the territory cell. The game ends when the invader reaches the territory or the defender catches the invader before it reaches the territory. We suppose both players start from the initial state s_1 shown in Fig. 3.2(a). There are three non-terminal states (s_1, s_2, s_3) in this game shown in Fig. 3.2. If the invader moves to the right cell and the defender happens to move left, then both players reach the state s_2 in Fig. 3.2(b). If the invader moves down and the defender moves up simultaneously, then they will reach the state s_3 in Fig. 3.2(c). In states s_2 and s_3, if the invader is smart enough, it can always reach the territory no matter what action the defender will take. Therefore, starting from the initial state s_1, a clever defender will try to intercept the invader by guessing which direction the invader will go.

We define $dist_{P_1P_2}$ as the Manhattan distance (taxicab metric) between players P_1 and P_2 in a grid world. If the players' coordinates in the grid are (x_{P_1}, y_{P_1}) for player

P_1 and (x_{P_2}, y_{P_2}) for player P_2, then the Manhattan distance is calculated as

$$dist_{P_1P_2} = |x_{P_1} - x_{P_2}| + |y_{P_1} - y_{P_2}|. \tag{3.16}$$

We now define the reward functions for the players. The reward function for the defender is defined as

$$R_D = \begin{cases} dist_{IT}, \text{ defender captures the invader;} \\ -10, \quad \text{invader reaches the territory.} \end{cases} \tag{3.17}$$

where

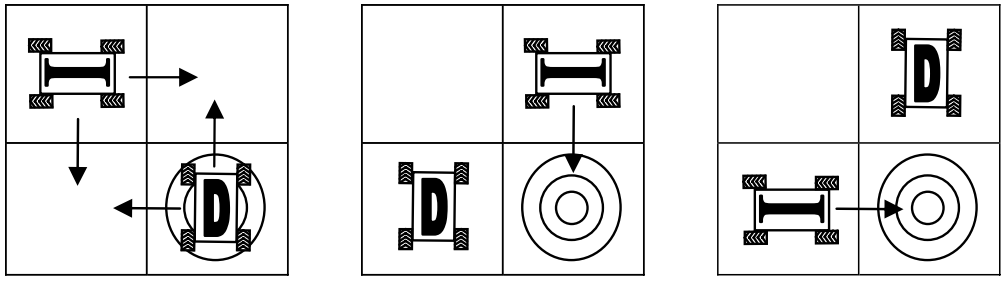
$$dist_{IT} = |x_I(t_f) - x_T| + |y_I(t_f) - y_T|.$$

The reward function for the invader is given by

$$R_I = \begin{cases} -dist_{IT}, \text{ defender captures the invader;} \\ 10, \qquad \text{invader reaches the territory.} \end{cases} \tag{3.18}$$

The reward functions in (3.17) and (3.18) are also used in the 6×6 grid game.

Before the simulation, we can simply solve this game similar to solving Example 2.5. In states s_2 and s_3, a smart invader will always reach the territory without being intercepted. The value of the states s_2 and s_3 for the defender will be $V_D(s_2) = -10$ and $V_D(s_3) = -10$. We set the discount factor as 0.9 and we can get $Q_D^*(s_1, a_{left}, o_{right}) = \gamma V_D(s_2) = -9$, $Q_D^*(s_1, a_{up}, o_{down}) = \gamma V_D(s_3) = -9$, $Q_D^*(s_1, a_{left}, o_{down}) = 1$ and $Q_D^*(s_1, a_{up}, o_{right}) = 1$, as shown in Tab. 3.3(a). Under the Nash equilibrium, we define the probabilities of the defender moving up and left as $\pi_D^*(s_1, a_{up})$ and $\pi_D^*(s_1, a_{left})$ respectively. The probabilities of the invader moving down and right are denoted as $\pi_I^*(s_1, o_{up})$ and $\pi_I^*(s_1, o_{left})$ respectively. Based on



(a) Initial positions of the players: state s_1

(b) invader in top-right vs. defender in bottom-left: state s_2

(c) Invader in bottom-left vs. defender in top-right: state s_3

Figure 3.2: A 2×2 grid game

the Q values in Tab. 3.3(a), we can find the value of the state s_1 for the defender by solving a linear programming problem shown in Tab. 3.3(b). The approach for solving a linear programming problem can be found in Sect. 2.3.1. After solving the linear constraints in Tab. 3.3(b), we get the value of the state s_1 for the defender as $V_D(s_1) = -4$ and the Nash equilibrium strategy for the defender as $\pi^*_D(s_1, a_{up}) = 0.5$ and $\pi^*_D(s_1, a_{left}) = 0.5$. For a two-player zero-sum game, we can get $Q^*_D = -Q^*_I$. Similar to the approach in Tab. 3.3, we can find the minimax solution of this game for the invader as $V_I(s_1) = 4$, $\pi^*_I(s_1, o_{down}) = 0.5$ and $\pi^*_I(s_1, o_{right}) = 0.5$. Therefore, the Nash equilibrium strategy of the invader is moving down or right with probability 0.5 and the Nash equilibrium strategy of the defender is moving up or left with probability 0.5.

We now test how learning algorithms can help the players learn the NE without knowing the model of the environment. We first apply the minimax-Q algorithm to the game. To better examine the performance of the minimax-Q algorithm, we use the same parameter settings as in [8]. We use the ε-greedy policy as the exploration-exploitation strategy. The ε-greedy policy is defined such that the player chooses an

Table 3.3: Minimax solution for the defender in the state s_1

(a) Q values of the defender for the state s_1

		Defender	
	Q_D^*	up	left
Invader	down	-9	1
	right	1	-9

(b) linear constraints for the defender in the state s_1

Objective: Maximize V

$$(-9) \cdot \pi_D(s_1, a_{up}) + (1) \cdot \pi_D(s_1, a_{left}) \geq V$$

$$(1) \cdot \pi_D(s_1, a_{up}) + (-9) \cdot \pi_D(s_1, a_{left}) \geq V$$

$$\pi_D(s_1, a_{up}) + \pi_D(s_1, a_{left}) = 1$$

action randomly from the player's action set with a probability ε and a greedy action with a probability $1 - \varepsilon$. The greedy parameter ε is given as 0.2. The learning rate α is chosen such that the value of the learning rate will decay to 0.01 after one million iterations. The discount factor γ is set to 0.9. We run the simulation for 100 iterations. The number of iterations represents the number of times the step 2 is repeated in Algorithm 3.1. After learning, we plot the players' learned strategies in Fig. 3.3. The result shows that the players' strategies converge to the Nash equilibrium after 100 iterations.

We now apply the WoLF-PHC algorithm to the 2×2 grid game. According to the parameter settings in [10], we set the learning rate α as $1/(10 + t/10000)$, δ_w as $1/(10 + t/2)$ and δ_l as $3/(10 + t/2)$ where t is the number of iterations. The number of iterations denotes the number of times the step 2 is repeated in Algorithm 3.5. The result in Fig. 3.4 shows that the players' strategies converge close to the Nash equilibrium after 15000 iterations.

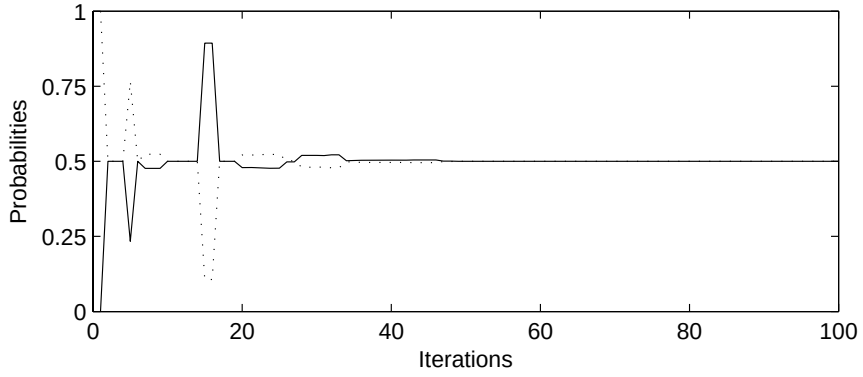
In the second simulation, the invader plays a stationary strategy against the defender at state s_1 in Fig. 3.2(a). The invader's fixed strategy is moving right with probability 0.8 and moving down with probability 0.2. Then the optimal strategy for the defender against this invader is moving up all the time. We apply both algorithms to the game and examine the learning performance for the defender. Fig. 3.5(a) shows that, using the minimax-Q algorithm, the defender's strategy fails to

converge to its optimal strategy. Whereas, Fig. 3.5(b) shows that the WoLF-PHC algorithm does converge to the defender's optimal strategy against the invader.

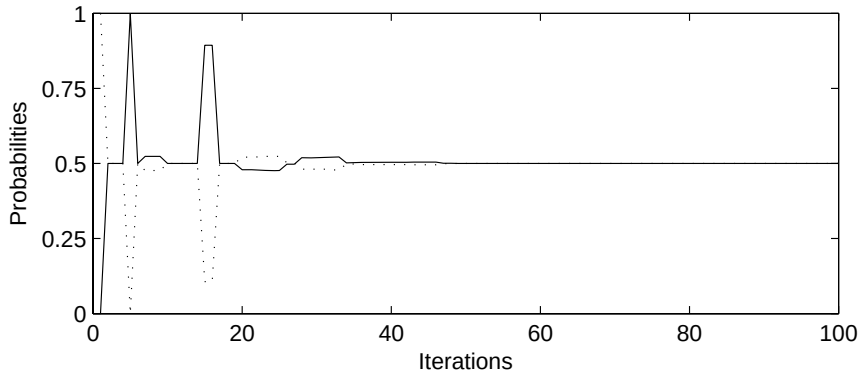
In the 2×2 grid game, the first simulation verified the convergence property of the minimax-Q and WoLF-PHC algorithms. According to Tab. 3.1, there is no proof of convergence for the WoLF-PHC algorithm. But simulation results in Fig. 3.4 showed that the players' strategies converged to the Nash equilibrium when both players used the WoLF-PHC algorithm. Under the rationality criterion, the minimax-Q algorithm failed to converge to the defender's optimal strategy in Fig. 3.5(a), while the WoLF-PHC algorithm showed the convergence to the defender's optimal strategy after learning.

6×6 Grid Game

We now change the 2×2 grid game to a 6×6 grid game. The playing field of the 6×6 grid game is defined in Section 3.3.1. The territory to be guarded is represented by a cell located at $(5, 5)$ in Fig. 3.6. The position of the territory will not be changed during the simulation. The initial positions of the invader and defender are shown in Fig. 3.6(a). The number of actions for each player has been changed from 2 in the 2×2 grid game to 4 in the 6×6 grid game. Both players can move up, down, left or right. The grey cells in Fig. 3.6(a) is the area where the defender can reach before the invader. Therefore, if both players play their equilibrium strategies, the invader can move to the territory as close as possible with the distance of 2 cells shown in Fig. 3.6(b). Different from the previous 2×2 grid game where we show the convergence of the players' strategies during learning, in this game, we want to show the average learning performance of the players during learning. We add a testing phase to evaluate the learned strategies after every 100 iterations. The number of

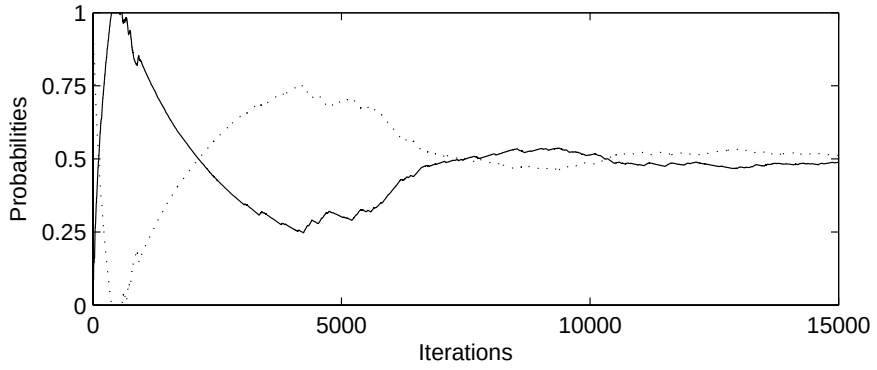


(a) Defender's strategy $\pi_D(s_1, a_{left})$ (solid line) and $\pi_D(s_1, a_{up})$(dash line)

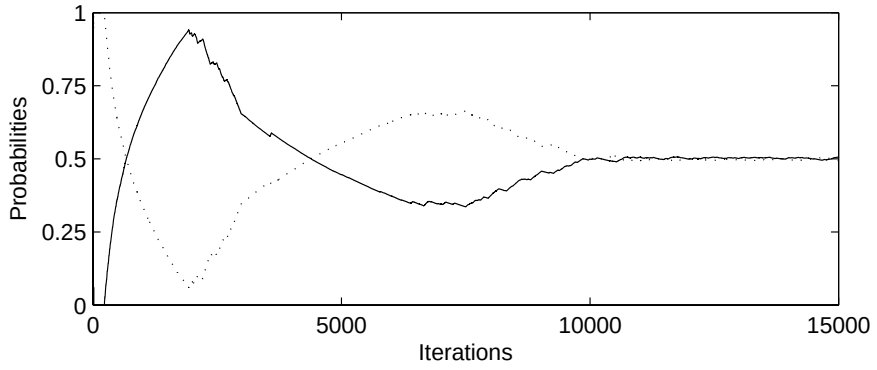


(b) Invader's strategy $\pi_I(s_1, o_{down})$ (solid line) and $\pi_I(s_1, o_{right})$ (dash line)

Figure 3.3: Players' strategies at state s_1 using the minimax-Q algorithm in the first simulation for the 2×2 grid game

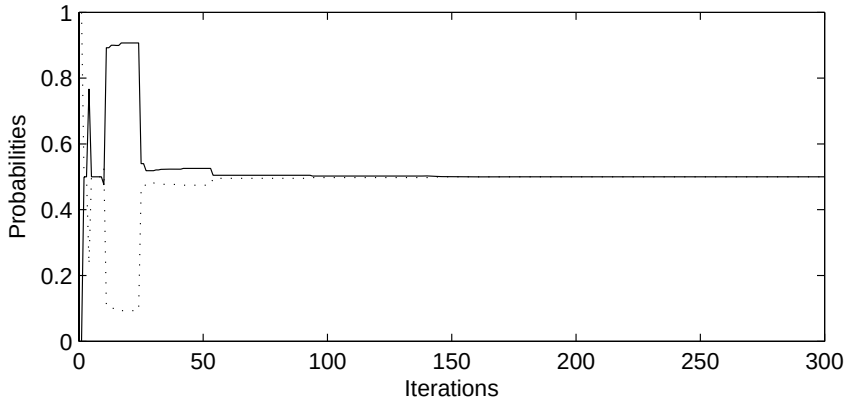


(a) Defender's strategy $\pi_D(s_1, a_{left})$ (solid line) and $\pi_D(s_1, a_{up})$(dash line)

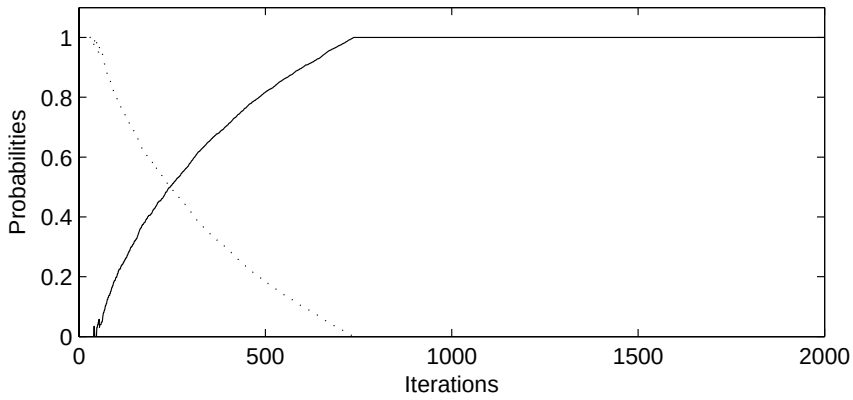


(b) Invader's strategy $\pi_I(s_1, o_{down})$ (solid line) and $\pi_I(s_1, o_{right})$ (dash line)

Figure 3.4: Players' strategies at state s_1 using the WoLF-PHC algorithm in the first simulation for the 2×2 grid game

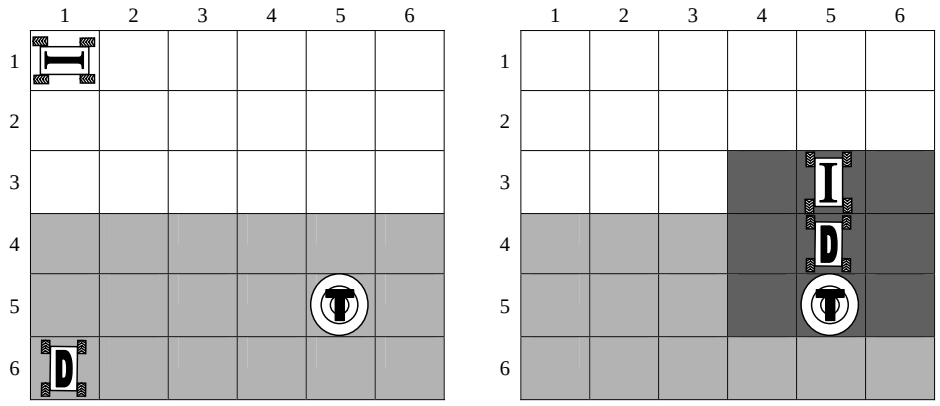


(a) Minimax-Q learned strategy of the defender at state s_1 against the invader using a fixed strategy. Solid line: probability of defender moving up; Dash line: probability of defender moving left



(b) WoLF-PHC learned strategy of the defender at state s_1 against the invader using a fixed strategy. Solid line: probability of defender moving up; Dash line: probability of defender moving left

Figure 3.5: Defender's strategy at state s_1 in the second simulation for the 2×2 grid game



(a) Initial positions of the players

(b) One of the terminal positions of the players

Figure 3.6: A 6×6 grid game

iterations denotes the number of times the step 2 is repeated in Algorithm 3.1 or Algorithm 3.5. A testing phase includes 1000 runs of the game. In each run, the learned players start from their initial positions shown in Fig. 3.6(a) and end at the terminal time. For each run, we find the final distance between the invader and the territory at the terminal time. Then we calculate the average of the final distance over 1000 runs. The result of a testing phase, which is the average final distance over 1000 runs, is collected after every 100 iterations.

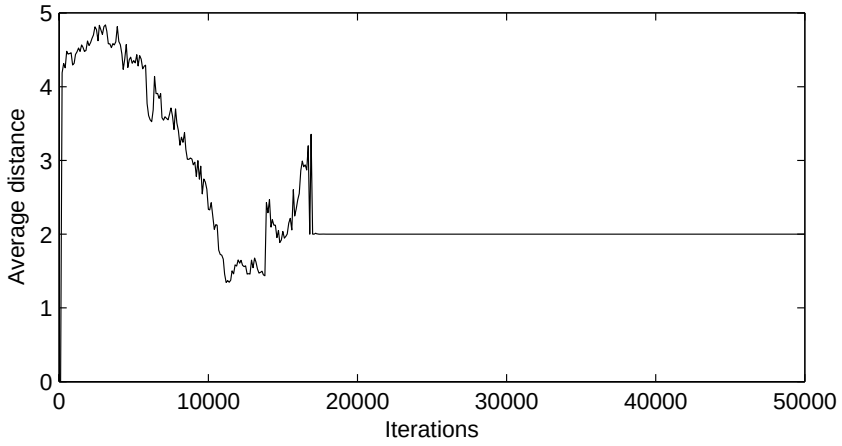
We use the same parameter settings as in the 2×2 grid game for the minimax-Q algorithm. In the first simulation, we test the convergence property by using the same learning algorithm for both players. Fig. 3.7(a) shows the learning performance when both players used the minimax-Q algorithm. In Fig. 3.7(a), the x-axis denotes the number of iterations and the y-axis denotes the result of the testing phase (the average of the final distance over 1000 runs) for every 100 iterations. The learning curve in Fig. 3.7(a) is based on one learning trial including 50000 iterations. From the result in Fig. 3.7(a), the averaged final distance between the invader and the territory converges to 2 after 50000 iterations. The sharp changes in the curve is due to the fact

that each player updates its Q-function by performing linear programming at each iteration. As shown in Fig. 3.6(b), distance 2 is the final distance between the invader and the territory when both player play their Nash equilibrium strategies. Therefore, Fig. 3.7(a) indicates that both players' learned strategies converge close to their Nash equilibrium strategies. Then we use the WoLF-PHC algorithm to simulate again. We set the learning rate α as $1/(4+t/50)$, δ_w as $1/(1+t/5000)$ and δ_l as $4/(1+t/5000)$. We run the simulation for 200000 iterations. The result in Fig. 3.7(b) shows that the averaged final distance converges close to the distance of 2 after the learning.

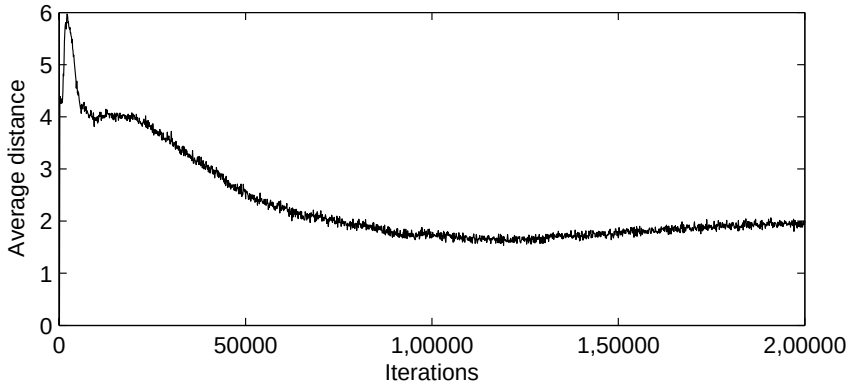
In the second simulation, we fix the invader's strategy to a random-walk strategy, which means that the invader can move up, down, left or right with equal probability. Similar to the first simulation, the learning performance of the algorithms are tested based on the result of a testing phase after every 100 iterations. In a testing phase, we play the game 1000 runs and average the final distance between the invader and the territory at the terminal time for each run over 1000 runs.

We test the learning performance of both algorithms applied for the defender in the game and compare them. The results are shown in Fig. 3.8(a) and 3.8(b). Using the WoLF-PHC algorithm, the defender can intercept the invader further away from the territory (distance of 6.6) than using the minimax-Q algorithm (distance of 5.9). Therefore, based on the rationality criterion in Tab. 3.1, the WoLF-PHC learned defender can achieve better performance than the minimax-Q learned defender as playing against a random-walk invader.

In the above 6×6 grid game, under the convergence criterion, the learning algorithms are tested by finding the averaged final distance for every 100 iterations. The simulation results in Fig. 3.7 showed that, after learning, the averaged final distance converged close to 2 which is also the final distance under the players' Nash equilibrium strategies. Under the rationality criterion, Fig. 3.8 showed that the WoLF-PHC

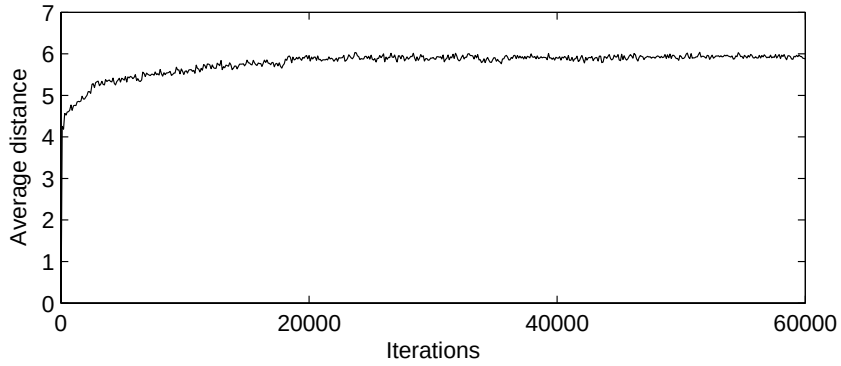


(a) Result of the minimax-Q learned strategy of the defender against the minimax-Q learned strategy of the invader.

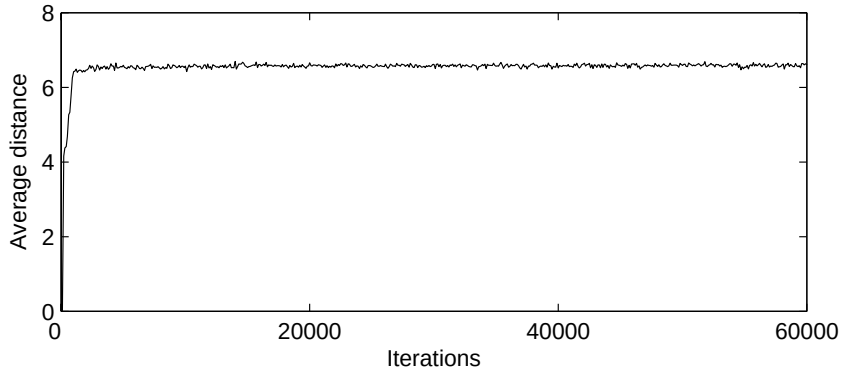


(b) Result of the WoLF-PHC learned strategy of the defender against the WoLF-PHC learned strategy of the invader.

Figure 3.7: Results in the first simulation for the 6×6 grid game



(a) Result of the minimax-Q learned strategy of the defender against the invader using a fixed strategy.



(b) Result of the WoLF-PHC learned strategy of the defender against the invader using a fixed strategy.

Figure 3.8: Results in the second simulation for the 6×6 grid game

learned defender intercepted the random-walk invader at the averaged final distance 6.6 than the minimax-Q learned defender at the averaged final distance 5.9 after learning.

3.4 Summary

In this chapter, we presented and compared four multi-agent reinforcement learning algorithms in stochastic games. The comparison is based on three properties: applicability, rationality and convergence and was shown in Table 3.1.

Then we proposed a grid game of guarding a territory as a two-player zero-sum stochastic game. The invader and the defender try to learn to play against each other using multi-agent reinforcement learning algorithms. The minimax-Q algorithm and WoLF-PHC algorithm were applied to the game. The comparison of these two algorithms was studied and illustrated in simulation results. Both the minimax-Q algorithm and the WoLF-PHC algorithm showed the convergence to the players' Nash equilibrium strategies in the game of guarding a territory for the 2×2 and 6×6 cases. For the rationality property, the defender with the WoLF-PHC algorithm achieved better performance than the defender with the minimax-Q algorithm when playing against a stationary invader. Although there is no theoretical proof of convergence for the WoLF-PHC algorithm, but simulations showed that the WoLF-PHC satisfied the convergence property for the guarding a territory game.

Chapter 4

Decentralized Learning in Matrix Games

4.1 Introduction

Multi-agent learning algorithms have received considerable attention over the past two decades [6, 45]. Among multi-agent learning algorithms, decentralized learning algorithms have become an attractive research field. Decentralized learning means that there is no central learning strategy for all of the agents. Instead, each agent learns its own strategy. Decentralized learning algorithms can be used for players to learn their Nash equilibria in games with incomplete information [28, 46–48]. When an agent has "incomplete information" it means that the agent does not know its own reward function, nor the other players' strategies nor the other players' reward functions. The agent only knows its own action and the received reward at each time step. The main challenge for designing a decentralized learning algorithm with incomplete information is to prove that the players' strategies converge to a Nash equilibrium.

There are a number of multi-agent learning algorithms proposed in the literature that can be used for two-player matrix games. Lakshmivarahan and Narendra [46] presented a linear reward-inaction approach that can guarantee the convergence to

a Nash equilibrium under the assumption that the game only has strict Nash equilibria in pure strategies. The linear reward-penalty approach, introduced in [47], can converge to the value of the game if the game has Nash equilibria in fully mixed strategies with the proper choice of parameters. Bowling and Veloso proposed a WoLF-IGA approach that can guarantee the convergence to a Nash equilibrium for two-player two-action matrix games and the Nash equilibrium can be in fully mixed strategies or in pure strategies. However, the WoLF-IGA approach is not a completely decentralized learning algorithm since the player has to know its opponent's strategy at each time step. Dahl [49, 50] proposed a lagging anchor approach for two-player zero-sum matrix games that can guarantee the convergence to a Nash equilibrium in fully mixed strategies. But the lagging anchor algorithm is not a decentralized learning algorithm because each player has to know its reward matrix.

In this chapter, we evaluate the learning automata algorithms L_{R-I} [46] and L_{R-P} [47], the gradient ascent algorithm WoLF-IGA [10] and the lagging anchor algorithm [49]. We then propose the new L_{R-I} lagging anchor algorithm. The L_{R-I} lagging anchor algorithm is a combination of learning automata and gradient ascent learning. It is a completely decentralized algorithm and as such, each agent only needs to know its own action and the received reward at each time step. We prove the convergence of the L_{R-I} lagging anchor algorithm to Nash equilibria in two-player two-action general-sum matrix games. Furthermore, the Nash equilibrium can be in games with pure or fully mixed strategies. We then simulate three matrix games to test the performance of our proposed learning algorithm.

The motivation for this research is to develop a decentralized learning algorithm for teams of mobile robots. In particular, we are interested in robots that learn to work together for security applications. We have structured these applications as stochastic games such as the guarding a territory game or the pursuit-evasion game. These games have multiple states and multiple players. In Section 4.3, we make

theoretical advances that prove convergence of our proposed L_{R-I} lagging anchor algorithm for two-player two-action general-sum matrix games. We further extend the works to the grid game introduced by Hu and Wellman [14] and we demonstrate the practical performance of the proposed algorithm.

The main **contribution in this chapter** is

- Propose a decentralized learning algorithm called the L_{R-I} lagging anchor algorithm for matrix games,
- Prove the convergence of the L_{R-I} lagging anchor algorithm to Nash equilibria in two-player two-action general-sum matrix games,
- Propose a practical L_{R-I} lagging anchor algorithm for players to learn their Nash equilibrium strategies in general-sum stochastic games,
- Simulate three matrix games and demonstrate the performance of the proposed practical algorithm in Hu and Wellman [14]'s grid game.

The above contribution is an extension of the work we published in [51]. In [51], we proved the convergence of the L_{R-I} lagging anchor algorithm for two-player two-action zero-sum matrix games. In this chapter, we extend the L_{R-I} lagging anchor algorithm to two-player two-action general-sum matrix games. The works in this chapter have been accepted for publication and will appear in [52].

We first review the multi-agent learning algorithms in matrix games based on the learning automata scheme and the gradient ascent scheme in section 4.2. In Sect. 4.3, we introduce the new L_{R-I} lagging anchor algorithm and provide the proof of convergence to Nash equilibria in two-player two-action general-sum matrix games. Simulations of three matrix games are also illustrated in Sect. 4.3 to show the convergence of our proposed L_{R-I} lagging anchor algorithm. In Sect. 4.4, we

propose a practical L_{R-I} lagging anchor algorithm for stochastic games and show the convergence of this practical algorithm in Hu and Wellman [14]'s grid game.

4.2 Learning in Matrix Games

Learning in matrix games can be expressed as the process of each player updating its strategy according to the received reward from the environment. A learning scheme is used for each player to update its own strategy toward a Nash equilibrium based on the information from the environment. In order to address the limitations of the previously published multi-agent learning algorithms for matrix games, we divide these learning algorithms into two groups. One group is based on learning automata [53] and another group is based on gradient ascent learning [54].

4.2.1 Learning Automata

Learning automation is a learning unit for adaptive decision making in an unknown environment [53,55]. The objective of the learning automation is to learn the optimal action or strategy by updating its action probability distribution based on the environment response. The learning automata approach is a completely decentralized learning algorithm since each agent only knows its own action and the received reward from the environment. The information of the reward function and other agents' strategies is unknown to the agent. We take the match pennies game presented in Example 2.2 for example. Without knowing the reward function in (2.33) and its opponent's strategy, using decentralized learning algorithms, the agent needs to learn its own equilibrium strategy based on the received immediate reward at each time step.

Learning automation can be represented as a tuple (A, r, p, U) where $A = \{a_1, \cdots, a_m\}$ is the player's action set, $r \in [0, 1]$ is the reinforcement signal, p is

the probability distribution over the actions and U is the learning algorithm which is used to update p. We present two typical learning algorithms based on learning automata: the linear reward-inaction (L_{R-I}) algorithm and the linear reward-penalty (L_{R-P}) algorithm.

Linear Reward-Inaction Algorithm

The linear reward-inaction (L_{R-I}) algorithm for player $i(i = 1, ..., n)$ is defined as follows

$$\begin{aligned} p_c^i(k+1) &= p_c^i(k) + \eta r^i(k)(1 - p_c^i(k)) \text{ if } a_c \text{ is the current action at } k \\ p_j^i(k+1) &= p_j^i(k) - \eta r^i(k) p_j^i(k) \qquad \text{for all } a_j^i \neq a_c^i \end{aligned} \tag{4.1}$$

where k is the time step, the superscripts and subscripts on p denote different players and each player's different action respectively, $0 < \eta < 1$ is the learning parameter, $r^i(k)$ is the response of the environment given player i's action a_c^i at k and p_c^i is the probability distribution over player i's action $a_c^i(c = 1, \cdots, m)$.

The learning procedure for the L_{R-I} algorithm is listed as follows:

Algorithm 4.1 L_{R-I} algorithm for player i

1: Initialize $\pi_i(a_i)$ for $a_i \in A_i$. Choose the step size η.
2: **for** Each iteration **do**
3: Select action a_c at current state s based on the current strategy $\pi_i(\cdot)$
4: Take action a_c and observe the reward r
5: Update player i's policy $\pi_i(\cdot)$

$$\begin{aligned} p_c^i(k+1) &= p_c^i(k) + \eta r^i(k)(1 - p_c^i(k)) \text{ if } a_c \text{ is the current action at } k \\ p_j^i(k+1) &= p_j^i(k) - \eta r^i(k) p_j^i(k) \qquad \text{for all } a_j^i \neq a_c^i \end{aligned}$$

6: **end for**

For a common payoff game with n players or a two-player zero-sum game, if each player uses the L_{R-I} algorithm, then the L_{R-I} algorithm guarantees the convergence

to strict Nash equilibria in pure strategies [46, 53]. This convergence is under the assumption that the game only has strict Nash equilibria in pure strategies. If the game has Nash equilibria in mixed strategies, then there is no guarantee of convergence to the Nash equilibria.

Example 4.1. We present two examples to show the learning performance of the L_{R-I} algorithm. The first game is the modified matching pennies game in Example 2.3. This game has a Nash equilibrium in pure strategies which are both players' first actions. We apply the L_{R-I} algorithm for both players. We set the step size η as 0.001 and run the simulation for 30000 iterations. Figure 4.1 shows the players' learning process where p_1 denotes the probability of player 1 choosing its first action and q_1 denotes the probability of player 2 choosing its first action. The Nash equilibrium strategies in this game are both players' first actions. Starting from the initial condition $(p_1 = 0.3, q_1 = 0.3)$, Figure 4.1 shows that players' strategies converge to the Nash equilibrium $(p_1 = 1, q_1 = 1)$ after learning.

The second game we simulate is the matching pennies game. In this game, based on the study in Example 2.2, there exists a Nash equilibrium in mixed strategies. Given the reward function in (2.33), the players' Nash equilibrium strategies are $(p_1 = 0.5, q_1 = 0.5)$. Similar to the previous example, we set the step size η as 0.001. We run the simulation for 30000 iterations to test if the players' strategies converge to the Nash equilibrium after the simulation. Figure 4.2 shows the result. The players' strategies starts from $(p_1 = 0.3, q_1 = 0.3)$ and runs circles around the equilibrium point $(p_1 = 0.5, q_1 = 0.5)$.

The above two examples show that the L_{R-I} algorithm can be applied to the game that has Nash equilibria in pure strategies, and is not applicable for a game with Nash equilibria in mixed strategies.

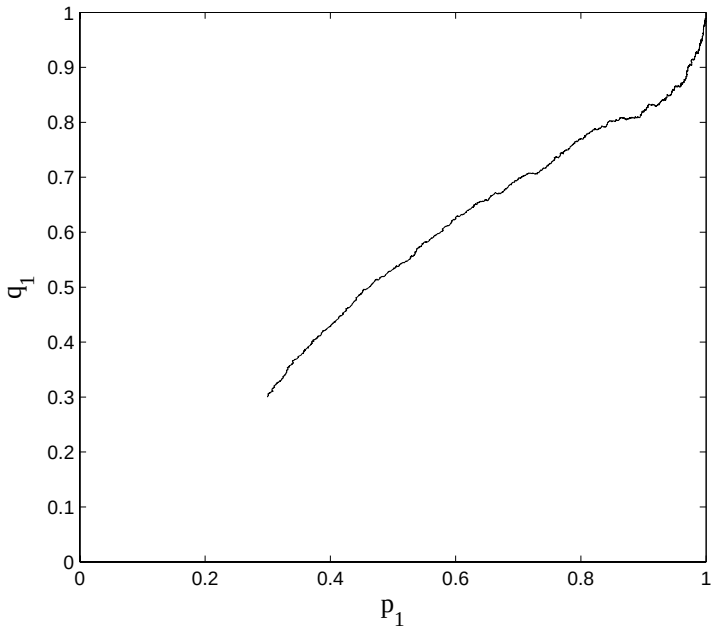


Figure 4.1: Players' learning trajectories using L_{R-I} algorithm in the modified matching pennies game

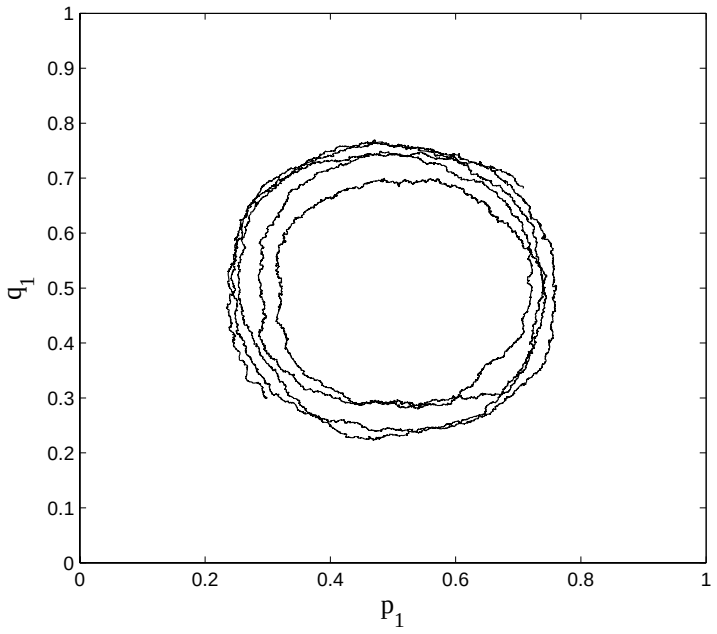


Figure 4.2: Players' learning trajectories using L_{R-I} algorithm in the matching pennies game

Linear Reward-Penalty Algorithm

The linear reward-penalty (L_{R-P}) algorithm for player i is defined as follows

$$\begin{aligned} p_c^i(k+1) &= p_c^i(k) + \eta_1 r^i(k)[1 - p_c^i(k)] - \eta_2[1 - r^i(k)]p_c^i(k) \\ p_j^i(k+1) &= p_j^i(k) - \eta_1 r^i(k)p_j^i(k) + \eta_2[1 - r^i(k)]\left[\frac{1}{m-1} - p_j^i(k)\right] \text{ (for all } a_j^i \neq a_c^i \text{)} \end{aligned} \tag{4.2}$$

where a_c^i is the current action that player i has taken, $0 < \eta_1, \eta_2 < 1$ are learning parameters and m is the number of actions in the player's action set.

In a two-player zero-sum matrix game, if each player uses the L_{R-P} and chooses $\eta_2 < \eta_1$, then the L_{R-P} algorithm can be made to converge arbitrarily close to the optimal solution [47].

The learning procedure for the L_{R-P} algorithm is listed as follows:

Algorithm 4.2 L_{R-P} algorithm for player i

1: Initialize $\pi_i(a_i)$ for $a_i \in A_i$. Choose the step size η_1 and η_2 .
2: **for** Each iteration **do**
3: Select action a_c at current state s based on the current strategy $\pi_i(\cdot)$
4: Take action a_c and observe the reward r
5: Update player i's policy $\pi_i(\cdot)$

$$\begin{aligned} p_c^i(k+1) &= p_c^i(k) + \eta_1 r^i(k)[1 - p_c^i(k)] - \eta_2[1 - r^i(k)]p_c^i(k) \\ p_j^i(k+1) &= p_j^i(k) - \eta_1 r^i(k)p_j^i(k) \\ &\quad + \eta_2[1 - r^i(k)]\left[\frac{1}{m-1} - p_j^i(k)\right] \text{ (for all } a_j^i \neq a_c^i \text{)} \end{aligned}$$

6: **end for**

Example 4.2. We first test the learning convergence for the matching pennies game. We set the step size η_1 as 0.001 and η_2 as 0.0005. We run the simulation for 30000 iterations. Simulation result in Fig. 4.3 shows that the players' learning strategies are

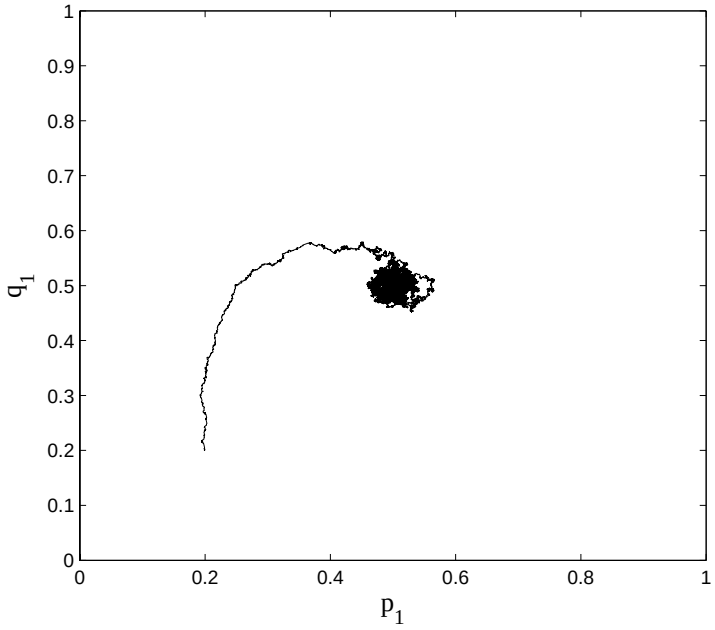


Figure 4.3: Players' learning trajectories using L_{R-P} algorithm in the matching pennies game

moving close to the Nash equilibrium strategies ($p_1 = 0.5, q_1 = 0.5$). We also test the learning convergence for modified matching pennies game. We use the same parameter settings and run the simulation for 30000 iterations. Fig. 4.4 shows that the players' strategies failed to converge to the Nash equilibrium strategies ($p_1 = 1, q_1 = 1$).

Based on the above two examples, we can verify that the L_{R-P} algorithm converges to the Nash equilibria if the game has Nash equilibria in mixed strategies, and fails to converge to the Nash equilibria if the game only has Nash equilibria in mixed strategies.

4.2.2 Gradient Ascent Learning

Gradient ascent learning can be used to update the player's strategy in the direction of the current gradient [54]. At each iteration, the player will adjust its strategy based on

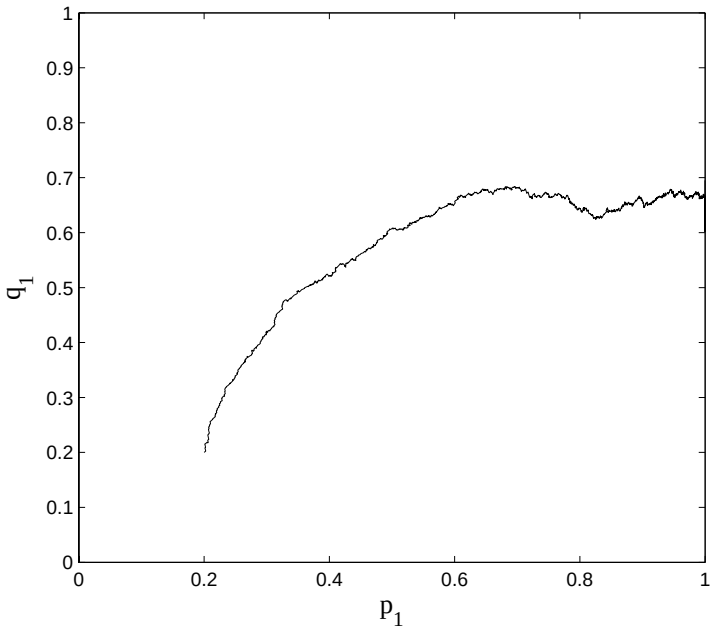


Figure 4.4: Players' learning trajectories using L_{R-P} algorithm in the modified matching pennies game

its gradient in order to increase its expected reward in the long run. Using a gradient ascent learning algorithm, Singh et al. [54] showed that the players' strategies do not converge to Nash equilibria for the general case of matrix games. In other words, there is no learning algorithm that can be applied to all types of matrix games. However, there are a number of gradient ascent learning algorithms that can guarantee the convergence to Nash equilibria for specific matrix games such as two-player matrix games. We present two gradient ascent learning algorithms that can guarantee the convergence to the Nash equilibria in two-player two-action general-sum matrix games and two-player zero-sum matrix games. They are the WoLF-IGA algorithm [10] and the lagging anchor algorithm [49].

WoLF-IGA Algorithm

The "Win or Learn Fast" infinitesimal gradient ascent (WoLF-IGA) algorithm was introduced by Bowling and Veloso [10] for two-player two-action matrix games. As a gradient ascent learning algorithm, the WoLF-IGA algorithm allows the player to update its strategy based on the current gradient and a variable learning rate. The value of the learning rate is smaller, when the player is winning and the learning rate is larger when the player is losing. We use the term p_1 as the probability of player 1 choosing the first action. Then $1 - p_1$ is the probability of player 1 choosing the second action. Accordingly, q_1 is the probability of player 2 choosing the first action and $1 - q_1$ is the probability of player 2 choosing the second action. The updating rules of the WoLF-IGA algorithm are as follows

$$p_1(k+1) = p_1(k) + \eta\alpha_1(k)\frac{\partial V_1(p_1(k), q_1(k))}{\partial p_1} \tag{4.3}$$

$$q_1(k+1) = q_1(k) + \eta\alpha_2(k)\frac{\partial V_2(p_1(k), q_1(k))}{\partial q_1} \tag{4.4}$$

$$\alpha_1(k) = \begin{cases} \alpha_{\min}, \text{ if } V_1(p_1(k), q_1(k)) > V_1(p_1^*, q_1(k)) \\ \alpha_{\max}, \text{ otherwise} \end{cases}$$

$$\alpha_2(k) = \begin{cases} \alpha_{\min}, \text{ if } V_2(p_1(k), q_1(k)) > V_2(p_1(k), q_1^*) \\ \alpha_{\max}, \text{ otherwise} \end{cases}$$

where η is the step size, $\alpha_i (i = 1, 2)$ is the learning rate for player $i (i = 1, 2)$, $V_i(p_1(k), q_1(k))$ is the expected reward of player i at time k given the current two players' strategy pair $(p_1(k), q_1(k))$, and (p_1^*, q_1^*) are equilibrium strategies for the players. In a two-player two-action matrix game, if each player uses the WoLF-IGA algorithm with $\alpha_{\max} > \alpha_{\min}$, the players' strategies converge to a Nash equilibrium as the step size $\eta \to 0$ [10].

This algorithm is a gradient ascent learning algorithm that can guarantee the convergence to a Nash equilibrium in fully mixed or pure strategies for two-player two-action general-sum matrix games. However, this algorithm is not a decentralized learning algorithm. This algorithm requires the knowledge of $V_1(p_1^*, q_1(k))$ and $V_2(p_1(k), q_1^*)$ in order to choose the learning parameters $\alpha_{\min}$ and $\alpha_{\max}$ accordingly. In order to obtain $V_1(p_1^*, q_1(k))$ and $V_2(p_1(k), q_1^*)$, we need to know each player's reward matrix and its opponent's strategy at time k. Whereas, in a decentralized learning algorithm the agents would only have their own actions and the received reward at time k.

The Lagging Anchor Algorithm

The lagging anchor algorithm for two-player zero-sum games was introduced by Dahl [49]. As a gradient ascent learning method, the lagging anchor algorithm updates the players' strategies according to the gradient. We denote player 1's strategy as a vector $\mathbf{v} = [p_1, p_2, \cdots, p_{m_1}]^T$ which is the probability distribution over all the possible actions of player 1. Accordingly, player 2's strategy is denoted as a vector $\mathbf{w} = [q_1, q_2, \cdots, q_{m_2}]^T$ where m_1 and m_2 denote the number of actions. The updating rules are listed as follows

$$\begin{aligned}
\mathbf{v}(k+1) &= \mathbf{v}(k) + \eta \mathbf{P}_{m_1} R_1 Y(k) + \eta\eta_d(\bar{\mathbf{v}}(k) - \mathbf{v}(k)) \\
\bar{\mathbf{v}}(k) &= \bar{\mathbf{v}}(k) + \eta\eta_d(\mathbf{v}(k) - \bar{\mathbf{v}}(k)) \\
\mathbf{w}(k+1) &= \mathbf{w}(k) + \eta \mathbf{P}_{m_2} R_2 X(k) + \eta\eta_d(\bar{\mathbf{w}}(k) - \mathbf{w}(k)) \\
\bar{\mathbf{w}}(k) &= \bar{\mathbf{w}}(k) + \eta\eta_d(\mathbf{w}(k) - \bar{\mathbf{w}}(k))
\end{aligned} \tag{4.5}$$

where η is the step size, $\eta_d > 0$ is the anchor drawing factor, $\mathbf{P}_{m_i} = \mathbf{I}_{m_i} - (1/m_i)\mathbf{1}_{m_i}\mathbf{1}_{m_i}^T$ is a matrix used to maintain the summation of the elements in the vector $\mathbf{v}$ or $\mathbf{w}$ to be one. The term $Y(k)$ is a unit vector corresponding to the actions of player 2. If

the m_2th action in player 2's action set is selected at time k, then the m_2th element in $Y(k)$ is set to 1 and the other elements in $Y(k)$ are zeros. Similarly, $X(k)$ is the unit vector corresponding to the actions of player 1 and R_1 and R_2 are the reward matrices for player 1 and 2 respectively. In (4.5), $\bar{\mathbf{v}}$ and $\bar{\mathbf{w}}$ are the anchor parameters for $\mathbf{v}$ and $\mathbf{w}$ respectively which can be represented as the weighted average of the players' strategies. In a two-player zero-sum game with only Nash equilibria in fully mixed strategies, if each player uses the lagging anchor algorithm, then the players' strategies converge to a Nash equilibrium as the step size $\eta \to 0$ [50].

This algorithm guarantees the convergence to a Nash equilibrium in fully mixed strategies. However, the convergence to a Nash equilibrium in pure strategies has never been discussed. Furthermore, the lagging anchor algorithm in (4.5) requires full information of the player's reward matrices R_1 and R_2. Therefore, the lagging anchor algorithm is not a decentralized learning algorithm.

Table 4.1 compares these algorithms based on the allowable number of actions for each player, the convergence to pure strategies or fully mixed strategies and the level of decentralization. From this table, only the WoLF-IGA algorithm can guarantee the convergence to both pure and mixed strategy Nash equilibrium. But it is not a decentralized learning algorithm. Although the L_{R-I} algorithm and the L_{R-P} algorithm are decentralized learning algorithms, neither of them can guarantee the convergence to both pure and mixed strategy NE. Therefore, the main motivation of this work is to design a decentralized learning algorithm which can guarantee the convergence to both pure and mixed strategy NE as shown in Table 4.1.

Table 4.1: Comparison of learning algorithms in matrix games

Applicability	Existing algorithms				Our proposed algorithm
	L_{R-I}	L_{R-P}	WoLF-IGA	lagging anchor	L_{R-I} lagging anchor
Allowable actions	no limit	2 actions	2 actions	no limit	2 actions
Convergence	pure NE	fully mixed NE	both	fully mixed NE	both
Decentralized?	Yes	Yes	No	No	Yes

4.3 L_{R-I} Lagging Anchor Algorithm

In this section, we design an L_{R-I} lagging anchor algorithm which is a completely decentralized learning algorithm and can guarantee the convergence to Nash equilibria in both pure and fully mixed strategies. We take the L_{R-I} algorithm defined in (4.1) as the updating law of the player's strategy and add the lagging anchor term in (4.5). Then the L_{R-I} lagging anchor algorithm for player i is defined as follows

$$\left.\begin{aligned} p_c^i(k+1) &= p_c^i(k) + \eta r^i(k)[1 - p_c^i(k)] + \eta[\bar{p}_c^i(k) - p_c^i(k)] \\ \bar{p}_c^i(k+1) &= \bar{p}_c^i(k) + \eta[p_c^i(k) - \bar{p}_c^i(k)] \end{aligned}\right\} \text{if } a_c^i \text{ is the action taken at time } k$$

$$\left.\begin{aligned} p_j^i(k+1) &= p_j^i(k) - \eta r^i(k) p_j^i(k) + \eta[\bar{p}_j^i(k) - p_j^i(k)] \\ \bar{p}_j^i(k+1) &= \bar{p}_j^i(k) + \eta[p_j^i(k) - \bar{p}_j^i(k)] \end{aligned}\right\} \text{for all } a_j^i \neq a_c^i \qquad (4.6)$$

where η is the step size and $(\bar{p}_c^i, \bar{p}_j^i)$ are the lagging parameters for (p_c^i, p_j^i). The idea behind the L_{R-I} lagging anchor algorithm is that we consider both the player's current strategy and the long-term average of the player's previous strategies at the

same time. We expect that the player's current strategy and the long-term average strategy will be drawn towards the equilibrium point during learning.

To analyze the above L_{R-I} lagging anchor algorithm, we use the ordinary differential equation (ODE) approach. The behavior of the learning algorithm can be approximated by ODEs as the step size goes to zero [56]. We first separate the equations in (4.6) into two parts: the L_{R-I} part and the lagging anchor part. The L_{R-I} part is the same equation as in (4.1). Thathachar and Sastry [56] provided the procedure to find the equivalent ODEs for the L_{R-I} part. The analysis on approximating ODEs for the L_{R-I} algorithm can be obtained in Appendix A in [56]. We skip the details here. According to 4.1, the obtained ODE for the L_{R-I} algorithm is given as

$$\dot{p}_c^i = \sum_{j=1}^{m_i} p_c^i p_j^i (d_c^i - d_j^i) \tag{4.7}$$

where d_c^i is the expected reward of player i given that player i is choosing action a_c^i and the other players are following their current strategies.

Combining the above ODE of the L_{R-I} algorithm in (4.7) with the ODEs for the lagging anchor part of our algorithm, we can find the equivalent ODEs for our L_{R-I} lagging anchor algorithm as

$$\begin{aligned} \dot{p}_c^i &= \sum_{j=1}^{m_i} p_c^i p_j^i (d_c^i - d_j^i) + (\bar{p}_c^i - p_c^i) \\ \dot{\bar{p}}_c^i &= p_c^i - \bar{p}_c^i \end{aligned} \tag{4.8}$$

Based on the above ODEs for our proposed L_{R-I} lagging anchor algorithm, we now present the following theorem.

Theorem 4.1. *We consider a two-player two-action general-sum matrix game and assume the game only has a Nash equilibrium in fully mixed strategies or strict Nash equilibria in pure strategies. If both players follow the L_{R-I} lagging anchor algorithm,*

when the step size $\eta \to 0$, then the following is true regarding the asymptotic behavior of the algorithm.

- *All Nash equilibria are asymptotically stable.*
- *Any equilibrium point which is not a Nash equilibrium is unstable.*

Proof: Given a two-player two-action general-sum game as

$$R_1 = \begin{bmatrix} r_{11} & r_{12} \\ \\ r_{21} & r_{22} \end{bmatrix}, R_2 = \begin{bmatrix} c_{11} & c_{12} \\ \\ c_{21} & c_{22} \end{bmatrix}. \tag{4.9}$$

We denote p_1 as the probability of player 1 taking its first action and q_1 as the probability of player 2 taking its first action. Then the L_{R-I} lagging anchor algorithm becomes

$$\begin{aligned} \dot{p}_1 &= \sum_{j=1}^{2} p_1 p_j (d_1^1 - d_j^1) + (\bar{p}_1 - p_1) \\ \dot{\bar{p}}_1 &= p_1 - \bar{p}_1 \\ \dot{q}_1 &= \sum_{j=1}^{2} q_1 q_j (d_1^2 - d_j^2) + (\bar{q}_1 - q_1) \\ \dot{\bar{q}}_1 &= q_1 - \bar{q}_1 \end{aligned} \tag{4.10}$$

where $d_1^1 = r_{11}q_1 + r_{12}(1 - q_1)$, $d_2^1 = r_{21}q_1 + r_{22}(1 - q_1)$, $d_1^2 = c_{11}p_1 + c_{21}(1 - p_1)$ and $d_2^2 = c_{12}p_1 + c_{22}(1 - p_1)$. Then (4.10) becomes

$$\begin{aligned} \dot{p}_1 &= p_1(1 - p_1)[u_1 q_1 + r_{12} - r_{22}] + (\bar{p}_1 - p_1) \\ \dot{\bar{p}}_1 &= p_1 - \bar{p}_1 \\ \dot{q}_1 &= q_1(1 - q_1)[u_2 p_1 + c_{21} - c_{22}] + (\bar{q}_1 - q_1) \\ \dot{\bar{q}}_1 &= q_1 - \bar{q}_1 \end{aligned} \tag{4.11}$$

where $u_1 = r_{11} - r_{12} - r_{21} + r_{22}$ and $u_2 = c_{11} - c_{12} - c_{21} + c_{22}$. If we let the right hand side of the above equation equal to zero, we then get the equilibrium points of the above equations as $(p_1^*, q_1^*) = (0,0), (0,1), (1,0), (1,1), ((c_{22} - c_{21})/u_2, (r_{22} - r_{12})/u_1)$. To study the stability of the above learning dynamics, we use a linear approximation of the above equations around the equilibrium point $(p_1^*, q_1^*, p_1^*, q_1^*)$. Then the linearization matrix J is given as

$$J_{(p_1^*, q_1^*)} = \begin{bmatrix} (1-2p_1^*)(u_1 q_1^* + r_{12} - r_{22}) - 1 & 1 & p_1^*(1-p_1^*)u_1 & 0 \\ 1 & -1 & 0 & 0 \\ q_1^*(1-q_1^*)u_2 & 0 & (1-2q_1^*)(u_2 p_1^* + c_{21} - c_{22}) - 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \tag{4.12}$$

If we substitute each of the equilibrium points $(0,0), (0,1), (1,0), (1,1)$ into (4.12), we get

$$J_{\text{pure}} = \begin{bmatrix} -e_1 - 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & -e_2 - 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \tag{4.13}$$

where

$$e_1 = r_{22} - r_{12}, e_2 = c_{22} - c_{21} \quad \text{for } (0,0); \tag{4.14}$$

$$e_1 = r_{21} - r_{11}, e_2 = c_{21} - c_{22} \quad \text{for } (0,1); \tag{4.15}$$

$$e_1 = r_{12} - r_{22}, e_2 = c_{12} - c_{11} \quad \text{for } (1,0); \tag{4.16}$$

$$e_1 = r_{11} - r_{21}, e_2 = c_{11} - c_{12} \quad \text{for } (1,1). \tag{4.17}$$

The eigenvalues of the above matrix J_{pure} are $\lambda_{1,2} = 0.5[-(e_1 + 2) \pm \sqrt{e_1^2 + 4})]$ and $\lambda_{3,4} = 0.5[-(e_2 + 2) \pm \sqrt{e_2^2 + 4})]$. In order to obtain a stable equilibrium point, the real parts of the eigenvalues of J_{pure} must be negative. Therefore, the equilibrium point is asymptotically stable if

$$\begin{aligned} 0.5[-(e_{1,2} + 2) \pm \sqrt{e_{1,2}^2 + 4})] < 0 \Rightarrow \\ e_{1,2} + 2 > \sqrt{e_{1,2}^2 + 4} \Rightarrow \\ e_{1,2} > 0 \end{aligned} \tag{4.18}$$

For the equilibrium point $((c_{22} - c_{21})/u_2, (r_{22} - r_{12})/u_1)$, the linearization matrix becomes

$$J_{\text{mixed}} = \begin{bmatrix} -1 & 1 & p_1^*(1 - p_1^*)u_1 & 0 \\ 1 & -1 & 0 & 0 \\ q_1^*(1 - q_1^*)u_2 & 0 & -1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix}. \tag{4.19}$$

The characteristic equation of the above matrix is

$$\lambda^4 + 4\lambda^3 + (4 + e_3)\lambda^2 + 2e_3\lambda + e_3 = 0 \tag{4.20}$$

where $e_3 = -p_1^*(1 - p_1^*)q_1^*(1 - q_1^*)u_1u_2$. We set up the Routh table to analyze the

locations of the roots in (4.20) as follows

λ^4	1	$4+e_3$	e_3
λ^3	4	$2e_3$	
λ^2	$4+0.5e_3$	e_3	
λ^1	$(e_3^2+4e_3)/(4+0.5e_3)$		
λ^0	e_3		

(4.21)

Based on the Routh-Hurwitz stability criterion, if (4.20) is stable, then all the coefficients of (4.20) must be positive and all the elements in the first column of the Routh table in (4.21) are positive. In order to meet the Routh-Hurwitz stability criterion, we must have $e_3 > 0$. Therefore, the equilibrium point $((c_{22}-c_{21})/u_2, (r_{22}-r_{12})/u_1)$ is asymptotically stable if

$$e_3 = -p_1^*(1-p_1^*)q_1^*(1-q_1^*)u_1u_2 > 0 \Rightarrow u_1u_2 < 0 \tag{4.22}$$

Case 1: strict Nash equilibrium in pure strategies

We first consider that the game only has strict Nash equilibrium in pure strategies. Without loss of generality, we assume that the Nash equilibrium in this case is both players' first actions. According to the definition of a strict Nash equilibrium in (2.22), if the Nash equilibrium strategies are both players' first actions, we get

$$r_{11} > r_{21}, c_{11} > c_{12}. \tag{4.23}$$

Since the Nash equilibrium in this case is the equilibrium point $(1,1)$, we can get $e_1 = r_{11} - r_{21} > 0$ and $e_2 = c_{11} - c_{12} > 0$ based on (4.17) and (4.23). Therefore, the

stability condition in (4.18) is satisfied and the equilibrium point $(1, 1)$ which is the Nash equilibrium in this case is asymptotically stable.

We now test the other equilibrium points. We first consider the equilibrium point $((c_{22} - c_{21})/u_2, (r_{22} - r_{12})/u_1)$. According to (4.22), if this equilibrium point is stable, we must have $u_1 u_2 < 0$. To be a valid inner point in the probability space (unit square), the equilibrium point $((c_{22} - c_{21})/u_2, (r_{22} - r_{12})/u_1)$ must satisfy

$$\begin{cases} 0 < (c_{22} - c_{21})/u_2 < 1 \\ 0 < (r_{22} - r_{12})/u_1 < 1 \end{cases} \tag{4.24}$$

If $u_1 u_2 < 0$, we get

$$\begin{cases} r_{11} > r_{21}, r_{22} > r_{12} \\ c_{11} < c_{12}, c_{22} < c_{21} \end{cases} \text{ if } u_1 > 0, u_2 < 0 \tag{4.25}$$

$$\begin{cases} r_{11} < r_{21}, r_{22} < r_{12} \\ c_{11} > c_{12}, c_{22} > c_{21} \end{cases} \text{ if } u_1 < 0, u_2 > 0 \tag{4.26}$$

However, the conditions in (4.25) and (4.26) conflict with the inequalities in (4.23). Therefore, the inequality $u_1 u_2 < 0$ will not hold and the equilibrium point $((c_{22} - c_{21})/u_2$,$(r_{22} - r_{12})/u_1)$ is unstable in Case 1.

For the equilibrium points $(0, 1)$ and $(1, 0)$, based on (4.15), (4.16) and (4.18), the stability conditions are $r_{21} > r_{11}, c_{21} > c_{22}$ for $(0, 1)$ and $r_{12} > r_{22}, c_{12} > c_{11}$ for $(1, 0)$. However, these stability conditions conflict with the inequalities $r_{11} > r_{21}, c_{11} > c_{12}$ in (4.23). Therefore, the equilibrium points $(0, 1)$ and $(1, 0)$ are unstable in Case 1.

For the equilibrium point $(0, 0)$, the stability condition is $r_{22} > r_{12}, c_{22} > c_{21}$ based on (4.14) and (4.18). From (2.22), we can find that this stability condition also meets

the requirement for a strict Nash equilibrium (both players' second actions) in (2.22). Therefore, the equilibrium point $(0, 0)$ is stable only if it is also a Nash equilibrium point.

Thus, the Nash equilibrium point is asymptotically stable while any equilibrium point which is not a Nash equilibrium is unstable.

Case 2: Nash equilibrium in fully mixed strategies

We now consider that the game only has Nash equilibrium in fully mixed strategies. Singh et al. [54] showed that a Nash equilibrium in fully mixed strategies for a two-player two-action general-sum matrix game has the from of

$$(p_1^{NE}, q_1^{NE}) = \left[\frac{c_{22} - c_{21}}{u_2}, \frac{r_{22} - r_{12}}{u_1}\right] \tag{4.27}$$

where (p_1^{NE}, q_1^{NE}) denotes the Nash equilibrium strategies over players' first actions which happens to be the equilibrium point of (4.11). According to (4.22), the equilibrium point $((c_{22} - c_{21})/u_2, (r_{22} - r_{12})/u_1)$ is asymptotically stable if $u_1 u_2 < 0$. If we assume $u_1 u_2 > 0$, we get

$$\begin{cases} 0 < (c_{22} - c_{21})/u_2 < 1 \\ 0 < (r_{22} - r_{12})/u_1 < 1 \end{cases} (u_1 u_2 > 0) \tag{4.28}$$

The above equation (4.28) can be derived as follows

$$\begin{cases} r_{11} > r_{21}, r_{22} > r_{12} \\ c_{11} > c_{12}, c_{22} > c_{21} \end{cases} \text{if } u_1 > 0, u_2 > 0 \tag{4.29}$$

$$\begin{cases} r_{11} < r_{21}, r_{22} < r_{12} \\ c_{11} < c_{12}, c_{22} < c_{21} \end{cases} \text{if } u_1 < 0, u_2 < 0 \tag{4.30}$$

According to (2.22), the above equations contain multiple Nash equilibria in pure strategies: $(p_1^{NE}, q_1^{NE}) = (1,1), (0,0)$ if $u_1 > 0, u_2 > 0$ and $(p_1^{NE}, q_1^{NE}) = (0,1), (1,0)$ if $u_1 < 0, u_2 < 0$. However, under our assumption, the game in Case 2 only has a Nash equilibrium in fully mixed strategies and Nash equilibria in pure strategies do not exist. Therefore, we always have $u_1 u_2 < 0$ in Case 2 and the equilibrium point $((c_{22} - c_{21})/u_2, (r_{22} - r_{12})/u_1)$, which is also the Nash equilibrium point, is asymptotically stable.

For the other equilibrium points, based on (4.14)-(4.17) and (4.18), the stability conditions become

$$r_{22} > r_{12}, c_{22} > c_{21} \quad \text{for } (0,0); \tag{4.31}$$

$$r_{21} > r_{11}, c_{21} > c_{22} \quad \text{for } (0,1); \tag{4.32}$$

$$r_{12} > r_{22}, c_{12} > c_{11} \quad \text{for } (1,0); \tag{4.33}$$

$$r_{11} > r_{21}, c_{11} > c_{12} \quad \text{for } (1,1). \tag{4.34}$$

As already noted, the game in Case 2 only has a Nash equilibrium in fully mixed strategies and we always have $u_1 u_2 < 0$. Then the inequalities in (4.25) and (4.26) are true in Case 2. However, the stability conditions in (4.31)-(4.34) for the equilibrium points $(0,0), (0,1), (1,0), (1,1)$ conflict with the inequalities in (4.25) and (4.26). Therefore, the equilibrium points other than $((c_{22} - c_{21})/u_2, (r_{22} - r_{12})/u_1)$ are unstable in this case.

Thus we can conclude that the Nash equilibrium point is asymptotically stable while the other equilibrium points are unstable in Case 2.

Table 4.2: Examples of two-player matrix games

(a) Matching Pennies	(b) Prisoner's Dilemma	(c) Rock-Paper-Scissors
$R_1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$,	$R_1 = \begin{bmatrix} 5 & 0 \\ 10 & 1 \end{bmatrix}$,	$R_1 = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$,
$R_2 = -R_1$	$R_2 = (R_1)^T$	$R_2 = -R_1$.
NE in fully mixed strategies	NE in pure strategies	NE in fully mixed strategies

4.3.1 Simulation

We now simulate three matrix games to show the performance of the proposed L_{R-I} lagging anchor algorithm. The first game is the matching pennies game as shown in Example 2.2. This game is a two-player zero-sum game and each player has two actions: Heads or Tails. If both players choose the same action, then player 1 gets a reward 1 and player 2 gets a reward -1. If the actions are different, then player 1 gets -1 and player 2 gets 1. Based on the reward matrix in Table 4.2 (a) and the solutions in Example 2.2, the Nash equilibrium in this game is in fully mixed strategies such that each player plays Heads and Tails with a probability of 0.5. We set the step size $\eta = 0.001$ in (4.6) and $p_1(0) = q_1(0) = 0.2$. We run the simulation for 30000 iterations. In Fig. 4.5, starting from (0.2, 0.2), the players' probabilities of taking their first actions move close to the Nash equilibrium point (0.5, 0.5) as the learning proceeds.

The second game we simulate is a two-player general-sum game called the prisoner's dilemma. In this game, we have two players and each player has two actions:

confess or stay silent. A player receives a reward of 10 if it confesses and the other player remains silent, or receives a reward of 0 if it remains silent and the other player confesses. If both players stay silent, each player receives a reward of 5. If they both confess, each player receives a reward of 1. The reward matrix is shown in Table 4.2 (b) where one player's reward matrix is the transpose of the other player's reward matrix. This game has a unique Nash equilibrium in pure strategies which is both players confessing. We set the step size $\eta = 0.001$ in (4.6) and $p_1(0) = q_1(0) = 0.5$. We run the simulation for 30000 iterations. Figure 4.6 shows that the players' strategies move close to the Nash equilibrium strategies (both players' second actions) as the learning proceeds.

The third game we simulate is the rock-paper-scissors game. This game has two players and each player has three actions: rock, paper and scissors. A winner in the game is determined by the following rules: paper defeats rock, scissors defeat paper, and rock defeats scissors. The winner receives a reward of 1 and the loser receives -1. If both players choose the same action, each player gets 0. The reward matrix is shown in Table 4.2 (c). This game has a Nash equilibrium in fully mixed strategies which is each player choosing any action with the same probability of 1/3. We set the step size $\eta = 0.001$, $p_1(0) = q_1(0) = 0.6$ and $p_2(0) = q_2(0) = 0.2$. We run the simulation for 50000 iterations. Although we only prove the convergence for two-player two-action games, the result in Fig. 4.7 shows that the proposed L_{R-I} lagging anchor algorithm may be applicable to a two-player matrix game with more than two actions.

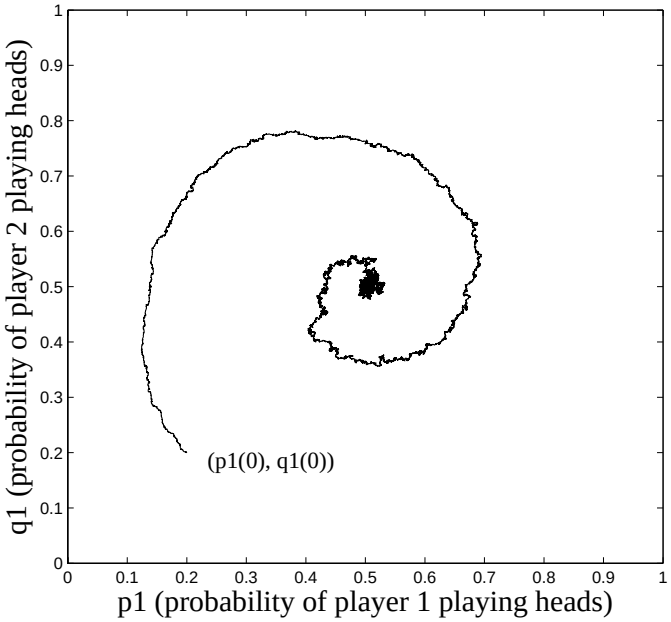


Figure 4.5: Trajectories of players' strategies during learning in matching pennies

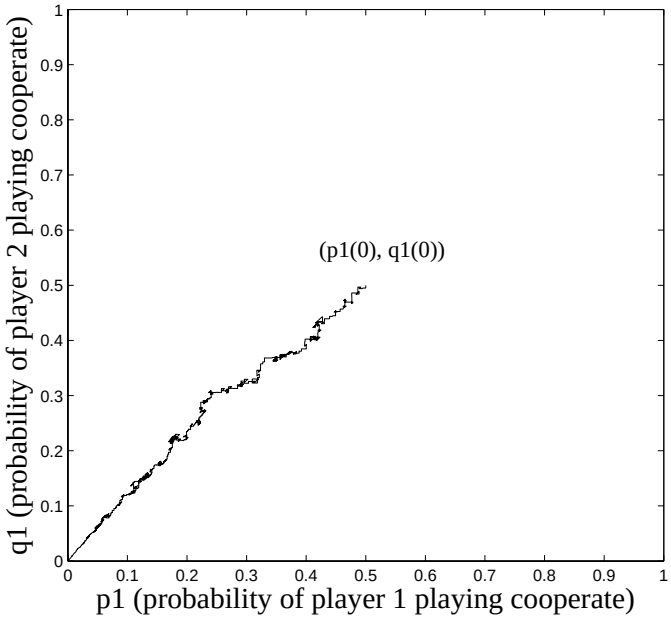


Figure 4.6: Trajectories of players' strategies during learning in prisoner's dilemma

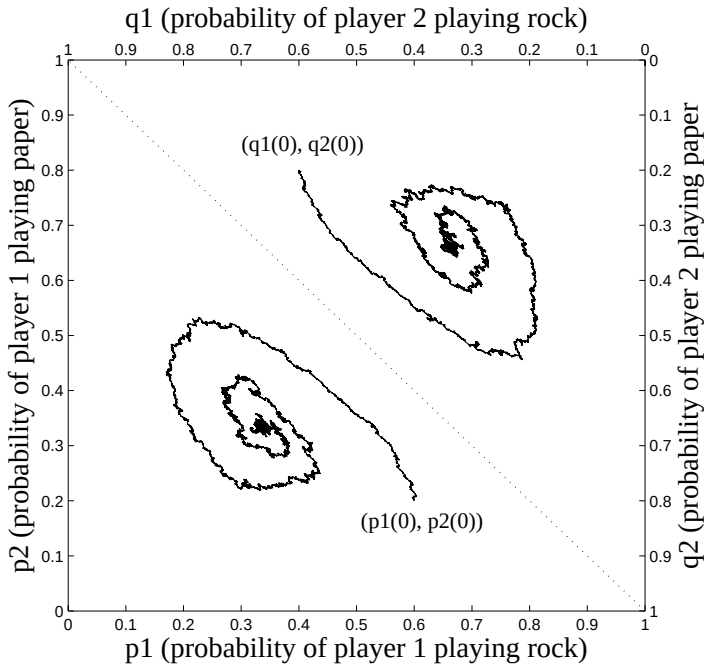


Figure 4.7: Trajectories of players' strategies during learning in rock-paper-scissors

4.4 Extension of Matrix Games to Stochastic Games

The proposed L_{R-I} lagging anchor algorithm is designed based on matrix games. In the literature, decentralized learning algorithms are also applied to stochastic games [10, 28, 48]. In [28, 48], if each player uses the linear reward-inaction algorithm, the players' strategies will converge to Nash equilibria under the assumption that the stochastic game only has Nash equilibria in pure strategies. In [10], a practical decentralized learning algorithm called the WoLF-PHC algorithm is applied to stochastic games. Although there is no convergence proof for the WoLF-PHC algorithm, the empirical results showed the convergence of this algorithm in [10]. In this section, inspired by the WoLF-PHC algorithm, we design a practical decentralized

learning algorithm for stochastic games based on the L_{R-I} lagging anchor approach in (4.6). The practical algorithm is shown in Algorithm 4.3.

Algorithm 4.3 A practical L_{R-I} lagging anchor algorithm for player i

1: Initialize $Q_i(s, a_i) \leftarrow 0$ and $\pi_i(s, a_i) \leftarrow \dfrac{1}{|A_i|}$. Choose the learning rate α, η and the discount factor γ.
2: **for** Each iteration **do**
3: Select action a_c at current state s based on mixed exploration-exploitation strategy
4: Take action a_c and observe the reward r and the subsequent state s'
5: Update $Q_i(s, a_c)$

$$Q_i(s, a_c) = Q_i(s, a_c) + \alpha\big[r_i + \gamma \max_{a'} Q_i(s', a') - Q_i(s, a_c)\big]$$

6: Update the player's policy $\pi_i(s, \cdot)$

$$\begin{aligned}
\pi_i(s, a_c) &= \pi_i(s, a_c) + \eta Q_i(s, a_c)\big[1 - \pi_i(s, a_c)\big] + \eta\big[\bar{\pi}_i(s, a_c) - \pi_i(s, a_c)\big] \\
\bar{\pi}_i(s, a_c) &= \bar{\pi}_i(s, a_c) + \eta\big[\pi_i(s, a_c) - \bar{\pi}_i(s, a_c)\big] \\
\pi_i(s, a_j) &= \pi_i(s, a_j) - \eta Q_i(s, a_c)\pi_i(s, a_j) + \eta\big[\bar{\pi}_i(s, a_j) - \pi_i(s, a_j)\big] \\
\bar{\pi}_i(s, a_j) &= \bar{\pi}_i(s, a_j) + \eta\big[\pi_i(s, a_j) - \bar{\pi}_i(s, a_j)\big] \\
&\text{(for all } a_j \neq a_c\text{)}
\end{aligned}$$

7: **end for**
$\big(Q_i(s, a_i)$ is the action-value function, $\pi_i(s, a_i)$ is the probability of player i taking action a_i at state s and a_c is the current action taken by player i at state $s\big)$

We now apply Algorithm 4.3 to a stochastic game to test the performance. The stochastic game we simulate is a general-sum grid game introduced by Hu and Wellman [14]. The game runs under a 3×3 grid field as shown in Fig. 4.8(a). We have two players whose initial positions are located at the bottom left corner for player 1 and the bottom right corner for player 2. Both players try to reach the goal denoted as "G" in Fig. 4.8(a). Each player has four possible moves which are moving up, down, left or right unless the player is on the sides of the grid. In Hu and Wellman's

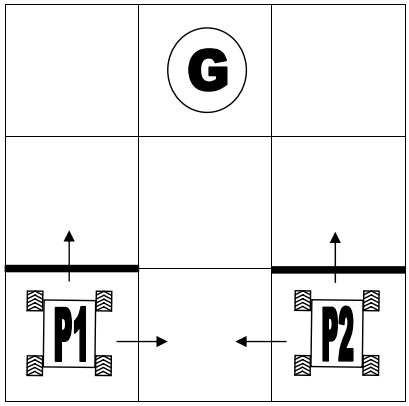
game, the movement that will take the player to a wall is ignored. Since we use the exact same game as Hu and Wellman, the possible actions of hitting a wall have been removed from the players' action sets. For example, if the player is at the bottom left corner, its available moves are moving up or right. If both players move to the same cell at the same time, they will bounce back to their original positions. The two thick lines in Fig. 4.8(a) represent two barriers such that the player can pass through the barrier with a probability of 0.5. For example, if player 1 tries to move up from the bottom left corner, it will stay still or move to the upper cell with a probability of 0.5. The game ends when either one of the players reaches the goal. To reach the goal in minimum steps, the player needs to avoid the barrier and first move to the bottom center cell. Since both players cannot move to the bottom center cell simultaneously, the players need to cooperate such that one of the players has to take the risk and move up. The reward function for player i ($i = 1, 2$) in this game is defined as

$$R_i = \begin{cases} 100 & \text{player } i \text{ reaches the goal} \\ -1 & \text{both players move to the same cell (except the goal)} \\ 0 & \text{otherwise} \end{cases} \tag{4.35}$$

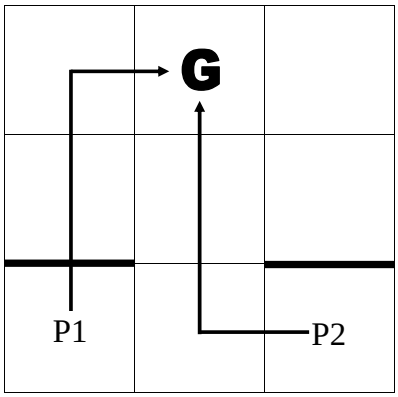
According to [14], this grid game has two Nash equilibrium paths as shown in Fig. 4.8(b) and Fig. 4.8(c). Starting from the initial state, the Nash equilibrium are player 1 moving up and player 2 moving left or player 1 moving right and player 2 moving up.

We set the step size as $\eta = 0.001$, the learning rate as $\alpha = 0.001$ and the discount factor as $\gamma = 0.9$. The mixed exploration-exploitation strategy is chosen such that the player chooses a random action with probability 0.05 and the greedy action with probability 0.95. We run the simulation for 10000 episodes. An episode is when the game starts with the players' initial positions and ends when either one of the players

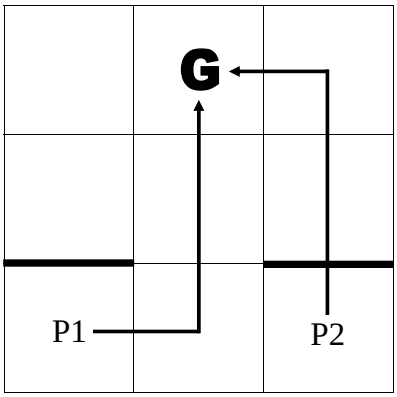
reaches the goal. Figure 4.9 shows the result of two players' learning trajectories. We define p_1 as player 1 's probability of moving up and q_1 as player 2 's probability of moving up from their initial positions. The result in Fig. 4.9 shows that two players' strategies at the initial state converge to one of the two Nash equilibrium strategies (player 1 moving right and player 2 moving up). Therefore, the proposed practical L_{R-I} lagging anchor algorithm may be applicable to general-sum stochastic games.



(a) Grid Game



(b) Nash equilibrium path 1



(c) Nash equilibrium path 2

Figure 4.8: Hu and Wellman's grid game

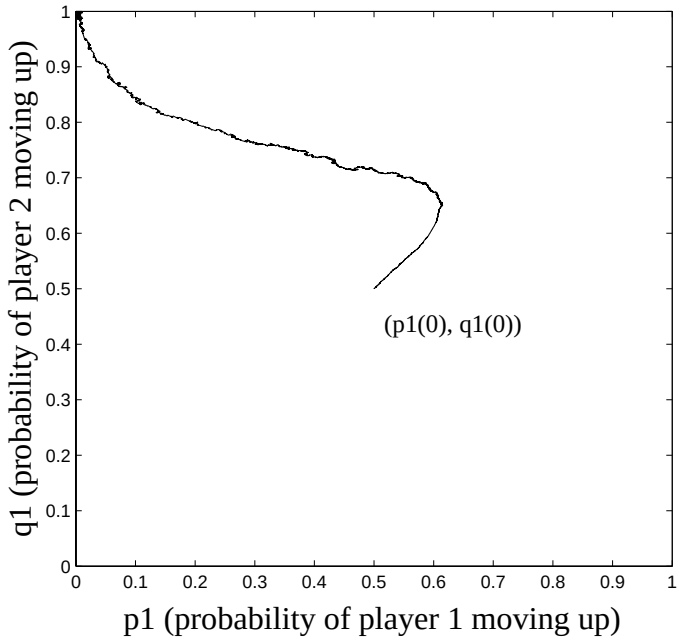


Figure 4.9: Learning trajectories of players' strategies at the initial state in the grid game

4.5 Summary

In this chapter, we investigated the existing learning algorithms for matrix games. The analysis of the existing learning algorithms showed that the learning automata technique including the L_{R-I} algorithm and the L_{R-P} algorithm is a good candidate for decentralized learning. But none of them can guarantee the convergence to Nash equilibria in both pure and fully mixed strategies. We provided two examples to show that the L_{R-I} algorithm can only converge to the Nash equilibria in pure strategies and the L_{R-P} algorithm can only converge to the Nash equilibria in fully mixed strategies. The WoLF-IGA algorithm can converge to both pure and mixed strategy Nash equilibria, but it is not a decentralized learning algorithm. The lagging anchor algorithm considers the player's current strategy and the long term average strategy during learning. Although the lagging anchor algorithm can guarantee the convergence to a Nash equilibrium in fully mixed strategies, it is not a decentralized

learning algorithm.

Inspired by the concept of lagging anchor, we proposed an L_{R-I} lagging anchor algorithm as a completely decentralized learning algorithm. We proved that the L_{R-I} lagging anchor algorithm can guarantee the convergence to a Nash equilibrium in pure or fully mixed strategies in two-player two-action general-sum matrix games. Through simulations, we showed the performance of the proposed L_{R-I} lagging anchor algorithm in three matrix games and the practical L_{R-I} lagging anchor algorithm in a general-sum stochastic game. Simulation results showed the possibility of applying the proposed L_{R-I} lagging anchor algorithm to a two-player matrix game with more than two actions and the possibility of applying the proposed practical L_{R-I} lagging anchor algorithm to general-sum stochastic games.

Chapter 5

Potential-Based Shaping in Stochastic Games

5.1 Introduction

In reinforcement learning, one may suffer from the temporal credit assignment problem [5] where a reward is received after a sequence of actions. The delayed reward will lead to difficulty in distributing credit or punishment to each action from a long sequence of actions and this will cause the algorithm to learn slowly. An example of this problem can be found in some episodic tasks such as a soccer game [57] where the player is only given credit or punishment after a goal is scored. If the number of states in the soccer game is large, it will take a long time for a player to learn its equilibrium policy.

The term shaping was first introduced by Skinner [58] in experimental psychology. Shaping was later implemented in the field of machine learning by Selfridge, Sutton, and Barto [59] such that the learning agent was first assigned to learn several easy tasks before achieving a complex task. Reward shaping is a technique to improve the learning performance of a reinforcement learner by introducing shaping rewards to the environment [60,61]. When the state space is large, the delayed reward will slow down

the learning dramatically. To speed up the learning, the learner may apply shaping rewards to the environment as a supplement to the delayed reward. In this way, a reinforcement learning algorithm can improve its learning performance by combining a "good" shaping reward function with the original delayed reward.

The applications of reward shaping can be found in the literature [60–63]. Gullapalli and Barto [60] demonstrated the application of shaping to a key-press task where a robot was trained to press keys on a keyboard. Dorigo and Colombetti [62] applied shaping policies for a robot to perform a predefined animate-like behavior. Mataric [61] presented an intermediate reinforcement function for a group of mobile robots to learn a foraging task. Randlv and Alstrm [63] combined reinforcement learning with shaping to make an agent learn to drive a bicycle to a goal.

The theoretical analysis of reward shaping can be found in the literature [64–66]. Ng et al. [64] presented a potential-based shaping reward that can guarantee the policy invariance for a single agent in a Markov decision process. Ng et al. proved that the optimal policy keeps unchanged after adding the potential-based shaping reward to an MDP environment. Following Ng et al., Wiewiora [65] showed that the effects of potential-based shaping can be achieved by a particular initialization of Q-values for agents using Q-learning. Asmuth et al. [66] applied the potential-based shaping reward to a model-based reinforcement learning approach.

The above articles focus on applications of reward shaping to a single agent in an MDP. For the applications of reward shaping in general-sum games, Babes, Munoz de Cote, and Littman [67] introduced a social shaping reward for players to learn their equilibrium policies in the iterated prisoner's dilemma game. But there is no theoretical proof of policy invariance under the reward transformation. In this chapter, we prove that the Nash equilibria under the potential-based shaping reward transformation [64] will also be the Nash equilibria for the original game for general-sum stochastic games. Note that the similar work of Devlin and Kudenko [68] was

published while our work was under review. But Devlin and Kudenko only proved sufficiency based on a proof technique introduced by Asmuth et al. [66], while we prove both sufficiency and necessity using a different proof technique in our work.

We then apply the potential-based shaping method to two grid games to test how reward shaping can affect the players' learning performance. The two grid games we simulate are the modified version of Hu and Wellman's [14] grid game and the guarding a territory game with two defenders and one invader. We first apply two reinforcement learning algorithms to these games which are Littman's [40] friend-or-foe Q-learning algorithm for the modified Hu and Wellman's [14] grid game and Bowling and Veloso's [10] WoLF-PHC algorithm for the three-player grid game of guarding a territory. Then we design two shaping rewards and combine them with these learning algorithms. We then run simulations to test the performance of these algorithms with different shaping rewards. Comparison of the learning algorithms with and without shaping rewards is illustrated based on simulation results.

The main **contributions in this chapter** are:

- Extend the potential-based shaping method from MDPs to multi-player general-sum stochastic games.
- Prove the policy invariance for the potential-based shaping method in multi-player general-sum stochastic games.
- Apply two reinforcement learning algorithms combined with the potential-based shaping method to two grid games.
- Run simulations and compare the learning performance of players with and without shaping rewards.

The contribution on the proof of policy invariance for general-sum stochastic games has been published in [69].

In this chapter, we first present shaping rewards in MDPs in Sect. 5.2. The proof of policy invariance under reward transformations for general-sum stochastic games is provided in Sect. 5.3. In Sect. 5.4, we present two grid games to test the performance of two learning algorithms based on different shaping reward functions.

5.2 Shaping Rewards in MDPs

We first present the framework of shaping rewards in MDPs. Assume we have an MDP defined as a tuple $N = (S, A, Tr, \gamma, R)$ where S is the state space, A is the action space, γ is the discount factor and R is the reward function. An agent in the MDP wants to learn a desired policy using a learning algorithm. In order to help the learning algorithm to learn the desired policy, one can add additional shaping rewards to the MDP to guide the learning algorithm. With the additional shaping rewards added to the original MDP, we get a transformed MDP $N' = (S, A, Tr, \gamma, R')$, where R' is the new reward function in the transformed MDP. The new reward function R' is given as $R' = R + F$ where $F : S \times A \times S \to \mathbb{R}$ is a bounded real-valued function called the shaping reward function [64].

The transformed MDP N' includes the same state space, action space and transition function as the original MDP N. In the transformed MDP N', instead of receiving the reward $R(s, a, s')$, we will receive the new reward $R(s, a, s') + F(s, a, s')$ for moving from s to s' on action a. The aim of the reward transformation is to help the learning algorithm to learn the desired or optimal policy in the original MDP. Therefore, we want to know what forms of shaping reward function F can guarantee that the optimal policy in N' will also be optimal in N.

In [64], Ng et al. introduced a potential-based shaping function that can guarantee the policy invariance under reward transformations. The potential-based shaping

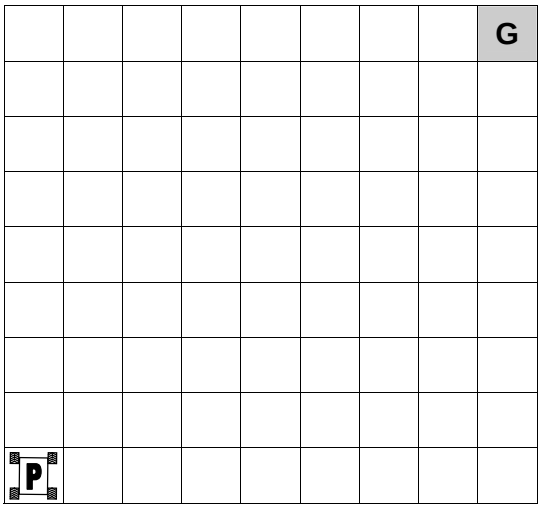


Figure 5.1: An example of reward shaping in MDPs

function F, which is a difference of potentials, has the form of

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s), \tag{5.1}$$

where γ is the discount factor given in the MDP and the shaping potential $\Phi(\cdot) : S \rightarrow \mathbb{R}$ is a real-valued function. Without prior knowledge of the transition function Tr and the reward function R in N, Ng et al. proved that the potential-based shaping functions F are the only form of F that will guarantee consistency with the optimal policy in N. The details of the proof can be found in [64].

Example 5.1. We now modify Example 2.1 presented in Sect. 2.2.1 to show how the potential-based shaping function can help the player learn its optimal policy. We present a 9×9 grid as the playing field shown in Fig. 5.1. The player starts from the bottom left corner and tries to reach the goal labeled as "G" on the opposite corner. The player has 4 possible actions: moving up, down, left and right. The player takes an action a and moves one cell each time. If the player's chosen action is taking the player off the grid, the player will stand still. Similar to Example 2.1, the transition

function is set to 1 for each movement and the reward function is given as

$$R(s, a, s') = -1, \quad \text{for each movement} \tag{5.2}$$

To encourage the player to move towards the goals, we choose $F(s, a, s')$ such that the player receives a higher reward when the next state s' is closer to the goal. We define the shaping potential $\Phi(s)$ as

$$\Phi(s) = -dist_{PG}(s) \tag{5.3}$$

where $dist_{PG}(s)$ is the Manhattan distance between player's current state and the goal.

Based on (5.1), the reward function in (5.2) is transformed to

$$\begin{aligned} R'(s, a, s') &= R(s, a, s') + F(s, a, s') \\ &= -1 + dist_{PG}(s) - \gamma dist_{PG}(s') \end{aligned} \tag{5.4}$$

We now simulate the game using Q-learning described in Sect. 2.2.2. The discount factor is set to 1. We define one trial as 200 episodes or 200 runs of the game. Each episode or run starts from the initial position of the player, which is located at the bottom left corner, and ends when the player reaches the goal. We record the total steps for the player to reach the goal at each episode. Then we average them over 40 trials. The simulation results are shown in Fig. 5.2, where the dash line is the learning result without the shaping function and the solid line is the result with the shaping function. Compared with the learning without shaping, the potential-based shaping function defined in (5.4) helped the player speed up the learning process.

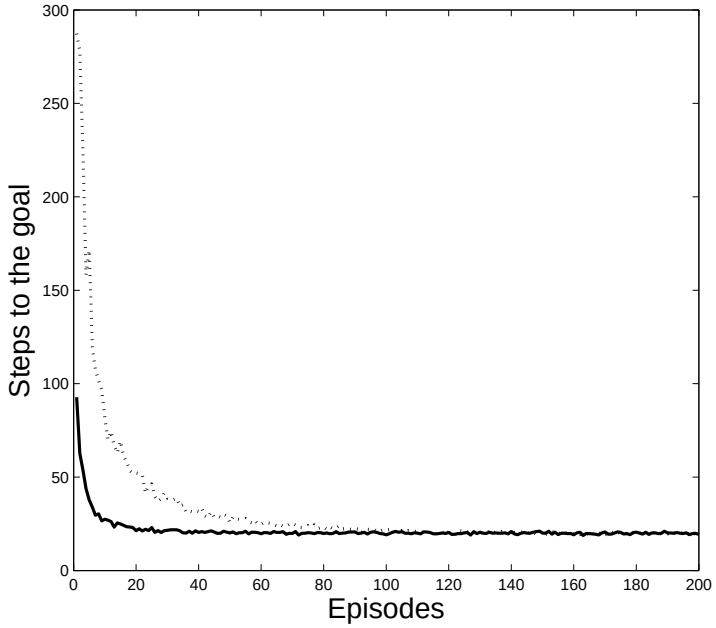


Figure 5.2: Simulation results with and without the shaping function in Example 5.1

5.3 Potential-Based Shaping in General-Sum Stochastic Games

Ng et al. [64] presented a reward shaping method to deal with the credit assignment problem by adding a potential-based shaping reward to the environment. The combination of the shaping reward with the original reward may improve the learning performance of a reinforcement learning algorithm and speed up the convergence to the optimal policy. The existing theoretical studies on potential-based shaping methods in the literature only consider the case of a single agent in an MDP [64–66]. In our research, we extend the potential-based shaping method from Markov decision processes to multi-player stochastic games. We prove that the Nash equilibria under the potential-based shaping reward transformation will be the Nash equilibria for the original game under the framework of general-sum stochastic games. To be consistent

with the terms used in [64], in this chapter, we use the term "**policy**" to replace the term "**strategy**" defined in Sect. 2.4 for the players in stochastic games. These two terms are interchangeable.

We define a potential-based shaping reward $F_i(s, s')$ for player i as

$$F_i(s, s') = \gamma\Phi_i(s') - \Phi_i(s), \tag{5.5}$$

where $\Phi : S \rightarrow \mathbb{R}$ is a real-valued shaping function and $\Phi(s_T) = 0$ for any terminal state s_T. We define a multi-player stochastic game as a tuple $M = (S, A_1, \ldots, A_n, Tr, \gamma, R_1, \ldots, R_n)$ where S is a set of states, $A_1, \ldots, A_n$ are players' action sets, Tr is the transition function, γ is the discount factor, and $R_i(s, a_1, \ldots, a_n, s')(i = 1, \ldots, n)$ is the reward function for player i. After adding the shaping reward function $F_i(s, s')$ to the reward function $R_i(s, a_1, \ldots, a_n, s')$, we define a transformed multi-player stochastic game as a tuple $M' = (S, A_1, \ldots, A_n, Tr, \gamma, R'_1, \ldots, R'_n)$ where $R'_i(i = 1, \ldots, n)$ is the new reward function given by $R'_i(s, a_1, \ldots, a_n, s') = F_i(s, s') + R_i(s, a_1, \ldots, a_n, s')$. Inspired by Ng et al.'s proof of policy invariance in an MDP in [64] , we prove the policy invariance in a multi-player general-sum stochastic game as follows.

Theorem 5.1. *Given an n-player discounted stochastic game* $M = (S, A_1, \ldots, A_n, Tr, \gamma, R_1, \ldots, R_n)$, *we define a transformed n-player discounted stochastic game* $M' = (S, A_1, \ldots, A_n, Tr, \gamma, R_1 + F_1, \ldots, R_n + F_n)$ *where* $F_i \in S \times S$ *is a shaping reward function for player i. We call* F_i *a potential-based shaping function if* F_i *has the form of (5.5). Then, the potential-based shaping function* F_i *is a necessary and sufficient condition to guarantee the Nash equilibrium policy invariance such that*

- *(Sufficiency) If* F_i $(i = 1, \ldots, n)$ *is a potential-based shaping function, then every Nash equilibrium policy in* M' *will also be a Nash equilibrium policy in* M

(and vice versa).

- *(Necessity) If F_i $(i = 1, \ldots, n)$ is not a potential-based shaping function, then there may exist a transition function Tr and reward function R such that the Nash equilibrium policy in M' will not be the Nash equilibrium policy in M.*

The proof is conducted as follows. In the proof of sufficiency, we first present the definition of the Nash equilibrium in the stochastic game M. Then the definition of the Nash equilibrium in M' can be derived from equations in (2.49)-(2.51). We then find that the definition of the Nash equilibrium in M' is exactly the same definition of the Nash equilibrium in M. In the proof of necessity, we first build a stochastic game in M. Then we choose a shaping function which is not potential based and add it to M. We find that the Nash equilibrium policy in M' is no longer the same Nash equilibrium policy in M.

Proof. (*Proof of Sufficiency*)
Based on (2.51), a Nash equilibrium in the stochastic game M can be represented as a set of policies such that for all $i = 1, \ldots, n, s \in S$ and $\pi_{M_i} \in \Pi$

$$\sum_{a_1,\ldots,a_n \in A_1 \times \cdots \times A_n} Q^*_{M_i}(s, a_1, \ldots, a_n)\pi^*_{M_1}(s, a_1) \cdots \pi^*_{M_i}(s, a_i) \cdots \pi^*_{M_n}(s, a_n) \geq$$
$$\sum_{a_1,\ldots,a_n \in A_1 \times \cdots \times A_n} Q^*_{M_i}(s, a_1, \ldots, a_n)\pi^*_{M_1}(s, a_1) \cdots \pi_{M_i}(s, a_i) \cdots \pi^*_{M_n}(s, a_n). \quad (5.6)$$

We subtract $\Phi_i(s)$ on both sides of (5.6) and get

$$\sum_{a_1,\ldots,a_n \in A_1 \times \cdots \times A_n} Q^*_{M_i}(s, a_1, \ldots, a_n)\pi^*_{M_1}(s, a_1) \cdots \pi^*_{M_i}(s, a_i) \cdots \pi^*_{M_n}(s, a_n) - \Phi_i(s) \geq$$
$$\sum_{a_1,\ldots,a_n \in A_1 \times \cdots \times A_n} Q^*_{M_i}(s, a_1, \ldots, a_n)\pi^*_{M_1}(s, a_1) \cdots \pi_{M_i}(s, a_i) \cdots \pi^*_{M_n}(s, a_n) - \Phi_i(s). \quad (5.7)$$

Since $\sum_{a_1,\ldots,a_n \in A_1 \times \cdots \times A_n} \pi_{M_1}(s, a_1) \cdots \pi_{M_i}(s, a_i) \cdots \pi_{M_n}(s, a_n) = 1$ $(\forall s \in S)$, we can

get

$$\sum_{a_1,\ldots,a_n \in A_1 \times \cdots \times A_n} [Q^*_{M_i}(s, a_1, \ldots, a_n) - \Phi_i(s)]\pi^*_{M_1}(s, a_1) \cdots \pi^*_{M_i}(s, a_i) \cdots \pi^*_{M_n}(s, a_n) \geq$$
$$\sum_{a_1,\ldots,a_n \in A_1 \times \cdots \times A_n} [Q^*_{M_i}(s, a_1, \ldots, a_n) - \Phi_i(s)]\pi^*_{M_1}(s, a_1) \cdots \pi_{M_i}(s, a_i) \cdots \pi^*_{M_n}(s, a_n). \quad (5.8)$$

We define

$$\hat{Q}_{M'_i}(s, a_1, \ldots, a_n) = Q^*_{M_i}(s, a_1, \ldots, a_n) - \Phi_i(s). \quad (5.9)$$

Then we can get

$$\sum_{a_1,\ldots,a_n \in A_1 \times \cdots \times A_n} \hat{Q}_{M'_i}(s, a_1, \ldots, a_n)\pi^*_{M_1}(s, a_1) \cdots \pi^*_{M_i}(s, a_i) \cdots \pi^*_{M_n}(s, a_n) \geq$$
$$\sum_{a_1,\ldots,a_n \in A_1 \times \cdots \times A_n} \hat{Q}_{M'_i}(s, a_1, \ldots, a_n)\pi^*_{M_1}(s, a_1) \cdots \pi_{M_i}(s, a_i) \cdots \pi^*_{M_n}(s, a_n). \quad (5.10)$$

We now use some algebraic manipulations to rewrite the action-value function under the Nash equilibrium in (2.50) for player i in the stochastic game M as

$$\begin{aligned} Q^*_{M_i}(s, a_1, \ldots, a_n) - \Phi_i(s) &= \sum_{s' \in S} Tr(s, a_1, \ldots, a_n, s') \big[R_{M_i}(s, a_1, \ldots, a_n, s') + \\ &\quad \gamma V^*_{M_i}(s') + \gamma\Phi_i(s') - \gamma\Phi_i(s')\big] - \Phi_i(s). \end{aligned} \quad (5.11)$$

Since $\sum_{s' \in S} Tr(s, a_1, \ldots, a_n, s') = 1$, the above equation becomes

$$\begin{aligned} Q^*_{M_i}(s, a_1, \ldots, a_n) - \Phi_i(s) &= \sum_{s' \in S} Tr(s, a_1, \ldots, a_n, s') \left[R_{M_i}(s, a_1, \ldots, a_n, s')\right. \\ &\quad \left. + \gamma\Phi_i(s') - \Phi_i(s) + \gamma V^*_{M_i}(s') - \gamma\Phi_i(s')\right]. \end{aligned} \quad (5.12)$$

According to (2.49), we can rewrite the above equation as

$$\begin{aligned} Q^*_{M_i}(s,a_1,\ldots,a_n)-\Phi_i(s) &= \sum_{s'\in S} Tr(s,a_1,\ldots,a_n,s')\left[R_{M_i}(s,a_1,\ldots,a_n,s')+\gamma\Phi_i(s')\right. \\ -\Phi_i(s)+\gamma &\sum_{a_1,\ldots,a_n\in A_1\times\cdots\times A_n} Q^*_{M_i}(s',a'_1,\ldots,a'_n)\pi^*_{M_1}(s',a'_1)\cdots\pi^*_{M_i}(s',a'_n)-\gamma\Phi_i(s')\Big] \\ &= \sum_{s'\in S} Tr(s,a_1,\ldots,a_n,s')\left\{R_{M_i}(s,a_1,\ldots,a_n,s')+\gamma\Phi_i(s')\right. \\ -\Phi_i(s)+\gamma &\sum_{a_1,\ldots,a_n\in A_1\times\cdots\times A_n} \left[Q^*_{M_i}(s',a'_1,\ldots,a'_n)-\Phi_i(s')\right]\pi^*_{M_1}(s',a'_1)\cdots\pi^*_{M_i}(s',a'_n)\Big\}. \end{aligned} \tag{5.13}$$

Based on the definitions of $F_i(s,s')$ in (5.5) and $\hat{Q}_{M'_i}(s,a_1,\ldots,a_n)$ in (5.9), the above equation becomes

$$\begin{aligned} \hat{Q}_{M'_i}(s,a_1,\ldots,a_n) &= \sum_{s'\in S} Tr(s,a_1,\ldots,a_n,s')\left[R_{M_i}(s,a_1,\ldots,a_n,s')+F_i(s,s')\right. \\ &+\gamma \sum_{a_1,\ldots,a_n\in A_1\times\cdots\times A_n} \hat{Q}_{M'_i}(s',a'_1,\ldots,a'_n)\,\pi^*_{M_1}(s',a'_1)\cdots\pi^*_{M_i}(s',a'_n)\Big] \end{aligned} \tag{5.14}$$

Since equations (5.10) and (5.14) have the same form as equations (2.49)-(2.51), we can conclude that $\hat{Q}_{M'_i}(s,a_1,\ldots,a_n)$ is the action-value function under the Nash equilibrium for player i in the stochastic game M'. Therefore, we can obtain

$$\hat{Q}_{M'_i}(s,a_1,\ldots,a_n)=Q^*_{M'_i}(s,a_1,\ldots,a_n)=Q^*_{M_i}(s,a_1,\ldots,a_n)-\Phi_i(s). \tag{5.15}$$

If the state s is the terminal state s_T, then we have $\hat{Q}_{M'_i}(s_T,a_1,\ldots,a_n) = Q^*_{M_i}(s_T,a_1,\ldots,a_n)-\Phi_i(s_T)=0-0=0$. Based on (5.10) and $\hat{Q}_{M'_i}(s,a_1,\ldots,a_n) = Q^*_{M'_i}(s,a_1,\ldots,a_n)$, we can find that the Nash equilibrium in M is also the Nash equilibrium in M'. Then the state-value function under the Nash equilibrium in the

stochastic game M' can be given as

$$\begin{aligned} V^*_{M'_i}(s) &= \sum_{a_1,\ldots,a_n \in A_1\times\cdots\times A_n} Q^*_{M'_i}(s,a_1,\ldots,a_n)\pi^*_{M'_1}(s,a_1)\cdots\pi^*_{M'_i}(s,a_i)\cdots\pi^*_{M'_n}(s,a_n) \\ &= \sum_{a_1,\ldots,a_n \in A_1\times\cdots\times A_n} Q^*_{M_i}(s,a_1,\ldots,a_n)\pi^*_{M_1}(s,a_1)\cdots\pi^*_{M_i}(s,a_i)\cdots\pi^*_{M_n}(s,a_n) \\ &\quad -\Phi_i(s) \\ &= V^*_{M_i}(s) - \Phi_i(s). \end{aligned} \tag{5.16}$$

(*Proof of Necessity*)

If F_i $(i = 1,\ldots,n)$ is not a potential-based shaping function, we will have $F_i(s,s') \neq \gamma\Phi_i(s') - \Phi_i(s)$. Then we can build a stochastic game M by giving the following transition function Tr and player 1's reward function $R_{M_1}(\cdot)$

$$\begin{aligned} Tr(s_1,a_{11},a_2,\ldots,a_n,s_3) &= 1, \\ Tr(s_1,a_{12},a_2,\ldots,a_n,s_2) &= 1, \\ Tr(s_2,a_1,\ldots,a_n,s_3) &= 1, \\ Tr(s_3,a_1,\ldots,a_n,s_3) &= 1, \\ R_{M_1}(s_1,a_1,\ldots,a_n,s_3) &= \frac{\Delta}{2}, \\ R_{M_1}(s_1,a_1,\ldots,a_n,s_2) &= 0, \\ R_{M_1}(s_2,a_1,\ldots,a_n,s_3) &= 0, \\ R_{M_1}(s_3,a_1,\ldots,a_n,s_3) &= 0, \end{aligned} \tag{5.17}$$

where $a_i (i = 1,\ldots,n)$ represents any possible action $a_i \in A_i$ from player i, and a_{11} and a_{12} represent player 1's action 1 and action 2 respectively. Equation $Tr(s_1,a_{11},a_2,\ldots,a_n,s_3) = 1$ in (5.17) denotes that, given the current state s_1, player 1's action a_{11} will lead to the next state s_3 no matter what joint action the other players take. Based on the above transition function and reward function, we can get the

game model including states (s_1, s_2, s_3) shown in Figure 5.3. Similar to Ng et al.'s [64] proof of necessity, we define $\Delta = F_1(s_1, s_2) + \gamma F_1(s_2, s_3) - F_1(s_1, s_3)$. Without loss of generality, we assume $F_1(s_3, s_3) = 0$ since replacing $F_1(s, s')$ with $F_1(s, s') - F_1(s_3, s_3)$ does not affect the player's equilibrium policy [64]. Based on (2.49), (2.50), (5.15), (5.16) and (5.17), we can obtain player 1's action-value function at state s_1 in M and M'

$$\begin{aligned}
Q^*_{M_1}(s_1, a_{11}, \dots) &= \frac{\Delta}{2}, \\
Q^*_{M_1}(s_1, a_{12}, \dots) &= 0, \\
Q^*_{M'_1}(s_1, a_{11}, \dots) &= F_1(s_1, s_2) + \gamma F_1(s_2, s_3) - \frac{\Delta}{2}, \\
Q^*_{M'_1}(s_1, a_{12}, \dots) &= F_1(s_1, s_2) + \gamma F_1(s_2, s_3).
\end{aligned}$$

Then the Nash equilibrium policy for player 1 at state s_1 is

$$\pi^*_{M_1}(s_1, a_1) = \begin{cases} a_{11} \text{ if } \Delta > 0, \\ \\ a_{12} \text{ otherwise} \end{cases}, \quad \pi^*_{M'_1}(s_1, a_1) = \begin{cases} a_{12} \text{ if } \Delta > 0, \\ \\ a_{11} \text{ otherwise} \end{cases}. \tag{5.18}$$

Therefore, in the above case, the Nash equilibrium policy for player 1 at state s_1 in M is not the Nash equilibrium policy in M'. □

The above analysis shows that the potential-based shaping reward with the form of $F_i(s, s') = \gamma \Phi_i(s') - \Phi_i(s)$ guarantees the Nash equilibrium policy invariance. Now the question becomes how to select a shaping function $\Phi_i(s)$ to improve the learning performance of the learner. Ng et al. [64] showed that $\Phi_i(s) = V^*_{M_i}(s)$ is an ideal candidate for improving the player's learning performance in an MDP. We substitute

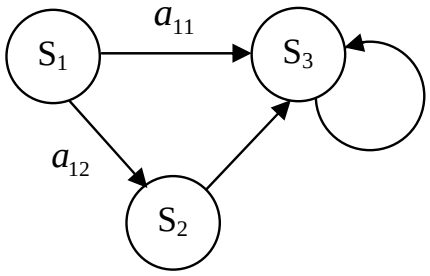


Figure 5.3: Possible states of the stochastic model in the proof of necessity

$\Phi_i(s) = V^*_{M_i}(s)$ into (5.14) and get

$$\begin{aligned}
\hat{Q}_{M'_i}(s, a_1, \ldots, a_n) &= Q^*_{M'_i}(s, a_1, \ldots, a_n) \\
&= \sum_{s' \in S} Tr(s, a_1, \ldots, a_n, s') \left[R_{M_i}(s, a_1, \ldots, a_n, s') + F_i(s, s') \right. \\
&+ \gamma \sum_{a_1, \ldots, a_n \in A_1 \times \cdots \times A_n} Q^*_{M'_i}(s', a'_1, \ldots, a'_n)\, \pi^*_{M_1}(s', a'_1) \cdots \pi^*_{M_i}(s', a'_n) \Big] \\
&= \sum_{s' \in S} Tr(s, a_1, \ldots, a_n, s') \left[R_{M_i}(s, a_1, \ldots, a_n, s') + F_i(s, s') \right. \\
&\quad \left. + \gamma(V^*_{M_i}(s') - \Phi_i(s')) \right] \\
&= \sum_{s' \in S} Tr(s, a_1, \ldots, a_n, s') \left[R_{M_i}(s, a_1, \ldots, a_n, s') + F_i(s, s') \right]. \quad (5.19)
\end{aligned}$$

Equation (5.19) shows that the action-value function $Q^*_{M'_i}(s, a_1, \ldots, a_n)$ in state s can be easily obtained by checking the immediate reward $R_{M_i}(s, a_1, \ldots, a_n, s') + F_i(s, s')$ that player i received in state s'. However, in practical applications, we will not have all the information of the environment such as $Tr(s, a_1, \ldots, a_n, s')$ and $R_i(s, a_1, \ldots, a_n, s')$. This means that we cannot find a shaping function $\Phi_i(s)$ such that $\Phi_i(s) = V^*_{M_i}(s)$ without knowing the model of the environment. Therefore, the goal for designing a shaping function is to find a $\Phi_i(s)$ as a "good" approximation to $V^*_{M_i}(s)$.

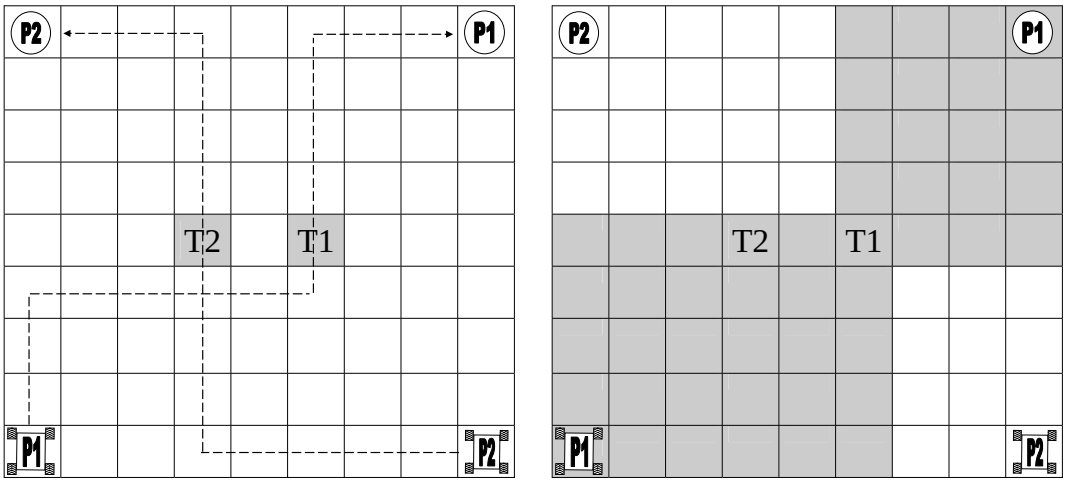
5.4 Simulation and Results

In this section, we set up two grid games to test the performance of the reinforcement learning algorithms with and without the shaping rewards. We apply the friend-or-foe Q-learning algorithm presented in Sect. 3.2.3 and the WoLF-PHC algorithm presented in Sect. 3.2.4 to the modified Hu and Wellman's [14] grid game and the three-player grid game of guarding a territory respectively. We show how the convergence can be affected by different shaping functions through simulation results.

5.4.1 Hu and Wellman's Grid Game

The first game we test is the modified version of the game introduced by Hu and Wellman [14]. In [14], the game runs in a 3×3 grid field. To better show the effect of reward shaping to a game, we increase the size of the game to a 9×9 grid shown in Fig. 5.4. We have two players located at the bottom of the playing field where player 1 is at the bottom left corner and player 2 is at the bottom right corner. Player 1 tries to reach its goal at the upper right corner and the player 2 tries to reach its goal at the upper left corner. Before the two players reach their goals, each player has to pass one specific cell located in the middle of the field shown as T1 and T2 respectively in Fig. 5.4(a). Each player has four possible moves which are moving up, down, left or right unless the player is on the sides of the grid. For example, if the player is at the bottom left corner, its available moves are moving up or right. If the two players move to the same cell at the same time, they will bounce back to their original positions. The game ends when either one of the players passes the specific cell and reaches its goal.

Now we define the reward function and transition function for the game. For simplicity, the transition function is set as 1 for all the possible moves such that the



(a) Possible optimal paths of the two players (b) Region G_1 (grey cells) containing all the optimal paths for player 1

Figure 5.4: A modified Hu and Wellman's grid game

player moves from one cell to another with probability 1. For the reward function, we give the player a reward of 100 if the player passes the specific cell and reaches the goal. If the player reaches the goal without passing the specific cell, no reward will be given to the player. If two players move to the same cell, each player bounces back and receives a reward of -1. Similar to the original game introduced by Hu and Wellman [14], this game has a global equilibrium point with multiple optimal paths. One of the possible optimal paths is shown in Fig. 5.4(a) where two dash lines show the least moves for two players to reach the goal. Fig. 5.4(a) shows that the smallest number of moves to reach the goal is 16. We can also find the boundary of all the possible optimal paths for each player. For example, the grey cells in Fig. 5.4(b) show the region G_1 containing all possible optimal paths for player 1. In other words, it is impossible for player 1 to enter the white cells in Fig. 5.4(b) and still reach the goal in 16 moves. Then we can find the value of the state for all the states as given in (2.49). For example, if we choose 0.9 as the discount factor γ, the value of the state

for player 1 can be calculated as

$$V_1^*(s) = \begin{cases} 0.9^{dist_{1G}-1} \cdot 100 & \text{if } dist_{1G} \leq dist_{2G} \text{ and } P1 \in G_1 \\ 0 & \text{otherwise} \end{cases} \tag{5.20}$$

where $dist_{1G}$ is the Manhattan distance between the player 1 and its goal P_1 at the current time step, $dist_{2G}$ is the Manhattan distance between the player 2 and its goal P_2 at the current time step, and $P1 \in G_1$ represents that player 1 remains inside the region G_1 shown in Fig. 5.4(b).

Littman [40] formulated the Hu and Wellman's [14] game as a cooperative game and apply the friend-Q algorithm to find the Nash equilibrium to the game. Due to the similarity between this game and the Hu and Wellman's game, we can also apply the friend-Q algorithm for this grid game. The friend-Q algorithm is the special case of the friend-or-foe Q-learning algorithm where the player treats all other players as its friends. In this case, step 6 of Algorithm 3.3 in Sect. 3.2.3 becomes

$$V_i(s) = \max_{\pi_1(s,\cdot),...,\pi_n(s,\cdot)} \sum_{a_1,...,a_n \in A_1 \times \cdots \times A_n} Q_i(s, a_1, ..., a_n)\pi_1(s, a_1) \cdots \pi_n(s, a_n). \tag{5.21}$$

We choose the following parameters in the friend-Q algorithm. We define the learning rate as $\alpha(t) = 1/(1 + t/500)$ where t is the time step. We adopt a 0.05-greedy exploration strategy such that the player chooses an action randomly from its action set with a probability 0.05 and the greedy action with probability 0.95. The values of $\alpha(t)$ and ε are chosen based on the parameter settings in [10]. To better compare the performance of the learning algorithms and the shaping reward, we will keep the value of $\alpha(t)$ and ε the same for all the games in this section. For a single training episode, the game starts with the players' initial positions and ends when the terminal condition is reached. We use $NS_i(TE)$ to denote the number of steps taken for player

i to reach its goal at the TEth training episode. We define one trial as one single learning period including 1000 training episodes. In this game, we run 100 trials and average the result of each training episode over 100 trials. The averaged result of each training episode over 100 trials is given as

$$\overline{NS}_i(TE) = \frac{1}{100}\sum_{Trl=1}^{100} NS_i^{Trl}(TE), \text{ for } TE = 1, ..., 1000 \tag{5.22}$$

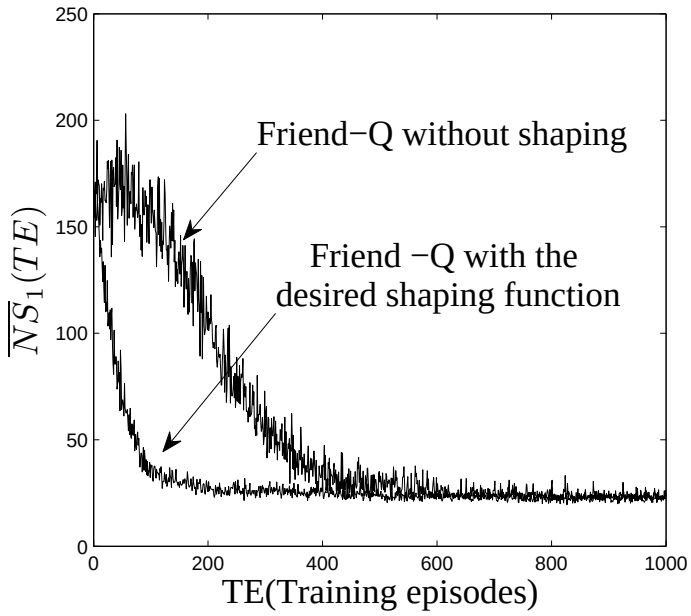
where i represents player i, and $NS_i^{Trl}(TE)$ denotes the number of steps for player i to reach the goal at the TEth training episode in trial # Trl. We now add shaping reward functions to the friend-Q algorithm. As discussed in Section 5.3, a desired shaping reward function has the form of

$$F_i(s, s') = \gamma V_{M_i}^*(s') - V_{M_i}^*(s). \tag{5.23}$$

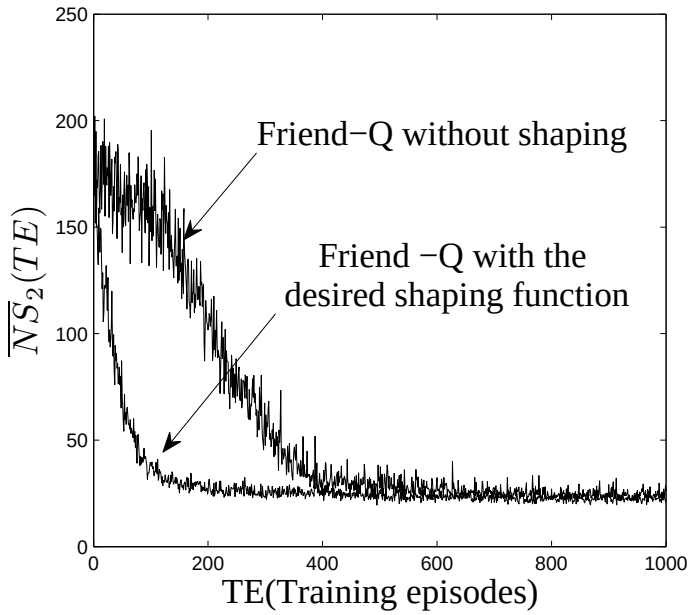
For player 1, we can calculate $V_{M_1}^*(\cdot)$ from (5.20) and substitute it into (5.23) to get the desired shaping reward function. Similarly, we can also calculate $V_{M_2}^*(\cdot)$ for player 2 and get the desired potential-based shaping function for player 2. After simulations we compare the performance of the players with and without the desired shaping function. The results are shown in Fig. 5.5. As for the convergence to the optimal path(16 steps to the goal), both players converge faster with the help of the desired shaping function.

5.4.2 A Grid Game of Guarding a Territory with Two Defenders and One Invader

The second game we considered is a three-player grid game of guarding a territory. In this section, we extend the two-player grid game of guarding a territory introduced



(a) Averaged steps for player 1 to reach the goal



(b) Averaged steps for player 2 to reach the goal

Figure 5.5: Learning performance of friend-Q algorithm with and without the desired reward shaping

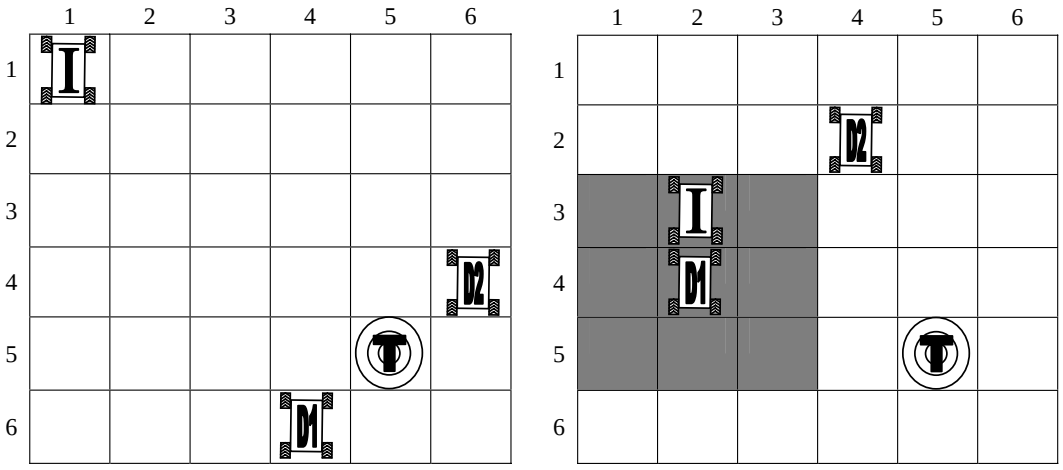
in [37] to a three-player grid game of guarding a territory with two defenders and one invader. Two defenders in the game try to prevent an invader from entering the territory. The goal for the invader is to invade a territory or move as close as possible to the territory. The goal of the two defenders is to intercept the invader and keep the invader as far as possible away from the territory. The game is defined as follows

- We take a 6×6 grid as the playing field shown in Figure 5.6. The territory is represented by a cell named T at $(5, 5)$ in Figure 5.6(a). The position of the territory remains unchanged. The invader starts from the upper-left corner and the defenders start at positions (6, 4) and (4, 6) respectively.

- Each player has four possible actions. It can move up, down, left or right unless it is on the sides of the grid. For example, if the invader is located at the top-left corner, it can only have two actions: move down or right. At each time step, each player takes one action and move to an adjacent cell simultaneously.

- The nine gray cells centered around the defender, shown in Figure 5.6(b), are the terminal region where the invader will be captured by the defenders. A successful invasion by the invader is defined in the situation where the invader reaches the territory before the capture or the capture happens at the territory. The game ends when either one of the defenders captures the invader or a successful invasion by the invader happens. Then a new trial starts with the same initial positions of the players.

- The goal of the invader is to reach the territory without interception or move to the territory as close as possible if the capture must happen. On the contrary, the aim of the defenders is to intercept the invader at a location as far as possible from the territory.



(a) Initial state of the players when the game starts

(b) One terminal state of the players when the game ends

Figure 5.6: A grid game of guarding a territory with two defenders and one invader

The terminal time is defined as the time when the invader reaches the territory or is intercepted by the defenders. We define the payoff as the Manhattan distance between the invader and the territory at the terminal time:

$$\text{Payoff} = |x_I(t_f) - x_T| + |y_I(t_f) - y_T| \tag{5.24}$$

where $(x_I(t_f), y_I(t_f))$ is the invader's position at the terminal time t_f and (x_T, y_T) is the territory's position. If we represent the two defenders as a team, then the invader tries to minimize the payoff while the defender team tries to maximize the payoff.

We now define the transition probability function and reward function in the game. For simplicity, the transition probability function for all the possible moves is set to 1 which means that the players have deterministic moves. The reward function for

the defender i ($i = 1, 2$) is defined as

$$R_{D_i} = \begin{cases} Dist_{IT}, & \text{defender } i \text{ captures the invader} \\ -10, & \text{invader reaches the territory} \\ 0, & \text{others} \end{cases} \tag{5.25}$$

where

$$Dist_{IT} = |x_I(t_f) - x_T| + |y_I(t_f) - y_T|.$$

The reward function for the invader is given by

$$R_I = \begin{cases} -Dist_{IT}, & \text{defender captures the invader} \\ 10, & \text{invader reaches the territory} \\ 0, & \text{others} \end{cases} \tag{5.26}$$

In this grid game, we have three players playing on a 6×6 grid. Considering each player has four possible actions, each action-value Q function in (2.50) contains *players' joint action* $\times$ *players' joint space* $= 4^3 \times 36^3$ elements. The friend-or-foe Q-learning algorithm needs to know all the players' actions in order to compute the value of the state using the linear programming method at each time step. If we use the friend-or-foe Q-learning algorithm, we have to deal with the problem of memory requirement due to the large size of the Q function and the problem of computational complexity due to the use of linear programming. The WoLF-PHC algorithm is a policy online learning approach that can update each player's policy based on the player's own action and the received reward. Thus the dimension of the action-value Q function decreases from $4^3 \times 36^3$ to 4×36^3.

Based on the above analysis, we choose the WoLF-PHC algorithm for the three-player grid game of guarding a territory and study the learning performance of the players with and without reward shaping. Our aim is to test how the shaping function can affect the learning performance of the players. To do that, we design two different shaping reward functions and compare them through simulation results. We first define the following shaping function called *Shaping* 1:

$$\begin{aligned} \Phi_I(s) &= -dist_{IT} \\ \Phi_{D_i}(s) &= -dist_{D_iI}, \quad (i = 1, 2) \end{aligned} \tag{5.27}$$

where $dist_{IT}$ is the Manhattan distance between the invader and the territory at the current state s, and $dist_{D_iI}$ is the Manhattan distance between the defender i ($i = 1, 2$) and the invader at the current state s. To compare the learning performance of the WoLF-PHC learning algorithm with and without the shaping function, we run two simulations at the same time. The first simulation, called **S1**, applies the WoLF-PHC learning algorithm without the shaping function to all the players. The second simulation, called **S2**, applies the WoLF-PHC learning algorithm with the shaping function to all the players. Each simulation includes 2000 **training episodes**. After every 200 training episodes in each simulation, we set up a **testing phase** to test the performance of the learners. In a testing phase, two **tests** are performed. We denote $t1$ for the first test and $t2$ for the second test. In the first test, we let the invader from **S1** (Simulation 1) play against the two defenders from **S2** (Simulation 2) for 50 runs. In the second test, we let the invader from **S2** play against the two defenders from **S1** for 50 runs. A single **run** of the game is when the game starts at the players' initial positions and ends at a terminal state. The result of each run is the distance between the invader and the territory at the terminal time. For each

test in the **testing phase**, we average the result from each run over 50 runs and get

$$\overline{dist}_{IT,t1} = \frac{1}{50} \sum_{run=1}^{50} dist_{IT,t1}^{run}(s_T) \tag{5.28}$$

$$\overline{dist}_{IT,t2} = \frac{1}{50} \sum_{run=1}^{50} dist_{IT,t2}^{run}(s_T) \tag{5.29}$$

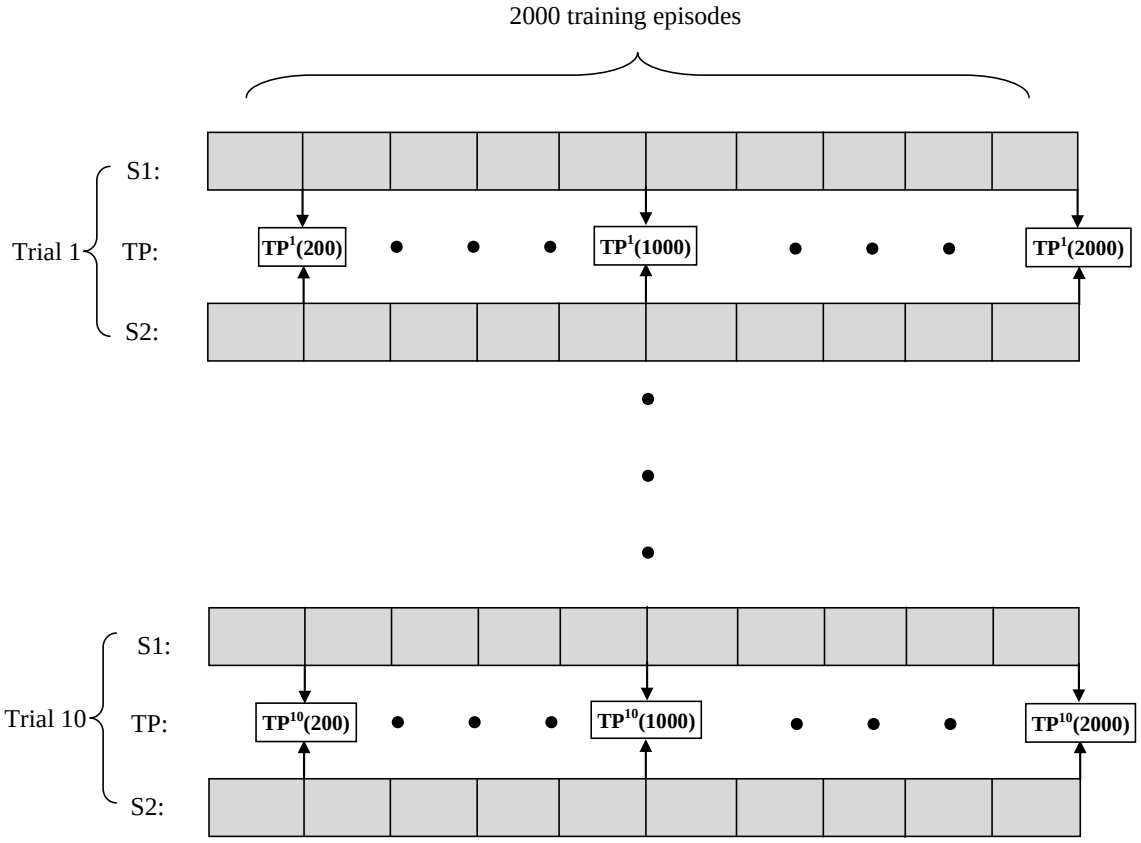
where $\overline{dist}_{IT,t1}$ denotes the average result of 50 runs for test 1 and $\overline{dist}_{IT,t2}$ denotes the average result of 50 runs for test 2 in the testing phase. In each testing phase, we calculate the average distance $\overline{dist}_{IT,t1}$ and $\overline{dist}_{IT,t2}$ in (5.28) and (5.29). We define one trial as one run of *Simulation* 1 and *Simulation* 2. After 10 trials, we average the result of each testing phase over the 10 trials. The result is given as

$$\overline{Dist}_{IT,t1}(TE) = \frac{1}{10} \sum_{Trl=1}^{10} \overline{dist}_{IT,t1}^{Trl}(TE) \tag{5.30}$$

$$\overline{Dist}_{IT,t2}(TE) = \frac{1}{10} \sum_{Trl=1}^{10} \overline{dist}_{IT,t2}^{Trl}(TE) \tag{5.31}$$

where $\overline{Dist}_{IT,t1}(TE)$ is the average distance at the TEth training episode over 10 trials for test 1, and $\overline{Dist}_{IT,t2}(TE)$ is the average distance at the TEth training episode over 10 trials for test 2. We illustrate the simulation procedure in Fig. 5.7.

For simplicity, we use the same parameter settings as in the previous game for all the simulations. Table 5.1 shows the result where $\overline{Dist}_{IT,t1}(TE)$ and $\overline{Dist}_{IT,t2}(TE)$ denote the average results of test 1 and test 2 in a testing phase at the TEth training episode over 10 trials. In Table 5.1, the values in test 1 (second column) are smaller than the values in test 2 (third column). This implies that the invader from simulation 1 (without the shaping function) can move closer to the territory than the invader from simulation 2 (with the shaping function). In other words, the defenders from simulation 1 can keep the invader further away from the territory than the defenders



TP: testing phase,
TE: training episode,
***TP*^{Trl}(*TE*)**: the testing phase at the *TE*th training episode for trial # *Trl*. In each testing phase, we calculate the average distance $\overline{dist}_{IT,t1}$ for test 1 and $\overline{dist}_{IT,t2}$ for test 2
S1: simulation 1 where players play the game without shaping for 2000 training episodes,
S2: simulation 2 where players play the game with shaping for 2000 training episodes,
Trail: each trial contains two simulations (S1 and S2) running at the same time.

Figure 5.7: Simulation procedure in a three-player grid game of guarding a territory

in simulation 2. For example, the results at 200th training episode in Table 5.1 shows that the invader from simulation 1 can move close to the territory at an average distance of 4.72 when playing against the defenders from simulation 2, while the invader from simulation 2 can only move close to the territory at an average distance of 4.91 when playing against the defenders from simulation 1. In simulation 1, all the players are trained without using shaping functions. In simulation 2, all the players are trained using the shaping function provided in (5.27). Therefore, Table 5.1 verifies that the shaping function *Shaping* 1 in (5.27) does not help the players achieve a better performance.

Table 5.1: Comparison of WoLF-PHC learning algorithms with and without shaping: Case 1

Training Episode (TE)	Testing phase at TEth training episode		Who has better performance
	Test 1 ($\overline{Dist}_{IT,t1}(TE)$): Invader from S1 v.s. Defenders from S2	Test 2 ($\overline{Dist}_{IT,t2}(TE)$): Defenders from S1 v.s. Invader from S2	
200	4.72	4.91	players from S1
400	4.84	5.02	players from S1
600	4.89	5.11	players from S1
800	5.03	5.16	players from S1
1000	5.02	5.16	players from S1
1200	4.95	5.04	players from S1
1400	5.03	5.07	players from S1
1600	5.03	5.13	players from S1
1800	5.09	5.33	players from S1
2000	4.97	5.31	players from S1

According to the payoff given in (5.24), the defenders's goal is to keep the invader away from the territory. Based on *Shaping* 1 in (5.27), the goal becomes that two

defenders try to move close to the invader. Therefore, we need to redesign our shaping function that can better represent the goal of the defenders in the game. We define a new shaping function, called *Shaping* 2, as follows

$$\begin{aligned} \Phi_I(s) &= -dist_{IT} \\ \Phi_{D_i}(s) &= dist_{D_iT} - dist_{D_iI}, \quad (i = 1, 2) \end{aligned} \tag{5.32}$$

where $dist_{D_iT}$ is the Manhattan distance between the defender i (i=1,2) and the territory at the current time step. Equation (5.32) implies that the defender i's aim is to intercept the invader while moving away from the territory. Therefore, this new shaping function stays closer to the real goal of the defenders in the game rather than the previous shaping function in (5.27). We now apply the new shaping function *Shaping* 2 to the players and run simulations again. Table 5.2 shows that the values in the second column for test 1 are greater than the values in the third column for test 2. This implies that, compared with the defenders from simulation 1 (without the shaping function), the defenders from simulation 2 (with the new shaping function) can keep the invader further away from the territory. Although the new shaping function might not be the ideal shaping function, the aid of this shaping function does improve the learning performance of the players.

5.5 Summary

A potential-based shaping method can be used to deal with the temporal credit assignment problem and speed up the learning process in MDPs. In this chapter, we extended the potential-based shaping method from Markov decision processes to general-sum stochastic games. We proved that the potential-based shaping reward applied to a general-sum stochastic game will not change the original Nash equilibrium

Table 5.2: Comparison of WoLF-PHC learning algorithms with and without shaping: Case 2

Training Episode (TE)	Testing phase at TEth training episode		Who has better performance
	Test 1 ($\overline{Dist}_{IT,t1}(TE)$): Invader from S1 v.s. Defenders from S2	Test 2 ($\overline{Dist}_{IT,t2}(TE)$): Defenders from S1 v.s. Invader from S2	
200	5.06	4.86	players from S2
400	5.28	4.96	players from S2
600	5.39	5.02	players from S2
800	5.30	4.90	players from S2
1000	5.06	4.89	players from S2
1200	5.38	4.97	players from S2
1400	5.38	4.98	players from S2
1600	5.31	4.94	players from S2
1800	5.35	4.97	players from S2
2000	5.20	5.13	players from S2

of the game. The proof of policy invariance in Sect. 5.3 has the potential to improve the learning performance of the players in a stochastic game.

Under the framework of stochastic games, two grid games were studied in this chapter. We applied Littman's friend-or-foe Q-learning algorithm to the modified Hu and Wellman's grid game. Then we applied the WoLF-PHC learning algorithm to the game of guarding a territory with two defenders and one invader. To speed up the players' learning performance, we designed two different potential-based shaping rewards to the game of guarding a territory. Simulation results showed that a good shaping function can improve the learning performance of the players, while a bad shaping function can also worsen the learning performance of the players.

Chapter 6

Reinforcement Learning in Differential Games

Future security applications will involve robots protecting critical infrastructure [42]. The robots work together to prevent the intruders from crossing the secured area. They will have to adapt to an unpredictable and continuously changing environment. Their goal is to learn what actions to take in order to get optimum performance in security tasks. This chapter addresses the learning problem for robots working in such an environment. We model this application as the "guarding a territory" game. The differential game of guarding a territory was first introduced by Isaacs [36]. In the game, the invader tries to get as close as possible to the territory while the defender tries to intercept and keep the invader as far as possible away from the territory. The Isaacs' guarding a territory game is a differential game where the dynamic equations of the players are differential equations.

A player in a differential game needs to learn what action to take if there is no prior knowledge of its optimal strategy. Learning in differential games has attracted attention in [11–13, 70, 71]. In these articles, reinforcement learning algorithms are applied to the players in the pursuit-evasion game. The study on guarding a territory game can be found in [43, 44, 72], but there is no investigation on how the players can

learn their optimal strategies by playing the game. In our research, we assume the defender has no prior knowledge of its optimal strategy nor the invader's strategy. We investigate how reinforcement learning algorithms can be applied to the differential game of guarding a territory.

In reinforcement learning, a reinforcement learner may suffer from the temporal credit assignment problem where a player's reward is delayed or only received at the end of an episodic game. When a task has a very large state space or continuous state space, the delayed reward will slow down the learning dramatically. For the game of guarding a territory, the only reward received during the game is the distance between the invader and the territory at the end of the game. Therefore, it is extremely difficult for a player to learn its optimal strategy based on this very delayed reward.

To deal with the temporal credit assignment problem and speed up the learning process, one can apply reward shaping to the learning problem. As discussed in Chapter 5, shaping can be implemented in reinforcement learning by designing intermediate shaping rewards as an informative reinforcement signal to the learning agent and reward the agent for making a good estimate of the desired behavior [5, 63, 73]. The idea of reward shaping is to provide an additional reward as a hint, based on the knowledge of the problem, to improve the performance of the agent.

Traditional reinforcement learning algorithms such as Q-leaning may lead to the curse of dimensionality problem due to the intractable continuous state space and action space. To avoid this problem, one may use fuzzy systems to represent the continuous space [74]. Fuzzy reinforcement learning methods have been applied to the pursuit-evasion differential game in [11–13]. In [12], we applied a fuzzy actor-critic learning (FACL) algorithm to the pursuit-evasion game. Experimental results showed that the pursuer successfully learned to capture the invader in an effective way [12]. In this chapter, we apply fuzzy reinforcement learning algorithms to the differential game of guarding a territory and let the defender learn its Nash equilibrium strategy

by playing against the invader. To speed up the defender's learning process, we design a shaping reward function for the defender in the game. Moreover, we apply the same FACL algorithm and shaping reward function to a three-player differential game of guarding a territory including two defenders and one invader. We run simulations to test the learning performance of the defenders in both cases.

The main contributions of this chapter are:

- Apply fuzzy reinforcement learning algorithms to the defender in the differential game of guarding a territory.
- Design a shaping reward function for the defender to speed up the learning process.
- Run simulations to test the learning performance of the defenders in both the two-player and the three-player differential game of guarding a territory.

This chapter is organized as follows. We first review the differential game of guarding a territory in Sect. 6.1. The fuzzy Q-learning (FQL) and fuzzy actor-critic reinforcement learning are presented in Sect. 6.2. Reward shaping is discussed in Sect. 6.3. Simulation results are presented in Sect. 6.4.

6.1 Differential Game of Guarding a Territory

We consider a two-player zero-sum differential game with system dynamics described as

$$\dot{\bar{x}}(t) = f(\bar{x}(t), \bar{\phi}(t), \bar{\psi}(t), t), \quad \bar{x}(t_0) = \bar{x}_0 \tag{6.1}$$

where $\bar{x}(t) \in \mathbf{R}^n$ is the state vector of dimension n, function $f(\cdot)$ determines the dynamics of the system, $\bar{\phi}$ and $\bar{\psi}$ are the strategies played by each player. The

payoff, represented as $P(\bar{\phi}, \bar{\psi})$, is given in the form

$$P(\bar{\phi}, \bar{\psi}) = h(t_f, \bar{x}(t_f)) + \int_{t_0}^{t_f} g(\bar{x}(t), \bar{\phi}, \bar{\psi}, t)ds \tag{6.2}$$

where t_f is the terminal time (or the first time the states $\bar{x}(t)$ intersect a given final condition), $h(\cdot)$ is the payoff at the terminal time, $g(\cdot)$ is the integral payoff and functions $h(\cdot)$ and $g(\cdot)$ are chosen in order to achieve an objective. We assumed that the player who uses strategy $\bar{\phi}$ wants to maximize the payoff $P(\cdot)$, whereas the player using strategy $\bar{\psi}$ wants to minimize it. Therefore, the objective of the game is to find control signals $\bar{\phi}^*$ and $\bar{\psi}^*$ such that [75]

$$P(\bar{\phi}, \bar{\psi}^*) \leq P(\bar{\phi}^*, \bar{\psi}^*) \leq P(\bar{\phi}^*, \bar{\psi}), \quad \forall\, \bar{\phi},\, \bar{\psi} \tag{6.3}$$

where $P(\bar{\phi}^*, \bar{\psi}^*)$ is the value of the game and $(\bar{\phi}^*, \bar{\psi}^*)$ is the saddle point containing both players' Nash equilibrium strategies.

The Isaacs' guarding a territory game is a two-player zero-sum differential game. The invader's goal is to reach the territory. If the invader cannot reach the territory, it at least moves to a point as close as possible to the territory [36]. Accordingly, the defender tries to intercept the invader at a point as far as possible from the territory [36]. We denote the invader as I and the defender as D in Fig. 6.1. The dynamics of the invader I and the defender D are defined as

$$\dot{x}_D(t) = \sin\theta_D, \;\; \dot{y}_D(t) = \cos\theta_D \tag{6.4}$$

$$\dot{x}_I(t) = \sin\theta_I, \;\; \dot{y}_I(t) = \cos\theta_I \tag{6.5}$$

$$-\pi \leq \theta_D \leq \pi, \;\; -\pi \leq \theta_I \leq \pi$$

where θ_D is the defender's strategy and θ_I is the invader's strategy.

In order to simplify the problem, we establish a relative coordinate frame centered at the defender's position with its y'-axis in the direction of the invader's position as shown in Fig. 6.1. The territory is represented by a circle with center T (x'_T, y'_T) and radius R. Different from θ_D and θ_I in the original coordinate frame, we define u_D as the defender's strategy and u_I as the invader's strategy in relative coordinates.

Based on (6.2), the payoff for this game is defined as

$$P_{ip}(u_D, u_I) = \sqrt{(x'_I(t_f) - x'_T)^2 + (y'_I(t_f) - y'_T)^2} - R \tag{6.6}$$

where ip denote the players' initial positions, R is the radius of the target and t_f is the terminal time. The terminal time is the time when the invader reaches the territory or the invader is intercepted before it reaches the territory. The above payoff indicates how close the invader can move to the territory if both players start from their initial positions and follow their stationary strategies u_D and u_I thereafter. In this game, the invader tries to minimize the payoff P while the defender tries to maximize it.

In Fig. 6.1, we draw the bisector BC of the segment ID. According to the dynamics of the players in (6.4) and (6.5), the players can move in any direction instantaneously with the same speed. Therefore, the region above the line BC is where the invader can reach before the defender and the region below the line BC is where the defender can reach before the invader. We draw a perpendicular line TO to the bisector BC through the point T. Then point O is the closest point on the line BC to the territory T. Starting from the initial position (I, D), if both players play their optimal strategies, the invader can only reach point O as its closest point to the territory.

The value of the game can be found as the shortest distance between the line BC

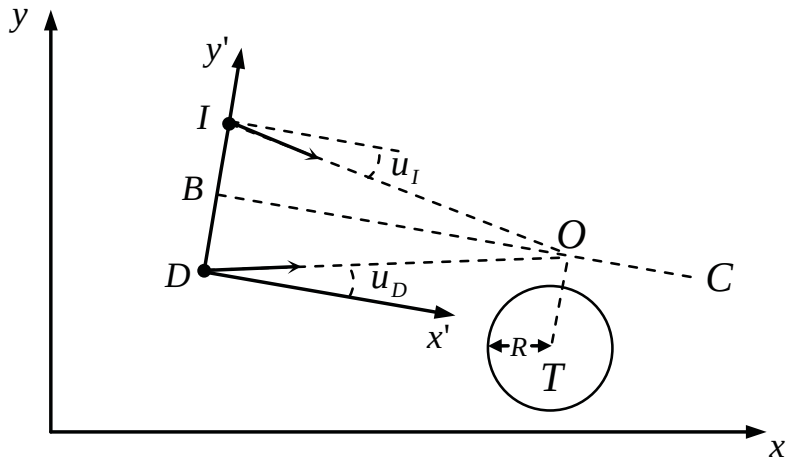


Figure 6.1: The differential game of guarding a territory

and the territory. We define the value of the game as

$$P(u_D^*, u_I^*) = \|\overrightarrow{TO}\| - R \tag{6.7}$$

where u_D^* and u_I^* are the players' Nash equilibrium strategies given by

$$u_D^* = \angle\overrightarrow{DO}, \tag{6.8}$$

$$u_I^* = \angle\overrightarrow{IO}, \tag{6.9}$$

$$-\pi \leq u_D^* \leq \pi, -\pi \leq u_I^* \leq \pi.$$

6.2 Fuzzy Reinforcement Learning

The value of the game in (6.7) is obtained based on the assumption that both players play their Nash equilibrium strategies. In practical applications, one player may not know its own Nash equilibrium strategy or its opponent's strategy. Therefore, learning algorithms are needed to help the player learn its equilibrium strategy. Most of the learning algorithms applied to differential games, especially to the pursuit-evasion

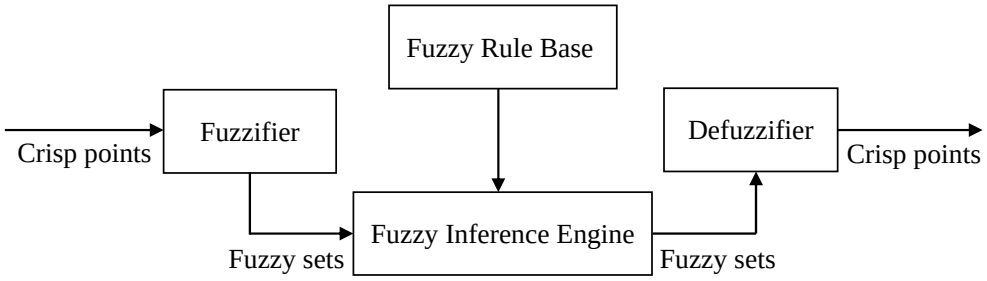


Figure 6.2: Basic configuration of fuzzy systems

game, are based on reinforcement learning algorithms [11–13].

The players' Nash equilibrium strategies given in (6.8) and (6.9) are continuous. A typical reinforcement learning approach such as Q-learning needs to discretize the action space and the state space. However, when the continuous state space or action space is large, the discrete representation of the state or action is computationally intractable [23]. Wang [76] proved that a fuzzy inference system (FIS) is a universal approximator which can approximate any nonlinear function to any degree of precision. Therefore, one can use fuzzy systems to generate continuous actions of the players or represent the continuous state space.

In this chapter, we present two fuzzy reinforcement learning algorithms for the defender to learn to play against an invader. The two fuzzy reinforcement learning methods are the fuzzy actor-critic learning (FACL) and fuzzy Q-learning (FQL), which are based on actor-critic learning and Q-learning respectively. In fuzzy reinforcement learning methods, the parameters of fuzzy systems are tuned by reinforcement signals [77].

The fuzzy system in this chapter, as shown in Fig. 6.2, is implemented by Takagi-Sugeno (TS) rules with constant consequents [78]. It consists of L rules with n fuzzy variables as inputs and one constant number as the consequent. Each rule

l ($l = 1, \ldots, L$) is of the form

$$\begin{aligned} &\text{rule } l : \text{IF } x_1 \text{ is } F_1^l, \cdots, \text{ and } x_n \text{ is } F_n^l \\ &\text{THEN } u = c^l \end{aligned} \tag{6.10}$$

where $\bar{x} = (x_1, \cdots, x_n)$ are the inputs passed to the fuzzy controller, F_i^l is the fuzzy set related to the corresponding fuzzy variable, u is the rule's output, and c^l is a constant that describes the center of a fuzzy set. If we use the product inference for fuzzy implication [76], t -norm, singleton fuzzifier and center-average defuzzifier, the output of the system becomes

$$U(\bar{x}) = \frac{\sum_{l=1}^{L}((\prod_{i=1}^{n} \mu^{F_i^l}(x_i)) \cdot c^l)}{\sum_{l=1}^{L}(\prod_{i=1}^{n} \mu^{F_i^l}(x_i))} = \sum_{l=1}^{L} \Phi^l c^l \tag{6.11}$$

where $\mu^{F_i^l}$ is the membership degree of the fuzzy set F_i^l and

$$\Phi^l = \frac{\prod_{i=1}^{n} \mu^{F_i^l}(x_i)}{\sum_{l=1}^{L}(\prod_{i=1}^{n} \mu^{F_i^l}(x_i))}. \tag{6.12}$$

6.2.1 Fuzzy Q-Learning

Among fuzzy reinforcement learning algorithms, one may use a fuzzy Q-learning algorithm to generate a global continuous action for the player based on a predefined discrete action set [74,79,80]. We assume that the player has m possible actions from an action set $A = \{a_1, a_2, \cdots, a_m\}$. To generate the player's global continuous action,

we use the following form of fuzzy IF-THEN rules

$$\begin{aligned} \text{rule } l \ : \ \text{IF } x_1 \text{ is } F_1^l, \cdots, \text{ and } x_n \text{ is } F_n^l \\ \text{THEN } \ u = a^l \end{aligned} \tag{6.13}$$

where a^l is the chosen action from the player's discrete action set A for rule l. The action a^l is chosen based on an exploration-exploitation strategy [5]. In this chapter, we use the ε-greedy policy as the exploration-exploitation strategy. The ε-greedy policy is defined such that the player chooses a random action from the player's discrete action set A with a probability ε and a greedy action with a probability $1-\varepsilon$. The greedy action is the action that gives the maximum value in an associated Q-function. Then we have

$$a^l = \begin{cases} \text{random action from } A & \text{Prob}(\varepsilon) \\ \\ \arg\max\limits_{a \in A} \big(Q(l,a)\big) & \text{Prob}(1-\varepsilon) \end{cases} \tag{6.14}$$

where $Q(l,a)$ is the associated Q-function given the rule l and the player's action $a \in A$. Based on (6.11), the global continuous action at time t becomes

$$U_t(\bar{x}_t) = \sum_{l=1}^{L} \Phi_t^l a_t^l \tag{6.15}$$

where Φ_t^l is given by (6.12), $\bar{x}_t = (x_1, x_2, \ldots, x_n)$ are the inputs, L is the number of fuzzy IF-THEN rules and a_t^l is the chosen action in (6.14) for rule l at time t.

Similar to (6.15), we can generate the global Q-function by replacing c^l in (6.11) with $Q_t(l, a_t^l)$ and get

$$\mathbf{Q}_t(\bar{x}_t) = \sum_{l=1}^{L} \Phi_t^l Q_t(l, a_t^l). \tag{6.16}$$

We can also define $\mathbf{Q}_t^*(\bar{x}_t)$ as the global Q-function with the maximum Q-value for each rule. Then (6.16) becomes

$$\mathbf{Q}_t^*(\bar{x}_t) = \sum_{l=1}^{L} \Phi_t^l \max_{a \in A} Q_t(l, a) \tag{6.17}$$

where $\max_{a \in A} Q_t(l, a)$ denotes the maximum value of $Q_t(l, a)$ for all $a \in A$ in rule l.

Given (6.16) and (6.17), we define the temporal difference error as

$$\widetilde{\varepsilon}_{t+1} = r_{t+1} + \gamma \mathbf{Q}_t^*(\bar{x}_{t+1}) - \mathbf{Q}_t(\bar{x}_t) \tag{6.18}$$

where $\gamma \in [0, 1)$ is the discount factor and r_{t+1} is the received reward at time $t + 1$. According to [74], the update law for the Q-function is given as

$$Q_{t+1}(l, a_t^l) = Q_t(l, a_t^l) + \eta \widetilde{\varepsilon}_{t+1} \Phi_t^l, \quad (l = 1, ..., L) \tag{6.19}$$

where η is the learning rate.

The FQL algorithm is summarized in Algorithm 6.1.

Algorithm 6.1 FQL algorithm

1: Initialize $Q(\cdot) = 0$ and $\mathbf{Q}(\cdot) = 0$;
2: **for** Each time step **do**
3: Choose an action for each rule based on (6.14) at time t;
4: Compute the global continuous action $U_t(\bar{x}_t)$ in (6.15);
5: Compute $\mathbf{Q}_t(\bar{x}_t)$ in (6.16);
6: Take the global action $U_t(\bar{x}_t)$ and run the game;
7: Obtain the reward r_{t+1} and the new inputs $\bar{x}_{t+1}$ at time $t + 1$;
8: Compute $\mathbf{Q}_t^*(\bar{x}_{t+1})$ in (6.17);
9: Compute the temporal difference error $\widetilde{\varepsilon}_{t+1}$ in (6.18);
10: Update $Q_{t+1}(l, a_t^l)$ in (6.19) for $l = 1, ..., L$;
11: **end for**

Example 6.1. We present an example to show the learning performance of the FQL

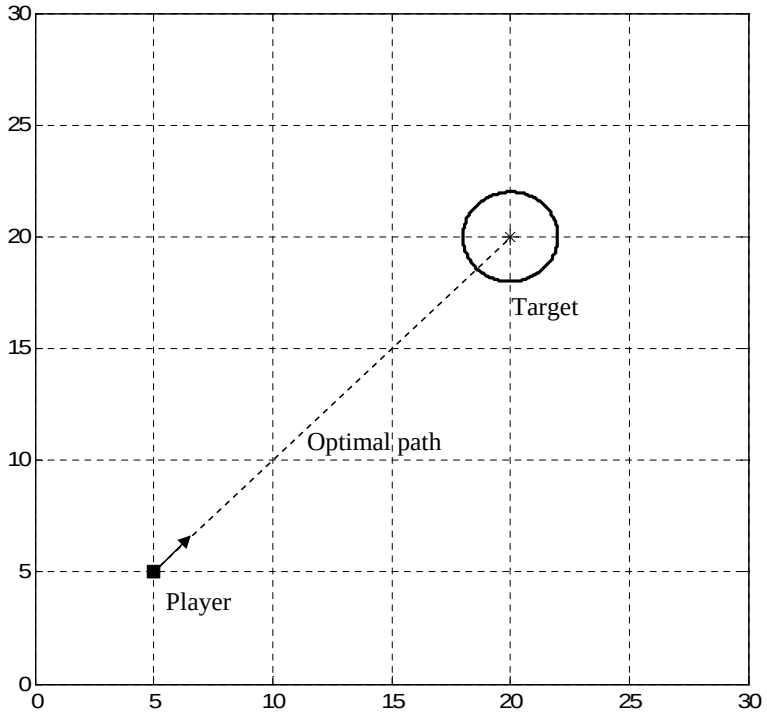


Figure 6.3: An example of FQL algorithm

algorithm. In this example, we let a player move towards a target. The playing field is a two-dimensional space shown in Fig. 6.3. The player starts from its initial position at $(5, 5)$ and tries to reach the target. The target is a circle with the center at $(20, 20)$ and a radius of 2 units. The player's speed is 1 unit/second and it can move to any direction spontaneously. The goal of the player is to reach the target in minimum time. The optimal strategy for the player is going straight to the target. The game starts from the player's initial position at $(5, 5)$ and ends when the player reaches the target or the edges of the playing field. If the player starts from the initial position $(5, 5)$, then the optimal path is the straight line between the player's initial position $(5, 5)$ and the center of the target at $(20, 20)$.

We now apply the FQL algorithm in Algorithm 6.1 to the game. The player uses the FQL algorithm to learn its optimal path. In order to apply the FQL algorithm to

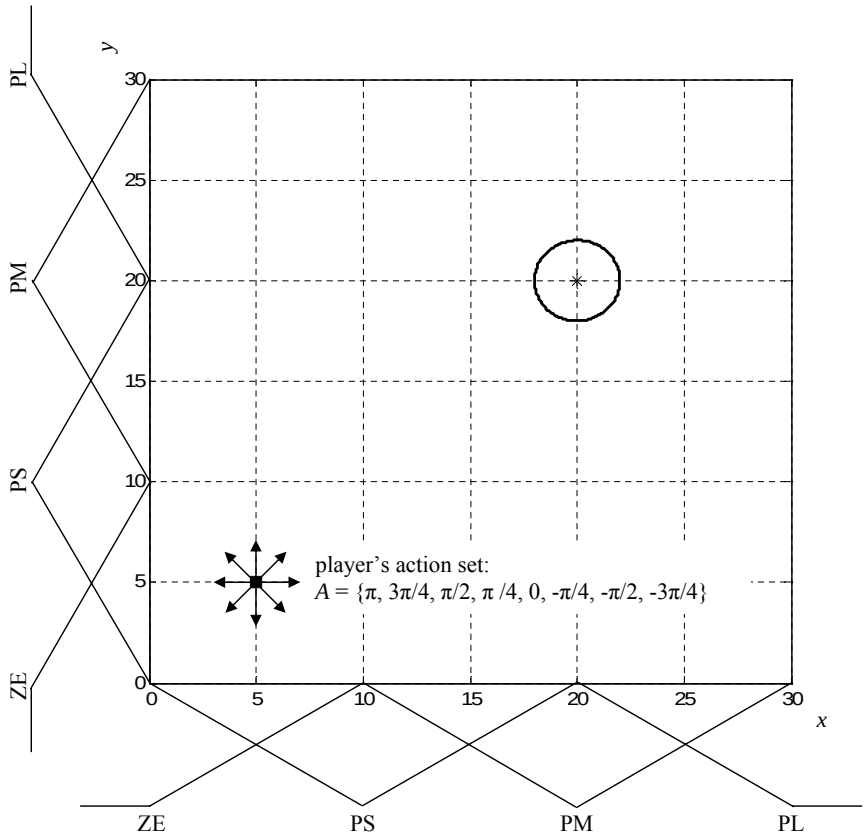


Figure 6.4: An example of FQL algorithm: action set and fuzzy partitions

a game in a continuous domain, the continuous action space needs to be discretized into an action set A. For this game, we discretize the player's action space into an action set with 8 actions. These 8 actions are the player's turning angles given as $A = \{\pi, 3\pi/4, \pi/2, \pi/4, 0, -\pi/4, -\pi/2, -3\pi/4\}$. In this example, we use fuzzy systems to represent the continuous state space. We define four fuzzy sets for each coordinate in the state space. To reduce the computational load, the fuzzy membership function $\mu^{F_i^l}(x_i)$ in (6.11) is defined as a triangular membership function (MF). Fig. 6.4 shows the membership functions for the coordinates on the plane. The number of fuzzy rules is $4 \times 4 = 16$. Each rule l has the associated $Q(l, a)$ where $l = 1, ..., 16$ and

$a \in A$. For example, rule 1 has the form of

$$\text{rule 1: IF } x \text{ is ZE and } y \text{ is ZE}$$
$$\text{THEN } u = \begin{cases} \text{random action from } A & \text{Prob}(\varepsilon) \\ \arg\max_{a \in A}\big(Q(1,a)\big) & \text{Prob}(1-\varepsilon) \end{cases} \tag{6.20}$$

where $Q(1,a)$ is the associated Q function for rule 1.

For each movement at time t, the player receives a reward signal r_{t+1}. The player's goal is to reach the target in the shortest path or shortest time. Therefore, we present the following reward function r as

$$r = dist_{PT}(t) - dist_{PT}(t+1) \tag{6.21}$$

where $dist_{PT}(t)$ denotes the distance between the player and the target at time t. This reward function encourages the player to move towards the target. For example, if the player moves closer to the target, the player receives a positive reward. If the player's action leads to the opposite direction to the target, the player receives a negative reward.

We set the following parameters for the FQL algorithm. The discount factor γ in (6.18) is set to 0.9 and the learning rate α in (6.19) is set to 0.1. The exploration parameter ε is chosen as 0.2. We run the simulation for 200 episodes. Fig. 6.5 shows the result where the lower line is the player's moving trajectory before learning and the upper line is the player's moving trajectory after learning.

6.2.2 Fuzzy Actor-Critic Learning

In fuzzy Q-learning, one has to define the player's action set A based on the knowledge of the player's continuous action space. Suppose we do not know how large the action

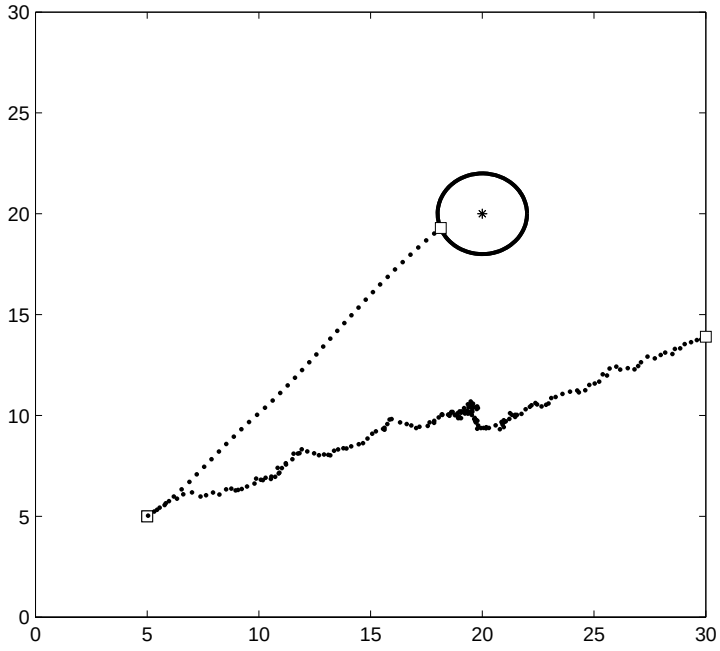


Figure 6.5: An example of FQL algorithm: simulation results

space is or the exact region the action space is in, the determination of the action set becomes difficult. Moreover, the number of elements in the action set will be prohibitively large when the action space is large. Correspondingly, the dimension of the Q function in (6.19) will be intractably large. To avoid this, we present in this section a fuzzy actor-critic learning method.

The actor-critic learning system contains two parts: one is to choose the optimal action for each state called the actor, and the other is to estimate the future system performance called the critic. Figure 6.6 shows the architecture of the actor-critic learning system. The actor is represented by an adaptive fuzzy controller which is implemented as a FIS. We also propose to implement the critic as a FIS. We have implemented the adaptive fuzzy critic in [13, 81]. We showed that the adaptive fuzzy critic in [13] performed better than the neural network proposed in [23]. In the implementation proposed in this chapter, we only adapt the output parameters

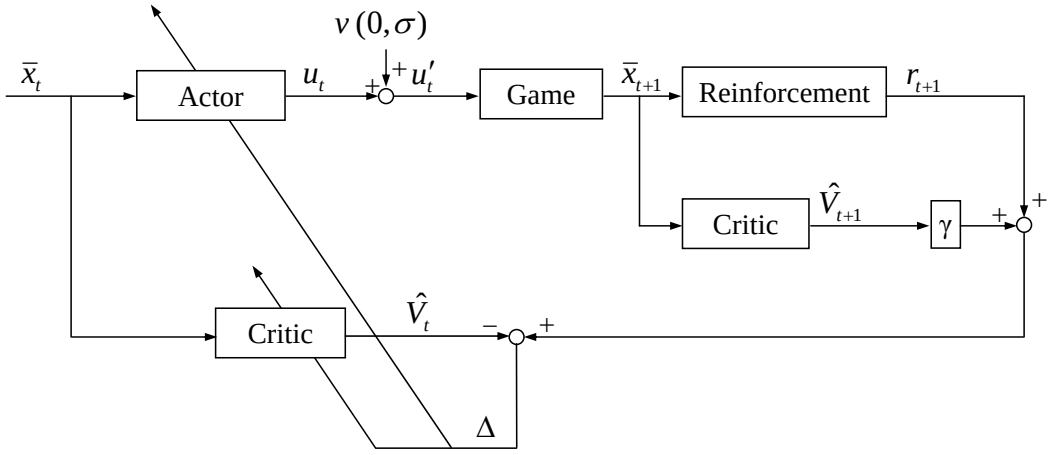


Figure 6.6: Architecture of the actor-critic learning system

of the fuzzy system, whereas in [13] the input and output parameters of the fuzzy system are adapted which is a more complex adaptive algorithm. The reinforcement signal r_{t+1} is used to update the output parameters of the adaptive controller and the adaptive fuzzy critic as shown in Fig. 6.6.

The actor is represented by an adaptive fuzzy controller which is implemented by TS rules with constant consequents. Then the output of the fuzzy controller becomes

$$u_t = \sum_{l=1}^{L} \Phi^l w_t^l \tag{6.22}$$

where w^l is the output parameter of the actor.

In order to promote exploration of the action space, a random white noise $v(0, \sigma)$ is added to the generated control signal u. The output parameter of the actor w^l is adapted as

$$w_{t+1}^l = w_t^l + \beta \Delta \left(\frac{u_t' - u_t}{\sigma} \right) \frac{\partial u}{\partial w^l} \tag{6.23}$$

where $\beta \in (0, 1)$ is the learning rate for the actor.

In order to avoid large adaptation steps in the wrong direction [24], we use only the sign of the prediction error Δ and the exploration part $(u_t' - u_t)/\sigma$ in (6.23). Then

equation (6.23) becomes

$$w_{t+1}^l = w_t^l + \beta \text{sign}\left\{\Delta\left(\frac{u_t' - u_t}{\sigma}\right)\right\}\frac{\partial u}{\partial w^l} \tag{6.24}$$

where

$$\frac{\partial u}{\partial w^l} = \frac{\prod_{i=1}^{n} \mu^{F_i^l}(x_i)}{\sum_{l=1}^{L}(\prod_{i=1}^{n} \mu^{F_i^l}(x_i))} = \Phi_t^l. \tag{6.25}$$

The task of the critic is to estimate the value function over a continuous state space. The value function is the expected sum of discounted rewards defined as

$$V_t = E\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\right\} \tag{6.26}$$

where t is the current time step, r_{t+k+1} is the received immediate reward at the time step $t+k+1$ and $\gamma \in [0, 1)$ is a discount factor.

After each action selection from the actor, the critic evaluates the new state to determine whether things have gone better or worse than expected. For the critic in Fig. 6.6, we assume TS rules with constant consequents [24]. The output of the critic $\hat{V}$ is an approximation to V given by

$$\hat{V}_t = \sum_{l=1}^{L} \Phi^l \zeta_t^l \tag{6.27}$$

where t denotes a discrete time step, ζ_t^l is the output parameter of the critic defined as c^l in (6.10) and Φ^l is defined in (6.12).

Based on the above approximation $\hat{V}_t$, we can generate a prediction error Δ as

$$\Delta = r_{t+1} + \gamma \hat{V}_{t+1} - \hat{V}_t. \tag{6.28}$$

This prediction error is then used to train the critic. Supposing it has the parameter ζ^l to be adapted, the adaptation law would then be

$$\zeta_{t+1}^l = \zeta_t^l + \alpha\Delta\frac{\partial\hat{V}}{\partial\zeta^l} \tag{6.29}$$

where $\alpha \in (0,1)$ is the learning rate for the critic. We set $\beta < \alpha$, where β is given in (6.23), so that the actor will converge slower than the critic to prevent instability in the actor [81]. Also the partial derivative is easily calculated to be

$$\frac{\partial\hat{V}}{\partial\zeta^l} = \frac{\prod_{i=1}^{n}\mu^{F_i^l}(x_i)}{\sum_{l=1}^{L}(\prod_{i=1}^{n}\mu^{F_i^l}(x_i))} = \Phi^l. \tag{6.30}$$

The FACL learning algorithm is summarized in Algorithm 6.2.

Algorithm 6.2 FACL algorithm

1: Initialize $\hat{V} = 0$, $\zeta^l = 0$ and $w^l = 0$ for $l = 1, ..., L$.
2: **for** Each time step **do**
3: Obtain the inputs $\bar{x}_t$.
4: Calculate the output of the actor u_t in (6.22).
5: Calculate the output of the critic $\hat{V}_t$ in (6.27).
6: Run the game for the current time step.
7: Obtain the reward r_{t+1} and new inputs $\bar{x}_{t+1}$.
8: Calculate $\hat{V}_{t+1}$ based on (6.27).
9: Calculate the prediction error Δ in (6.28).
10: Update ζ_{t+1}^l in (6.29) and w_{t+1}^l in (6.24).
11: **end for**

Example 6.2. We use the same example as introduced in Example 6.1. The player starts from the initial position at $(5, 5)$ and tries to reach the target at $(20, 20)$, as shown in Fig. 6.3. We apply the FACL algorithm in Algorithm 6.2 to the example. The fuzzy membership functions are chosen the same as the ones described in Fig. 6.4. The player's reward function is chosen the same as in (6.21). The parameters of

the FACL algorithm are chosen as follows. The learning rate α in (6.29) is set to 0.1 and β in (6.23) is set to 0.05. The discount factor is chosen as $\gamma = 0.9$ in (6.28).

We run the simulation for 200 episodes. Figure 6.7 shows the result. The lower line is the player's moving trajectory before learning. Since the initial value of w^l is set to zero, the output of the fuzzy controller in (6.22) which is the turning angle of the player is zero before learning. Thus, the player's moving trajectory is a horizontal line at the beginning. After 200 episodes of learning, the upper line in Figure 6.7 shows the player's moving path. After learning, the player's moving path is close to the optimal path which is a straight line between the player's initial position and the center of the target.

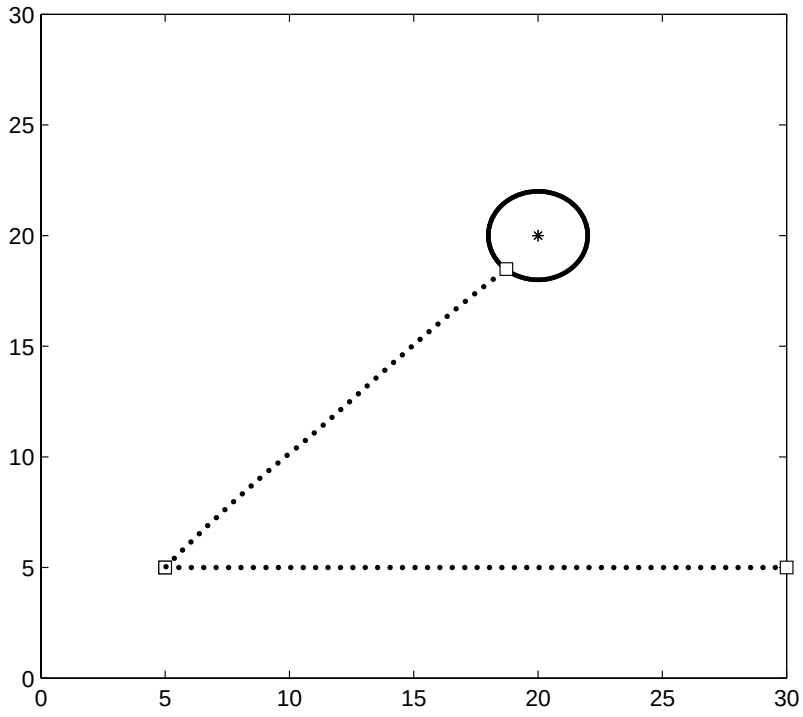


Figure 6.7: An example of FACL algorithm: simulation results

6.3 Reward Shaping in the Differential Game of Guarding a Territory

In reinforcement learning, the player may suffer from the temporal credit assignment problem where a reward is only received after a sequence of actions. For example, players in a soccer game are only given rewards after a goal is scored. This will lead to the difficulty in distributing credit or punishment to each action from a long sequence of actions. We define the terminal reward when the reward is received only at the terminal time. If the reinforcement learning problem is in the continuous domain with only a terminal reward, it is almost impossible for the player to learn without any information other than this terminal reward.

In the differential game of guarding a territory, the reward is received only when the invader reaches the territory or is intercepted by the defender. According to the payoff function given in (6.6), the terminal reward for the defender is defined as

$$R_D = \begin{cases} Dist_{IT} & \text{the defender captures the invader} \\ 0 & \text{the invader reaches the territory} \end{cases} \tag{6.31}$$

where $Dist_{IT}$ is the distance between the invader and the territory at the terminal time. Since we only have terminal rewards in the game, the learning process of the defender will be prohibitively slow. To solve this, one can use a shaping reward function for the defender to compensate for the lack of immediate rewards.

The purpose of reward shaping is to improve the learning performance of the player by providing an additional reward to the learning process. But the question is how to design good shaping reward functions for different types of games. In the

pursuit-evasion game, the immediate reward is defined as

$$r_{t+1} = Dist_{ID}(t) - Dist_{ID}(t+1) \tag{6.32}$$

where $Dist_{ID}(t)$ denotes the distance between the pursuer and the evader at time t. One might consider the above immediate reward as the shaping reward function for the differential game of guarding a territory. However, the immediate reward in (6.32) is not a good candidate for the shaping reward function in our game. The goal of the pursuer is to minimize the distance between the pursuer and the evader at each time step. Different from the pursuer, the goal of the defender in the differential game of guarding a territory is to keep the invader away from the territory. Since the defender and the invader have the same speed, the defender may fail to protect the territory if the defender keeps chasing after the invader all the time.

Based on the above analysis and the characteristics of the game, we design the following shaping reward function for the defender:

$$r_{t+1} = y'_T(t) - y'_T(t+1) \tag{6.33}$$

where $y'_T(t)$ and $y'_T(t+1)$ denote the territory's relative position of the y'-axis at time t and $t+1$ respectively.

The shaping reward function in (6.33) is designed based on the idea that the defender tries to protect the territory from invasion by keeping the territory and the invader on opposite sides. In other words, if the invader is on the defender's left side, then the defender needs to move in a direction where it can keep the territory as far as possible to the right side. As shown in the relative coordinates in Fig. 6.1, the invader is located on the positive side of the y'-axis. Then the goal of the defender in Fig. 6.1 is to keep the invader on the positive side of the y'-axis and move in a

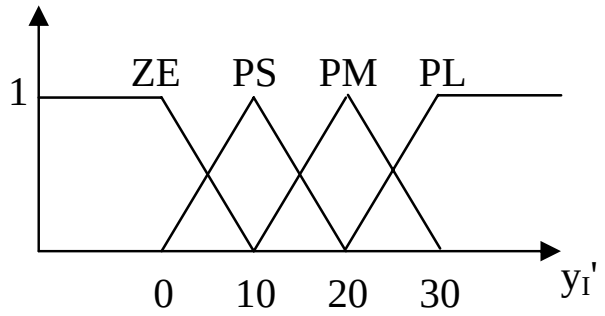
direction where it can keep the territory further to the negative side of the y'-axis.

6.4 Simulation Results

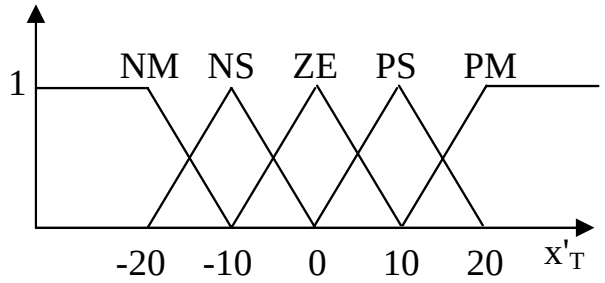
We assume that the defender does not have any information about its optimal strategy or the invader's strategy. The only information the defender has is the players' current positions. We apply the aforementioned FQL and FACL algorithms in Sect. 6.2 to the game and make the defender learn to intercept the invader. To compensate for the lack of immediate rewards, the shaping reward functions introduced in Sect. 6.3 are added to the FQL and FACL algorithms. Simulations are conducted to show the learning performance of the FQL and FACL algorithms based on different reward functions. Then we add one more defender to the game. We use the same FACL algorithm to both defenders independently. Each defender only has its own position and the invader's position as the input signals. Then the FACL algorithm becomes a completely decentralized learning algorithm in this case. We test, through simulations, how the two defenders can cooperate with each other to achieve good performance even though they do not directly share any information.

6.4.1 One Defender vs. One Invader

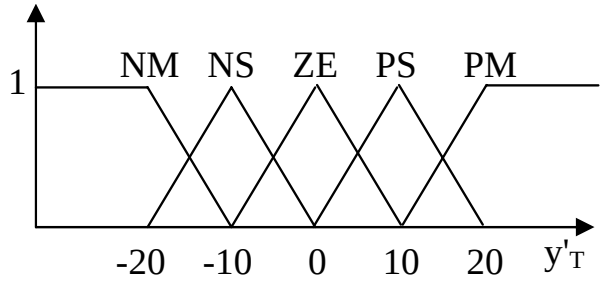
We first simulate the differential game of guarding a territory with one invade and one defender whose dynamics are given in (6.4) and (6.5). To reduce the computational load, $\mu^{F_i^l}(x_i)$ in (6.11) is defined as a triangular membership function (MF). In this game, we define 3 input variables which are the relative y-position y'_I of the invader, the relative x-position x'_T of the territory and the relative y-position y'_T of the territory. The predefined triangular membership functions for each input variable are shown in Fig. 6.8. The number of fuzzy rules applied to this game is $4 \times 5 \times 5 = 100$. The selection of the number of rules and the membership functions in the premise part of



(a) MFs for y'_I



(b) MFs for x'_T



(c) MFs for y'_T

Figure 6.8: Membership functions for input variables

the fuzzy rules is based on a priori knowledge of the game.

For the FQL algorithm, we pick the discrete action set A as

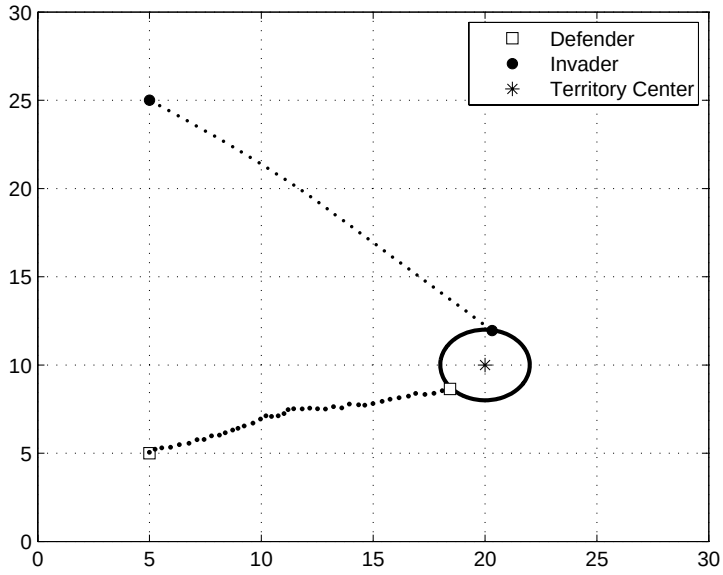
$$A = \{\pi, 3\pi/4, \pi/2, \pi/4, 0, -\pi/4, -\pi/2, -3\pi/4\}. \tag{6.34}$$

The ε-greedy policy in (6.14) is set to $\varepsilon = 0.2$. For the FACL algorithm, we set the learning rate $\alpha = 0.1$ in (6.29) and $\beta = 0.05$ in (6.23). The exploration policy in the

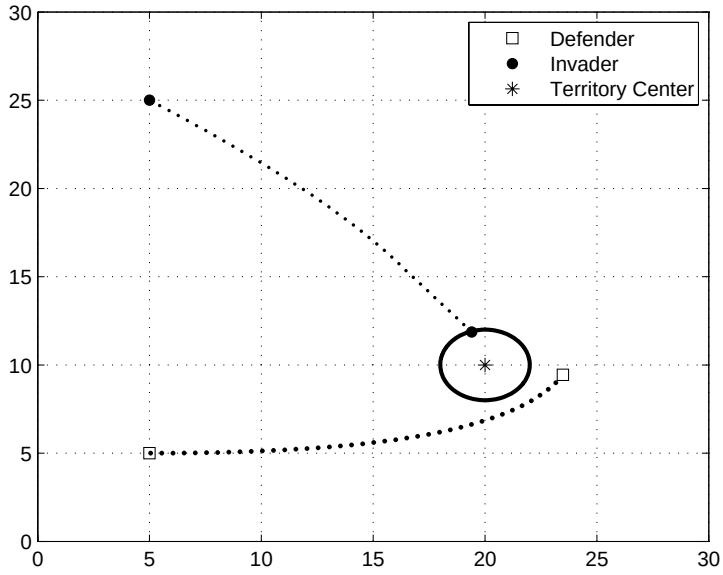
FACL algorithm is chosen as a random white noise $v(0, \sigma)$ with $\sigma = 1$. The discount factor determines the present value of future rewards [5]. To reduce the influence of the future rewards to the current state, we choose a small discount factor $\gamma = 0.5$ in (6.18) and (6.28).

We now define episodes and training trials for the learning process. An **episode** or a single run of the game is when the game starts at the players' initial positions and ends at a terminal state. A **terminal state** in this game is the state where the defender captures the invader or the invader enters the territory. A **training trial** is defined as one complete learning cycle which contains 200 training episodes. We set the invader's initial position at $(5, 25)$ for each training episode. The center of the territory is located at $(20, 10)$ with radius $R = 2$.

Example 6.3. We assume the invader plays its Nash equilibrium strategy all the time. The defender, starting at the initial position $(5, 5)$, learns to intercept the NE invader. We call the invader that always plays its Nash equilibrium strategy as the **NE invader**. We run simulations to test the performance of the FQL and FACL algorithms with different shaping reward functions introduced in Sect. 6.3. Figures 6.9 - 6.11 show the simulation results after one training trial including 200 training episodes. In Fig. 6.9, with only the terminal reward given in (6.31), the trained defender failed to intercept the invader. The same happened when the shaping reward function given in (6.32) was used to the FQL and the FACL algorithms, as shown in Fig. 6.10. As we discussed in Sect. 6.3, the shaping reward function in (6.32) is not a good candidate for this game. With the help of our proposed shaping reward function in (6.33), the trained defender successfully intercepted the invader, as shown in Fig. 6.11. This example verifies the importance of choosing a good shaping reward function for the FQL and FACL algorithms for our game.

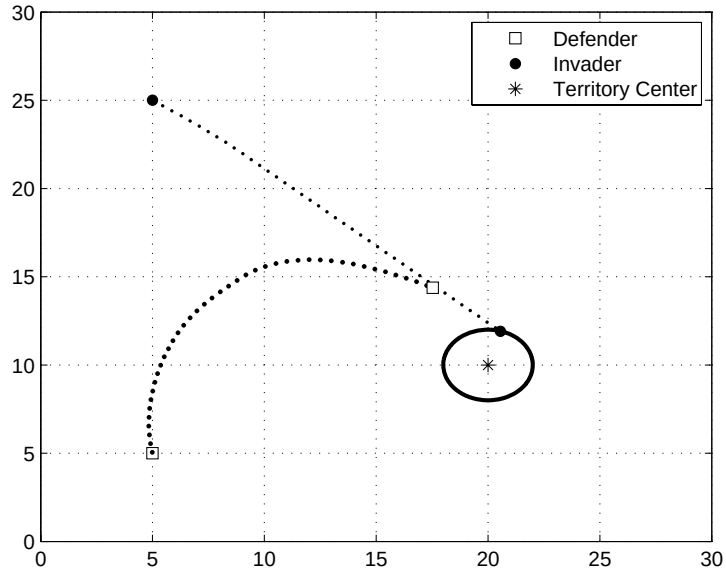


(a) Trained defender using FQL with no shaping function

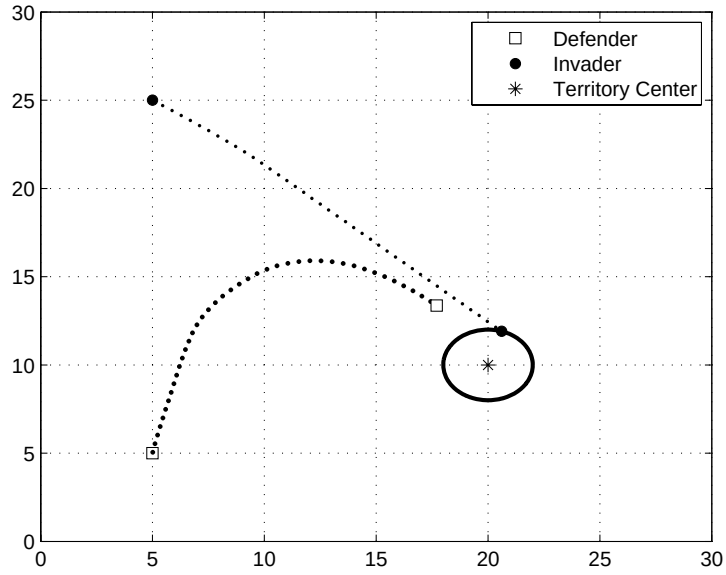


(b) Trained defender using FACL with no shaping function

Figure 6.9: Reinforcement learning with no shaping function in Example 6.3

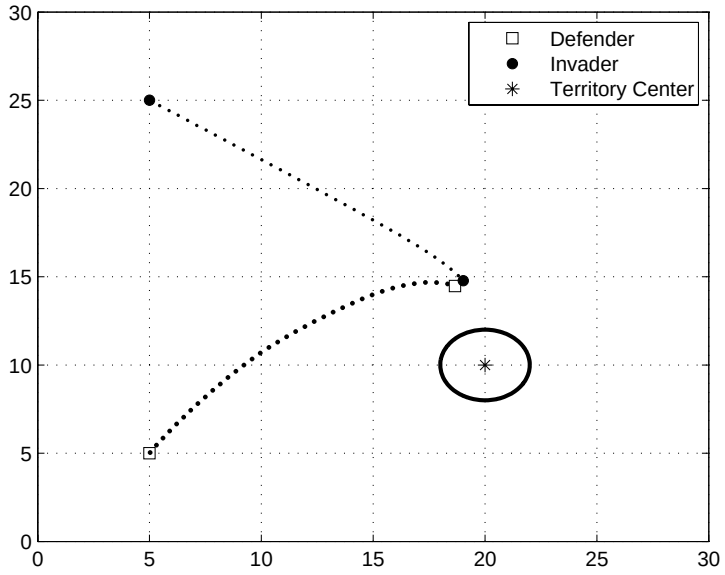


(a) Trained defender using FQL with the bad shaping function

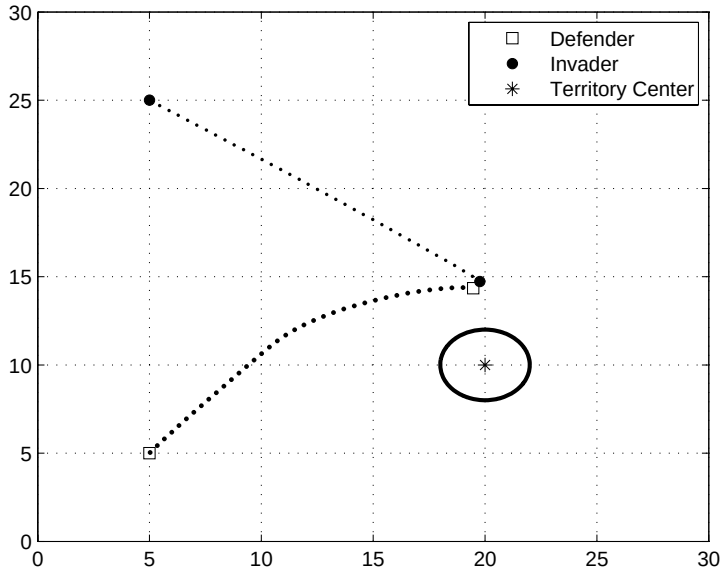


(b) Trained defender using FACL with the bad shaping function

Figure 6.10: Reinforcement learning with a bad shaping function in Example 6.3



(a) Trained defender using FQL with the good shaping function



(b) Trained defender using FACL with the good shaping function

Figure 6.11: Reinforcement learning with a good shaping function in Example 6.3

Example 6.4. In this example, we want to show the average performance of the FQL and FACL algorithms with the proposed shaping reward function given in (6.33).

The training process includes 20 training trials with 200 training episodes for each training trial. For each training episode, the defender randomly chooses one initial position from the defender's initial positions 1-4 shown in Fig. 6.12. After every 10 training episodes in each training trial, we set up a testing phase to test the performance of the defender trained so far. In the testing phase, we let the NE invader play against the trained defender and calculate the performance error as follows:

$$PE_{ip} = P_{ip}(u_D^*, u_I^*) - P_{ip}(u_D, u_I^*), \quad (ip = 1, \ldots, 6) \tag{6.35}$$

where ip represents the initial positions of the players, the payoffs $P_{ip}(u_D^*, u_I^*)$ and $P_{ip}(u_D, u_I^*)$ are calculated based on (6.6), and PE_{ip} denotes the calculated performance difference for players' initial positions ip. In this example, the invader's initial position is fixed during learning. Therefore the players' initial positions ip are represented as the defender's initial positions 1-6 shown in Fig. 6.12.

We use $PE_{ip}(TE)$ to represent the calculated performance error for the defender's initial position ip at the TEth training episode. For example, $PE_1(10)$ denotes the performance error calculated based on (6.35) for defender's initial position 1 at the 10th training episode. Then we average the performance error over 20 trials and get

$$\overline{PE}_{ip}(TE) = \frac{1}{20} \sum_{Trl=1}^{20} PE_{ip}^{Trl}(TE), \quad (ip = 1, \ldots, 6) \tag{6.36}$$

where $\overline{PE}_{ip}(TE)$ denotes the averaged performance error for players' initial position ip at the TEth training episode over 20 training trials.

Fig. 6.13 show the results where the average performance error $\overline{PE}_{ip}(TE)$ becomes smaller after learning for the FQL and the FACL algorithms. Note that the

defender's initial position 5 and 6 in Fig. 6.12 is not included in the training episodes. Although we did not train the defender's initial positions 5 and 6, the convergence of the performance errors PE_5 and PE_6 verify that the defender's learned strategy is close to its NE strategy. Compared with Fig. 6.13(a) for the FQL algorithm, the performance errors in Fig. 6.13(b) for the FACL algorithm converge closer to zero after the learning. The reason is that the global continuous action in (6.15) for the FQL algorithm is generated based on a fixed discrete action set A with only 8 elements given in (6.34). The closeness of the defender's learned action (strategy) to its NE action (strategy) is determined by the size of the action set A in the FQL algorithm. A larger size of the action set encourages the convergence of the defender's action (strategy) to its NE action (strategy), but the increasing dimension of the Q function will cause slow learning speed, as we discussed in the beginning of Sect. 6.2.2. For the FACL algorithm, the defender's global continuous action is updated directly by the prediction error in (6.28). In this way, the convergence of the defender's action (strategy) to its NE action (strategy) is better in the FACL algorithm.

6.4.2 Two Defenders vs. One Invader

We now add a second defender to the game with the same dynamics as the first defender as defined in (6.4). The payoff for this game is defined as

$$P(u_{D1}, u_{D2}, u_I) = \sqrt{(x'_I(t_f) - x'_T)^2 + (y'_I(t_f) - y'_T)^2} - R \tag{6.37}$$

where u_{D_1}, u_{D_2} and u_I are the strategies for defender 1, defender 2 and the invader respectively, and R is the radius of the target. Based on the analysis of the two-player game in Sect. 6.1, we can also find the value of the game for the three-player differential game of guarding a territory. For example, we call the grey region in

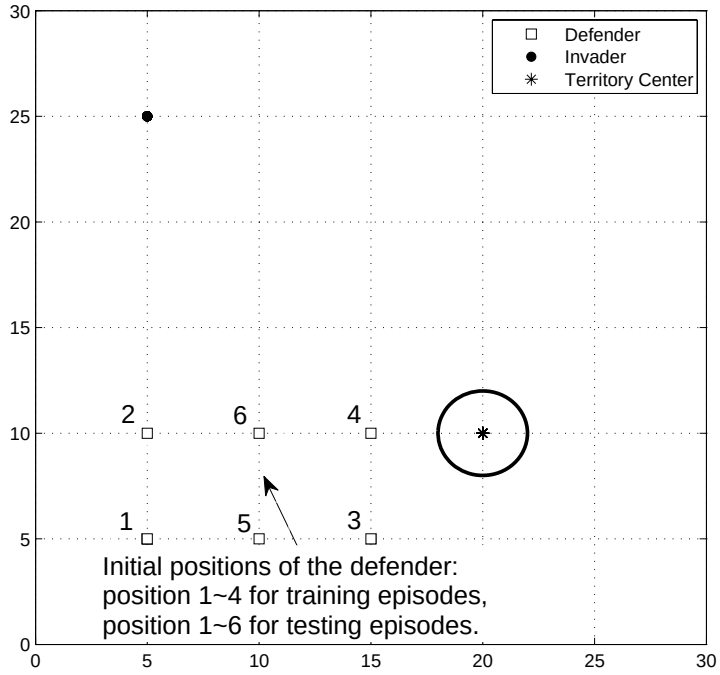
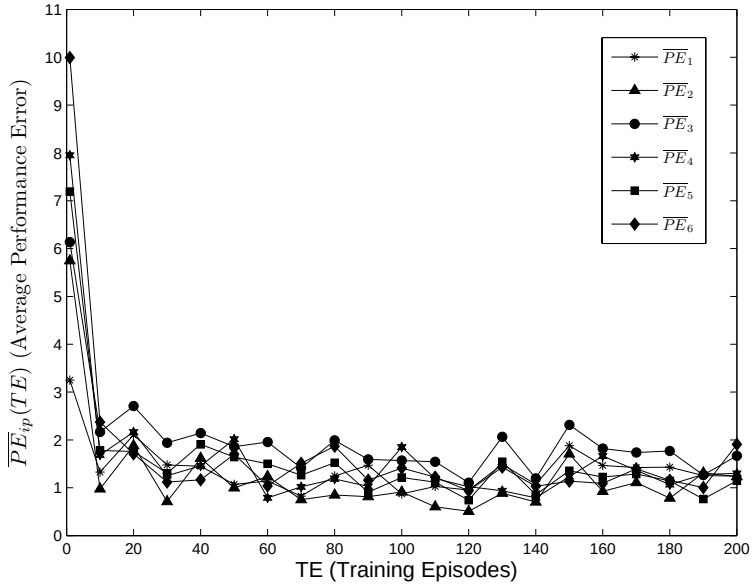


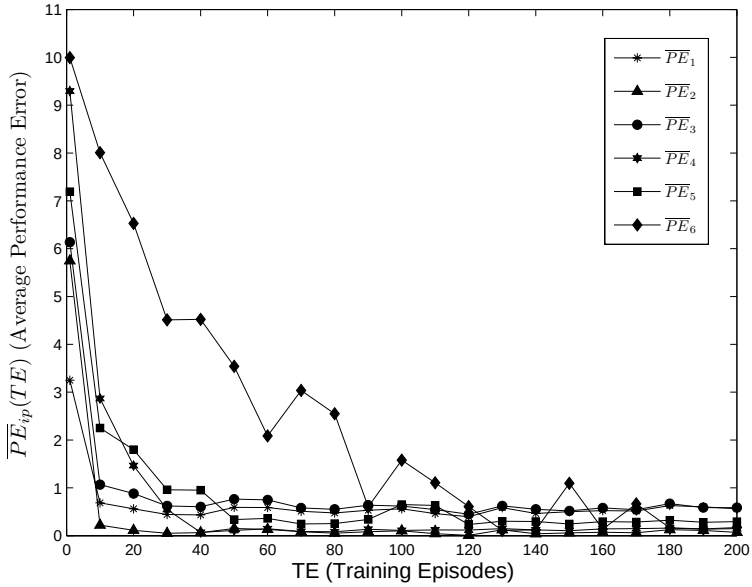
Figure 6.12: Initial positions of the defender in the training and testing episodes in Example 6.4

Fig. 6.14 as the invader's reachable region where the invader can reach before the two defenders. Then the value of the game becomes the shortest distance from the territory to the invader's reachable region. In Fig. 6.14, point O on the invader's reachable region is the closest point to the territory. Therefore, the value of the game becomes

$$P(u_{D_1}^*, u_{D_2}^*, u_I^*) = \|\overrightarrow{TO}\| - R \tag{6.38}$$



(a) Average performance error $\overline{PE}_{ip}(TE)$ $(ip = 1, ..., 6)$ in the FQL algorithm



(b) Average performance error $\overline{PE}_{ip}(TE)$ $(ip = 1, ..., 6)$ in the FACL algorithm

Figure 6.13: Example 6.4: Average performance of the trained defender vs. the NE invader

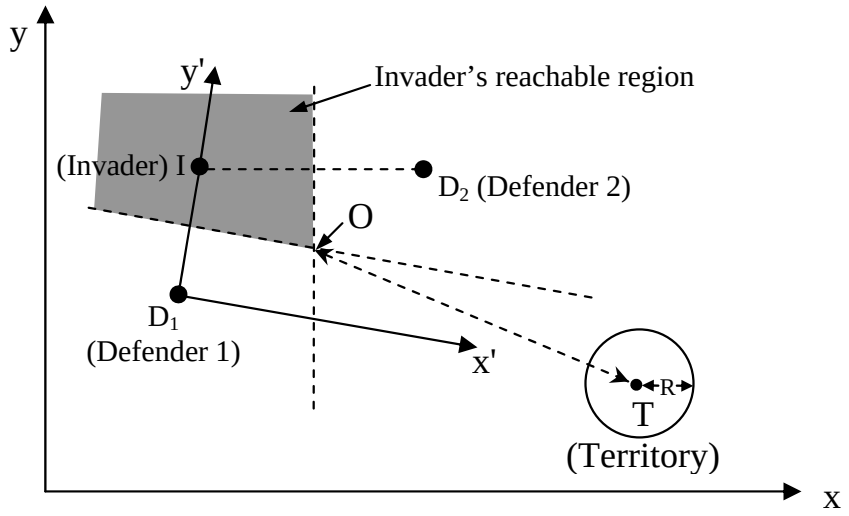


Figure 6.14: The differential game of guarding a territory with three players

where $u^*_{D_1}, u^*_{D_2}, u^*_I$ are the NE strategies for defender 1, defender 2 and the invader respectively. Based on (6.38), the players' NE strategies are given as

$$u^*_{D_1} = \angle\overrightarrow{D_1O}, \tag{6.39}$$

$$u^*_{D_2} = \angle\overrightarrow{D_2O}, \tag{6.40}$$

$$u^*_I = \angle\overrightarrow{IO}, \tag{6.41}$$

$$-\pi \leq u^*_{D_1} \leq \pi, -\pi \leq u^*_{D_2} \leq \pi, -\pi \leq u^*_I \leq \pi.$$

We apply the FACL algorithm to the game and make the two defenders learn to cooperate to intercept the invader. The initial position of the invader and the position of the target are the same as in the two-player game. Each defender in this game uses the same parameter settings of the FACL algorithm as in Sect. 6.4.1. Moreover, each defender only has the information of its own position and the invader's position without any information from the other defender. Each defender uses the same FACL algorithm independently, which makes the FACL algorithm a completely decentralized learning algorithm in this game.

Example 6.5. We assume the invader plays its Nash equilibrium strategy given in (6.41) all the time. The two defenders, starting at the initial position $(5, 5)$ for defender 1 and $(25, 25)$ for defender 2, learn to intercept the NE invader. Similar to the two-player game in Sect. 6.4.1, we run a single trial including 200 training episodes to test the performance of the FACL algorithm with different shaping reward functions given in Sect. 6.3. In Fig. 6.15, two defenders failed to intercept the NE invader with only the terminal reward and with the shaping reward function given in (6.32). On the contrary, with the proposed shaping reward function in (6.33), the two trained defenders successfully intercepted the NE evader after one training trial as shown in Fig. 6.16.

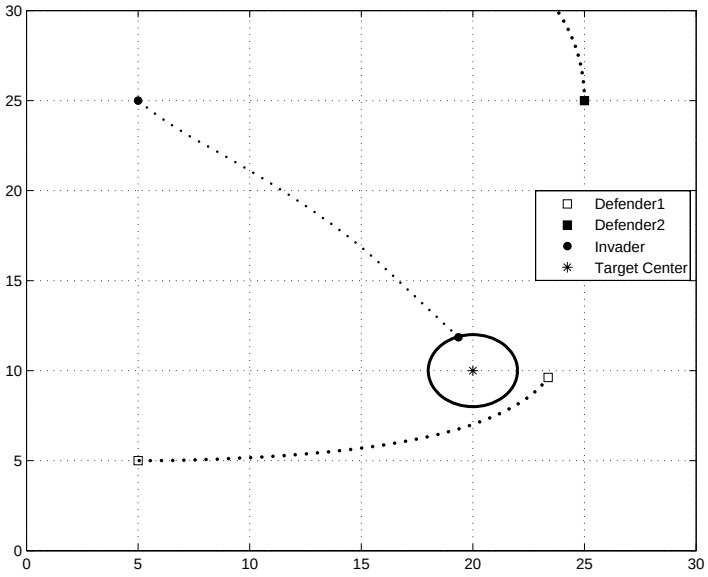
Example 6.6. In this example, we want to show the average performance of the FACL algorithm with our proposed shaping reward function for the three-player game. Similar to Example 6.4, we run 20 training trials with 200 training episodes for each training trial. For each training episode, the defender randomly chooses one initial position from the defender's initial positions 1-2 shown in Fig. 6.17(a).

After every 10 training episodes, we set up a testing phase to test the performance of the defender trained so far. The performance error in a testing phase is defined as

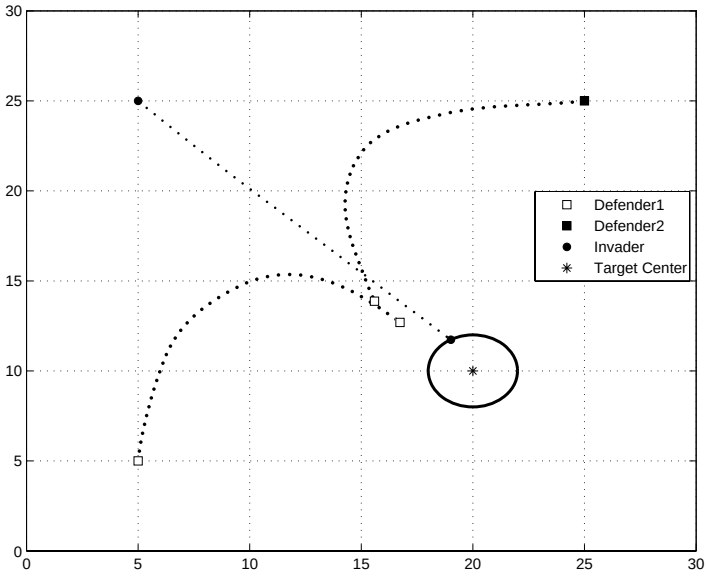
$$PE_{ip} = P_{ip}(u^*_{D_1}, u^*_{D_2}, u^*_I) - P_{ip}(u_{D_1}, u_{D_2}, u^*_I), \quad (ip = 1, \ldots, 4) \tag{6.42}$$

where ip represents the defender's initial positions 1-4 shown in Fig. 6.17(a), $P_{ip}(u^*_{D_1}, u^*_{D_2}, u^*_I)$ and $P_{ip}(u_{D_1}, u_{D_2}, u^*_I)$ are the payoffs calculated based on (6.37). Then we average the performance error over 20 trials and get

$$\overline{PE}_{ip}(TE) = \frac{1}{20} \sum_{Trl=1}^{20} PE_{ip}^{Trl}(TE), \quad (ip = 1, \ldots, 4) \tag{6.43}$$



(a) Two trained defenders using FACL with no shaping function vs. the NE invader after one training trial



(b) Two trained defenders using FACL with the bad shaping function vs. the NE invader after one training trial

Figure 6.15: Reinforcement learning without shaping or with a bad shaping function in Example 6.5

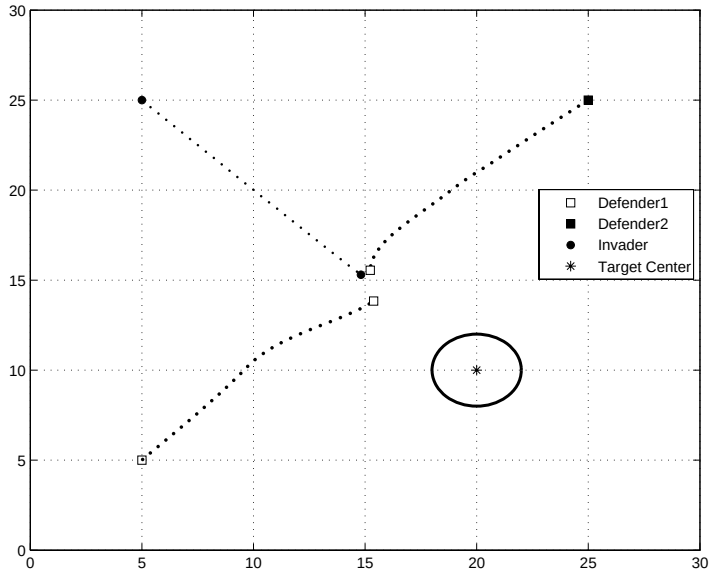


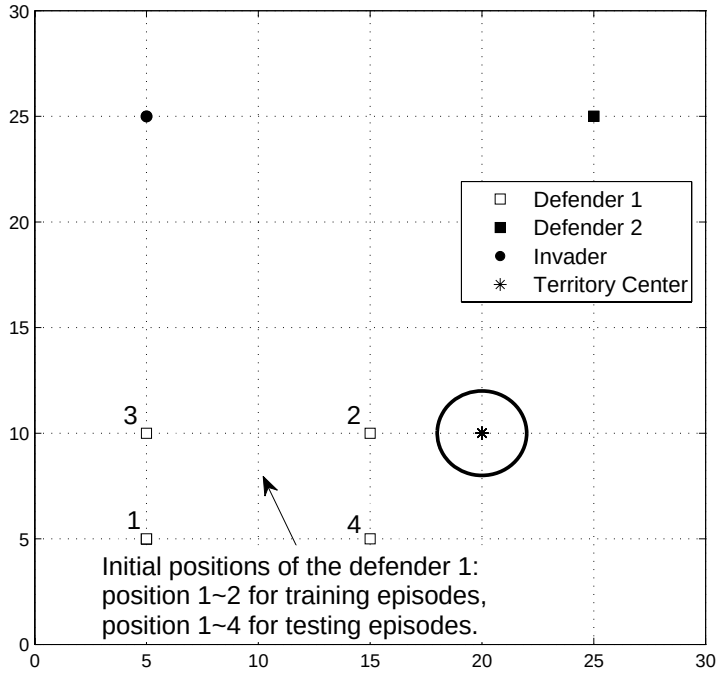
Figure 6.16: Two trained defenders using FACL with the good shaping function vs. the NE invader after one training trial in Example 6.5

where $\overline{PE}_{ip}(TE)$ denotes the averaged performance error for players' initial position ip at the TEth training episode over 20 training trials.

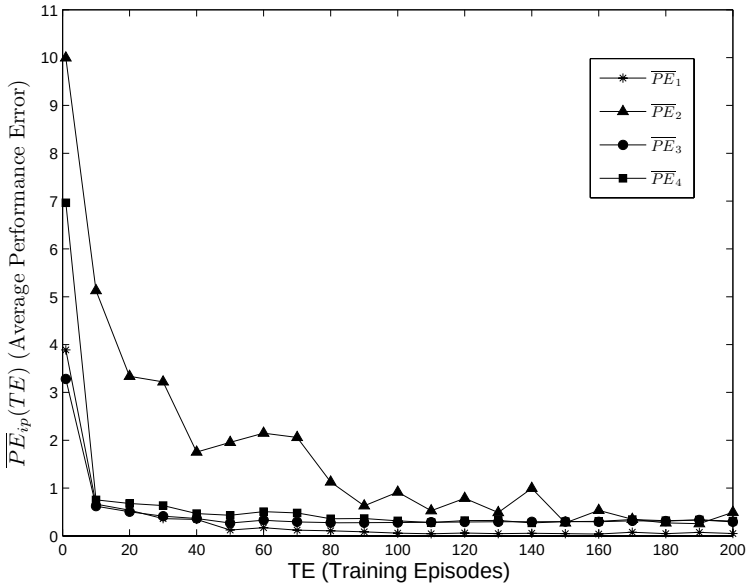
The simulation result in Fig. 6.17(b) shows that the average performance error $\overline{PE}_{ip}(TE)$ $(ip = 1, ..., 4)$ converges close to zero after 200 training episodes. Based on the simulation results, the two trained defenders successfully learned to cooperate with each other to intercept the NE invader. Compared with the two-player game with one invader and one defender in Fig. 6.11, two defenders can work together to keep the invader further away from the territory. Although there is no training performed for position 3 and 4, as shown in Fig. 6.17(a), the convergence of PE_3 and PE_4 in Fig. 6.17(b) verifies the good performance of two trained defenders. Simulation results also verify the effectiveness of the proposed shaping reward function to the FACL algorithm in the three-player differential game of guarding a territory.

6.5 Summary

This chapter presented the application of fuzzy reinforcement learning and reward shaping to the differential game of guarding a territory. The defender learns to keep the invader away from the territory with no prior knowledge of its optimal strategy. The defender's action is generated from the output of a fuzzy system. The membership functions of the consequence part of the fuzzy system are adjusted by the reinforcement signal. A shaping reward function was proposed to increase the speed of the defender's learning process. Simulation results showed that the fuzzy actor-critic learning method with reward shaping improves the overall performance of the defenders in both the two-player differential game of guarding a territory game and the three-player differential game of guarding a territory game with incomplete information.



(a) Initial positions of the players in the training and testing episodes



(b) Average performance error for the trained defenders vs. the NE invader

Figure 6.17: Example 6.6: Average performance of the two trained defenders vs. the NE invader

Chapter 7

Conclusion

In this dissertation we brought out several issues for multi-agent reinforcement learning in games. These issues include:

- Introduce Isaacs' guarding a territory game into a grid world.
- Develop a decentralized learning algorithm for a player in a multi-player game to learn its own equilibrium strategy with incomplete information.
- Analyze the affect of reward transformations on the player's learning performance and learning convergence.
- Study Isaacs' guarding a territory game as a learning problem.

In this final chapter we summarize our contributions to address these issues. We then describe some new directions for our future work based on these contributions.

7.1 Contributions

Four main contributions were presented in this dissertation.

1. In Chapter 3, we introduced the grid version of Isaacs' guarding a territory game. We defined a 6×6 grid as the playing field for the invader and the

defender. We investigated the grid game of guarding a territory under the framework of stochastic games and found the Nash equilibrium to the game. The grid game of guarding a territory was also considered as a test bed for reinforcement learning algorithms. We applied two reinforcement learning algorithms which are the minimax-Q algorithm and the WoLF-PHC algorithm to our grid game. We compared the performance of the two reinforcement learning algorithm through simulation results based on the properties of convergence and rationality described in 3.1.

2. In Chapter 4, we developed a decentralized learning algorithm called the L_{R-I} lagging anchor algorithm. In this algorithm, the player only needs to know its own action and the current state during learning. We proved that the L_{R-I} lagging anchor algorithm can guarantee the convergence to Nash equilibria in two-player two-action general-sum matrix games. Three matrix games were simulated to show the convergence of the proposed L_{R-I} lagging anchor algorithm. The convergence in the rock-paper-scissors game indicated that the L_{R-I} lagging anchor algorithm may be applicable to a two-player matrix game with more than two actions. Inspired by the WoLF-PHC algorithm, we extended the algorithm to stochastic games. We designed a practical decentralized learning algorithm for stochastic games based on the L_{R-I} lagging anchor algorithm. To test the performance of this practical algorithm, we provided Hu and Wellman's grid game and run the simulation. The convergence to a Nash equilibrium indicated the possibility of applying the practical version of the L_{R-I} lagging anchor algorithm to general-sum stochastic games.

3. In Chapter 5, we extended the idea of potential-based shaping from MDPs to multi-player general-sum stochastic games. We proved that the potential-based reward shaping method applied to a multi-player general-sum stochastic

game does not change the players' Nash equilibria in the game. To test how shaping rewards can affect the learning performance, we simulated two grid games: modified Hu and Wellman's grid game and the grid game of guarding a territory with two defenders and one invader. From simulation results, we found that good shaping rewards significantly improved the players' learning performance in both games.

4. In Chapter 6, we studied Isaacs' guarding a territory game as a reinforcement learning problem in a continuous domain. The defender in the game learns its Nash equilibrium strategy while playing against the Nash equilibrium invader. We applied two reinforcement learning algorithms which are the FQL and FACL algorithms to the game. To cope with the temporal credit assignment problem, we designed a shaping function to help the defender learn its equilibrium strategy. Furthermore, we added one more defender to Isaacs' game and applied the same FACL algorithm and shaping reward to the game. Each defender learned individually without knowing the other defender's action. Simulation results showed that the combination of the reinforcement leaarning algorithms and the shaping function helped the defenders achieve their desired performance in both the two-player and the three-player differential game of guarding a territory.

7.2 Future Work

This dissertation also opens up new future research directions for multi-agent reinforcement learning in games. Similar to the structure of the contributions, we divide these future directions into four categories.

1. In the grid game of guarding a territory, the players are playing in a 6×6 grid field. In future research, the size of the playing field can be increased from 6×6

to a bigger grid such as a 12×12 grid. Each player in the current grid game can only move one cell at a time. In the future, the players may have different speed such that the invaders may move faster than the defenders. We assume the number of defenders is greater than the number of invaders. Under this new scenario, one may want to find the Nash equilibrium for the new game or apply reinforcement learning algorithms to the new game.

2. In the current research, we proved that the L_{R-I} lagging anchor algorithm can guarantee the convergence to Nash equilibria for two-player two-action general-sum matrix games. In the future, one may extend the proof from two-player two-action general-sum matrix games to a wider range such as general-sum matrix games with more than two players. Also, one may continue investigating the practical L_{R-I} lagging anchor algorithm and its applications to stochastic games such as the multi-player grid game of guarding a territory.

3. We proved the policy invariance of potential-based reward shaping for multi-player general-sum stochastic games. But the shaping reward function has to be carefully defined based on the prior knowledge of the environment in order to achieve a good learning performance. In [82], shaping rewards are learned to improve performance on a rod positioning experiment. In [83], the potential function for shaping the reward is learned online in MDPs. In the future, for stochastic games, one may let each player learn its own shaping reward function by playing the game when the ideal shaping function is unknown to the players.

4. We had studied Isaacs' guarding a territory game as a reinforcement learning problem in a continuous domain. We applied reinforcement learning algorithms with the shaping reward function for the defenders to learn their equilibrium strategies. In the future, one may apply reinforcement learning algorithms for both the invader and the defender, and study how the players can learn their

equilibrium strategies using reinforcement learning.

Moreover, one can set up a new differential game of guarding a territory by adding more players in the game and choosing different speeds for each player. For example, three invaders try to invade the territory by playing against four defenders. The invaders are moving faster than the defenders. Then we assume the players have the least information from other players such that each player only knows all the players' position. The goal under this new scenario is to develop decentralized learning algorithms for players in the game and design a learning scheme for each player to learn its own shaping reward function.

In [12], we implemented the derived fuzzy controller on real robots to perform the pursuit-evasion game. Similar to the experiments in [12], one may consider the implementation of decentralized learning algorithms on autonomous mobile robots in the guarding a territory game.

List of References

[1] G. Weiss, ed., *Multiagent systems: a modern approach to distributed artificial intelligence*. Cambridge, MA: MIT Press, 1999.

[2] Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game Theoretic and Logical Foundations*. Cambridge: Cambridge University Press, 2008.

[3] M. Wooldridge, *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd ed., 2009.

[4] G. A. Kaminka, "Robots are agents, too!," *AgentLink News*, pp. 16–17, December 2004.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. Cambridge, Massachusetts: The MIT Press, 1998.

[6] L. Buşoniu, R. Babuška, and B. D. Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Trans Syst Man Cybern C*, vol. 38, no. 2, pp. 156–172, 2008.

[7] M. Benda, V. Jagannathan, and R. Dodhiawala, "On optimal cooperation of knowledge sources - an empirical investigation," Tech. Rep. BCS–G2010–28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, WA, USA, July 1986.

[8] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proceedings of the 11th International Conference on Machine Learning*, pp. 157–163, 1994.

[9] T. Watanabe and Y. Takahashi, "Hierarchical reinforcement learning using a modular fuzzy model for multi-agent problem," in *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pp. 1681 –1686, 2007.

[10] M. Bowling and M. Veloso, "Multiagent learning using a variable learning rate," *Artificial Intelligence*, vol. 136, no. 2, pp. 215–250, 2002.

[11] J. W. Sheppard, "Colearning in differential games," *Machine Learning*, vol. 33, pp. 201–233, 1998.

[12] S. N. Givigi, H. M. Schwartz, and X. Lu, "A reinforcement learning adaptive fuzzy controller for differential games," *Journal of Intelligent and Robotic Systems*, vol. 59, pp. 3–30, 2010.

[13] S. F. Desouky and H. M. Schwartz, "Self-learning fuzzy logic controllers for pursuit-evasion differential games," *Robotics and Autonomous Systems*, vol. 59, pp. 22–33, 2011.

[14] 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.

[15] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[16] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[17] D. P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*. Englewood Cliffs, NJ.: Prentice-Hall, 1987.

[18] R. A. Howard, *Dynamic Programming and Markov Processes*. MIT Press, 1960.

[19] D. P. Bertsekas, *Dynamic Programming and Optimal Control,3rd edn.* Athena Scientific, 2007.

[20] R. S. Sutton, *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, Amherst, MA, 1984.

[21] C. J. C. H. Watkins, *Learning from Delayed Rewards*. PhD thesis, Kings College, Cambridge, UK, 1989.

[22] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[23] X. Dai, C. Li, and A. B. Rad, "An approach to tune fuzzy controllers based on reinforcement learning for autonomous vehicle control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, no. 3, pp. 285–293, 2005.

[24] W. M. Buijtenen, G. Schram, R. Babuska, and H. B. Verbruggen, "Adaptive fuzzy control of satellite attitude by reinforcement learning," *IEEE Trans. Fuzzy Syst.*, vol. 6, no. 2, pp. 185–194, 1998.

[25] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*. John Wiley and Sons, 1944.

[26] D. Fudenberg and D. K. Levine, *The Theory of Learning in Games*. Cambridge: MIT Press, 1998.

[27] T. Başar and G. J. Olsder, *Dynamic Noncooperative Game Theory*. London, U.K.: SIAM Series in Classics in Applied Mathematics 2nd, 1999.

[28] P. Sastry, V. Phansalkar, and M. Thathachar, "Decentralized learning of Nash equilibria in multi-person stochastic games with incomplete information," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 5, pp. 769–777, 1994.

[29] M. J. Osborne, *An Introduction to Game Theory*. Oxford University Press, USA, 2003.

[30] G. Owen, *Game Theory*. San Diego, CA: Academic Press, 1995.

[31] L. S. Shapley, "Stochastic games," in *Proceedings of the National Academy of Sciences*, vol. 39, pp. 1095–1100, 1953.

[32] J. Filar and K. Vrieze, *Competitive Markov decision processes*. New York, NY, USA: Springer-Verlag New York, Inc., 1996.

[33] M. Bowling, *Multiagent Learning in the Presence of Agents with Limitations*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, May 2003.

[34] M. Bowling and M. Veloso, "Rational and convergent learning in stochastic games," in *the Seventeenth International Joint Conference on Artificial Intelligence*, pp. 1021–1026, 2001.

[35] M. Bowling, "Convergence and no-regret in multiagent learning," in *In Advances in Neural Information Processing Systems 17*, pp. 209–216, MIT Press, 2005.

[36] R. Isaacs, *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. New York, New York: John Wiley and Sons, Inc., 1965.

[37] X. Lu and H. M. Schwartz, "An investigation of guarding a territory problem in a grid world," in *American Control Conference (ACC), 2010*, pp. 3204 –3210, 2010.

[38] M. L. Littman and C. Szepesvári, "A generalized reinforcement-learning model: Convergence and applications," in *Proc. 13th International Conference on Machine Learning*, pp. 310–318, 1996.

[39] J. Hu and M. P. Wellman, "Multiagent reinforcement learning: Theoretical framework and an algorithm," in *Proceedings of the 15th International Conference on Machine Learning*, pp. 242–250, 1998.

[40] M. L. Littman, "Friend-or-foe q-learning in general-sum games," in *Proc. 18th International Conference on Machine Learning*, pp. 322–328, 2001.

[41] M. L. Littman, "Value-function reinforcement learning in markov games," in *Journal of Cognitive Systems Research*, vol. 2, pp. 55–66, 2001.

[42] R. Abielmona, E. Petriu, M. Harb, and S. Wesolkowski, "Mission-driven robotic intelligent sensor agents for territorial security," *Computational Intelligence Magazine, IEEE*, vol. 6, no. 1, pp. 55–67, 2011.

[43] K. H. Hsia and J. G. Hsieh, "A first approach to fuzzy differential game problem: guarding a territory," *Fuzzy Sets and Systems*, vol. 55, pp. 157–167, 1993.

[44] Y. S. Lee, K. H. Hsia, and J. G. Hsieh, "A strategy for a payoff-switching differential game based on fuzzy reasoning," *Fuzzy Sets and Systems*, vol. 130, no. 2, pp. 237–251, 2002.

[45] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000.

[46] S. Lakshmivarahan and K. S. Narendra, "Learning algorithms for two-person zero-sum stochastic games with incomplete information," *Mathematics of Operations Research*, vol. 6, no. 3, pp. 379–386, 1981.

[47] S. Lakshmivarahan and K. S. Narendra, "Learning algorithms for two-person zero-sum stochastic games with incomplete information: a unified approach," *SIAM Journal on Control and Optimization*, vol. 20, no. 4, pp. 541–552, 1982.

[48] P. Vrancx, K. Verbeeck, and A. Nowé, "Decentralized learning in markov games," *IEEE Trans Syst Man Cybern B*, vol. 38, no. 4, pp. 976–981, 2008.

[49] F. A. Dahl, "The lagging anchor algorithm: reinforcement learning in two-player zero-sum games with imperfect information," *Machine Learning*, vol. 49, pp. 5–37, 2002.

[50] F. A. Dahl, "The lagging anchor model for game learning — a solution to the crawford puzzle," *Journal of Economic Behavior & Organization*, vol. 57, pp. 287–303, 2005.

[51] X. Lu and H. M. Schwartz, "Decentralized learning in two-player zero-sum games: A LR-I lagging anchor algorithm," in *American Control Conference (ACC), 2011*, pp. 107 –112, 2011.

[52] X. Lu and H. M. Schwartz, "Decentralized learning in general-sum matrix games: An L_{R-I} lagging anchor algorithm," *International Journal of Innovative Computing, Information and Control*, vol. 8, 2012. to be published.

[53] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. Englewood Cliffs: Prentice Hall, 1989.

[54] S. P. Singh, M. J. Kearns, and Y. Mansour, "Nash convergence of gradient dynamics in general-sum games," in *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pp. 541–548, 2000.

[55] K. S. Narendra and M. A. L. Thathachar, "Learning automata - a survey," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. SMC-4, no. 4, pp. 323 –334, 1974.

[56] M. Thathachar and P. Sastry, *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Boston, Massachusetts: Kluwer Academic Publishers, 2004.

[57] M. Wiering, R. Salustowicz, and J. Schmidhuber, "Reinforcement learning soccer teams with incomplete world models," *Auton. Robots*, vol. 7, no. 1, pp. 77–88, 1999.

[58] B. F. Skinner, *The Behavior of Organisms: An Experimental Analysis*. New York: D. Appleton Century, 1938.

[59] O. G. Selfridge, R. S. Sutton, and A. G. Barto, "Training and tracking in robotics," in *Proceedings of the 9th international joint conference on Artificial intelligence*, pp. 670–672, 1985.

[60] V. Gullapalli and A. Barto, "Shaping as a method for accelerating reinforcement learning," in *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pp. 554 –559, 1992.

[61] M. J. Matarić, "Reward functions for accelerated learning," in *11th International Conference on Machine Learning*, pp. 181–189, 1994.

[62] M. Dorigo and M. Colombetti, "Robot shaping: developing autonomous agents through learning," *Artificial Intelligence*, vol. 71, pp. 321–370, 1994.

[63] J. Randløv and P. Alstrøm, "Learning to drive a bicycle using reinforcement learning and shaping," in *15th International Conference on Machine Learning*, pp. 463–471, 1998.

[64] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *16th International Conference on Machine Learning*, pp. 278–287, 1999.

[65] E. Wiewiora, "Potential-based shaping and q-value initialization are equivalent," *Journal of Artificial Intelligence Research*, vol. 19, pp. 205–208, 2003.

[66] J. Asmuth, M. L. Littman, and R. Zinkov, "Potential-based shaping in model-based reinforcement learning," in *Proc. 23rd AAAI Conference on Artificial Intelligence*, pp. 604–609, 2008.

[67] M. Babes, E. Munoz de Cote, and M. L. Littman, "Social reward shaping in the prisoner's dilemma," in *7th international joint conference on Autonomous agents and multiagent systems - Volume 3*, pp. 1389–1392, 2008.

[68] S. Devlin and D. Kudenko, "Theoretical considerations of potential-based reward shaping for multi-agent systems.," in *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, (Taipei, Taiwan), 2011.

[69] X. Lu, H. M. Schwartz, and S. N. Givigi, "Policy invariance under reward transformations for general-sum stochastic games," *Journal of Artificial Intelligence Research*, vol. 41, pp. 397–406, 2011.

[70] M. E. Harmon, L. C. Baird III, and A. H. Klopf, "Reinforcement learning applied to a differential game," *Adaptive Behavior*, vol. 4, no. 1, pp. 3–28, 1995.

[71] Y. Ishiwaka, T. Sato, and Y. Kakazu, "An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning," *Robotics and Autonomous Systems*, vol. 43, no. 4, pp. 245–256, 2003.

[72] Y. S. Lee, K. H. Hsia, and J. G. Hsieh, "A problem of guarding a territory with two invaders and two defenders," in *1999 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 3, pp. 863–868 vol.3, 1999.

[73] M. J. Matarić, "Reinforcement learning in the multi-robot domain," *Autonomous Robots*, vol. 4, pp. 73–83, 1997.

[74] L. Jouffe, "Fuzzy inference system learning by reinforcement methods," *IEEE Trans. Syst., Man, Cybern. C*, vol. 28, no. 3, pp. 338–355, 1998.

[75] A. E. Bryson and Y. Ho, *Applied Optimal Control: Optimization, Estimation, and Control*. Levittown, PA: Taylor & Francis, 1975. Rev. printing.

[76] L. X. Wang, *A Course in Fuzzy Systems and Control*. Englewood Cliffs, NJ: Prentice Hall, 1997.

[77] H. Berenji and P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcements," *Neural Networks, IEEE Transactions on*, vol. 3, no. 5, pp. 724 –740, 1992.

[78] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its application to modeling and control," *IEEE Trans. Syst., Man, Cybern.*, vol. 15, pp. 116–132, 1985.

[79] M. J. Er and C. Deng, "Online tuning of fuzzy inference systems using dynamic fuzzy Q-learning," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 34, no. 3, pp. 1478 –1489, 2004.

[80] A. Bonarini, A. Lazaric, F. Montrone, and M. Restelli, "Reinforcement distribution in fuzzy Q-learning," *Fuzzy Sets and Systems*, vol. 160, no. 10, pp. 1420 – 1443, 2009.

[81] S. N. Givigi, H. M. Schwartz, and X. Lu, "An experimental adaptive fuzzy controller for differential games," in *Proc. IEEE Systems, Man and Cybernetics'09*, pp. 3017 –3023, 2009.

[82] G. Konidaris and A. Barto, "Autonomous shaping: knowledge transfer in reinforcement learning," in *Proceedings of the 23rd international conference on Machine learning*, pp. 489–496, 2006.

[83] M. Grześ and D. Kudenko, "Online learning of shaping rewards in reinforcement learning," *Neural Networks*, vol. 23, pp. 541 – 550, 2010.