

California State University, San Bernardino

CSUSB ScholarWorks

---

Theses Digitization Project

John M. Pfau Library

---

2008

## Implementation of reinforcement learning in game strategy design

Chien-Yu Lin

Follow this and additional works at: <https://scholarworks.lib.csusb.edu/etd-project>



Part of the [Software Engineering Commons](#)

---

### Recommended Citation

Lin, Chien-Yu, "Implementation of reinforcement learning in game strategy design" (2008). *Theses Digitization Project*. 3497.

<https://scholarworks.lib.csusb.edu/etd-project/3497>

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact [scholarworks@csusb.edu](mailto:scholarworks@csusb.edu).

IMPLEMENTATION OF REINFORCEMENT LEARNING  
IN GAME STRATEGY DESIGN

---

A Project  
Presented to the  
Faculty of  
California State University,  
San Bernardino

---

In Partial Fulfillment  
of the Requirements for the Degree  
Master of Science  
in  
Computer Science

---

by

Chien-Yu Lin

December 2008

IMPLEMENTATION OF REINFORCEMENT LEARNING  
IN GAME STRATEGY DESIGN

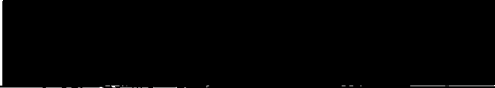
---

A Project  
Presented to the  
Faculty of  
California State University,  
San Bernardino

---

by  
Chien-Yu Lin  
December 2008

Approved by:

  
\_\_\_\_\_  
Dr. Haiyan Qiao, Chair, Computer Science

12/02/2008  
Date

  
\_\_\_\_\_  
Dr. Ernesto Gomez

  
\_\_\_\_\_  
Dr. David A. Turner

## ABSTRACT

Reinforcement learning is one type of machine learning. It is concerned with learning through the use of penalties and rewards. The main purpose of this project is to apply reinforcement learning in the design of game strategy. Using this approach, the computer learns the opponent's strategy, and learning takes place during each step of play. The reinforcement learning used in this project will be based on the Q-learning algorithm, and the game "Blackjack" is selected as the study model because of its simplicity and popularity.

In order to demonstrate that the strategy implemented with reinforcement learning performs better than pre-programmed strategies, an experimental approach is used. In the experiments, the winning percentages of different strategies with and without learning capabilities are compared when playing Blackjack. The experimental results show that the strategy with learning has a better performance than the pre-programmed strategies.

## ACKNOWLEDGMENTS

I would like to acknowledge my advisor, Dr. Haiyan Qiao for all the efforts that she had devoted to make this project possible. I would also like to thank her for all the time that she put into this project and for helping me whenever I ran into a problem. I would also like to thank Dr. Ernesto Gomez and Dr. David Turner for serving on my committee and giving me guidance along the way. I appreciate the opportunity that the faculty of Computer Science department gave me to pursue my Master of Science degree in Computer Science at California State University, San Bernardino.

## TABLE OF CONTENTS

ABSTRACT .....	iii
ACKNOWLEDGMENTS .....	iv
LIST OF TABLES .....	vii
LIST OF FIGURES .....	viii
CHAPTER ONE: INTRODUCTION	
1.1 Introduction .....	1
1.2 Problem .....	1
1.3 Background .....	2
1.4 Contribution .....	5
CHAPTER TWO: PROBLEM	
2.1 Introduction .....	7
2.2 Blackjack Game Rules .....	7
2.3 User Interface .....	9
2.4 Functions .....	11
CHAPTER THREE: METHODOLOGY	
3.1 Introduction .....	14
3.2 Reinforcement Learning Review .....	14
3.3 Q-learning Algorithm .....	16
CHAPTER FOUR: EXPERIMENTS	
4.1 Introduction .....	20
4.2 Game Configuration .....	21
4.3 Learning Process .....	23
4.4 Experiment Results .....	28
4.4.1 Ruled-based Greedy-Play .....	28
4.4.2 Learning from "Smart Player" .....	29

4.4.3 Learning from "Greedy Player" .....	33
4.4.4 Results Comparison .....	34
CHAPTER FIVE: CONCLUSION AND FUTURE DIRECTIONS	
5.1 Conclusion .....	36
5.2 Future Directions .....	37
REFERENCES .....	39

LIST OF TABLES

Table 1. Q-learning Algorithm .....18  
Table 2. Game Configuration Table ..... 22  
Table 3. Initial Data Table for  $S_0 = \{6, 19\}$  ..... 24  
Table 4. Data Table for  $S = \{6, 19\}$  in Round 1 ..... 25  
Table 5. Data Table for  $S = \{6, 19\}$  in Round 2 ..... 25  
Table 6. Initial Data Table for  $S_0 = \{6, 17\}$  in Round 3 ... 26  
Table 7. Data Table for  $S = \{6, 19\}$  in Round 3 ..... 27  
Table 8. Data Table for  $S = \{6, 17\}$  in Round 3 ..... 27  
Table 9. Result Table 1 ..... 32  
Table 10. Result Table 2 ..... 34



## LIST OF FIGURES

Figure 1. Illustration of Supervised Learning .....	4
Figure 2. Illustration of Reinforcement Learning .....	5
Figure 3. User Interface .....	10
Figure 4. User Interface in Different Mode .....	12
Figure 5. Sample of Rules.txt .....	31

## CHAPTER ONE

### INTRODUCTION

#### 1.1 Introduction

Reinforcement learning is one type of machine learning; it is concerned with learning through the use of penalties and rewards. Reinforcement learning is also considered as one of the suitable methods for game playing due to its capability to discover good strategies. In this project, this technique is applied in the design of game strategy.

This chapter presents the purpose and background of this project. The significance and contribution of the project is also discussed briefly.

#### 1.2 Problem

The main purpose of this project is to apply reinforcement learning to the design of game strategy. Although reinforcement learning is traditionally defined as a sequence of decisions and fit in game playing well, it is not well studied and applied in practice. In the gaming industry, the strategy used by computers to win a game is usually pre-programmed by game designers according to the game patterns or a set of rules. The strategies in computer games seldom have characteristics of learning, i.e., playing

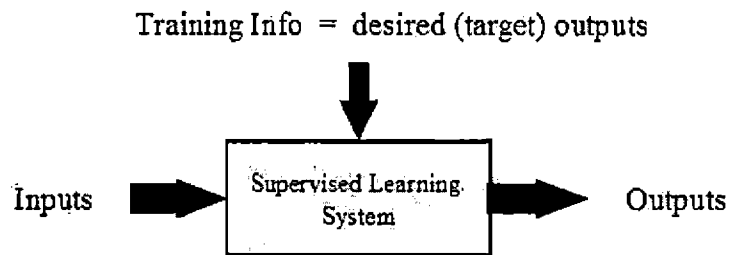
based on the behavior pattern of the opponent. This type of strategy usually makes players feel less challenged when playing against the computer opponents. Hence, to let games become more interesting, it is important to increase the variation and intelligence of the game. In my project, a computer software agent will be designed to learn the strategy of an opponent throughout the playing history, rather than respond to the opponent's action by following pre-programmed instructions or rules. By applying the reinforcement learning in game strategy design, the computer as a competitor will no longer be a pre-programmed robot. Through the learning process, the computer's actions will become more flexible and unpredictable.

### 1.3 Background

Machine learning studies the development of algorithms and techniques that enable a computer to "learn" through experience and improve performance over time. It is the core of artificial intelligence and also an important characteristic of computer intelligent behavior. Based on the developers' desired outcome of machine learning algorithm, machine learning is divided into different areas. For example, supervised learning generates a function that maps inputs to desired outputs (as explained in Figure 1.)

Unsupervised learning models a set of outputs which are unlabeled examples. In supervised learning, a set of inputs is assumed to be the cause of another set of outputs, while in unsupervised learning all inputs are assumed to be caused by a set of latent variables. Reinforcement learning is a sub-area of machine learning that is concerned with learning through the use of penalties and rewards (as explained in figure 2.) In other words, it is about how an agent ought to take actions in an environment based on current state and previous feedback so one can maximize the notion of long-term reward. The learning algorithm decides which action to take depending on finding the actions that yield the highest overall reward through trials and errors. The actions taken will affect the immediate and subsequent rewards. Hence, the reinforcement learning algorithm is a good method for approximating an optimal game strategy because it allows learning to take place during each step of play.

# Supervised Learning

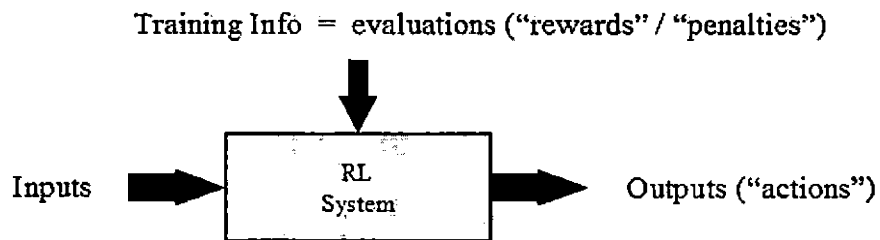


$$\text{Error} = (\text{target output} - \text{actual output})$$

Objective: Minimize error

Figure 1. Illustration of Supervised Learning

## Reinforcement Learning (RL)



Objective: Maximize reward or minimize penalty

Figure 2. Illustration of Reinforcement Learning

### 1.4 Contribution

Recent games industry puts emphasis on computer graphics and the game's fluency rather than the durability of a game's life. The advertisement of a new game usually uses a gorgeous 3D video to attract customers. However, customers easily lose their interest when they feel a game has no challenge even though its graphics is perfect. Especially the computers with pre-programmed strategies are easier to let players feel bored when players can predict

what the computer is going to do or respond on the next move. Hence, in order to extend a game's life, it is important to let the game become more competitive and flexible.

The contribution of the project is to combine game strategy design with reinforcement learning. The computer learns the player's strategy when the player is playing and allows learning to take place during each step of the play. The computer then has the capability to create an optimal game strategy through the playing history of the specific player. The computer game with learning capability will thus generate different strategies against different players. Therefore, it will increase variation and challenge of game play.

## CHAPTER TWO

### PROBLEM

#### 2.1 Introduction

Blackjack has all the basic elements of a game: players, actions, and payoff. It is selected as the study model because of its simplicity and popularity. In Blackjack, there is no absolute winning strategy because of the variation of the states and the randomness in a shuffled deck. However, players playing with certain strategies may have higher chances to win. It is desired that an intelligent software agent is designed to learn from smart and experienced human players so that the software agent can play against the dealer and defeat the dealer with a higher probability.

Chapter 2 introduces the basic Blackjack game rules. In the project, the Blackjack rules are the same as those in a casino. In this chapter, the detailed view of the user interface and functions of the program is present.

#### 2.2 Blackjack Game Rules

Blackjack is a widely played card game that can be found in casinos. The rule to win Blackjack is to obtain a total point of cards higher than the dealer's points but not



exceeding 21. The game works by assigning each card a point value. Cards from 2 to 10 are worth their face value, while Jacks, Queens, and Kings are worth 10 points. An ace is worth either 1 or 11 points, whichever is decided by players. According to casino's rules, each player is first given two cards and face up and the dealer also has two cards only one is faced up. After receiving the two cards, the player can choose his or her own actions. "Stay" is to stay with the current cards and take no card. "Hit" is to add a card to the hand to make the total card value to be closer to 21. A player may hit as many times as he wishes as long as his card value is not over 21. "Double Down" is when player holding two cards, the player can double his bet by hitting with only one more card and stay after that, and "Split" is having the pair of cards with the same values, the player can split his hand into two hands, Hence, based on player's card value and dealer's face up card's number, players can use their own strategy to decide hit or stay. However, for simplicity of the project, "Double down" and "Split" will not be considered in this project.

The dealer's actions are fixed. Based on the casino's blackjack rules, the dealer hits when his or her total points is less than 17 and stays when it is greater than 17. When the player has 21 points in the first two cards, that

means Blackjack, and the player automatically wins if the dealer does not have 21 points. However, if the dealer has Blackjack, this round is over and the dealer wins. Moreover, if the card value of the dealer or player is over twenty one, this round is also over, and this situation is called "Bust". Lastly, after the dealer finishes his action, and both dealer and player get the same card value, it means this round is a tie.

### 2.3 User Interface

The user interface is programmed in C++, and the graphic part is done using Graphic Device Interface (GDI) which is a class-based Application Programming Interface (API) for C++ programmer. The configuration of Blackjack user interface consists of dealer's stack of cards, player's stack of cards, mode-selection buttons, action buttons, and status (the card values, game result, and agent's action). In the user interface, the cards on the top are dealer's cards, and the cards on the bottom are owned by players. The numbers on the top of each set of cards are the total cards points, and game result shows in the middle of table when current round is finished. The buttons on the top left are mode-selection buttons. Users can choose the different modes

to operate the program. There are four modes, which are "Manual", "Auto", "Learner" and "Greedy". Each function will be introduced in Section 2.4. The buttons on the right are action buttons. Users can play Blackjack by clicking "Hit" and "Stay" buttons or understand how the agent chooses its actions by clicking the "Next" button. The user interface is shown in Figure 3.

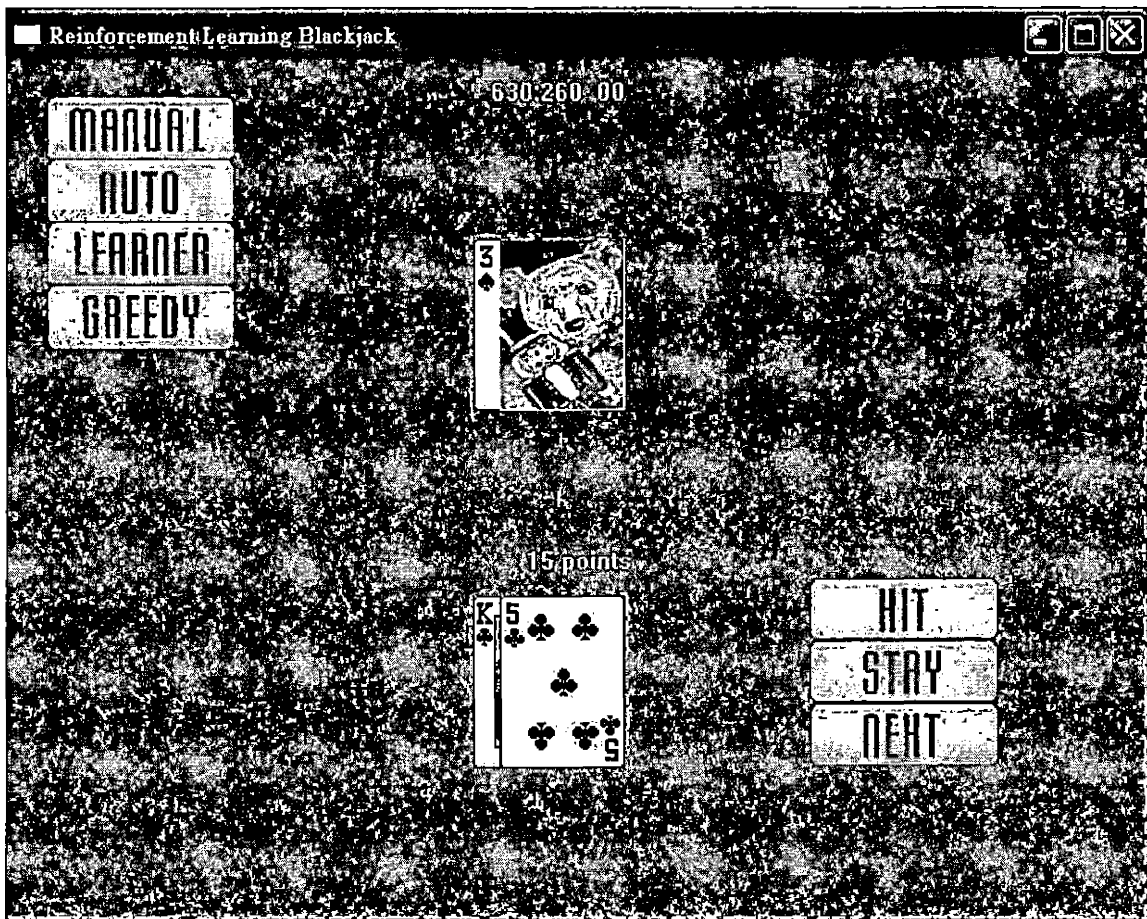


Figure 3. User Interface

## 2.4 Functions

A main objective of the project is to explore the feasibility and the advantages of learning in game strategy design. Therefore, in order to reveal the performance of reinforcement learning, we set up two pre-programmed game strategies which are named "Smart Player" and "Greedy Player". Interpreting "Smart Player" and "Greedy Player" in game play, main function is to let users play Blackjack and exhibit how different agents choose their actions. As mentioned before, there are four mode-selection buttons in the left of the user interface. The function of "Manual" button is to let human player play Blackjack manually. In this mode, players can choose their action "Hit" and "Stay" against the dealer, and learning agent learns the strategy of the player at the same time. The "Auto" button is to allow users to observe how the agent uses the pre-programmed strategy which is named "Smart player" to play Blackjack. In this mode, users can only click the "Next" button to see what is the next action that the agent will choose. When clicking the "Next" button, agent's action and description will be shown on the screen (see Figure 4).

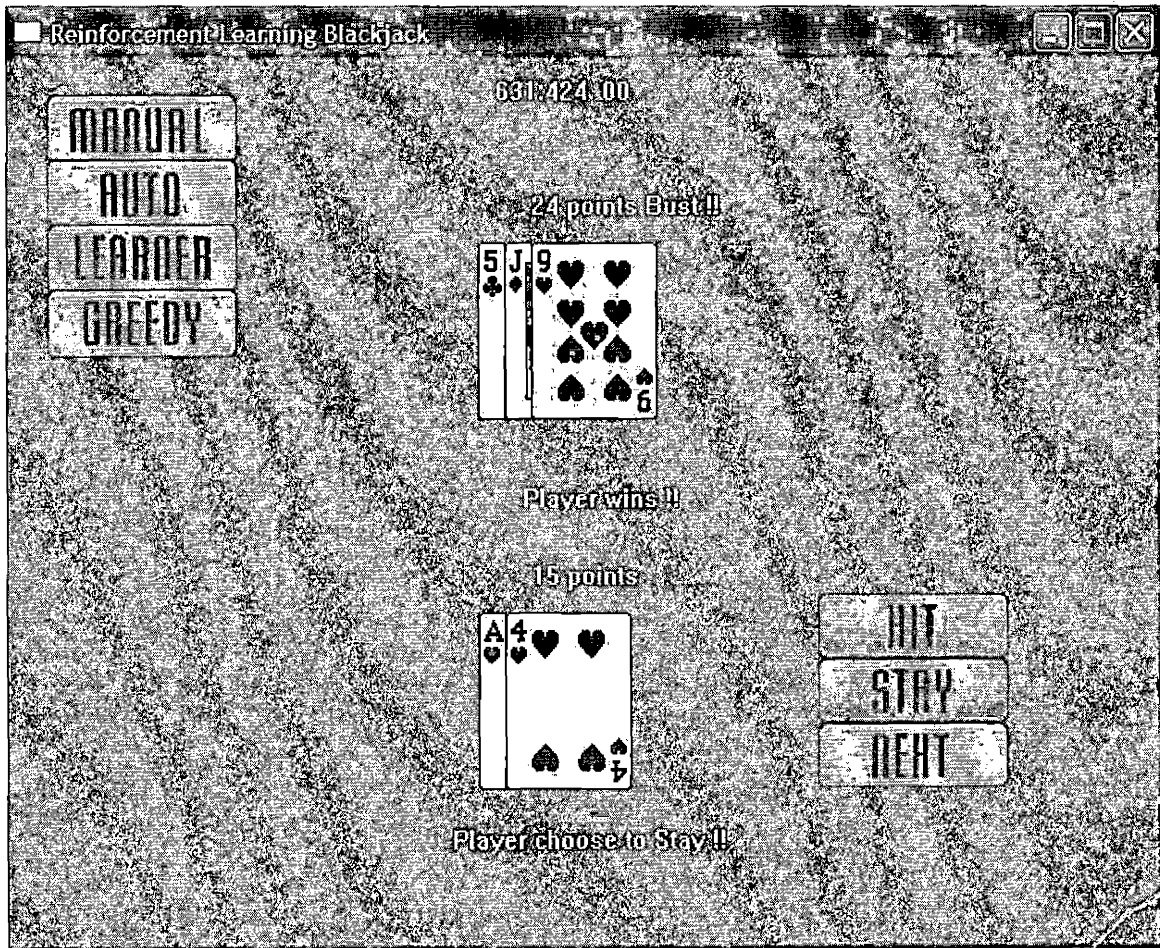


Figure 4. User Interface in Different Mode

The function of "Learner" button and "Greedy" button are similar with "Auto" Button. The difference between those three buttons is that different buttons present different agents. As implied by the name, the "Learner" button is to allow the user to observe how the agent uses the reinforcement learning algorithm to play Blackjack. From the learning process, the learning agent creates its own playing

strategy and shows its capability of decision making. However, the "Greedy" button allows the agent to use a different pre-programmed strategy, which is named "Greedy", to play Blackjack, and users can observe how the agent applies the Greedy strategy to play Blackjack.

## CHAPTER THREE

### METHODOLOGY

#### 3.1 Introduction

The most important element of the project is to explore learning in game play. Through learning, the computer observes human behavior, analyzes the goodness of actions in a state and thus plays against opponents smartly. This chapter reviews reinforcement learning and illustrates how reinforcement learning in game strategy design is implemented. Specifically, Q-learning, one of the reinforcement learning algorithms, is discussed as well.

#### 3.2 Reinforcement Learning Review

Reinforcement learning is a type of machine learning technique. This technique was discovered in late 1980s and has been studied until now. The main idea of reinforcement learning is to allow learning to take place while interacting with environments; its agent learns from the consequences of its action, rather than from being taught. The agent selects its actions through its experience and also by making new choices, which is practically trial and error learning. Moreover, the reinforcement learning agent receives a numerical reward when it makes a right choice,

and the agent chooses actions that "maximize the accumulated reward over time.

The widely used or standard framework of reinforcement learning is Markov Decision Process (MDP)[7]. It provides a mathematical framework for modeling decision-making in situation where outcomes are partially random and partially under the control of the decision maker. MDP is represented with a tuple  $(S, A, R, P(S))$ , where  $S$  is the set of states,  $A$  is the set of agent's actions, and  $R$  is a reward function  $R : S \times A \rightarrow R$  mapping state-action pair  $(s, a)$  of the environment to a reward. A reward function determines what is good in an immediate sense.  $P(s)$  is the set of discrete probability distributions over the states, i.e., the state transition probability.

The objective of reinforcement learning is to find a policy  $\pi$  that maximizes the expected sum of discounted rewards over time. A policy  $\pi$  which is defined as  $\pi:S \rightarrow A$ . To achieve the objective, we need to find out the optimal policy  $\pi$ . The policy  $\pi$  specifies the learning agent's behavior at given states in an environment. In other words, a policy is a mapping from observed states to actions which should be taken in those states. Therefore, a policy may be a simple function, a lookup table, and a search process involved extensive computation.



### 3.3 Q-learning Algorithm

Q-learning is a simple incremental algorithm developed from the theory of dynamic programming for reinforcement learning. To introduce the long term rewards of action a taking in state  $s$  in Q-learning, we define a value function  $V(s)$  as:

$$V^\pi(s) = R(\pi(s)) + \gamma \sum_{s'} (p_{s \rightarrow s'}[\pi(s)] V^\pi(s'))$$

The learning agent expects to receive  $R(\pi(s))$  immediately for performing the action  $a$  in state  $s$ , and then moves to a state that is 'worth'  $V^\pi(s')$  with probability  $p_{s \rightarrow s'}[\pi(s)]$  [4]. Therefore, the value function  $V(s)$  with policy  $\pi$  can be explained as the sum of immediate rewarded received plus the reward that will received at new state  $s'$  following the same strategy thereafter. In Q-learning, policies and the value function are represented by a two-dimensional lookup table indexed by state-action pairs. For a policy  $\pi$ , define Q values as:

$$Q^\pi(s, a) = R_s(a) + \gamma \sum_{s'} (p_{s \rightarrow s'}[\pi(s)] V^\pi(s'))$$

In other words the Q value is the expected discounted reward for choosing action  $a$  at state  $s$  and following policy  $\pi$  thereafter. The objective of Q-learning is to maintain an estimate of the  $Q^*$  which is  $Q^\pi$  for an optimal policy  $\pi$ . If

$a^*$  is an action at which the maximum is attained, and then an optimal policy can be formed as  $\pi^*(s)=a^*$ , it is straightforward to show that  $V^*(s)=\max_a Q^*(s,a)$  [4]. In Q-learning, the agent's experience consists of a sequence of distinct stages or episodes. In the  $n^{\text{th}}$  episode, the learning agent observes its current state  $s_n$  and then selects an action  $a_n$  [4]. Then it observes the subsequent state  $s'_n$  and receives an immediate payoff  $r_n$ , and adjust its  $Q_{n-1}$  values using a learning rate  $\alpha_n$ , according to:

$$Q_n(s,a) = \begin{cases} (1-\alpha_n)Q_{n-1}(s,a) + \alpha_n[r_n + \gamma V_{n-1}(s'_n)] & \text{if } s = s_n \text{ and } a = a_n, \\ Q_{n-1}(s,a) & \text{otherwise,} \end{cases}$$

Where

$$V_{n-1}(s') = \max_{a'} \{Q_{n-1}(s',a')\}$$

Each episode is equivalent to one training session; the agent explores the environment and gets the reward until it reaches the goal state.  $Q^\pi(s,a)$  allows the agent to compute the expected reward of being in state  $s$ , taking action  $a$ , and thereafter following policy  $\pi$ . Let the state at time  $t$  be  $r_t$ , and assume that the learning agent then chooses action  $a_t$ . The immediate result is that a reward  $r_t$  is received by the learner. The parameter  $\alpha$  is referred to as

the learning rate that determines the size of the update made on each time-step.  $\gamma$  is referred to as the discount rate, which determines the value of future rewards.  $0 \leq \gamma < 1$  controls the affection of future rewards on the optimal decisions. If  $\gamma$  is closer to zero, the agent will tend to consider only immediate reward. On the other hand, the closer  $\gamma$  is to one the greater the weight of future reinforcements. The Q-learning algorithm is outlined below [8]:

Table 1. Q-learning Algorithm

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
Choose $a$ from $s$ using policy derived from $Q$
Take action $a$ , observe $r$ , and $s'$
$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
$s \leftarrow s'$
until $s$ is terminal

The advantage of Q-learning is that one does not need a model of the environment. In Q-learning, the optimal policy can be learned by interacting with the environment, and no

knowledge of the true transition probabilities or the reward function is necessary. The update rule is policy free as it is a rule that just relates Q values to other Q values. Q-learning can calculate the Q values directly from the elementary rewards observed.

## CHAPTER FOUR

### EXPERIMENTS

#### 4.1 Introduction

In this project, the main idea is to generate optimal game strategy by using the reinforcement learning algorithm. Through the process of learning, the playing strategy used by the learning agent will improve over time of playing. In other words, after the human player play Blackjack many rounds with a dealer, the computer will learn his or her strategy and apply the strategy which is similar but better than the human player's to play Blackjack. Therefore, if the player is a experienced smart player, the computer that learns from the smart player will use an "optimal" Blackjack strategy against dealer.

Theoretically, the learning agent will reveal a better performance than the pre-programmed strategy. To evaluate the performance of the reinforcement learning, the software agent with learning ability will be compared with the software agent that uses pre-programmed strategy. And the comparison results are given between the learning agent and the pre-programmed agent and reveals that the learning agent has a better performance. Chapter four introduces how to implement game strategy using reinforcement learning step by

step and what is pre-programmed game strategy for agent to learn and all results of the experiments will be provided and discussed.

#### 4.2 Game Configuration

The configuration of Blackjack game consists of main game program and reinforcement learning agent program which are developed by C++ language. The function of the main game program is to let players play Blackjack manually. During the time when human player is having fun with Blackjack, the learning agent learns human player's playing strategy. Moreover, when selecting the mode on main game program, program presents different action choice by different agent in order to let users realize how the agent learns. Reinforcement learning agent program uses Q-learning algorithm. The Q-learning algorithm is a good method for approximating an optimal Blackjack strategy, because it allows learning to take place during playing. Therefore, Blackjack can be easily formulated as an episodic task. In this project, the state representation of Q-learning algorithm consists the player's current total points which are between 2 to 21, and dealer's face up card which is

between 1 to 10. The action sets are 1 and 0 which means "Hit" and "Stay". In Q-learning formula:

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

$\alpha = 0.01$ ,  $\gamma = 0.9$ , and immediate reward  $r$  is 1 when this round the player wins, -1 when player loses, and 0 when player choose "Hit" but not bust, and game is still in process (shows in Table 2).

Table 2. Game Configuration Table

$S: \{S_1, S_2\}$ , where $S_1$ : Player's current total cards number: 2, 3, ..., 21 and $S_2$ : Dealer's face up card number: 1, 2, ..., 10
$A = \begin{cases} 1, & \text{Player chooses "Hit"}. \\ 0, & \text{Player chooses "Stay"}. \end{cases}$
$R = \begin{cases} 1, & \text{when player wins.} \\ -1, & \text{when player loses.} \\ 0, & \text{Game continues.} \end{cases}$
$P(s) = 1$ for all $s \in S$

For example, in the situation when dealer's face up card is 9 and player's total card number is 15, player chooses to "Hit", then player gets the "King" which means 10 points. So player's total card number is 25 which means "Bust". On the state of "dealer's card = 9 Player's card = 15", player's action is 1 and gets penalty which is -1 due to the bad choice. However, in the same situation, player chooses "Hit" and gets 2 points. The game is still in the process. On the state of "dealer's card = 9 Player's card = 15", player's action is 1 and gets reward 0.

#### 4.3 Learning Process

In the prior chapter, the concepts of reinforcement learning and Q-learning algorithm have been discussed. The detailed processes of how the computer uses reinforcement learning to learn and take actions will be presented as examples in the below:

The learning agent begins with no prior knowledge, i.e., every action value  $Q$  for each state is zero, and uses the Q-learning algorithm.

Round 1: Assuming that player gets two cards, one is K and another is 9. Dealer has a face-up card which is 6 and a face-down card in hand. The total points of player's cards



are 19. Because learning starts with no prior knowledge beyond the rules of the game, every action Q for each state is initially zero. Initially, the data table is:

Table 3. Initial Data Table for  $S_0 = \{6, 19\}$

Dealer's card=6, Player's card= 19, Action=0	Q-Value=0
Dealer's card=6, Player's card= 19, Action=1	Q-Value=0

The number behind the Action = 0 or 1 is the action value. It determines that agent will choose "Hit" or "Stay" when the same state takes place again. Now if the player chooses "Hit" in this state and loses in this round, the Q-Value of this state will be updated in -0.01.

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

$$Q(s, a) \leftarrow 0 + 0.01[-1 + 0.9*0 - 0] = -0.01$$

The data table will be updated:

Table 4. Data Table for  $S = \{6, 19\}$  in Round 1

Dealer's card=6, Player's card= 19, Action=0	Q-Value=0
Dealer's card=6, Player's card= 19, Action=1	Q-Value =-0.01

Round 2: player gets the same two cards as last round, one is K and another is 9. Dealer has a face-up card which is 6 and a face-down card in hand. According to the data table, the Q-Value of action "Stay" is bigger than the Q-Value of action "Hit". Hence, player chooses "Stay", assume that player wins this round. The Q-Value of this state will be updated in 0.01.

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

$$Q(s, a) \leftarrow 0 + 0.01[1 + 0.9*0 - 0] = 0.01$$

The data table will be changed to:

Table 5. Data Table for  $S = \{6, 19\}$  in Round 2

Dealer's card=6, Player's card= 19, Action=0	Q-Value =0.01
Dealer's card=6, Player's card= 19, Action=1	Q-Value =-0.01

Round 3: Player gets the different two cards with last round, one is 8 and another is 9. Dealer has a face-up card which is 6 and a face-down card in hand. The total points of player's cards are 17. Because this state is not experienced by the learning agent, the initial data table is:

Table 6. Initial Data Table for  $S_0 = \{6, 17\}$  in Round 3

Dealer's card=6, Player's card= 17, Action=0	Q-Value=0
Dealer's card=6, Player's card= 17, Action=1	Q-Value=0

In this round, player choose to "Hit" and then gets a 2 points card. The total points of player's cards are 19, and this round is still in process. The Q-Value will be updated with 0 reward as well.

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

$$Q(s, a) \leftarrow 0 + 0.01[0 + 0.9 * \max_{a'} Q(s', a') - 0]$$

$\max_{a'} Q(s', a')$  means the maximum Q-Value in the state of "Dealer's card =6 Player's card =19." which are:

Table 7. Data Table for S = {6, 19} in Round 3

Dealer's card=6, Player's card= 19, Action=0	Q-Value =0.01
Dealer's card=6, Player's card= 19, Action=1	Q-Value =-0.01

Hence, the  $\max_a Q(s', a')$  will be 0.01 and the Q-Value of this round will be updated by 0.00009.

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

$$Q(s, a) \leftarrow 0 + 0.01[0 + 0.9 * 0.01 - 0] = 0.00009$$

The data table will be changed to:

Table 8. Data Table for S = {6, 17} in Round 3

Dealer's card=6, Player's card= 17, Action=0	Q-Value=0
Dealer's card=6, Player's card= 17, Action=1	Q-Value =0.00009

The game continues until it is over. At each step of the game, the Q-Value corresponding to every s and a is updated as well.

Based on these operating rules, every state counts as long as the computer experience it, the action value of

these states will be generated. Moreover, the action value decides how much benefit will player get when choose this action if next time in the same state. Hence, when the computer agent chooses the action, it will choose an action with bigger action value in this state.

#### 4.4 Experiment Results

In order to let the computer completely learns human player's strategy, human player needs to play the game a great deal of rounds. So the computer can collect enough data for learning information. However, it is unreasonable for a human player to play thousand of rounds. Hence, the project allows the computer playing blackjack automatically based on pre-programmed strategy for many rounds. It records all data and generates rules for learning agent to use. In addition, this experiment compares the pre-programmed strategies to the reinforcement learning agent. Different pre-programmed game strategies will be introduced in the below:

##### 4.4.1 Ruled-based Greedy-Play

In chapter three, it mentioned that a comparison strategy studied as well as a pre-set strategy to compare with the performance of reinforcement learning strategy.

This pre-set strategy is named "Greedy" in this project, and its principle is to choose the action based on the short-term benefit. The rule of greedy-play algorithm is that player always hit until the total points of player's card are bigger than or equal to 17. The reason for the comparison group to use this game strategy is because dealer uses the same strategy. Since dealer always chooses to hit when his or her cards' points are less than 17, and stay when the cards' points are bigger than or equal to 17, so as long as dealer does not get a bust and player's card points are not more than 17, the player will lose in this round. For this greedy reason, this project uses this pre-set strategy "Greedy" to compare with a software agent which has learning ability. Experiment allows "Greedy" player to play Blackjack automatically and calculate the winning percentage to compare with the learning agent.

#### 4.4.2 Learning from "Smart Player"

As mentioned before, in order to avoid too little data collect from human players playing manually, this project sets a simple strategy for the reinforcement learning agent to learn automatically. The game strategy is called "Smart player" that has a basic concept as shown below:

Depending on dealer's face-up card number and player's total cards' points, player makes different decisions.

When player gets 17 - 20 in hand, player choose "Stay"

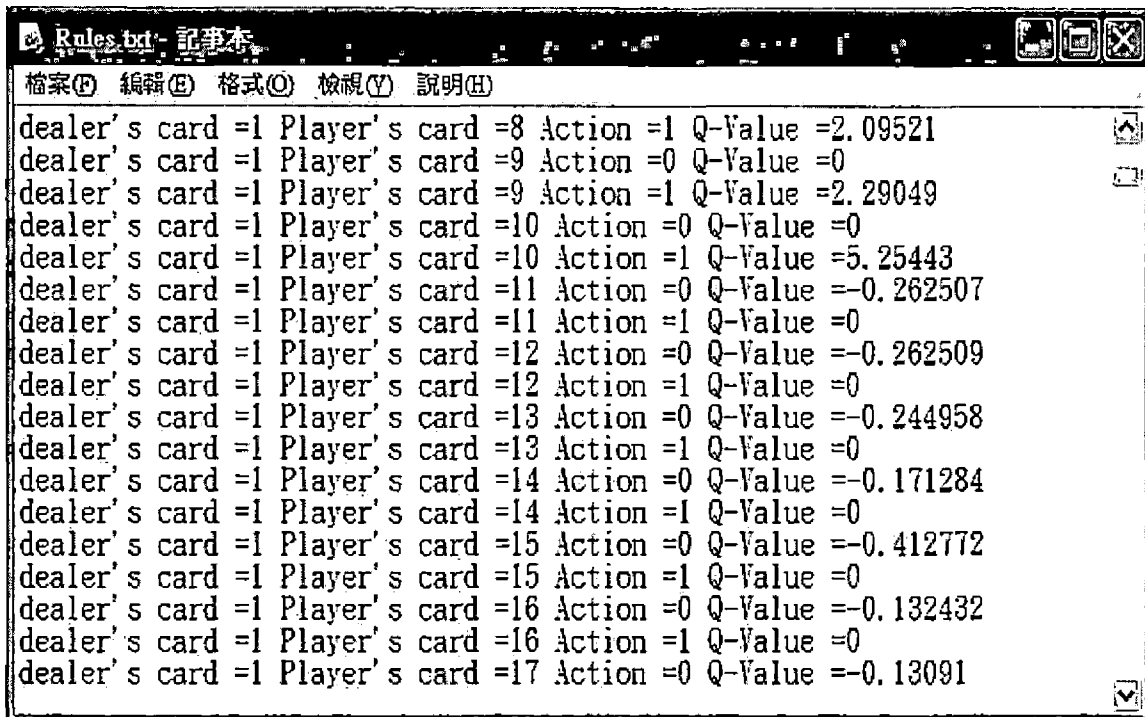
When player gets 12 - 16 in hand and dealer's face-up card number is 2 - 6, player choose "Stay"

When player gets 12 - 16 in hand and dealer's face-up card number is 7 - 10, J, Q, K, Ace, player choose "Hit"

When player gets 2 - 11 in hand, player choose "Hit"

Considering the different situation that might lead dealer or player to get a bust, these rules are basic optimal game strategies for any Blackjack situation in the long run.

After the computer uses "Smart Player" strategy to play Blackjack many rounds, the computer will output a .txt file for agent to read and use. This learning data is named "Rules.txt" (shows in Figure 5.) In this procedure, it can be called rule generation which means learning agent learns from "Smart Player". Hence, after generating these rules, the computer can use the "Smart player" strategy to play blackjack.



```
Rules.txt - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)
dealer's card =1 Player's card =8 Action =1 Q-Value =2.09521
dealer's card =1 Player's card =9 Action =0 Q-Value =0
dealer's card =1 Player's card =9 Action =1 Q-Value =2.29049
dealer's card =1 Player's card =10 Action =0 Q-Value =0
dealer's card =1 Player's card =10 Action =1 Q-Value =5.25443
dealer's card =1 Player's card =11 Action =0 Q-Value =-0.262507
dealer's card =1 Player's card =11 Action =1 Q-Value =0
dealer's card =1 Player's card =12 Action =0 Q-Value =-0.262509
dealer's card =1 Player's card =12 Action =1 Q-Value =0
dealer's card =1 Player's card =13 Action =0 Q-Value =-0.244958
dealer's card =1 Player's card =13 Action =1 Q-Value =0
dealer's card =1 Player's card =14 Action =0 Q-Value =-0.171284
dealer's card =1 Player's card =14 Action =1 Q-Value =0
dealer's card =1 Player's card =15 Action =0 Q-Value =-0.412772
dealer's card =1 Player's card =15 Action =1 Q-Value =0
dealer's card =1 Player's card =16 Action =0 Q-Value =-0.132432
dealer's card =1 Player's card =16 Action =1 Q-Value =0
dealer's card =1 Player's card =17 Action =0 Q-Value =-0.13091
```

Figure 5. Sample of Rules.txt

In order to prove that the strategy implemented by reinforcement learning reveal a better performance than pre-programmed strategy to play Blackjack, my experimental approach is to compare the winning percentage of three different strategies, Greedy-play strategy, Smart Player, and reinforcement learning agent, when playing Blackjack. The results of learning agent learning from "Smart Player" strategy will prove that the learning agent through the learning process gives a better performance than the pre-set strategy "Greedy-play" in Blackjack game. To calculate the



winning percentage of "Smart player", "Learning agent" and "Greedy-play" playing Blackjack, the approach is to run 10 times with different playing strategies and each time playing Blackjack 1000 rounds. The results of the experiment are as shown in Table 9.:

Table 9. Result Table 1

	Smart Player	Learner	Greedy
1	42.4%	39.1%	40.0%
2	39.5%	38.1%	36.3%
3	43.2%	39.9%	36.5%
4	40.1%	40.3%	39.7%
5	39.0%	37.5%	40.2%
6	43.1%	41.3%	38.2%
7	41.7%	42.8%	35.0%
8	41.5%	42.0%	41.2%
9	39.1%	38.8%	40.1%
10	42.8%	41.2%	37.8%

Average	41.24%	40.1%	38.56%
Standard Deviation	0.0159	0.0165	0.0195

#### 4.4.3 Learning from "Greedy Player"

In order to prove the strategy through the learning process will have a better performance than the original strategy, the experiment sets another programmed strategy "Greedy Player" to be another model for the reinforcement learning agent to learn. Based on the same learning elements and situation, learning agent learns from greedy-play strategy. The rule for greedy-play strategy is that player chooses stay whenever the total value of player's card is bigger than or equal to 17. Under this circumstance, the computer plays Blackjack with greedy-play strategy for many rounds and then output a .txt file with greedy-play rules. Moreover, learning agent learns this strategy and uses it to against dealer. Since reinforcement learning agent learns different strategies, the agent might choose different actions with the same state. The experimental approach is to allow learning agent which learns "Greed-play" strategy to run 10 times and each time playing Blackjack 1000 rounds. The results of the experiment are as shown in Table 10.:

Table 10. Result Table 2

	Learner	Greedy
1	38.3%	35.8%
2	39.1%	39.2%
3	41.2%	38.3%
4	40.7%	40.1%
5	41.1%	37.2%
6	39.6%	41.1%
7	39.1%	39.3%
8	40.2%	38.2%
9	40.1%	40.0%
10	37.0%	39.9%
Average	39.64%	38.91%
Standard Deviation	0.0125	0.0149

#### 4.4.4 Results Comparison

According to the results of the experiment (Table 9.), the average of winning percentage using reinforcement learning is bigger than the pre-programmed strategies

"Greedy-play." It shows that the agent is learning useful information during training process and using a better strategy against dealer. This result presents that the strategy implemented by reinforcement learning reveal a better performance than pre-programmed strategy to play Blackjack. However, the winning percentage of "Smart player" is the biggest. It is because the "Smart player" strategy is based on basic optimal game strategies for any Blackjack situation in the long run. Even learning agent learns from "Smart player" strategy, its average of winning percentage can just approach the winning percentage of "Smart player," but hard to exceed it.

In another experiment with the same elements and conditions, the computer learns from the programmed strategy "Greedy-play" instead of "Smart Player" strategy. According to the results(table 10.), the agent learning form "Greedy-play," its average of winning percentage is bigger than the original "Greedy-play" strategy. This result presents that the strategy through the learning process will present a better performance than the original strategy in Blackjack game, under the circumstance that the original strategy is not optimal.

## CHAPTER FIVE

### CONCLUSION AND FUTURE DIRECTIONS

#### 5.1 Conclusion

This project explored the methods of reinforcement learning as a mean of determining an optimal game strategy of Blackjack. From the above experiment, it can be concluded that the reinforcement learning strategy does have better performance in playing Blackjack than pre-programmed strategy. Reinforcement learning agent is able to select the best actions in each state. Although some selections might be wrong due to bad experience, agent gets more learning experience through learning process, and when learning is finished, the game strategy will become an "optimal" strategy.

In the pre-programmed game strategy, the style of playing in a game has always been pre-set, which would make the game lack for excitement and variation. In other words, if the reinforcement learning is implemented in game strategy design of computers, players would not be able to guess computer's next move, which will make the entertainment of the game improve.

## 5.2 Future Directions

The future direction of the project is to apply reinforcement learning to other types of games, such as Real-Time Strategy games, Role-Playing games, Virtual Life Games, Sports games, Shooting games and Massively Multiplayer game etc. Nevertheless, to enable the computer learns human player's game strategy and uses it against players in a complex game is very challenging. The limitation of reinforcement learning into a complex game design is the scalability. When state space and action space are huge, it takes longer to learn. The learning can be conducted off-line, so the computation time is a real problem. The problem is the opportunity to model the large space of actions and states. The reinforcement learning's convergence to optimal strategy is under the assumption that every state and action has been visited infinitely often. However, this assumption is not tenable in a large scale game. One possible resolution would be the integration of reinforcement learning with prior-knowledge of the game or basic rules of the game. Even though with the limitation that it is hard to converge to optimal strategy in a large scale game in terms of the time and exploration of the game, reinforcement learning still provides a promising and

satisfactory solution since it involves learning and updates the strategy from the history of play.

## REFERENCES

- [1] Blackjack, <http://en.wikipedia.org/wiki/Blackjack>
- [2] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: an Introduction, MIT Press, Cambridge, MA, 1998.
- [3] Junling H. and Michael P. Wellman, Nash Q-Learning for General-Sum Stochastic Games, Journal Of Machine Learning Research 4 1039-1069, 2003.
- [4] Watkins, C. and Dayan, P. Technical note: Q-learning. Machine Learning, 8(3/4):279-292, May 1992.
- [5] Kaelbling, L., Littman, M. L. and Moore, A. W., Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research, 4:237-285, 1996.
- [6] Shoham, Y. and Powers, R., and Grenager, T., Multi-Agent Reinforcement Learning: a Critical Survey. Computer Science Department of University Stanford, 2003.
- [7] Michael L. Littman, Markov Games as a Framework for Multi-agent Reinforcement Learning. Eleventh International Conference on Machine Learning, 1994.
- [8] Charles D. Granville, Applying Reinforcement Learning to Blackjack Using Q-Learning. University of Oklahoma, 2004