第二十二屆旺宏科學獎

成果報告書

參賽編號：SA22-239

姓名：周宗穎

作品名稱：實現強化學習於多人吹牛骰子遊戲之探討

參賽類別：資訊

關鍵字：不完全資訊賽局、經驗法則、強化學習

摘要

本研究使用 Python 程式語言實作吹牛骰子的對局程式，包含亂數選擇、經驗法則與強化學習，透過三種對局程式互相對戰，探討不同情況下的勝率分布，以及強化學習的效果與特徵。實作後我們發現，吹牛骰子遊戲沒有固定的遊玩位置優勢，但前一手玩家的遊玩特性會對勝率有很大的影響；強化學習在訓練的過程中會學習與模仿當前對手的策略，且會發生過擬合的現象；記錄完整手牌對勝率有提升，但會增加訓練所需次數與空間；強化學習中高分的策略多與機率分析和優勢手牌有關。

吹牛骰子是一款常見的賽局遊戲，也有很多對該遊戲的 AI 研究，但有關強化學習的內容較少，且多針對兩人賽局。本研究較大的創新有幾點，包含吹牛骰子本身遊戲特性、強化學習的各項實現特性、將研究拓展至三人賽局以及使用數學方式分析各項性質和理論。我們也期望將來能延伸探討多人賽局下的規律與特性，並透過對不完全資訊賽局的發現，推廣到其他應用層面。

壹、前言

一、研究動機

2016 年，Google DeepMind 研發的 AlphaGo 擊敗了韓國棋士李世乭，此後亦有越來越多圍棋 AI 出現，如 AlphaGo Zero 和 AlphaZero。號稱「人類對抗 AI 最後一道防線」的圍棋逐漸失守，掀起了一波 AI 熱潮。在圍棋之前，和圍棋並稱「世界三大棋類」的西洋棋與象棋也早已被 AI 攻破。隨著時間的推進，AI 逐漸精通了完全資訊賽局，在該領域擊敗人類。而在不完全資訊賽局方面，也有德州撲克軟體 Libratus 和麻將軟體 Suphx 的出現，且都有不俗的表現。「吹牛骰子」是一個經典的不完全資訊賽局，之前我看了某一件作品，並查找相關的文獻，感到很有興趣，便決定延伸該題目。曾經研究過一款名為 2048 的小遊戲，為一需考慮機率的完全資訊賽局，有許多 AI 的實現，其中便包括強化學習。強化學習在該遊戲上取得了不錯的成績，那麼強化學習是否也能運用在不完全資訊賽局上？於是展開了此研究。

二、研究目的

（一）實作兩人和三人賽局的強化學習，使之學習吹牛骰子的對局策略，並與經驗法則互相對戰，分析其對戰過程與數據，討論強化學習於吹牛骰子實現之成效

（二）藉由強化學習與經驗法則的實現，討論吹牛骰子遊戲的各項性質

（三）透過對強化學習和手牌組成的討論，尋找強化學習的性質與特徵，或是和人類策略的相似之處

（四）期望透過對不完全資訊賽局的研究，推廣至其他應用

三、研究架構圖

本研究的研究架構圖如下：

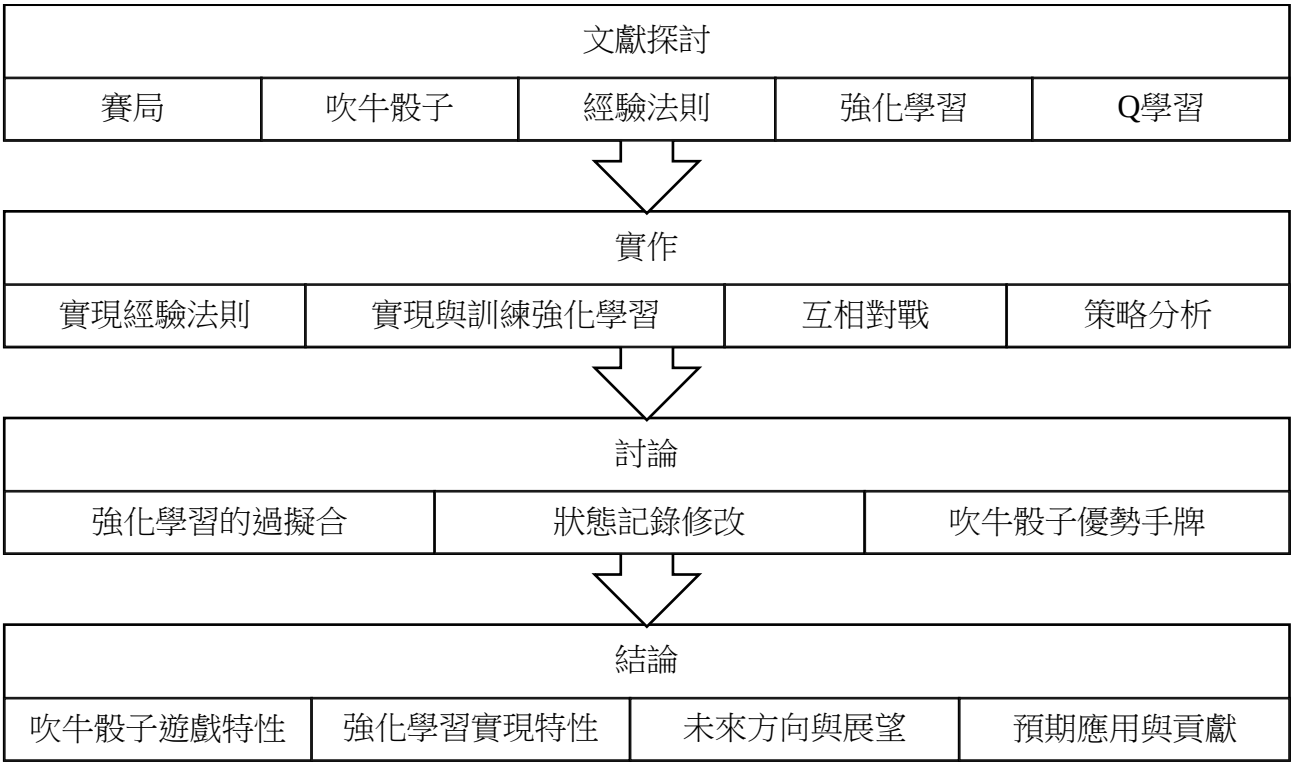

圖 1-1、研究架構圖

貳、研究設備及器材

一、電腦規格

（一）型號：Lenovo Legion 5 15IMH05

（二）處理器：Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz

（三）記憶體：8.00 GB

（四）系統：Windows 10 家用版，64 位元作業系統

二、Python 程式語言，使用平台：Google Colab（全名 Colaboratory）

三、Microsoft Excel：處理大量實驗數據和資料、繪製圖表

參、文獻探討

一、賽局

賽局最通用的定義為兩個或多個玩家，在理性的前提下，為追求己身目標而造成行為相互衝突而處於的一種對抗狀態。賽局有三大要素：參賽者、行動與報酬，三大要素皆清楚者為完全資訊賽局，任一不清楚則為不完全資訊賽局。

賽局的架構可依「有機率/無機率」、「完全資訊/不完全資訊」分為四種，表一為幾種常見的賽局遊戲之分類。

	完全資訊賽局	不完全資訊賽局
不需考慮機率	象棋、西洋棋、圍棋	走私者、戰艦遊戲
需考慮機率	西洋雙陸棋、大富翁	吹牛骰子、撲克

表 3-1、常見賽局遊戲的分類

本次主要的研究對象為吹牛骰子，即一需考慮機率的不完全資訊賽局。

二、吹牛骰子

（一）吹牛骰子簡介

「吹牛骰子」，又稱「大話骰」，是一種骰子遊戲，可兩人或多人遊玩。吹牛骰子在世界各地經過各種演變，演變出許多不同的名稱、規則。目前主要的玩法有多人共用一副骰子的 individual hand，和每人各持有一副骰子的 common hand，而 common hand 較為常見。

（二）吹牛骰子的玩法

以下以三人遊玩的 common hand 為例。

假設遊戲中依序有 A、B、C 三位玩家。遊戲開始時，每人擁有五顆骰子放在骰盅(或任何可藏起骰子的容器)內，而遊戲在其中一方抓牌時結束。

1.叫牌

在每回合中，第一位輪到的玩家 A 必須執行「叫牌」動作，也就是喊出「n 個 m」，代表 A 認為這場地上點數為 m 的骰子個數至少有 n 顆。例如 A 喊出「3 個 2」，代表 A 認為場上所有玩家的骰子中，2 點的骰子至少有 3 顆。

2.玩家 B 的選擇

(1)選擇不相信

若玩家 B 選擇不相信，則 B 可執行「抓牌」動作，將所有骰盅打開。若打開骰盅後發現牌型符合玩家 A 所猜測或者更大，則這回合玩家 B 落敗。若不符合玩家 A 所猜測，則玩家 A 落敗。

(2)選擇相信

若玩家 B 選擇相信，則玩家 B 也必須執行「叫牌」動作，且喊出的「N 個 M」中，必須符合 N > n，或 N = n 且 M > m。

3.玩家 C 的選擇

若玩家 B 選擇叫牌，則玩家 C 也必須選擇相信或不相信玩家 B。若選擇抓牌，則勝負在玩家 B、C 之間產生。若玩家 C 選擇叫牌，則輪到玩家 A 進行選擇。

4.遊戲持續進行，直到有一方抓牌為止。

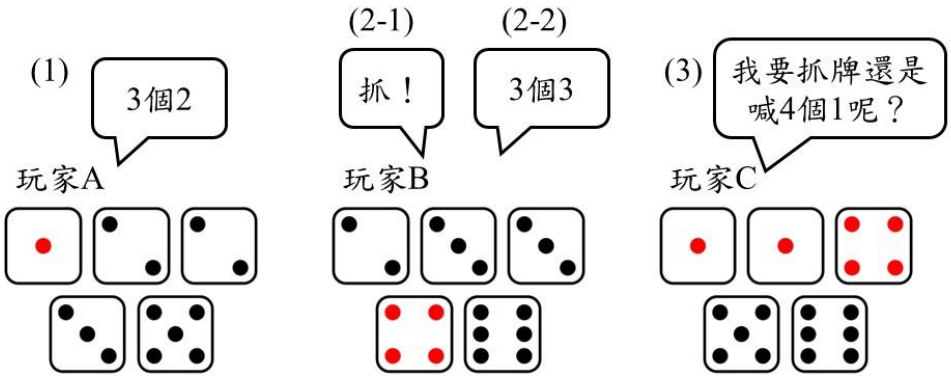

圖 3-1、吹牛骰子遊戲流程示例圖

（三）遊戲複雜度分析

以 2 人遊戲為例，遊戲時，每位玩家的骰子互不相關，因此共有$H\,{}^{6}_{5} = 252$ 種可能牌型，而在 2 人遊玩的情況下，盤面上共有 $252^2 = 63504$ 種可能。喊牌可能性共有 $6 \times 10 = 60$ 種，而 2 位玩家在每場遊戲中約有 5 次行動機會。因此遊戲複雜度約為 $63504 \times 60^5 = 4.938 \times 10^{13}$。以此亦可計算得 3 人遊戲的複雜度約為$9.449 \times 10^{16}$。

三、經驗法則

（一）經驗法則簡介

經驗法則是玩家在經歷一定的遊戲場數後，依據遊玩經驗歸納出的簡單的建議規則，

優點是能在當下的局面快速有效率的得到一個不錯的解。經驗法則帶有主觀性，決定的好壞取決於玩家對經驗的判斷與否，越複雜的賽局越難保證經驗法則的正確性。

經驗法則常常是一個遊戲的初學者最先學習的方法。當賽局樹龐大到難以展開，或賽局本身含有不完全資訊時，經驗法則便可以派上用場，常作為輔助與不同的方法結合。

（二）吹牛骰子主要的經驗法則

一般而言，吹牛骰子會運用到下列幾種經驗法則：

1.對手是否在說謊：透過手中骰子的數量，推測出對方所喊骰子數量為真的機率，判斷是否抓牌。

2.對手手牌的判讀：透過對手喊牌的狀況，判斷對手可能持有的點數。

3.考慮說謊：新手常因不敢說謊而將自己逼入絕境，有些高手則會選擇說謊，說謊的技巧亦是透過經驗累積來取得進步。

4.要不要抓牌：藉由自己的牌判斷對手喊牌是否正確，決定是否抓牌。

5.要喊什麼牌：若不考慮說謊，自己的牌便是喊牌時的主要資訊。運氣差時，手中重複的牌較少，喊牌的選擇也較少，則需要考慮說謊。說謊的被抓與否，或造成他人誤抓與否，與玩家的經驗有關。

（三）機率分析

以兩人吹牛骰子為例，若對方喊牌 a 個 m 點，我方擁有 b 個 m 點，設樣本空間 S 為 5x2 個骰子的所有組合，A 為場上擁有大於等於 a 個 m 點的事件，B 為己方恰擁有 b 個 m 點的事件，則對方喊牌為誠實的機率可表示為：

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

而對方說謊的機率則為$1 - P(A|B)$。該規則也適用於我方喊牌時對方之判斷。若抓牌獲勝則代表對方說謊，若對方說謊則抓牌會獲勝，因此對方說謊的機率約可視為勝率。可以此來討論是否抓牌，喊牌時亦可使用機率分析自己喊牌的安全性（誠實的機率）。

四、強化學習

（一）強化學習簡介

強化學習（Reinforcement Learning，簡稱 RL）是機器學習中的一個領域，是除了監

督學習和非監督學習之外的第三種基本的機器學習方法，強調如何基於環境而行動，以取得最大化的預期利益。

在 RL 環境中，你不是直接教代理要做什麼、怎麼做，而是對代理的每個動作都給予一個獎勵，獎勵可以為正面或負面。代理會傾向於執行會得到正面獎勵的動作。這是一個試誤的學習流程。

強化學習的特點為「不再受限於人類認知」，其學習完全不依靠人類數據和經驗，透過自我學習、反思和獨特的創造力，學習特別快，可在短時間內超越人類。專家說該算法摒棄向人類學習的需求，有可能是對現有人工智慧算法的拓展，更可能成為主流。

目前強化學習廣泛應用在各項領域的人工智慧，包括無人駕駛、工業自動化、金融貿易、自然語言處理、醫療保健、遊戲等等，其中遊戲最廣為人知的應用即為圍棋軟體 AlphaGo Zero，透過神經網路與自我學習的方式，在 40 天的訓練後表現超越世界排名第一的棋王柯潔。

（二）強化學習的要素

1.代理：執行智能決策的軟體程式，即 RL 中的學習者，藉由與環境互動並做出動作，再根據所採取的動作收到獎勵。

2.策略函數：代表選擇某個動作來達到目標的方式，用以定義代理在環境中的行為。代理會根據策略決定要做的動作。策略通常以符號π表示。

3.價值函數：代表助理在某個狀態中有多好。它與某個策略是相依的，常以$v(s)$表示。它等於助理從初始狀態開始後所收到的總期望獎勵。價值函數可以有多個，最佳價值函數代表該函數在所有狀態下得到了最高的價值，擁有最佳價值函數的策略則為最佳策略。

4.模型：代表代理在某個環境的表現。學習可分為模型式（model-based）與無模型（model-free）學習。模型式學習中，代理會運用先前所學到的資訊完成任務，而在無模型學習中，代理只仰賴執行正確動作所得到的試誤經驗。

（三）常見 RL 演算法的步驟

1.首先，代理會執行某個動作來與環境互動。

2.代理執行某個動作，並從一個狀態轉移到下一個狀態。

3.根據所執行的動作，代理會收到一個獎勵。

4.代理會根據獎勵來理解到這個動作是好是壞。

5.如果代理收到了一個正向獎勵，代理就會傾向於執行這個動作，不然代理就會試著去做其他會產生正向獎勵的動作。

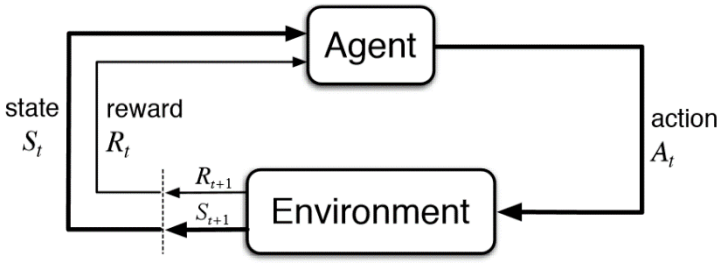

圖 3-2、RL 演算法的流程圖

五、Q 學習

（一）Q 學習簡介

Q 學習(Q-Learning)是強化學習中常用的一種演算法，也是本研究採用的實現方法。Q 學習透過記錄下學習過的策略，告訴代理什麼情況下採取什麼動作會有最大的獎勵值。

Q 學習會將每個行動的獎勵值存在一個 Q-table 中，用來計算在某個狀態下執行某個動作時，可以期望得到最大的獎勵值為多少。這個表可以引導我們選出每個狀態下最好的動作。

（二）Q 學習演算法的步驟

1.初始化 Q-table：建立一個 Q-table，一開始每一格的 Q 值皆為 0。

2.透過 epsilon-貪婪策略(Epsilon-Greedy Policy)選取一個動作：代理會有 epsilon 的機率隨機選擇動作，或是 1-epsilon 的機率根據 Q-table 選擇已知的最佳動作。

3.更新 Q 值：假設現在的狀態為S，選取的動作為A，下一個狀態為S'，獎勵為r，學習率為α，折扣因子為γ，則 Q 值的更新公式為：

$$Q(S,A) = Q(S,A) + \alpha\big(r + \gamma maxQ(S',A) - Q(S,A)\big)$$

4.重複步驟 2、3 直至最終狀態。

肆、研究過程或方法

一、Python 程式實作(程式碼請見附錄)

（一）吹牛骰子程式

將吹牛骰子的幾個重要步驟寫成函式，並透過主程式互相呼叫的方式完成遊戲。該程式裡面包含了幾個函式：

1.遊戲主程式：輸入對戰次數，並於每場對戰初始化各項數據，包括：

(1)玩家手牌：每位玩家的手牌，透過「產生手牌」函式完成。

(2)叫牌紀錄：遊戲過程中每一次叫牌，一開始包含 0 個 0。

每場遊戲由第一位玩家開始喊牌，然後玩家依序行動，若勝者產生則進入下場遊戲，最後統計每位玩家的勝場數。每場遊戲的流程圖如下：

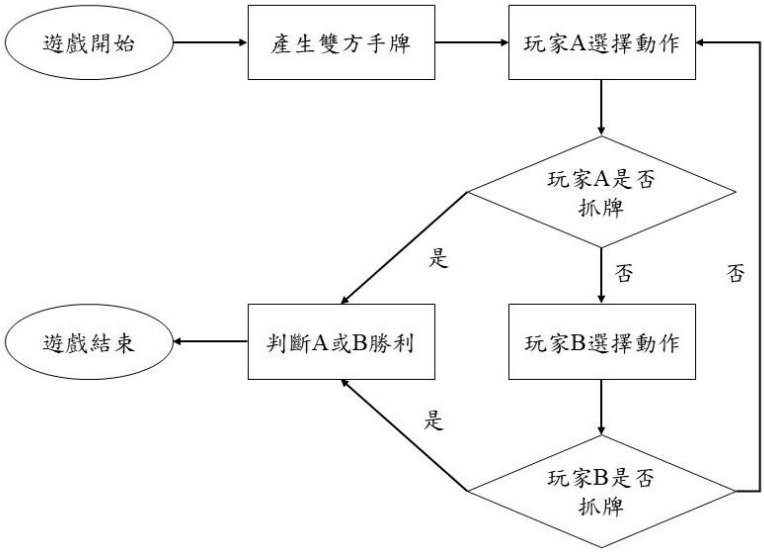

圖 4-1、吹牛骰子遊戲流程圖

2.產生手牌：使用亂數產生的方式，給玩家產生五個點數，並記錄下場面上所有骰子點數的數量，亦可產生特定相同個數的手牌。

3.窮舉所有合法叫牌：在本程式設計中，叫牌透過選取一個合法走步實現，於是每次叫牌時會呼叫此函式，依規則窮舉出所有合法的叫牌，回傳給玩家挑選。

4.檢驗抓牌是否成功：當玩家選擇抓牌會呼叫此函式，依照規則判斷抓牌成功與否，若成功則抓牌者勝利，若失敗則被抓牌者勝利。

（二）強化學習程式

設計上是在吹牛骰子的程式當中新增強化學習，一樣透過函式互相呼叫的方式完成學習。含有強化學習的吹牛骰子分為兩種 class（類別）形成的 object（物件）：

1.盤面：多了強化學習元素的吹牛骰子主程式，裡面的強化學習函式包含：

(1)__init__（初始化類別）：設定每位玩家的遊玩類型（亂數選擇、經驗法則、強化學習），以及遊戲的基本參數。

(2)遊戲主程式：與吹牛骰子的遊戲主程式相同，但在產生勝者時會呼叫「更新 Q 值」來更新強化學習玩家的 Q-table。另外還有一種不學習的遊戲主程式，即強化學習玩家在對戰過程中不會進行學習，用來進行對戰測試。

(3)更新 Q 值：根據玩家的勝負，令玩家呼叫函式更新其 Q-table 之數值。

2.玩家：盤面中的每一位玩家，包含其遊玩類型和以下幾種函式：

(1)__init__（初始化類別）：初始化強化學習之各項參數，包括獎勵、學習率、折扣因子，以及該玩家的 Q-table。

(2)選擇動作：根據其遊玩類型選擇動作，若為強化學習則透過 epsilon-貪婪策略選取一個動作，並記錄沒見過的盤面。

(3)更新 Q 值：每場遊戲結束時，就會呼叫此函式，根據遊戲結果和本身參數，以公式更新其 Q-table 中的數值。

（三）強化學習策略設計

1.策略組成

在 Q-table 中，每項策略符合(s, a)架構，由狀態和動作所組成。我們認為強化學習每回合只需要知道喊牌的歷史(先前喊牌)，便可依照自己持有的手牌，進行喊牌或抓牌的判斷。我們將記錄的策略分為三部分：

(1)喊牌歷史(A)：可記錄一次(當前喊牌)或多次，每一次喊牌為一組點數和個數共兩項，若記錄 n 次歷史則有 2n 項，若喊牌不足 n 次則補上[0 0]。

(2)我方持有手牌(B)：分為部分手牌和完整手牌。若記錄部分手牌，則記錄我方喊牌點數與對應的持有個數共兩項，若為抓牌則統一為[-1 -1]；若記錄完整手牌，則有六種點數的個數共六項。

(3)我方喊牌(C)：固定為一組點數和個數共兩項，若為抓牌則記錄為[-1 -1]。

其中 A、B 為狀態(state)，C 為動作(action)。每個策略會記錄一個 Q 值，初始值皆為 0，在遊玩過程中會依據每局勝負發生變化。

2.策略舉例

(1)部分手牌的策略

若記錄一次喊牌，其形態如[A A B(C) B C]。以[5 2 6 4 3]為例，本策略即代表：「上一次喊牌 2 個 5，我有 4 個 6，我要喊 3 個 6」。

狀態(A)		狀態、動作(BC)	狀態(B)	動作(C)
5	2	6	4	3

表 4-1、[5 2 6 4 3]策略舉例與說明

若記錄兩次喊牌，其形態如[A A A A B(C) B C]。如[5 2 4 1 -1 -1 -1]，即代表：「上一次喊牌 2 個 5，再上一次喊牌 1 個 4，我要抓牌」。依此類推。

(2)完整手牌的策略

若記錄一次喊牌，其形態如[A A B B B B B B C C]。以[5 2 1 1 0 1 0 2 6 2]為例，本策略即代表「上一次喊牌 2 個 5，我總共有 1 個 1、1 個 2、1 個 4、2 個 6，我要喊 2 個 6」。

狀態(A)		狀態(B)						動作(C)	
5	2	1	1	0	1	0	2	6	2

表 4-2、[5 2 1 1 0 1 0 2 6 2]策略舉例與說明

若記錄兩次喊牌，其形態如[A A A A B B B B B B C C]。如[5 2 4 1 0 0 0 1 0 4 6 3]，即代表：「上一次喊牌 2 個 5，再上一次喊牌 1 個 4，我總共有 1 個 4、4 個 6，我要喊 3 個 6」。依此類推。

3.遊玩過程中的運作

遊戲過程中 Q 學習的流程圖如下：

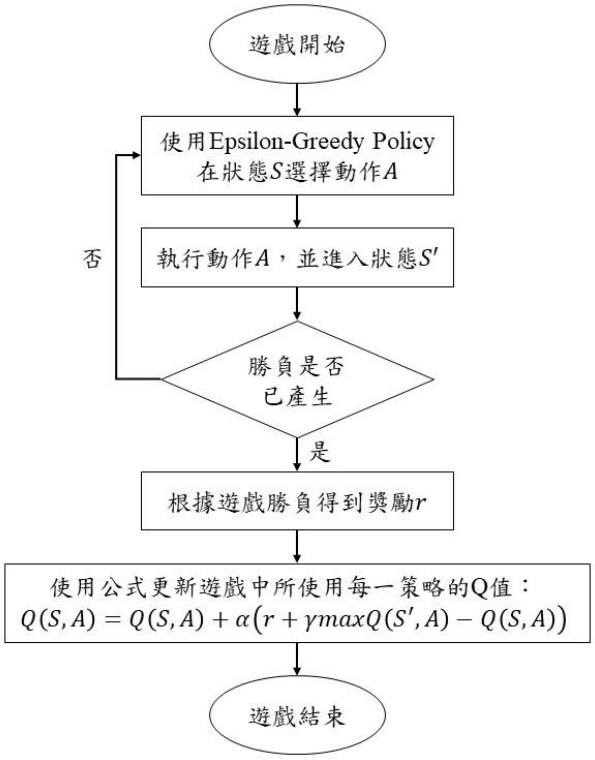

圖 4-2、Q 學習流程圖

每次遊玩中策略會做為一個索引值，當遊玩過程中出現某一狀態時，便會利用 epsilon-貪婪策略選取一個動作。每結束一場遊戲，便會根據勝負利用公式更新遊戲中所使用到的那幾項策略。

4.訓練一定程度後的 Q-table 舉例

以記錄一次歷史和部分手牌的策略為例，在訓練 1000 場後，Q-table 的其中幾項策略可能如下表：

動作(手牌狀態)＼喊牌狀態	對方喊 1 個 2	對方喊 3 個 3	對方喊 4 個 6
我有 4 個 5，喊 1 個 5	0.7951		
我有 4 個 5，喊 2 個 5	0.8516		
我有 4 個 5，喊 3 個 5	0.8705	0.7846	
我有 4 個 5，喊 4 個 5	0.8992	0.8901	
我有 4 個 5，喊 5 個 5	0.3079	0.5076	0.4853

表 4-3、Q-table 範例

上表中每一格即為策略（狀態與動作），格中的數字即為該策略的 Q 值，若為斜線則表示該策略不合法。

5.Q-table 空間理論最大值

Q-table 空間理論最大值，代表 Q-table 中最多含有幾項策略(完全探索)。策略數量的計算分為動作、手牌兩部分：

(1)動作

若玩家有k人，則喊牌可能性共有$6 \times 5k = 30k$ 種，加上抓牌共$30k + 1$種動作。設喊牌歷史記錄n次，加上我方喊牌(或抓牌)有$n + 1$組動作，歷史中記錄了m個[0 0]。喊牌必須越喊越大，因此可視為從$30k + 1$種動作中選出$n + 1 - m$種，即C_{n+1-m}^{30k+1}種組合。將$m = 0, 1, 2, \dots, n$等情況加起來，減去$m = n$ (歷史全為[0 0])時抓牌 1 種，即為策略中所有可能的動作組合。

(2)手牌

若為記錄部分手牌的策略，則手牌有 0~5 共 6 種可能，且抓牌時手牌只有 1 種可能，因此與動作數相乘後需減掉多餘的組合，得一般式：

$$\left[\left(\sum_{m=0}^{n} C_{n+1-m}^{30k+1}\right) - 1\right] \times 6 - \left[\left(\sum_{m=0}^{n} C_{n-m}^{30k}\right) - 1\right] \times 5 \text{ ，} 1 \le n \le 30k$$

若為記錄完整手牌的策略，則手牌有H_5^6種可能，直接與動作數相乘可得 Q-table 空間理論最大值一般式：

$$\left[\left(\sum_{m=0}^{n} C_{n+1-m}^{30k+1}\right) - 1\right] \times H_5^6 \text{ ，} 1 \le n \le 30k$$

例如兩人賽局中，Q-table 空間理論大小值如下：

	一次歷史喊牌	兩次歷史喊牌	三次歷史喊牌
部分手牌	11040	218130	3178160
完整手牌	476280	9545760	141053200

表 4-4、兩人賽局 Q-table 空間理論最大值舉例

二、經驗法則設計

研究者與強化學習模型對戰幾場後，發現人類玩家多會運用幾項特定的經驗法則，因此以保守型的人類玩家觀點寫出了三種經驗法則。本研究使用四種經驗法則，包含文獻中提供

的其中一種經驗法則，以及三種自己設計之經驗法則：

（一）經驗法則 0（來自文獻[1]）：

1.喊牌部分：從此狀況下個數較小的幾種喊牌裡隨機選擇一種。

2.抓牌部分：若對方喊牌數量與自己擁有的同種牌數量差距 ≧ 玩家人數時，即抓牌。

（二）經驗法則 1：

1.喊牌部分：必定選取此情況下最小的誠實喊牌，若無法誠實喊牌則選取最小喊牌。

2.抓牌部分：若自己無法誠實喊牌，且對方喊牌數量與自己相同個數最多的牌數量差距 ≧ 玩家人數時，即抓牌。

（三）經驗法則 2：開局選取最大的誠實喊牌，其他同經驗法則 1。

（四）經驗法則 3：開局略為說謊，選取最大的誠實喊牌並將個數加 1 作為喊牌，其他同經驗法則 1。

三、實驗設計

依照實驗進行邏輯，排列如下：

（一）對戰亂數選擇的強度：測試經驗法則與強化學習是否具有策略強度。

（二）經驗法則與強化學習對戰的優劣勢：

1.比較不同經驗法則間互相對戰的勝率，探討是否有克制性存在。

2.以單一經驗法則訓練的強化學習對戰經驗法則，觀察其優劣。

3.用相同類型、強度的玩家進行對戰，討論是否存在遊玩位置優勢。

（三）強化學習訓練變因對勝率的影響：

1.使用多種經驗法則，以不同的順序訓練強化學習，並測試其效果。

2.實驗強化學習固定在不同位置下訓練的效果。

3.改變強化學習狀態記錄中的手牌與歷史記錄，觀察其訓練效果。

（四）強化學習訓練特性探討：

1.討論強化學習的訓練次數、時間與空間等各項變因和效率。

2.對強化學習中分數較高的優勢策略進行分析。

（五）特定手牌的優劣性探討：驗證特定相同數量的手牌是否會對勝率產生影響。

伍、研究結果

以下實驗中，四種經驗法則將簡稱 EXP0~EXP3，強化學習模型將簡稱 RL，並以數字代表訓練用的經驗法則與順序，如 RL01 為先用 EXP0 訓練，再用 EXP1 訓練的強化學習模型。圖中紅色虛線為標準勝率，兩人賽局為 50%，三人賽局為 33.33%。

一、對戰亂數選擇的強度

我們以經驗法則與訓練 100000 場的強化學習對戰亂數選擇，勝率如下圖：

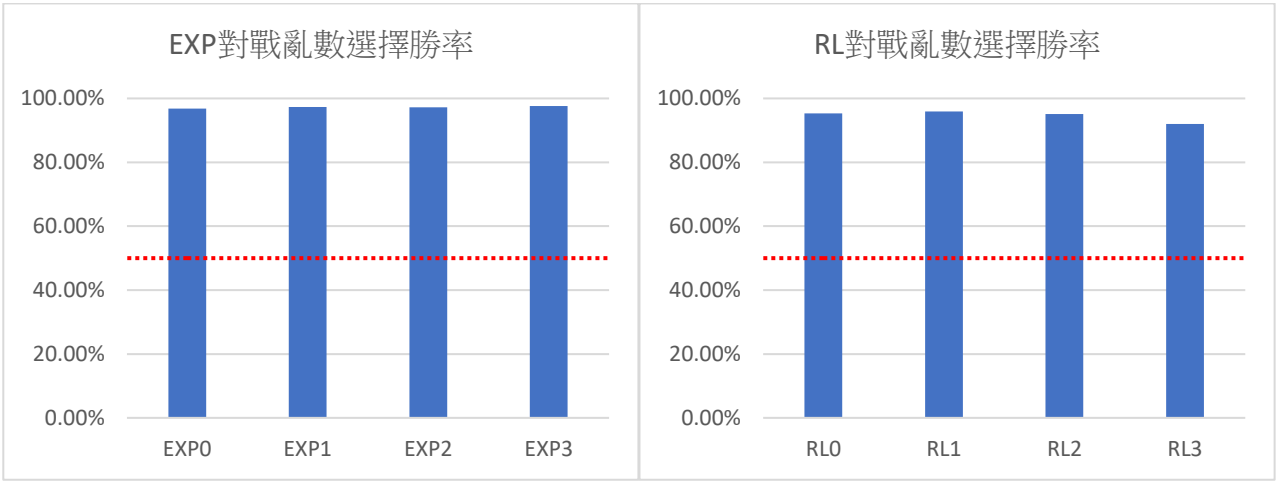

圖 5-1、EXP、RL 對戰亂數選擇之勝率

圖中可以看到，經驗法則與強化學習遠優於亂數選擇，具有一定的策略強度。

二、經驗法則與強化學習對戰的優劣勢

（一）經驗法則之間的勝率分布

1.經驗法則之間具有克制性

在兩人賽局使不同的經驗法則互相對戰，先手勝率如下圖：

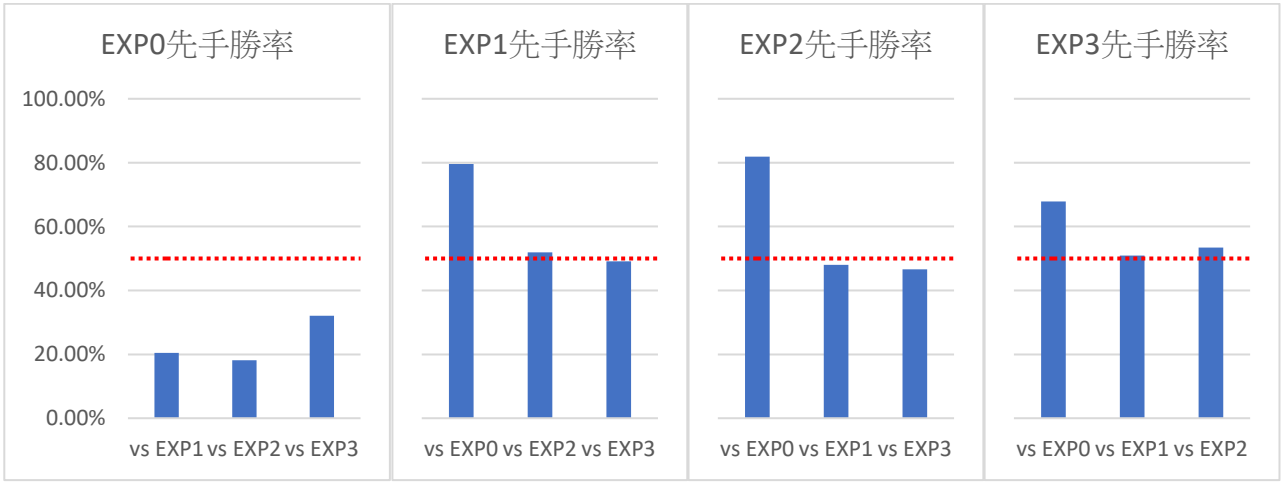

圖 5-2、不同 EXP 互相對戰之先手勝率

圖中可以看到：

EXP0 在對戰其他經驗法則時有明顯劣勢，顯示兩人賽局中 EXP1~EXP3 對 EXP0 具有克制性。其他經驗法則在對戰上勝率差距不大。

2.前一手玩家的遊玩特性對勝率有決定性的影響

三人賽局中，當 EXP 兩同一異，並依不同的順位進行對戰時，勝率如下：

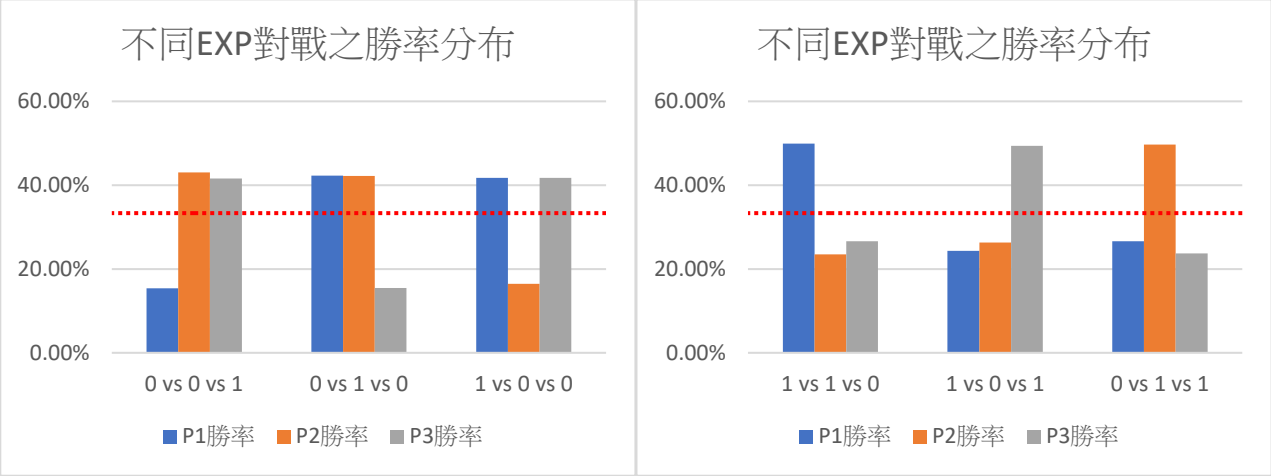

圖 5-3、兩同一異的 EXP 對戰勝率

圖中可以看到：

(1)前一手玩家為 EXP0 的玩家，不論 EXP0 或 EXP1 皆能取得較高的勝率。

(2)玩家之間的勝率關係隨玩家間相對位置關係固定，較不受絕對位置影響。

（二）強化學習對原來經驗法則的優勢

以強化學習與經驗法則進行對戰，其勝率如下圖：

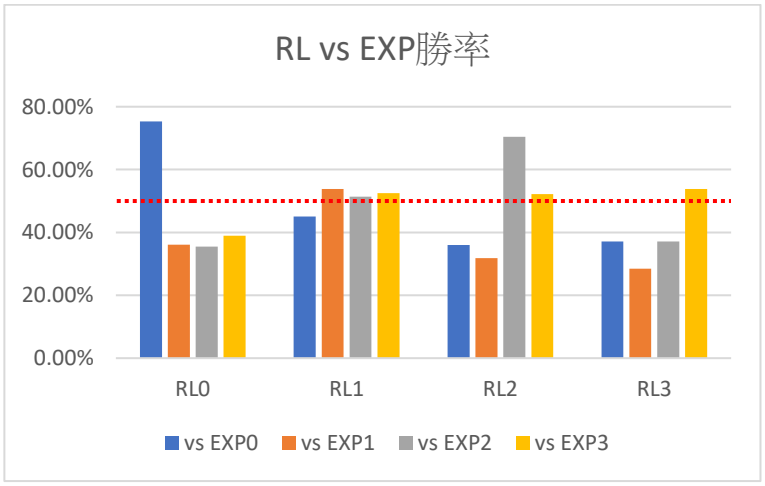

圖 5-4、RL 對戰 EXP 之勝率

圖中可以看到，當四種 RL 在對戰自己訓練用的 EXP 時，皆能取得過半的勝率；對

戰其他 EXP 時則較多處於劣勢。

（三）遊玩位置優勢的存在性

1.兩人賽局的先後手優勢

在兩人賽局使四種相同的 EXP 和 RL 互相對戰，先手勝率如下圖：

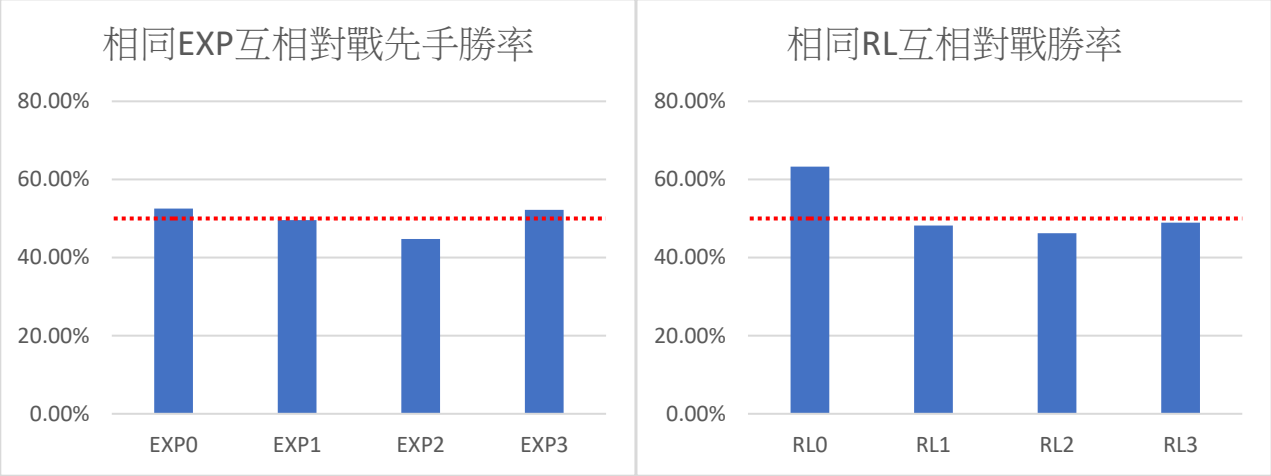

圖 5-5、相同 EXP、RL 互相對戰之勝率

圖中可以看到：

(1)EXP0 和 EXP3 具有些微的先手優勢，EXP2 有後手優勢，EXP1 則幾乎沒有優劣勢之分。

(2)RL0 有明顯的先手優勢，其他 RL 皆為後手些微優勢。

2.三人賽局的遊玩位置優勢

在三人賽局使三種相同 EXP 互相對戰時，其勝率如下圖：

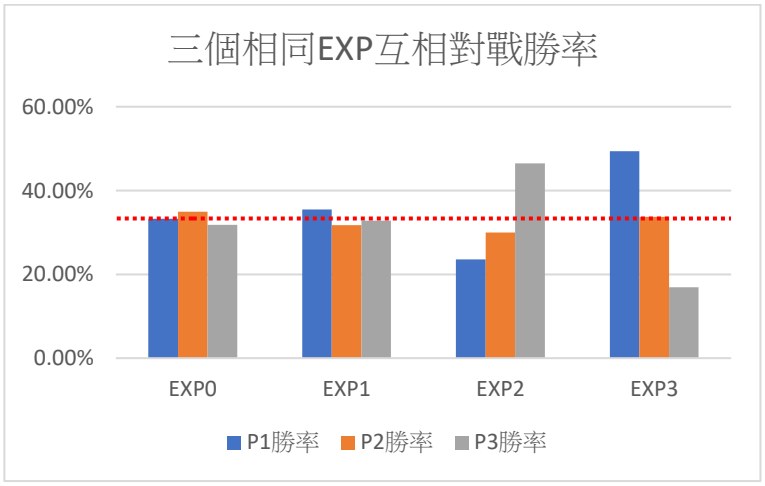

圖 5-6、三個 EXP 皆相同之對戰勝率

圖中可以看到：

(1)三個玩家皆為 EXP0 或 EXP1 時，勝率差距不大，分別以 P2 和 P1 有優勢。

(2)當三個玩家皆為 EXP2 時，P3 勝率明顯較高，EXP3 時則是 P1 具明顯優勢。

三、強化學習訓練變因對勝率的影響

（一）訓練對手數量與順序對勝率的影響

在兩人賽局中使用多個 EXP 依不同的順序訓練各 100000 場，得到數種不同的 RL，使其對戰四種經驗法則，其中幾項勝率如下圖：(完整實驗數據請見附錄)

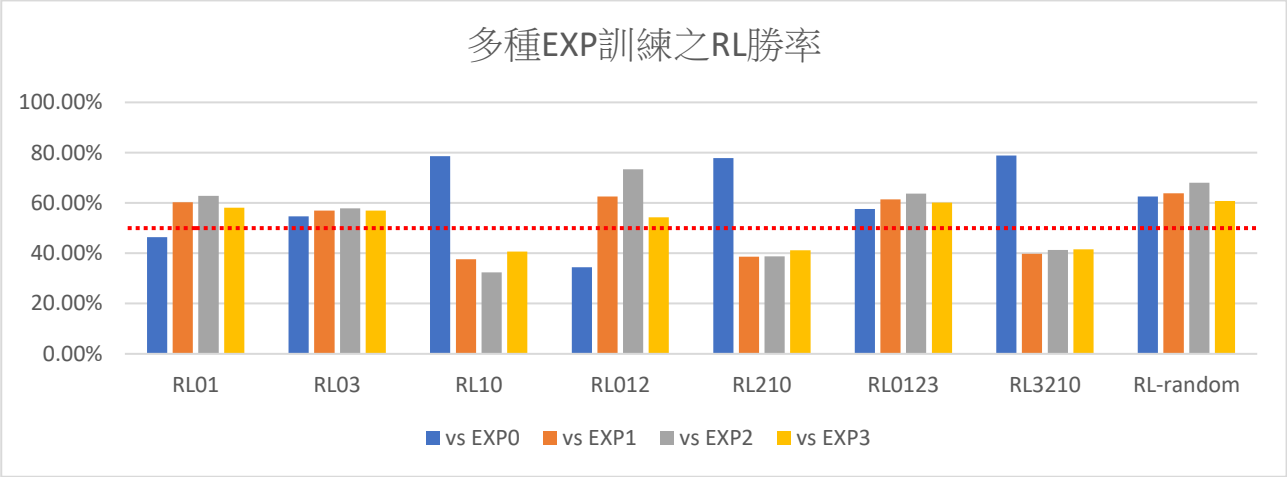

圖 5-7、多種 EXP 訓練的 RL 對戰 EXP 之勝率

其中 RL-random 是以相等機率隨機選取 EXP 進行訓練共 400000 場的 RL。

圖中可以看到：

(1)RL 面對最後訓練的 EXP 都會具有優勢，對先前訓練的 EXP 則會處於劣勢，其中因 EXP1~EXP3 性質接近，故勝率會呈現相近的結果。

(2)RL03 和 RL0123 為特例，面對先前訓練的 EXP0 依然具有優勢；RL-random 則是面對所有 EXP 都能取得優勢。

（二）固定訓練位置對勝率的影響

1.非訓練位置的劣勢

前面實驗中強化學習平均在各個位置進行訓練。我們在兩人賽局中將 RL0 分別固定在先手和後手訓練各 100000 場，之後使其對戰 EXP0，比較其勝率如下：

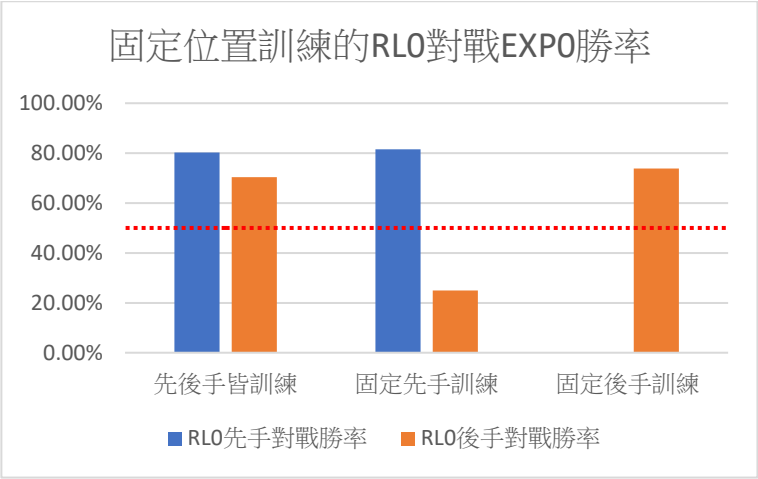

圖 5-8、固定先後手訓練的 RL0 對戰 EXP0 勝率

圖中可以看到：

(1)固定在先手訓練時，RL0 作為先手的勝率差距不大，但後手勝率會大幅降低。

(2)固定後手訓練的 RL0 作為後手的勝率差距不大，但先手勝率趨近於 0。

2.開局喊牌的決定性

在三人賽局中將 RL0 分別固定在 P1、P2 或 P3 訓練，之後使其對戰兩個 EXP0，比較其勝率如下：

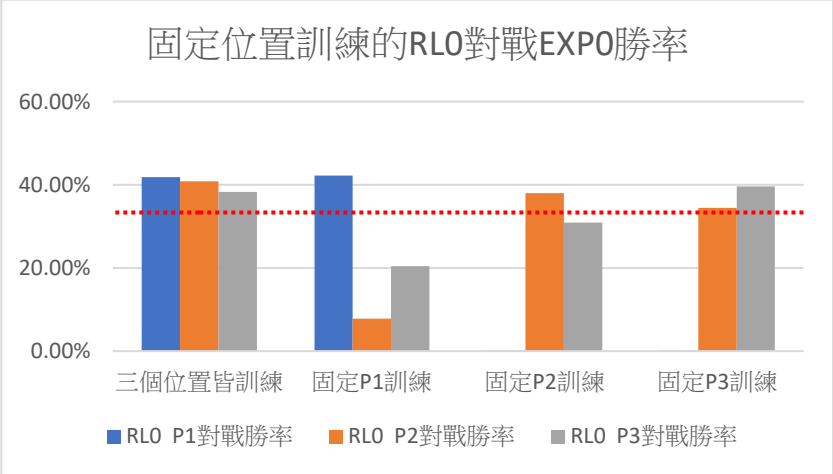

圖 5-9、固定三個位置訓練的 RL0 對戰 EXP0 勝率

圖中可以看到：

(1)分別固定在 P2 或 P3 訓練的 RL0，其作為 P1 的勝率趨近於 0；固定在 P1 訓練的 RL0，作為 P2 和 P3 的勝率會大幅下降，但不至於 0。推測是因為在 P1 對戰必須開局喊牌，若只在 P2 或 P3 訓練無法學到開局喊牌，因此做為 P1 時勝率趨近於 0，但 P1 仍能學到 P2 和 P3 的局中喊牌。

(2)固定在 P1 訓練的 RL0 僅在 P1 有優勢，固定在 P2 或 P3 訓練則是兩個位置都有一定強度。

（三）改變 Q-table 狀態紀錄對勝率的影響

前面實驗中強化學習的狀態紀錄皆為一次歷史、部分手牌。我們嘗試修改狀態記錄，探討記錄更多資訊是否會對勝率產生影響。

1.增加歷史紀錄的影響較小

我們將歷史記錄次數增加為兩次、三次，以部分手牌訓練 RL0 並與 EXP0 對戰，比較其勝率如下：

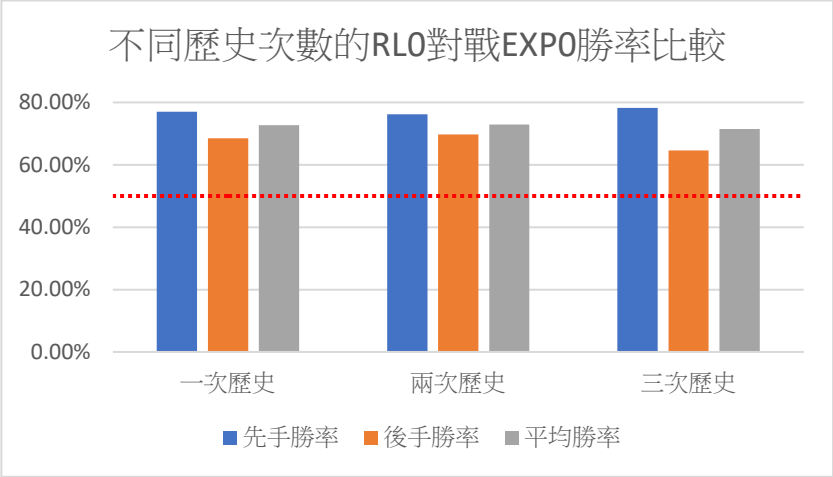

圖 5-10、不同歷史紀錄下 RL0 對戰 EXP0 的勝率

圖中可以看到，增加歷史對於 RL0 訓練的勝率影響不明顯，推測是因為本研究所使用之經驗法則多為保守型，喊牌有規律的緩慢增加，因此歷史較無參考價值。

2.記錄完整手牌對勝率和訓練次數的提升

我們記錄完整手牌，以一次歷史訓練 RL0 並與 EXP0 對戰，比較其勝率如下：

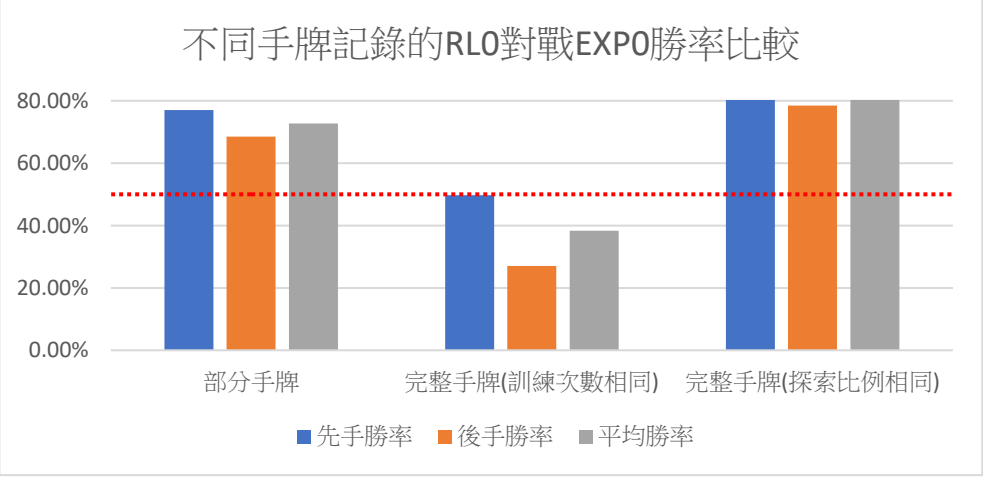

圖 5-11、不同手牌紀錄下 RL0 對戰 EXP0 的勝率

其中部分手牌的 RL0 為訓練 100000 次、狀態空間探索比例 30%。

圖中可以看到：

(1)在訓練 100000 次時，記錄完整手牌的 RL0 勝率明顯較部分手牌低。

(2)當策略探索比例達到 30%時，完整手牌的 RL0 勝率較部分手牌高，可知記錄完整手牌對勝率有提升，但訓練所需次數較多(3000000 次)。

四、強化學習訓練特性探討

（一）訓練效率分析

1.策略探索數量與空間

我們以記錄一次歷史和部分手牌的 RL0 進行策略分析，每 10000 場的策略數量變化如下圖：

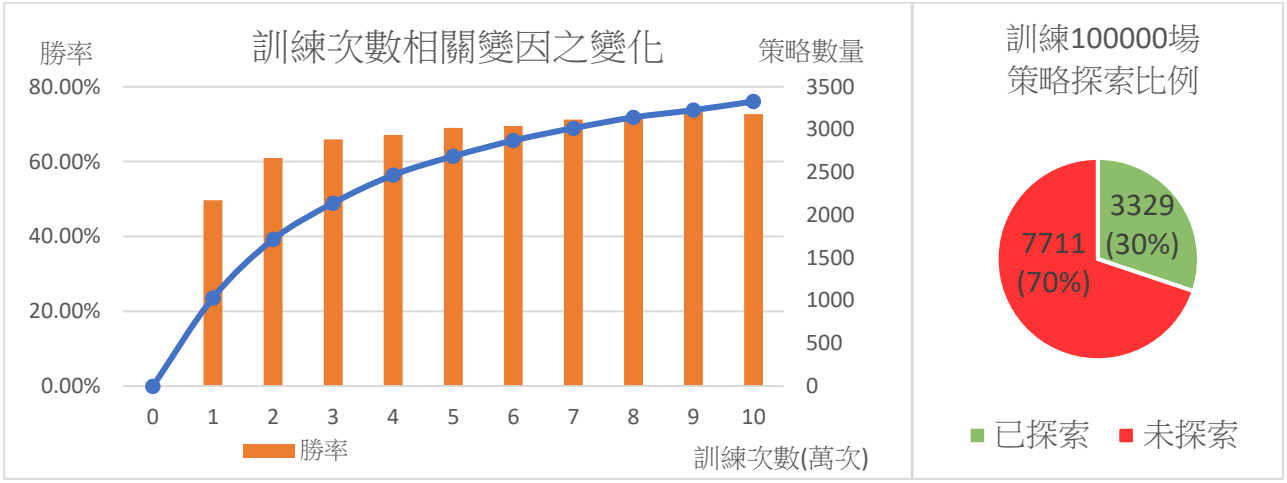

圖 5-12、訓練次數相關變因之變化

在圖 5-12 的左圖中可以看到，強化學習隨著訓練次數上升，策略數量和勝率的增加速度都會漸緩。推測該現象與策略的性質有關。前期強化學習可以快速的探索常見策略，因此策略數量和勝率上升較快。後期探索的策略則開始轉為罕見策略，因不易出現且僅用於極端情況，故增加的速度較小，且對勝率提升不大。

而在右圖中則可看到，訓練 100000 次後的 RL0 僅探索了策略總數的三成，可見罕見策略的占比較常見策略高出許多。推測探索完剩餘情況需要極大量的訓練時間，或是給予特定手牌組合。此外也有一部份策略為不可能策略（例如較少相同個數下極大喊牌），只有隨機選取才會探索到。

2.不同狀態記錄之間訓練時間的差異

我們記錄下不同狀態記錄的 RL0 訓練一定次數時所需的訓練時間，結果如下：

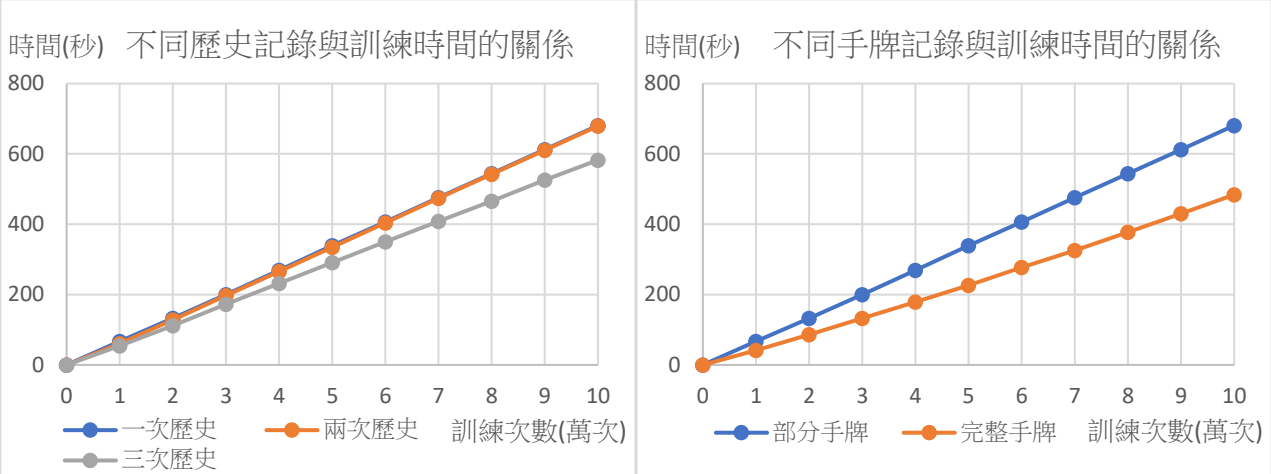

圖 5-13、RL0 訓練次數與訓練時間的關係圖

圖中可以看到，當強化學習記錄越多資料，相同次數的訓練時間會下降。推測是因為強化學習在根據 Q-table 選取動作時，只存取當前狀態的對應動作。若狀態記錄的資訊越多，重複的狀態越少出現，每個狀態對應被探索過的動作也會變小，因此選取速度較快。

（二）學習到的優勢策略分析

1.記錄一次歷史與部分手牌的 RL0

訓練 100000 場後，將 RL0 的 Q-table 中 Q 值最高的前 30 項策略進行分類，大致可分成以下四種：

狀態(A)		狀態、動作(BC)	狀態(B)	動作(C)
0	0	4	4	4
6	1	6	3	3
5	4	5	3	5
2	5	-1	-1	-1

表 5-1、優勢策略舉例與說明

(1)開局時誠實喊牌：6 個。以[0 0 4 4 4]為例：

對方喊 0 個 0，代表這是開局，尚未有人喊牌。此時我方有 4 個 4，並決定一開始就喊最大的 4 個 4。該策略 Q 值約為 0.8975。

(2)遊戲中誠實喊牌：18 個。以[6 1 6 3 3]為例：

對方喊 1 個 6，此時我方有 3 個 6，抓牌一定會失敗，於是決定喊 3 個 6。該策略 Q 值約為 0.892。

(3)略為說謊的喊牌：4 個。以[5 4 5 3 5]為例：

對方喊 4 個 5，此時我方有 3 個 5，對方只要有 1 個 5 便是誠實，抓牌風險非常大，於是決定喊 5 個 5。該策略 Q 值約為 0.8999。

(4)抓牌：2 個。以[2 5 -1 -1 -1]為例：

對方喊 5 個 2，我方決定抓牌，動作為[-1 -1 -1]。該策略無法看到我方手牌，但對方手牌有 5 個 2 的機率極小，我方抓牌獲勝的機率很大，且如果選擇喊牌則說謊的風險非常高。該策略 Q 值約為 0.8885。

2.記錄兩次或三次歷史，搭配部分手牌的 RL0

在記錄兩次或三次歷史的 RL0 中，Q 值最高的前 30 項策略同樣可分為前述四類，但抓牌策略的數量較高，分別有 21 和 26 個。

從高分策略的歷史紀錄中可以觀察到：

(1)觀察三次歷史記錄，發現強化學習在開局喊牌時多會喊較大的個數，推測與前述之開局誠實喊牌有關係。

(2)喊牌過程為慢慢加大，符合設計之經驗法則特性。

(3)抓牌的策略中，對方喊牌除了個數較大外，點數也與我方上輪喊牌不同，推測是因為我方持有的對應手牌較少，進一步確定對手更有可能說謊，因此抓牌獲勝機率高。

3.記錄一次歷史與完整手牌的 RL0

在完整手牌的 RL0 中，Q 值最高的前 30 項策略全部為抓牌。這些策略具有以下特徵：

(1)我方相同個數通常較少(2 個或都不相同)。

(2)對方喊牌個數高，且與我方對應手牌個數相差大，可進一步確定對方說謊的可能性，此時抓牌獲勝機率也更高。

五、特定手牌的優劣性探討

（一）吹牛骰子存在特定優勢手牌與劣勢手牌

在強化學習的高分策略中，我們觀察到其多具有高相同個數的特徵。我們讓對戰雙方皆為 EXP0，其中一方分配特定相同個數的手牌，另一方則分配隨機手牌，使其進行對戰，特定手牌一方的勝率如下：

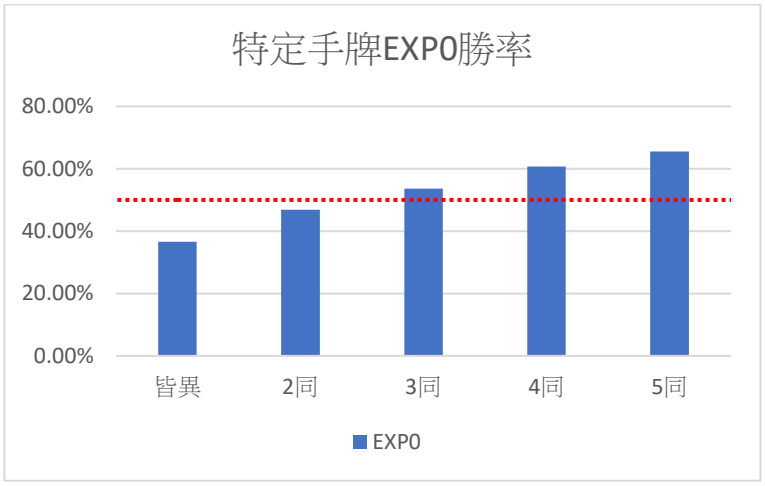

圖 5-14、特定手牌與隨機手牌的 EXP0 互相對戰之勝率

圖中可以看到：

(1)相同個數越高，勝率越高。

(2)當相同個數為 3 個以上時，能取得過半的勝率，屬於優勢手牌。

(3)當手牌相同點數為 2 個或皆不相同時勝率較低，屬於劣勢手牌。

（二）特定手牌優劣會受到玩家特性影響

1.經驗法則的相同趨勢

我們以其它 EXP 進行相同的實驗，共同進行比較，特定手牌一方的勝率如下：

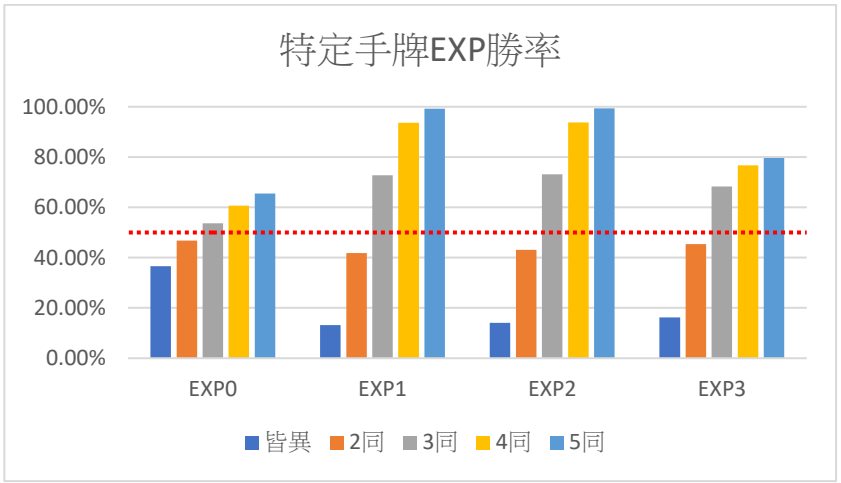

圖 5-15、特定手牌與隨機手牌的 EXP0~EXP3 互相對戰之勝率

圖中可以看到：

(1)EXP1~EXP3 的勝率趨勢與 EXP0 一致，但差距較 EXP0 更大。

(2)EXP3 在相同個數 4 和 5 時勝率的提升較另外兩種小。

2.強化學習不同於經驗法則的趨勢

我們以 RL 進行相同的實驗，共同進行比較，特定手牌一方的勝率如下：

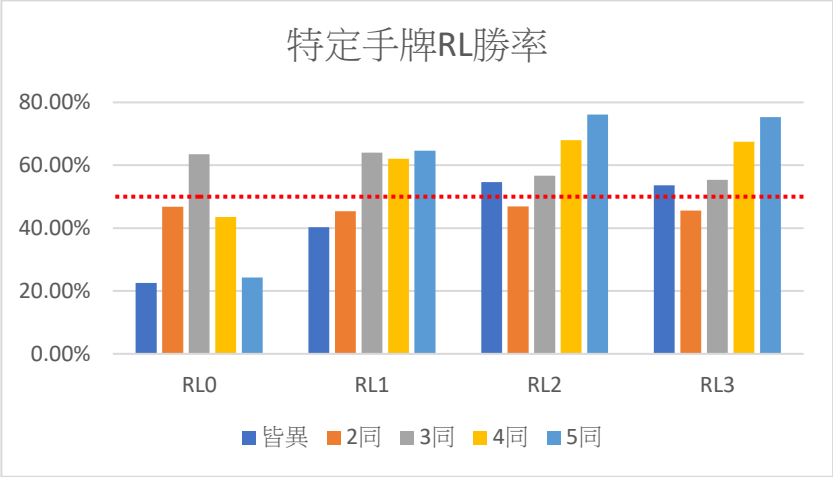

圖 5-16、特定手牌與隨機手牌的 RL0~RL3 互相對戰之勝率

圖中可以看到：

(1)RL0 只有 3 個相同時取得優勢，其他皆為劣勢

(2)RL1 在相同個數為 3 個以上時有優勢，2 個或皆不相同時為劣勢。

(3)RL2、RL3 呈現一樣的結果：2 個相同為劣勢手牌，其他皆為優勢手牌，且相同個數越高優勢越大。

（三）特定手牌互相對戰的優劣勢

1.經驗法則的規律性

我們讓對戰雙方皆為相同的 EXP，並分配特定相同個數的手牌給雙方進行對戰，其先手勝率如下：

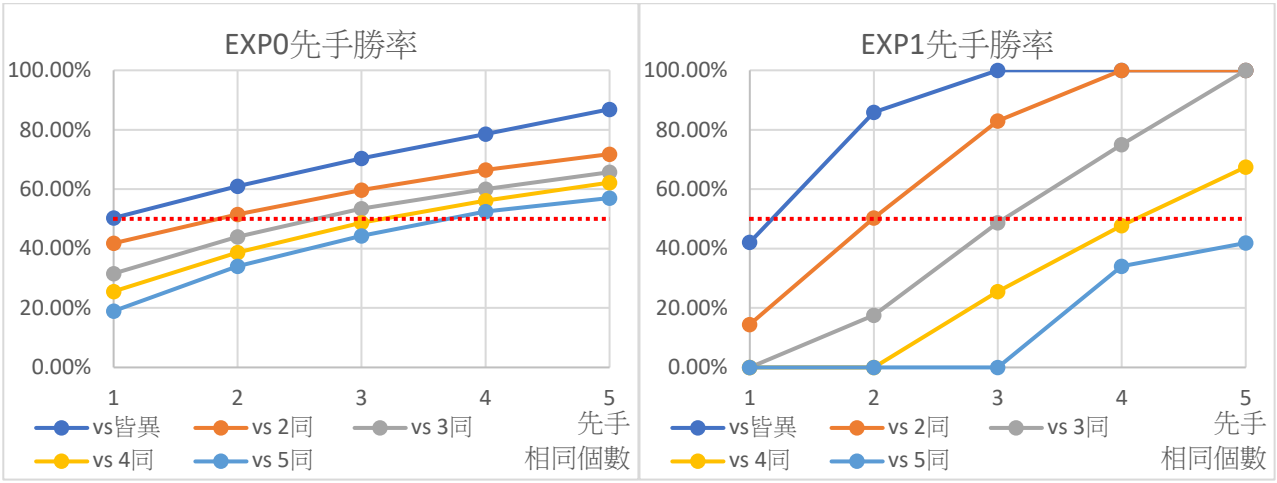

圖 5-17、最大相同個數不同之 EXP 互相對戰之先手勝率

圖中可以看到：

(1)在雙方手牌相同個數不同時，相同個數更多的一方會佔據優勢。

(2)相同個數差距越大，勝率差距也越大，而 EXP1 的差距較 EXP0 更大。

2.強化學習的特定性

我們讓對戰雙方皆為相同的 RL，並分配特定相同個數的手牌給雙方進行對戰，其先手勝率如下：

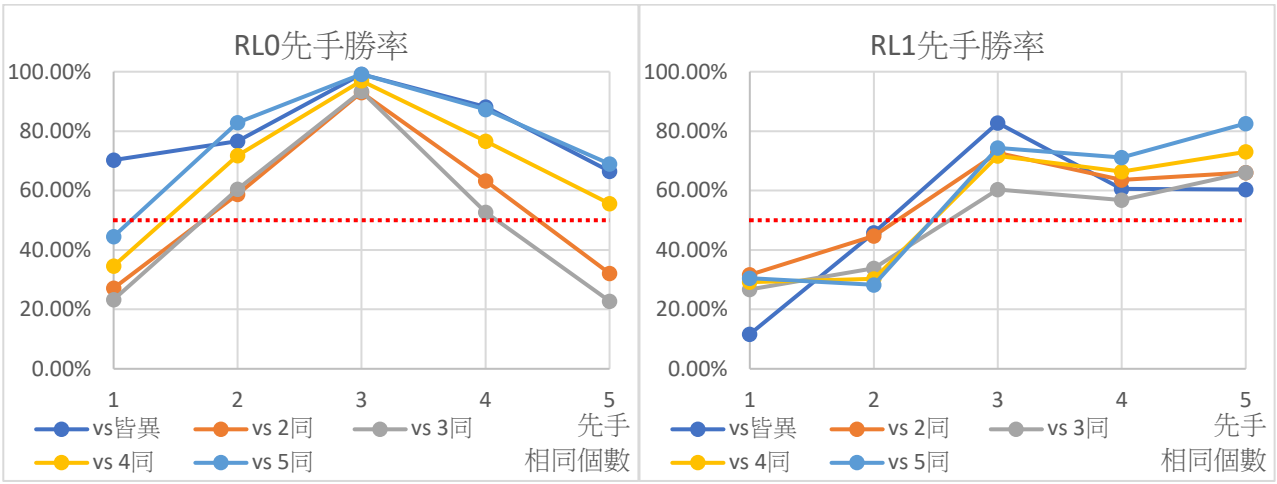

圖 5-18、最大相同個數不同之 RL 互相對戰之先手勝率

圖中可以看到：

(1)RL0 在先手相同個數為 3 個時，對戰每種對手時的勝率皆為最高，相同個數更多或更少則勝率皆下降，且多數情況先手皆具優勢。

(2)RL1 先手的相同手牌個數有 3 個以上時都會取得優勢，反之則必定會處於劣勢，在勝率變化上則較難看出一定規律。

陸、討論

一、過擬合的探討

（一）過擬合的現象

由實驗結果可發現：

1.使用一個經驗法則訓練的強化學習，面對訓練時的經驗法則皆能取得勝率過半的優勢，但面對其他經驗法則多半處於劣勢。

2.以多種經驗法則訓練的強化學習模型，面對先前訓練的經驗法則多半處於劣勢，面對之後訓練的經驗法則卻能取得很好的成績。

推測強化學習針對同一對手經歷大量訓練後，會對該對手形成針對性的策略，且對不同的對手缺乏泛化性，也就是過擬合的現象，此外還會稀釋先前學起來的策略，轉而傾向新的策略。

（二）過擬合的解決方法

本研究之經驗法則皆為固定的策略，因此與經驗法則的對戰可視為一種資料集，該現象則可解釋為：訓練次數過多使強化學習對原資料集產生過擬合，導致對其他資料集的勝率較差。

過擬合的現象可以透過擴大資料集、隨機產生資料集等方式解決，本次實驗中我們使用了隨機選擇經驗法則進行訓練的方式，其實就相當於一次面對四種經驗法則的資料集，並且隨機產生，結果也證實此種方法確實有效。

二、Q-table 狀態空間討論

（一）狀態紀錄資料與 Q-table 大小的關係

本研究所設計之強化學習策略，在記錄完整手牌及更多的喊牌歷史時，會使狀態空間大小明顯增加。先前在實驗的過程中，我們記錄下每場對局中共進行幾次動作(包含抓牌)：

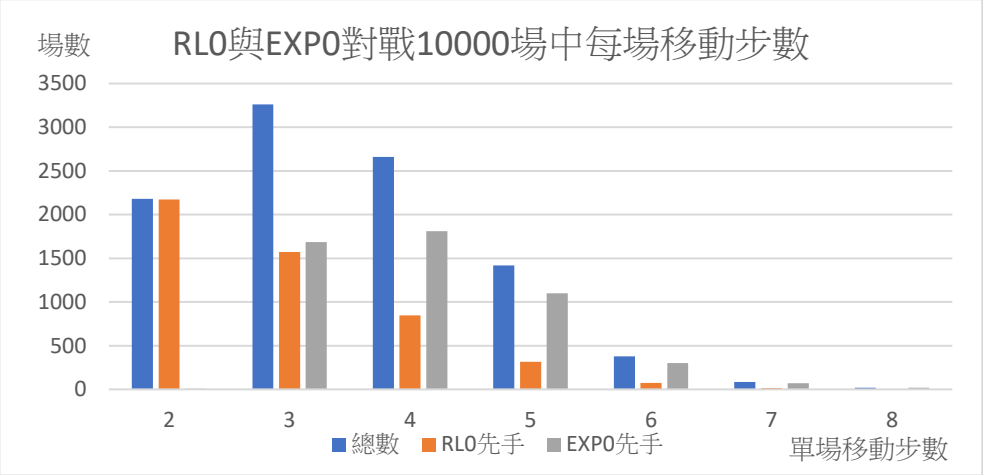

圖 6-1、RL0 對戰 EXP0 時每輪遊戲所動步數

圖中可以看到，RL0 在對戰保守型的 EXP0 時，每輪遊戲步數多介於 2~5 步之間，推測動作較多變化的積極型對手在對戰時，步數會較緩慢增加喊牌的保守型對手更少。

我們以圖中所示之最高步數，以公式計算不同歷史記錄下，部分手牌和完整手牌的狀態空間最大值，如下表：

歷史記錄次數	部分手牌狀態空間	較前一項增加倍數
1	11040	
2	218130	18.76
3	3178160	13.57
4	36434867	10.46
5	342279539	8.39
6	2709584919	6.92
7	18447516909	5.81

表 6-1、不同歷史記錄次數下部分手牌狀態空間最大值

表中可以看到，每多記錄一次歷史，都會使狀態空間顯著增加。結合前面的推測，我們認為強化學習如要增加歷史紀錄，最多只要記錄 5 次即可。

而記錄完整手牌時，狀態空間最大值如下表：

歷史記錄次數	完整手牌狀態空間	較部分手牌增加倍數
1	476280	42.14

2	9545760	42.76
3	141053220	43.38
4	1640238264	44.02
5	15632632008	44.67
6	125572868568	45.34
7	867669465348	46.03

表 6-2、不同歷史記錄次數下完整手牌狀態空間最大值

表中可以看到，記錄完整手牌時，狀態空間大小為部分手牌的 40 倍以上。由此可知訓練至相同探索程度時，記錄較多資訊的強化學習所需訓練次數必定增加。

（二）學到的高分策略的特徵

由實驗結果發現，高分的策略多具有一項共同特徵：相同個數高，無論是我方持有的手牌，或是抓對手的喊牌。推測與手牌的出現機率有關，相同個數越高的手牌出現機率越低，我方持有時更多喊牌選擇，對方喊牌時則容易說謊，因此可以使勝率上升，勝率越高時對應的 Q 值也越高。

三、特定手牌優劣性質討論

由實驗可以發現，經驗法則在手牌相同個數有差異時，相同個數高的一方會具有優勢，強化學習的高分策略也多包含高相同個數的特徵。推測與手牌的出現機率有關。經計算得不同相同個數出現的機率如下：

	都不相同	兩個相同	三個相同	四個相同	五個相同
出現機率	9.26%	69.44%	19.29%	1.93%	0.08%

表 6-3、不同相同個數出現的機率

由上表可發現，除了都不相同外，相同個數越高出現機率越低。當我方手牌持有相同個數越高時，可以誠實喊牌的個數也越高。本研究中經驗法則抓牌的規律皆僅考慮自己的手牌，因此喊牌越高時對手越容易選擇不相信抓牌，若對方選擇喊牌己方抓牌的勝率也很高，因此相同個數高的手牌通常會具有優勢。

柒、結論與未來方向

一、研究結果

（一）吹牛骰子遊戲特性

1.前一手玩家的差異會對勝率有決定性的影響。

2.相同玩家互相對戰時會存在位置優勢，但會隨不同玩家類型而改變。

3.相同個數較高的手牌在遊戲中具有優勢，通常能使玩家的勝率提升。

（二）強化學習於該遊戲實現特性

1.強化學習在訓練過程中會學習對手的策略，且可能發生過擬合的現象。若對所有對手同時進行隨機訓練，則可有效解決過擬合，提高強化學習的泛化性。

2.強化學習若僅於特定位置進行訓練，在其他位置對戰時勝率會下降。

3.狀態中增加歷史記錄會使狀態空間大小顯著增加，但對保守型的經驗法則對手較無幫助，推測對喊牌較不規律的積極型對手才會有用，若要記錄，最多 5 次即可。

4.狀態中記錄完整手牌能提升勝率，但探索至一定程度需要的空間和訓練次數也會顯著增加。

5.強化學習學得的策略，多與機率分析有關，相同個數大的手牌因出現機率較低，容易使人認定為說謊，出現在我方手牌或對方喊牌皆能提高勝率。

6.高分的策略多具有相同個數高、誠實喊牌或高說謊機率抓牌的特性，與保守型玩家的思考方式相近，顯示其模仿經驗法則之特性。

二、未來方向與展望

（一）增加探討更多類型的對手，以及強化學習對這些對手的學習狀況。

（二）嘗試以 Pygame 或其他方式製作可與真人對戰之介面，並透過網路等方式蒐集與人類對戰的紀錄，使強化學習接觸與學習更多種的策略。

（三）實作不一樣的強化學習演算法，例如 SARSA，並比較其學習效果。

（四）增加遊戲人數，推廣探討多人吹牛骰子的規律和特性。

（五）將強化學習的應用推廣到其他層面，期望能解決或改善更多問題。

三、預期應用與貢獻

（一）研究中所使用的分析方法，如利用數學方式計算各項數據和機率，可利用於其他性質類似之理論性研究。

（二）研究中發現之各項性質，包括吹牛骰子與強化學習等，可作為其他研究之參考，或進一步應用與討論之。

捌、參考文獻資料

一、王宇森（民 110）。*不完全資訊賽局對局策略實現之探討*。第二十屆旺宏科學獎佳作。

二、黃信翰（民 98）。*吹牛骰子之人工智慧研究*。國立臺灣師範大學資訊工程研究所碩士論文。

三、Sudharsan Ravichandiran。*用 Python 實作強化學習：使用 TensorFlow 與 OpenAI Gym*。

四、Noam Brown, Anton Bakhtin, Adam Lerer, Qucheng Gong(2020). *Combining Deep Reinforcement Learning and Search for Imperfect-Information Games*. Retrieved from https://arxiv.org/pdf/2007.13544.pdf.

附錄

一、多種經驗法則訓練強化學習的完整實驗數據

在「使用多種經驗法則訓練強化學習」實驗中，我們以正序和逆序和隨機等訓練順序，分別訓練出了 22 種 RL 模型，但有許多 RL 模型的對戰結果相似，故內文中僅保留 8 組數據進行對比。此處展示內文中未出現的完整實驗數據。

（一）兩種經驗法則

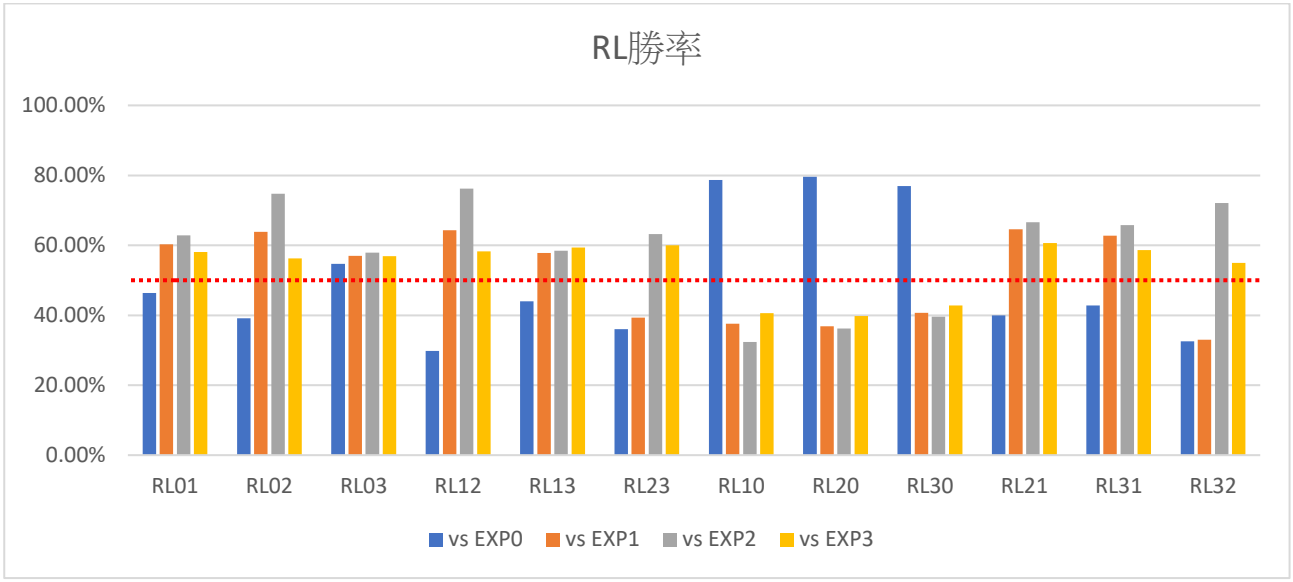

圖 a-1、兩種 EXP 訓練的 RL 對戰 EXP 之勝率

（二）三種經驗法則

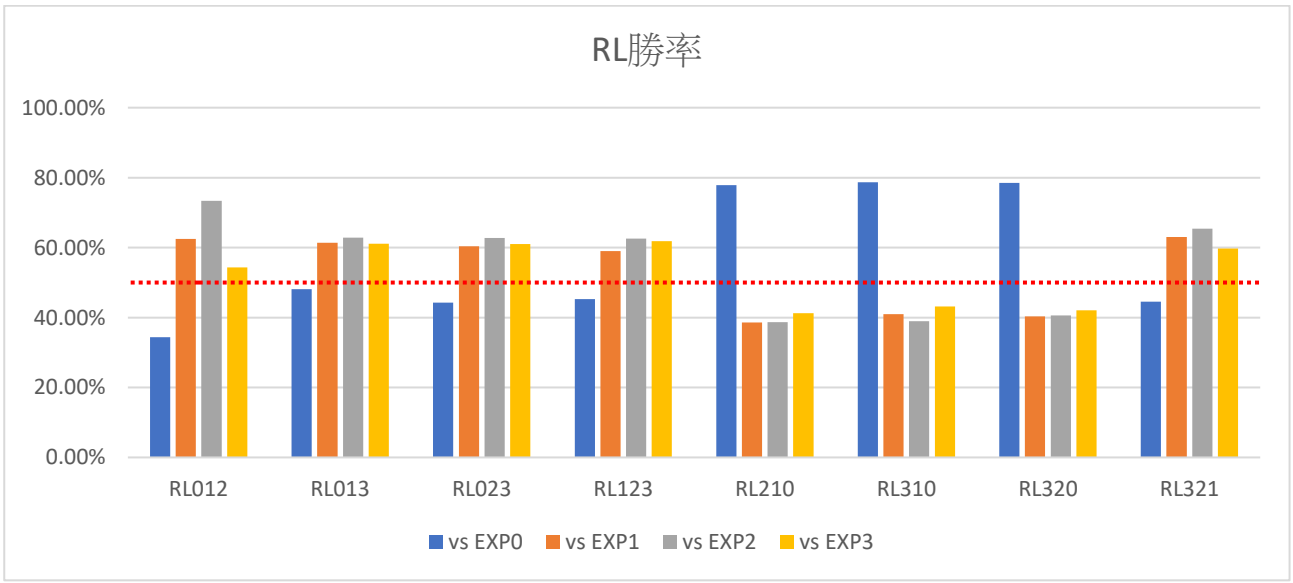

圖 a-2、三種 EXP 訓練的 RL 對戰 EXP 之勝率

二、程式碼節錄

此處展示兩人賽局中吹牛骰子與強化學習的部分程式碼。三人賽局的程式碼與兩人賽局大同小異，便不再重複展示。其中夾雜部分中文，是撰寫前期為方便理解而使用。

（一）吹牛骰子遊戲本體

1.產生手牌

```
def 亂數給定手牌(self):
    if self.same_p1 > 0:
        samedice = random.randrange(0, 6)
        self.先手手牌[samedice]+=self.same_p1
        self.所有手牌總和[samedice]+=self.same_p1
        for i in range(5-self.same_p1):
            dice = random.randrange(0, 6)
            while self.先手手牌[dice] != 0:
                dice = random.randrange(0, 6)
            self.先手手牌[dice]+=1
            self.所有手牌總和[dice]+=1
    else:
        for i in range(5):
            dice = random.randrange(0, 6)
            self.先手手牌[dice]+=1
            self.所有手牌總和[dice]+=1
    if self.same_p2 > 0:
        samedice = random.randrange(0, 6)
        self.後手手牌[samedice]+=self.same_p2
        self.所有手牌總和[samedice]+=self.same_p2
        for i in range(5-self.same_p2):
            dice = random.randrange(0, 6)
            while self.後手手牌[dice] != 0:
                dice = random.randrange(0, 6)
            self.後手手牌[dice]+=1
            self.所有手牌總和[dice]+=1
    else:
        for i in range(5):
            dice = random.randrange(0, 6)
            self.後手手牌[dice]+=1
            self.所有手牌總和[dice]+=1
```

2.窮舉所有合法叫牌

```
def 產生合法走步(self):
    all_act = []
    if self.playerSymbol==0: 玩家手牌 = self.先手手牌
    else: 玩家手牌 = self.後手手牌
    for 幾個 in range(1,11):
        for 幾點 in range(1,7):
            新的叫牌 = [幾點, 幾個]
            if self.叫牌是否合法(self.至今叫牌, 新的叫牌):
                all_act.append(新的叫牌)
    新的叫牌 = [-1, -1] #抓牌
    if self.叫牌是否合法(self.至今叫牌, 新的叫牌):
        all_act.append(新的叫牌)
    return all_act
```

3.檢驗抓牌是否成功

```
def 叫牌是否合法(self, 原叫牌, 新動作):
    if 新動作[1]==-1:
        return self.至今叫牌 != self.完全沒有叫牌
    幾點, 幾個 = 新動作[0], 新動作[1]
    return 幾個 > 原叫牌[1] or (幾個 == 原叫牌[1] and 幾點 > 原叫牌[0])
```

（二）盤面

1.初始化類別

```
def __init__(self, p1, p2):
    self.所有手牌總和 = np.zeros(6,dtype=int)
    self.先手手牌 = np.zeros(6,dtype=int)
    self.後手手牌 = np.zeros(6,dtype=int)
    self.至今叫牌 = [0, 0]
    self.完全沒有叫牌 = [0, 0]
    self.叫牌紀錄 = [0, 0]
    self.p1 = p1
    self.p2 = p2
    self.isEnd = False
    self.winner = 0
    self.stateHash = None
    self.playerSymbol = 0
    self.same_p1 = 0
    self.same_p2 = 0
    self.亂數給定手牌()
```

2.遊戲主程式

```
    def play(self, rounds, show=True, winword=True, count=False, reward=1,
penalty=0):
        先手, 後手 = 0, 0
        先手 old, 後手 old = 0, 0
        if count:
            rounds_count = np.zeros(62,dtype=int)
        for i in range(rounds):
            if i%1000 == 0:
                if show:
                  print("Rounds {}".format(i))
                  print(先手-先手 old, 後手-後手 old)
                先手 old, 後手 old = 先手, 後手
            while not self.isEnd:
                # P1
                act = self.產生合法走步()
                p1_action = self.p1.chooseAction(act, self.至今叫牌, self.
叫牌紀錄, self.先手手牌)
                st_np = [self.叫牌紀錄[0], self.叫牌紀錄[1]]
                for i in range (1, self.p1.history):
                    if i*2 >= np.size(self.叫牌紀錄):
                        st_np = np.concatenate([st_np, [0, 0]])
                    else:
                        st_np = np.concatenate([st_np, [self.叫牌紀錄
[i*2],self.叫牌紀錄[i*2+1]]])
                if self.p1.ptype == "RLnew":
                    st_np = np.concatenate([st_np, self.先手手牌])
                    st_np = np.concatenate([st_np, p1_action])
                else:
                    if p1_action[0] == -1:
                        st_action = [-1, -1, -1]
                    else:
                        st_action = [p1_action[0], self.先手手牌
[p1_action[0]-1], p1_action[1]]
                    st_np = np.concatenate([st_np, st_action])
                st_hash = str(st_np)
                self.p1.addState(st_hash)
                if p1_action[1]==-1:
                    if self.抓成功與否():
                    if count:
```

```
                    rounds_count[int(len(self.叫牌紀錄)/2)]+=1
                self.giveReward(reward, penalty)
                self.p1.reset()
                self.p2.reset()
                self.reset()
                break
            self.更新叫牌狀況(p1_action)

            # P2
            act = self.產生合法走步()
            p2_action = self.p2.chooseAction(act, self.至今叫牌, self.
叫牌紀錄, self.後手手牌)
            st_np = [self.叫牌紀錄[0], self.叫牌紀錄[1]]
            for i in range (1, self.p2.history):
                if(i*2 >= np.size(self.叫牌紀錄)):
                    st_np = np.concatenate([st_np, [0, 0]])
                else:
                    st_np = np.concatenate([st_np, [self.叫牌紀錄
[i*2],self.叫牌紀錄[i*2+1]]])
            if self.p2.ptype == "RLnew":
                st_np = np.concatenate([st_np, self.後手手牌])
                st_np = np.concatenate([st_np, p2_action])
            else:
                if p2_action[0] == -1:
                    st_action = [-1, -1, -1]
                else:
                    st_action = [p2_action[0], self.後手手牌
[p2_action[0]-1], p2_action[1]]
                st_np = np.concatenate([st_np, st_action])
            st_hash = str(st_np)
            self.p2.addState(st_hash)
            if p2_action[1]==-1:
                if self.抓成功與否():
                    self.winner = 1
                    self.isEnd, 後手 = True, 後手+1
                else:
                    self.winner = 0
                    self.isEnd , 先手 = True, 先手+1
                if count:
```

```
                    rounds_count[int(len(self.叫牌紀錄)/2)]+=1
                self.giveReward(reward, penalty)
                self.p1.reset()
                self.p2.reset()
                self.reset()
                break
            self.更新叫牌狀況(p2_action)
    if rounds%1000 == 0:
        if show:
            print("Rounds {}".format(rounds))
            print(先手-先手 old, 後手-後手 old)
        先手 old, 後手 old = 先手, 後手
    if winword:
        print("玩家 1 勝率："+str(先手*100/rounds)+'%')
    else:
        print(str(先手*100/rounds)+'%')
    if count:
        print("步數統計")
        for i in range(62):
            if rounds_count[i]>0:
                print(str(i)+"步 "+str(rounds_count[i]))
```

（三）玩家

1.初始化類別

```
def __init__(self, name, ptype="RL", history=1, do_shuffle=False,
para=[0.2, 0.1, 0.9]):
    self.name = name
    self.ptype = ptype
    self.states = []
    self.lr = para[0]
    self.epsilon = para[1]
    self.decay_gamma = para[2]
    self.states_value = {}
    self.do_shuffle = do_shuffle
    self.rand_area = [0.25, 0.5, 0.75]
    self.history = history
```

2.選擇動作

```
def chooseAction(self, actions, 至今叫牌, 叫牌紀錄, 我的手牌):
    if self.ptype=="RAND":
```

```
            return random.choice(actions)
        exparea = 2.0
        if self.ptype=="EXPRAND":
            exparea = random.random()
        if self.ptype=="EXP0" or exparea<=self.rand_area[0]:
            幾點, 幾個 = 至今叫牌
            cnt = min(len(actions)-1,5)
            r = random.randint(0,cnt)
            if 我的手牌[幾點-1]+1 < 幾個 :
                return actions[-1]
            else:
                return actions[r]
        elif self.ptype=="EXP1" or exparea<=self.rand_area[1]:
            幾點, 幾個 = 至今叫牌
            最多個 = 0
            for i in range(6):
                if 我的手牌[i] >= 最多個:
                    最多個 = 我的手牌[i]
                    最多點 = i+1
            if 最多個+1 < 幾個 :
                return actions[-1]
            else:
                return self.honestAction(actions, 我的手牌, [最多點, 最多個
+1])
        elif self.ptype=="EXP2" or exparea<=self.rand_area[2]:
            幾點, 幾個 = 至今叫牌
            最多個 = 0
            for i in range(6):
                if 我的手牌[i] >= 最多個:
                    最多個 = 我的手牌[i]
                    最多點 = i+1
            if 幾個 == 0:
                return [最多點, 最多個]
            elif 最多個+1 < 幾個 :
                return actions[-1]
            else:
                return self.honestAction(actions, 我的手牌, [最多點, 最多個
+1])
        elif self.ptype=="EXP3" or exparea<=1:
```

```
            幾點, 幾個 = 至今叫牌
            最多個 = 0
            for i in range(6):
                if 我的手牌[i] >= 最多個:
                    最多個 = 我的手牌[i]
                    最多點 = i+1
            if 幾個 == 0:
                return [最多點, 最多個+1]
            elif 最多個+1 < 幾個 :
                return actions[-1]
            else:
                return self.honestAction(actions, 我的手牌, [最多點, 最多個
+1])
        if self.ptype=="RL":
            if np.random.uniform(0, 1) <= self.epsilon:
                idx = np.random.choice(len(actions))
                action = actions[idx]
            else:
                value_max = -999
                if self.do_shuffle:
                    random.shuffle(actions)
                state = [叫牌紀錄[0], 叫牌紀錄[1]]
                for i in range (1, self.history):
                    if i*2 >= np.size(叫牌紀錄):
                        state = np.concatenate([state, [0, 0]])
                    else:
                        state = np.concatenate([state, [叫牌紀錄[i*2], 叫牌紀
錄[i*2+1]]])
                for a in actions:
                    if a[0] == -1:
                        new_a = [-1, -1, -1]
                    else:
                        new_a = [a[0], 我的手牌[a[0]-1], a[1]]
                    next_state = np.concatenate([state, new_a])
                    next_st_hash = str(next_state)
                    value = 0 if self.states_value.get(next_st_hash) is None
else self.states_value.get(next_st_hash)
                    if value >= value_max:
                        value_max = value
                        action = a
```

```
            return action
        if self.ptype=="RLnew":
            if np.random.uniform(0, 1) <= self.epsilon:
                idx = np.random.choice(len(actions))
                action = actions[idx]
            else:
                value_max = -999
                if self.do_shuffle:
                    random.shuffle(actions)
                state = [叫牌紀錄[0], 叫牌紀錄[1]]
                for i in range (1, self.history):
                    if i*2 >= np.size(叫牌紀錄):
                        state = np.concatenate([state, [0, 0]])
                    else:
                        state = np.concatenate([state, [叫牌紀錄[i*2], 叫牌紀
錄[i*2+1]]])
                state = np.concatenate([state, 我的手牌])
                for a in actions:
                    next_state = np.concatenate([state, a])
                    next_st_hash = str(next_state)
                    value = 0 if self.states_value.get(next_st_hash) is None
else self.states_value.get(next_st_hash)
                    if value >= value_max:
                        value_max = value
                        action = a
            return action
```

3.更新 Q 值

```
    def feedReward(self, reward):
        if self.ptype=="RL" or self.ptype=="RLnew":
            for st in reversed(self.states):
                if self.states_value.get(st) is None:
                    self.states_value[st] = 0
                self.states_value[st] += self.lr*(self.decay_gamma*reward -
self.states_value[st])
                reward = self.states_value[st]
```