國立臺灣師範大學
資訊工程研究所碩士論文

指導教授 : 林順喜 博士

不完全資訊賽局-蜜月橋牌之研究

On the Study of an Imperfect Information Game: Honeymoon Bridge

研究生 : 葉俊廷 撰

中華民國 九十八 年 六 月

摘要

近年來，許多人投入人工智慧的研究領域，博弈遊戲也是其中之一。在大多數的遊戲中，通常是藉由設計者根據自身的經驗或感覺訂定一些準則來達成人工智慧的效果，這種作法雖然效果好，但設計者需要精通該遊戲且對遊戲中各種可能情況都做詳盡考慮，才能制定出夠精確的準則。一旦經過大量遊戲後，這些準則將被使用者掌握，使用者就能清楚知道電腦程式的行為模式，也就有可能會大幅降低遊戲程式的勝率。

在本篇蜜月橋牌的研究中，希望能夠探討出以盡量不使用經驗法則，而以電腦模擬計算及參數的演化做到相同的效果。因為使用模擬的結果來做決策依據，所以在不同的兩局中，有著相同局面卻有可能做出不同的決策，程式中參數經由大量的遊戲後慢慢做修正，可以解決設計者制定參數時不夠準確的問題。

在這篇論文中，我們設計出一些方法及資料結構來加速程式對牌局的模擬及計算。目前程式的牌力，對牌面局勢的價值分析已有一定的準確性，但由於不使用經驗法則，在某些牌局做出的決策會不夠正確。大體來說，目前我們程式的牌力大約與一般業餘玩家相近。

ABSTRACT

The study of games has been identified as an important research area in artificial intelligence. In the development of most game programs, the designers usually use heuristics to build a rule-based system to play a reasonably intelligent strategy. However, those approaches do not provide any guarantees about the performance of the computed strategies. When the programs play with human players lots of times, the human player may easily find the weaknesses of the programs.

In the study of Honeymoon Bridge, we try to use simulation methods instead of rule-based methods. The program may make nondeterministic decisions in most situations because of the randomness involved in the computation. The parameters of the program can be gradually tuned according to the game results in order to improve its winning rate.

In this thesis, we design some data structures and algorithms to speed up the calculation and simulation for the Honeymoon Bridge game. Experimental results show that our program can compete with amateur players.

目錄

圖表目錄

第 一 章 緒論

第一節 研究背景及動機

隨著科技的進步，電腦的計算速度及能力也不斷提升，許多複雜或者需要大量計算的問題也慢慢被解決。在未來，愈來愈多的問題都可以在電腦上實作或應用。而電腦高速的運算能力恰好在棋類、牌類甚至各種博弈遊戲能夠得到極大的發揮，也有愈來愈多的學者投入這塊研究領域。

依照遊戲資訊的透明程度，可分為完全資訊賽局(Perfect Information Game)及不完全資訊賽局(Imperfect Information Game)。在完全資訊賽局如象棋(Chinese Chess)、西洋棋(Chess)、圍棋(Go)等都已取得重大成就，拜電腦高速運算的能力，遊戲盤面的搜尋深度及廣度都勝過以往，程式棋力已達人類高段棋手甚至超越人類的程度(註一)。但是在不完全資訊賽局，如德州撲克(Poker)、橋牌(Bridge)等，目前還較少有人研究。

在不完全資訊部分賽局的遊戲，尤其是牌類的遊戲，目前採用的演算法大都是由程式設計者根據專家的意見或者參考書籍的說明，將高手的經驗，以人工的方式，轉化成審局分數，但這種做法受限於設計者的專精程度及理解力，並非適合所有的盤面，因此我希望盡量減少設計者自己主觀意識的帶入，改由程式透過模擬來選擇走步，雖然這樣的做法效果不一定會優於前者，但我們認為這樣的作法才真正發揮了電腦博弈遊戲的長處。

註一:

西洋棋: 1997 深藍(Deep Blue) 與西洋棋冠軍卡斯帕洛夫比賽，取得二勝三和一負

2002 Deep Fritz 與冠軍克拉姆尼克比賽，取得 2 勝 2 負 4 和

2006 Deep Fritz 與冠軍克拉姆尼克比賽，取得 4 勝 2 負

圍棋 : 2008 周俊勳在全球九路電腦圍棋賽中和程式比賽，獲得 3 戰全勝

第 二 章 蜜月橋牌

第一節 蜜月橋牌介紹

橋牌(Bridge)是一種非常普遍的撲克牌遊戲，玩家以牌技及運氣獲取牌磴，具有非常高的趣味性，一般為四人遊戲，如圖 2-1 所示。但根據玩家人數衍生出了五人橋牌(拿破崙)、三人橋牌及蜜月橋牌(Honeymoon Bridge)等不同的玩法。比賽時，玩家將會分成兩邊對抗，當玩家人數超過三人，演算法的設計就必須考慮到同盟玩家的動作，這部分需要玩家間的默契及共識，很難以模擬的方式去計算，所以本研究的主題選擇了兩個玩家的蜜月橋牌。

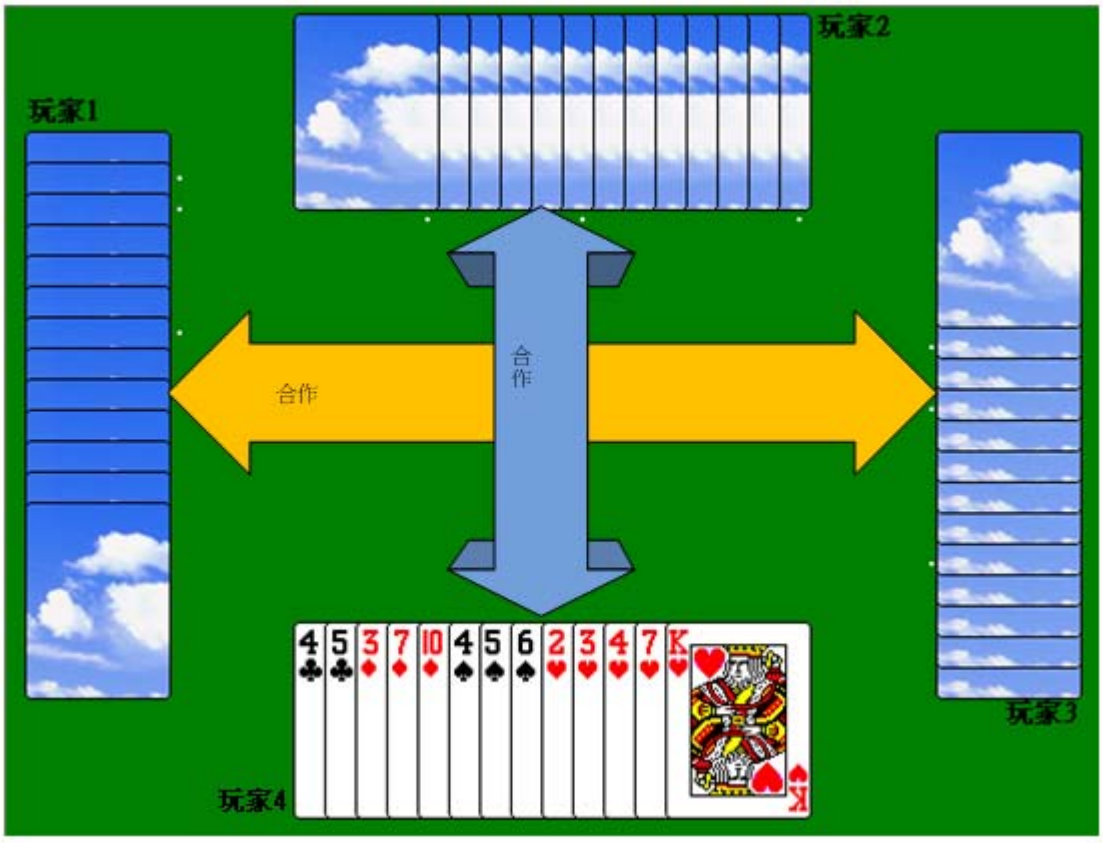


圖 2-1 四人橋牌玩家間的合作關係示意圖

第二節 蜜月橋牌規則

蜜月橋牌對戰時分為叫牌、換牌、打牌三個階段，以下將會詳細說明。

發牌使用五十二張標準撲克牌，首先雙方各抽取十三張牌，剩餘的二十六張牌放在牌桌中間，玩家此時只能看到自己手上的十三張牌，如圖 2-2。

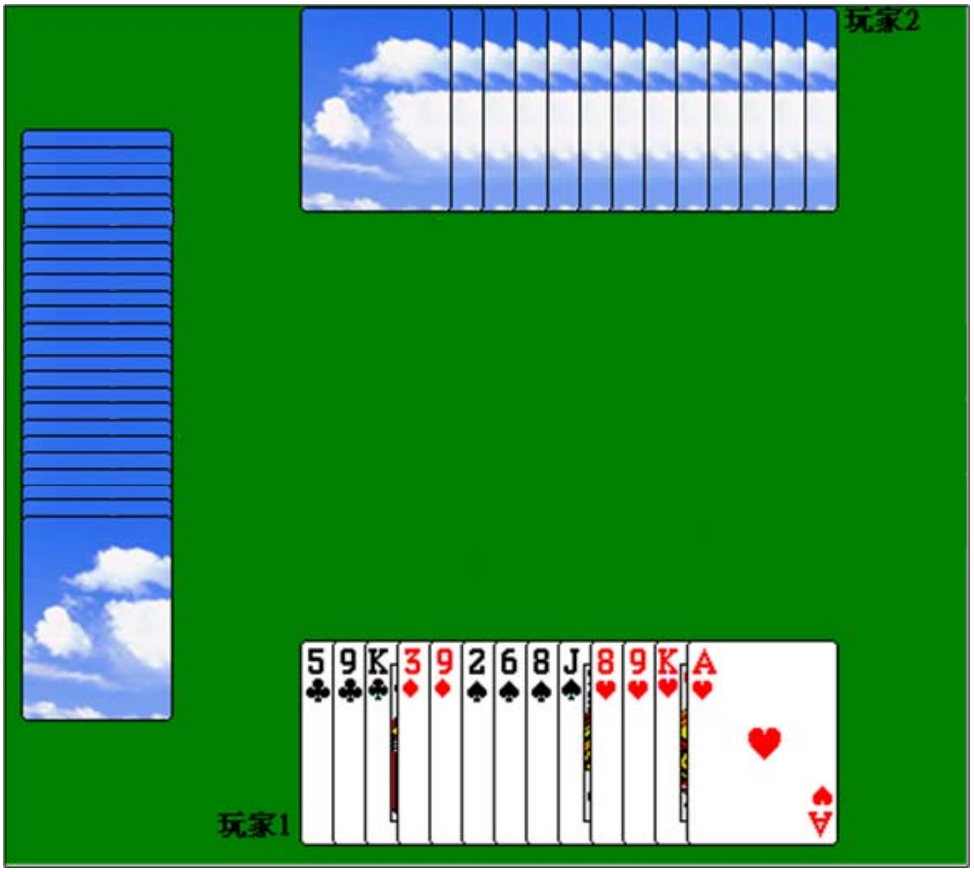


圖 2-2 遊戲開始的畫面

叫牌:

叫牌決定誰做莊方以及最後的合約，玩家此時會根據手牌的好壞出價(Bid)，以下介紹出價的規則及一些術語。

撲克牌的四個花色:

黑桃(spade) ♠ 、 紅心(heart)♥ 、方塊(diamond) ♦、 梅花(club)♣。

「叫價」會指定線位和花色，也就是指示了一個想打某合約的提議。想叫牌

的玩家必須叫出比前一個叫價更高的叫價。所謂比較高的叫價，是指叫價的線位較高，或是在同一線位上，但花色的等級較高。在叫價時花色大小關係如下:

黑桃(spade) ♠ > 紅心(heart)♥ > 方塊(diamond) ♦ > 梅花(club)♣

例如當前一家叫出 2♥ ， 接下來的人想選擇叫♠，只需要叫 2♠，但是若想叫的花色是♣，就必須叫到 3♣。當然，若是玩家認為自己的手牌很好，也可以直接喊出較高的線位，不給對手叫牌的機會。叫牌直到一方不叫價(pass)時結束，此時契約成立，合約磴數為叫到的線位加上六，這是叫到價的玩家在終局需達成的勝利條件，如果沒有達成就是對方獲勝。叫到的「花色」即為這場比賽的「王牌」，在此特別強調一點，王牌指的是花色，也就是屬於這個花色的全部 13 張牌都是王牌。

換牌:

一開始放在牌桌中間的二十六張牌，會先翻開上面的第一張，如圖 2-3。第一輪由上一階段(叫牌)未叫到牌的玩家先出牌，先出牌的玩家打出的牌稱為「引牌」。引牌可以是手上的任何一張牌，但後出牌的玩家則必須「跟出」同一花色（亦即，必須打出和引牌同樣花色的牌），除非他已經沒有該花色的牌。兩人「牌型」較大者可拿走牌桌中間翻開的那張牌加入自己的手牌中，且下一輪先出牌；牌型較小者拿走覆蓋在牌堆中的第一張牌，且這張牌**不需要讓對手看到**。重複此步驟直到將牌堆的二十六張牌抽完。在這個步驟中，玩家會盡量爭取想要的牌，以及盡量打掉手上不想要或者較弱的牌。

牌型的大小判斷如下:先比花色，再比數字大小。

花色大小順序: 王牌 > 引牌花色 > 其他花色。

數字大小: Ace > K > Q >J > 10 > 9 > 8 > 7 > 6 > 5 > 4 > 3 > 2。

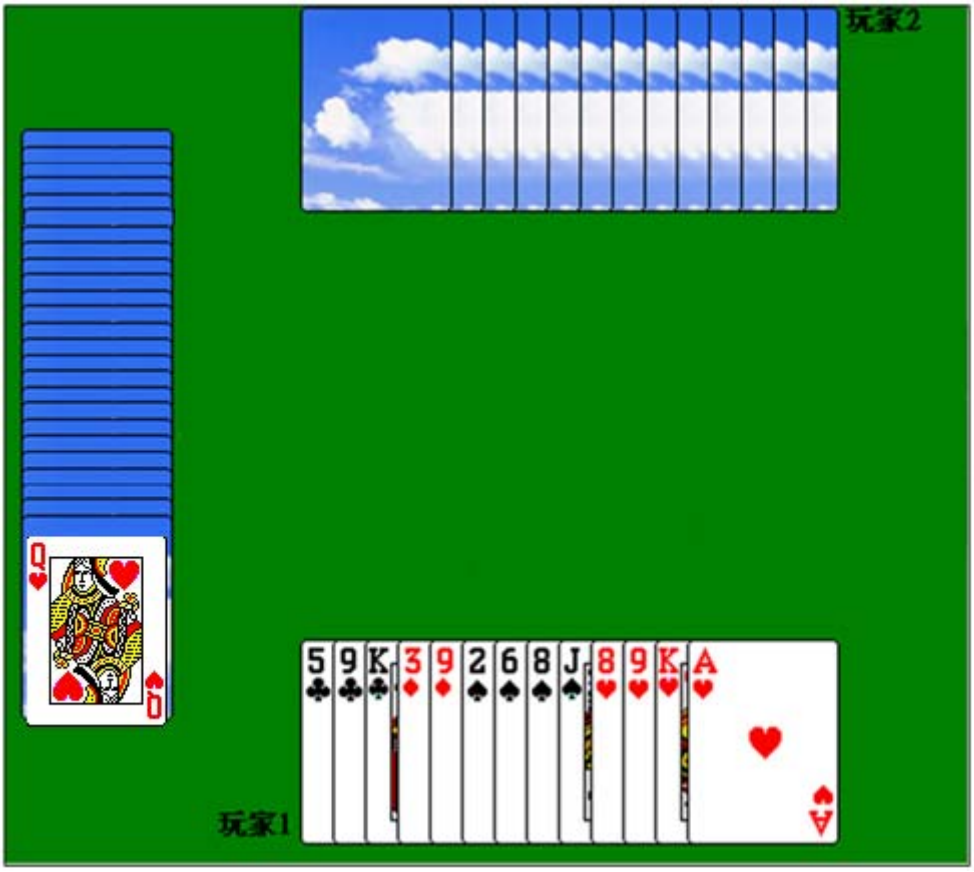


圖 2-3 換牌時的畫面

打牌:

換牌階段結束後，雙方手上各自擁有十三張牌，此時比賽正式開始計分，共有十三回合，由未喊到合約的玩家先出牌，每回合出牌牌型較大者獲得一牌磴，如同換牌階段，牌較大者下一回將會先出牌。十三回合結束時，雙方手牌也出完了，最後計算喊到合約的玩家是否達成勝利條件。

第三節 蜜月橋牌的求解的困難度

蜜月橋牌為不完全資訊的機率性遊戲，玩家一開始能看到的只有自己手上的十三張牌，人類玩家可以依照自己的經驗來判斷牌的好壞程度，但是要讓電腦能了解牌的好壞，所有可能的牌型組合為C_{13}^{52}種，我們不可能一一對所有可能的牌型去定義一個分數，所以我們必須要能設計出一個方法去估計牌的好壞程度。

在換牌階段，扣除最上面一張翻開的牌，在牌桌中間的牌堆，最少有一張，最多會有二十五張牌。假設牌堆中還有 n 張牌，隨著牌局進行，對手有 m 張牌是我們已經知道的(對手換牌時取走翻開的牌)，那麼對手手牌的可能性有 C_{13-m}^{n+13} 種，遊戲愈接近換牌開始階段，牌堆中會有愈多未換掉的牌，我們已知的資訊 m 會愈少，對手的可能的牌組也就愈多，如果要針對每一可能情況做展開，搜尋空間將會太過於龐大，如圖 2-4 所示。

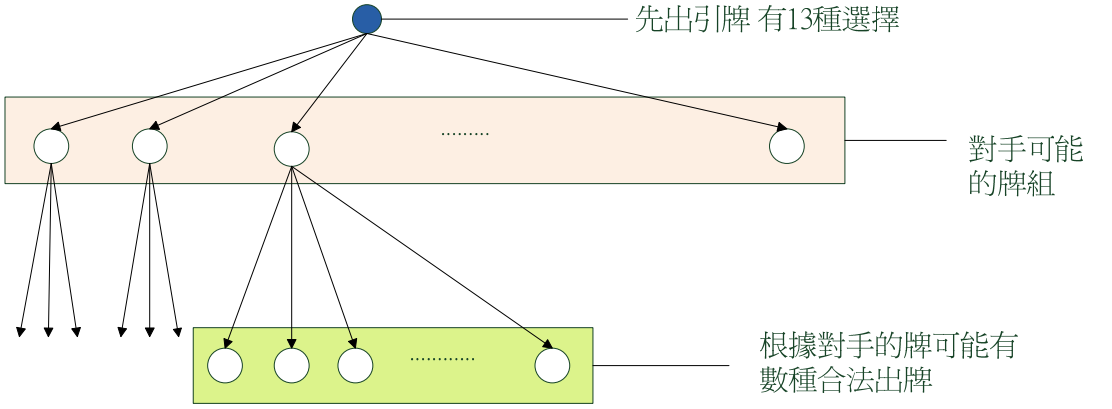


圖 2-4 蜜月橋牌換牌展開示意圖

這一類型的不完全資訊的遊戲賽局由於資訊的不足，因此可能出現的情況非常多，如果要對所有可能情況做搜尋，分支的廣度會非常的大，與一般完全資訊的賽局遊戲不同，完全資訊的賽局遊戲往往是受限於搜尋深度。因此在研究非完全資訊的賽局遊戲時，我認為首先要克服的就是遊戲分支度過高，造成無法在合理的遊戲時間內完成計算。

第 三 章 資料結構及演算法

第 一 節 遊戲樹

本研究蜜月橋牌的資料結構是以 4x13 的二維陣列來記錄每張牌的狀態，如圖 3-1 所示:

♠2	♠3	♠4	♠5	♠6	♠7	♠8	♠9	♠10	♠J	♠Q	♠K	♠A
♥2	♥3	♥4	♥5	♥6	♥7	♥8	♥9	♥10	♥J	♥Q	♥K	♥A
♦2	♦3	♦4	♦5	♦6	♦7	♦8	♦9	♦10	♦J	♦Q	♦K	♦A
♣2	♣3	♣4	♣5	♣6	♣7	♣8	♣9	♣10	♣J	♣Q	♣K	♣A

圖 3-1 蜜月橋牌二維 4 x 13 陣列示意圖

二維陣列運算速度雖然較一維陣列慢，但是使用二維陣列，可以區分出花色，對判斷出牌是否合法以及展開搜尋，可以加快程式的判斷以及方便撰寫。

3.1.1 遊戲樹

在介紹我的演算法之前，必須先介紹一些會用到的技術與方法，在許多的對弈遊戲，在展開或搜尋的過程中往往採用了遊戲樹(Game Tree)的方式呈現，如圖 3-2:

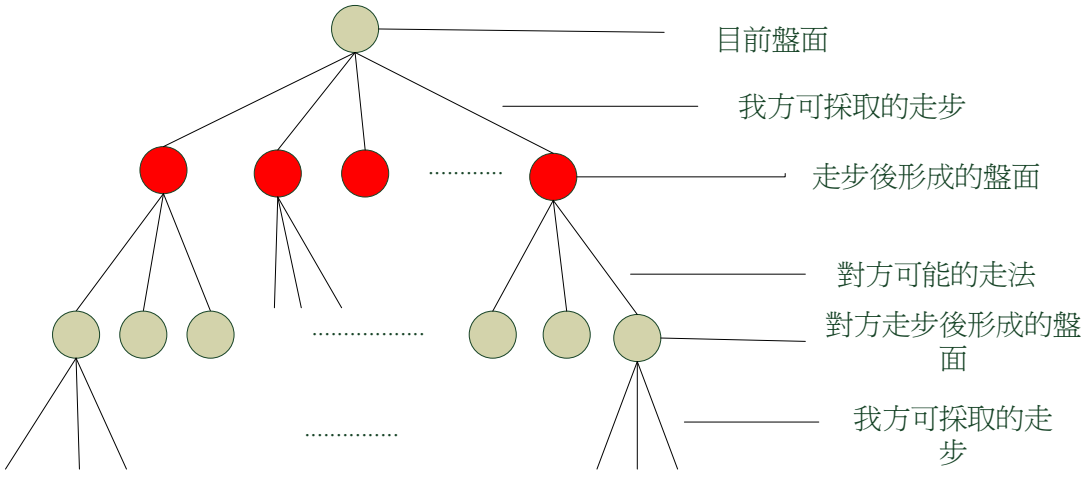


圖 3-2 遊戲樹

遊戲樹的分支度及深度視各種不同的遊戲而有所不同，如果有一種遊戲，雙

方每一回合各有 10 種走步選擇，假如搜尋深度設為 10 回合，就有10^{20}個葉節點，這樣龐大的搜尋量在合理的時間內是無法完成搜尋的，所以必須找出更可行的辦法。在蜜月橋牌這種不完全資訊的遊戲中，每一步都隱含著機率的存在，並不需要找出最佳解，只需要找出近似解即可。當對方的走步存在著一種機率情況，如圖 3-3，蒙地卡羅模擬法(Monte Carlo Simulation)就是一種可能的解決辦法。

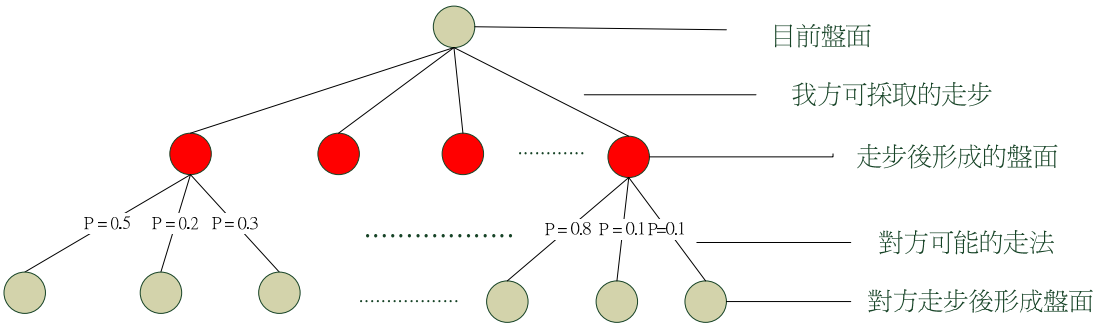


圖 3-3 走步機率不同的遊戲樹

3.1.2 蒙地卡羅模擬法(Monte Carlo Simulation)

蒙地卡羅法是利用統計取樣的方式來解決問題，得到一種近似的答案，基於大數法則的實證方法，當實驗的次數越多，其平均值也就會越趨近於理論值。以下是一種最常見的例子:使用蒙地卡羅模擬法求π。

如圖 3-4 以亂數設定 n 個點，其 x 座標及 y 座標介於 0~1，假設有 m 個點位於扇形($\frac{1}{4}$個圓)內，假如機率是均勻分布，m 跟 n 的比應該接近於扇形及方形的面積比，即$\frac{m}{n} = \frac{\pi}{4}$，所以可得到$\pi$的近似值。

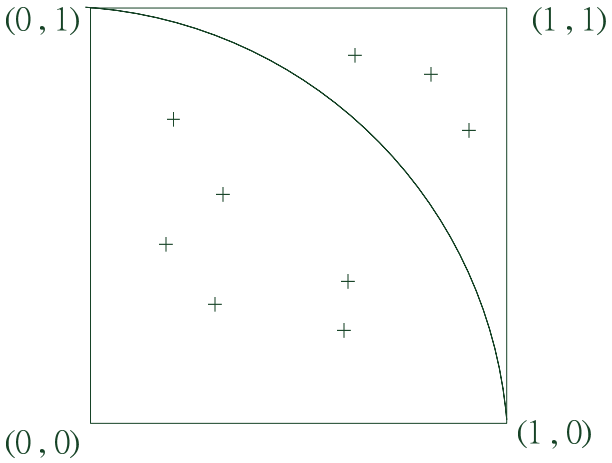


圖 3-4 蒙地卡羅法求π的亂數隨機分布圖

我們將蒙地卡羅法應用在遊戲樹，程式不預設展開路徑，由電腦亂數隨機發牌，這意謂著隨機選取展開路徑，根據蒙地卡羅法的精神，機率愈高的走步將會被展開愈多次，反之，機率愈低的走步將會較少被選擇，甚至沒有被選擇過。使用蒙地卡羅法，計算的是一種「期望值」，計算的是每一個走步的平均分數，而不是可能的最高分數，所以某些機率低的走步被忽略沒有展開計算到，並不會造成分數計算上太大的影響。在某些遊完全資訊的遊戲中，這些忽略可能會是致命的，但在不完全資訊的遊戲中，取樣是一種可以減少展開分支度，但又不失準確度的方法。

第二節 完全展開的搜尋演算法

在前一節中，說明了遊戲樹(Game Tree)及蒙地卡羅模擬法(Monte Carlo Simulation)是怎麼應用在遊戲中。

蒙地卡羅模擬法只能求出近似解或者是期望值，但在某些情況下，我們需要求出一個最佳解，例如:當蜜月橋牌進行到**打牌**的階段，如果玩家有記牌的話，就能知道對手手上的牌，這時候，做完全的展開搜尋顯然可以得到最佳打法。

Min-Max 搜尋演算法是一種假設雙方每一步都會採取最佳策略，進而推演出

最佳走法的一種深度優先的搜尋演算法。

3.2.1 賽局理論(Game Theory)

何謂最佳策略?這一點必須從 1928 年匈裔美籍數學家馮諾曼（Von Neumann, John）(1903-1957）提出的**賽局理論**[4]說起。賽局理論探討兩個經濟個體之間當有利害衝突時，謀略運用之學。賽局理論又稱「博奕理論」(Game Theory)，又可稱為「互動決策理論」，亦即針對一群決策者在決策時，所面臨的問題與戰略行為，所進行的一套有系統的策略式互動行為（interactive behavior）分析工具方法。

賽局理論是一種策略思考，尋求在利害衝突下的最適因應策略，透過策略推估，尋求自己的最大勝算或利益，其關鍵點是「**我的計算必須考慮你的計算，而你的計算也考慮了我的計算**」，更可以進一步推廣到多人賽局中，賽局中每一個人的決策，都必須考慮到其他人的決策，同時也影響著其他人的決策。

假設現在有一個零和的奇偶遊戲，甲乙兩人每次可以從 1 和 2 兩個數字挑一個出來，當兩人的和是奇數，玩家乙必須付給甲 3 元，當兩人的和是偶數 2 時，甲付給乙 2 元，和是 4 時，甲付給乙 4 元。對甲方而言，賽局樹如下:

甲 \ 乙	1	2
1	-2	+3
2	+3	-4

以直覺來看，甲不管出 1 或 2，贏的機率都是 $\frac{1}{2}$ ，但是出 2 輸了卻要輸 4 元，所以想全部都出 1，可以獲得最大的期望值 $\frac{-2+3}{2}$ = 0.5，但這種想法很明顯沒有考慮到乙的計算。乙也知道若以各 $\frac{1}{2}$ 的機率來作選擇，甲出 1 會較有利，所以

乙會改變策略，以較大的機率出 1 來獲得較大的利益。在根據對方的策略調整新策略的同時，對方也會跟著調整策略，反覆下去最後會有怎樣的結果?根據**納許平衡**(Nash equilibrium)[7]，雙方會各自找到一個最佳策略，無論對方如何出牌，都至少會有一最低限度的獲益。

假設甲有 P 的機率出 1，乙出 1 時甲的期望值為-2P + 3(1- P)，乙出 2 時甲的期望值為 3 P - 4(1- P)，為了確保一最低限度的獲益，甲應該讓出 1 及出 2 時的期望值相等。

$$-2P + 3(1- P) = 3 P - 4(1- P)$$

$$3 - 5 P = 7 P - 4$$

$$12 P = 7$$

$$P = \frac{7}{12}$$

當甲以 $\frac{7}{12}$的機率出 1，$\frac{5}{12}$的機率出 2 時，期望值為 -2(7/12)+3(1－5/12)=1/12 為最佳策略。同理，乙也可找到最佳策略，以$\frac{7}{12}$的機率出 1，$\frac{5}{12}$的機率出 2，期望值 －$\frac{1}{12}$ 為最佳，雙方也都各自找到最佳策略達成平衡，在這零和遊戲裡，只要雙方不背離找出的最佳策略，甲每一局平均可以贏乙 $\frac{1}{12}$ 元。

納許平衡，又稱為非合作賽局平衡，是博奕論的一個重要概念，在多人競爭的賽局裡，每個參與者有著相同的共識，基於這種共識，最後的決策結果會達到一種平衡，而「納許均衡」也意味著兩敗俱傷的可能性，沒有一位參賽者能單方面背離該策略而獲利。例如，在互不合作的前提下打價格戰，打到最後，出現一個均衡的結局是「無利潤」，這是「非合作的博弈均衡」，大家互相擠壓、排斥，誰都想獲得最大的利益，但是最後誰都沒有獲得利益，只從利己的角度出發，最後達到的結果卻是損人不利己， **囚犯困境**[3]是解釋那許平衡的經典例子。

囚犯困境由弗拉德（Merrill Flood）提出，這一個模型說明了遊戲雙方可能合作，也可能敵對，雙方各自以自身最大利益考量的前提下，做出決策。

現有合夥的兩個犯人甲、乙遭逮捕，分開偵訊甲、乙並給予兩種選擇:認罪、否認。甲、乙不同的選擇會產生以下的結果:

1. 其中一人認罪，另一人否認，認罪者可得到減刑，判刑 2 年，否認者判刑 10 年。
2. 兩人都認罪，獲得減刑，各判刑 2 年。
3. 兩人都否認，兩人都無罪釋放。

	甲認罪	甲否認
乙認罪	甲乙各判刑 2 年	甲判刑 10 年 乙判刑 2 年
乙否認	甲判刑 2 年 乙判刑 10 年	甲乙都獲得釋放

在這個例子裡，兩人都否認，可為全體帶來最大利益，但是在資訊不明的情況下，認罪可帶來確定的利益(減刑)，而否認卻可能因對方的認罪而加重刑責，在這種共識下，雖然認罪違反了雙方的最大利益，囚犯仍然可能會選擇認罪，造成「雙輸」的局面。

3.2.2 Min-Max Algorithm

在雙人對弈的「零和非合作賽局」中，因為是雙人的遊戲，所以對方的損失可以視為己方的利益，「殺敵一千，自損八百」這種在多人遊戲或者現實商場中不被考慮的策略，在雙人賽局中卻是可以被考慮的策略。不存在合作的關係，雙方的最佳策略的考量將會變成，己方的收益加上對方的損失，透過審局函數計算，在輪到我方時，會從所有候選策略中選出利益「最大」的策略，輪到對手時，對手也會選擇對他利益最大的策略，對手的最大利益也可以看成我方的「最小」利益，這也是 Min-Max 搜尋演算法[10]的原理。

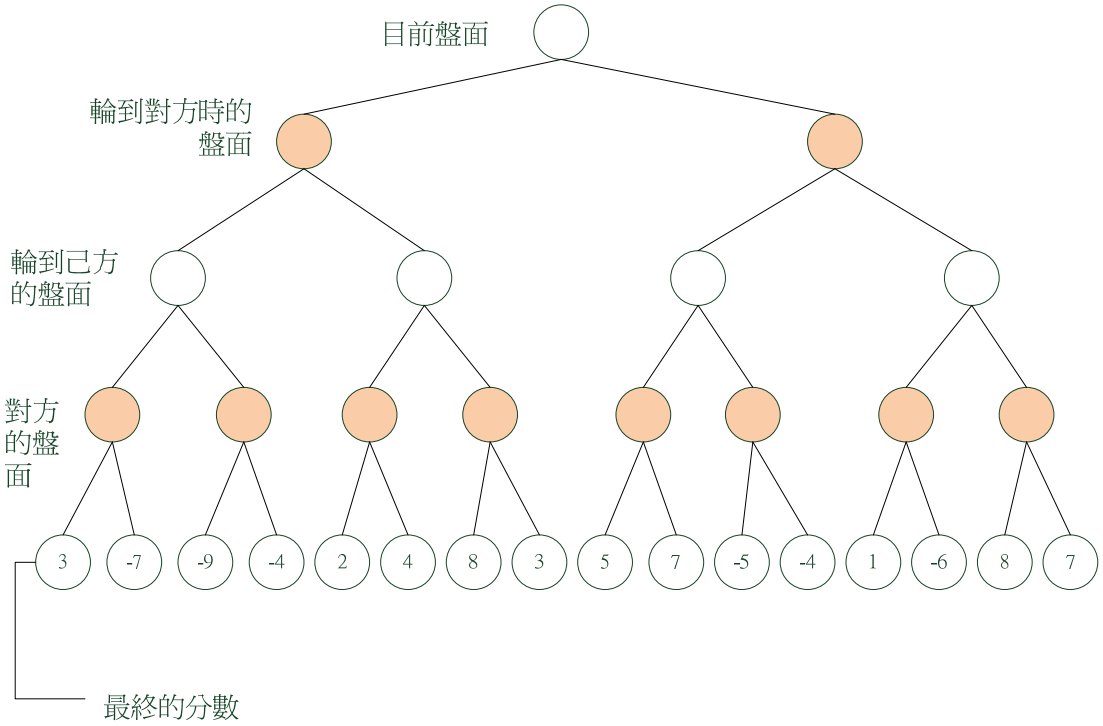


圖 3-5 Min-Max 演算法示意圖 1

圖 3-5 和 3-6 是雙方的決策遊戲樹，每一步雙方各有兩個選擇，根據 Min-Max 演算法來考慮，在我方的盤面，我們應該選擇分數最高的路徑，而輪到對方選擇時，我們假設對方會選擇對方分數最高(我方分數最低)的路徑，也就是假設雙方都**以自己最大利益做考量**，而達到納許平衡的情況，在雙方都不違反此原則下，這就是雙方的最佳策略。

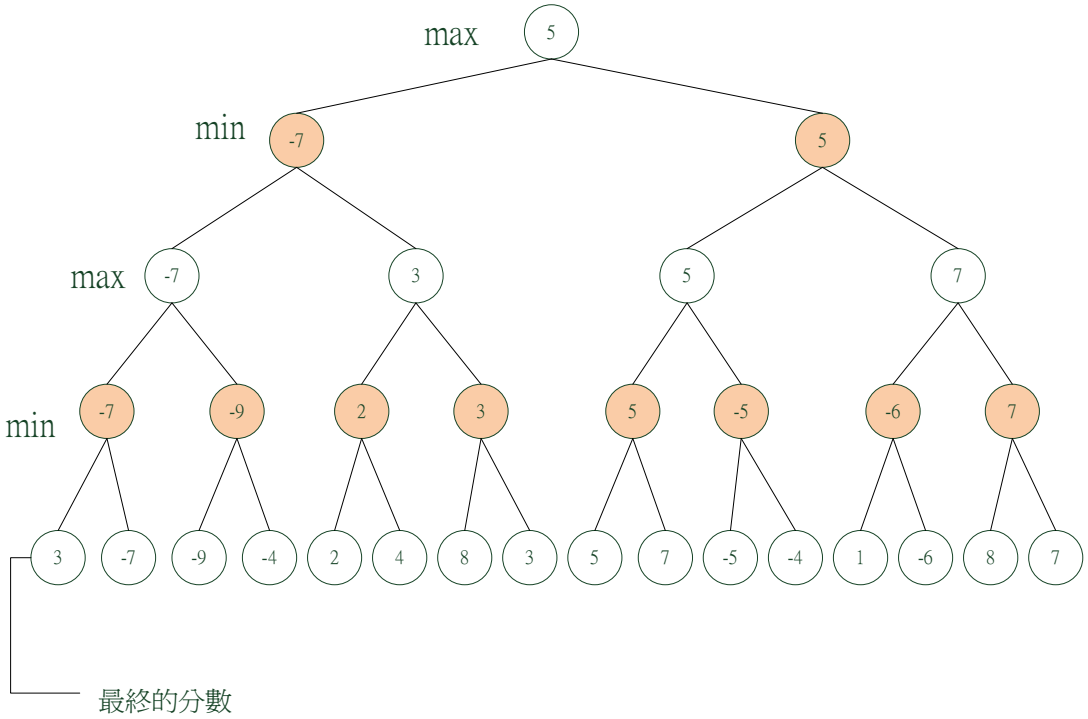


圖 3-6 Min-Max 演算法示意圖 2

第三節 叫牌演算法及資料結構

3.3.1 資料結構的設計

觀察蜜月橋牌的遊戲方式，可以發現，在進行**打牌**時，影響 Min-Max 演算法展開結果的是牌的「順序」，牌的大小差距並不會影響最後的結果，在蜜月橋牌的規則中只有分大或小，例如當某一磴當中，玩家 B 出了 Ace 去吃玩家 A 的 5，或者出了 6 去吃 5，結果都是一樣獲得一磴的積分，並不會因為出的牌特別地大或小有加分或減分，所以針對這種特性我們設計了一種只記錄順序的資料結構，如圖 3-6 及圖 3-7。

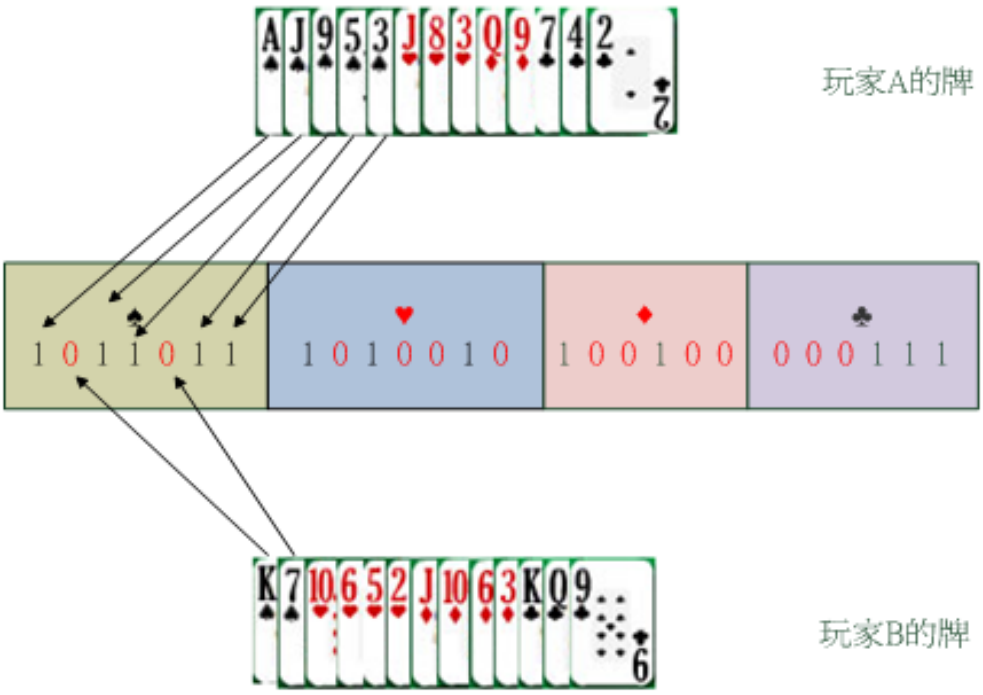


圖 3-7 玩家手牌排序圖 1

在圖 3-7 中，我們以花色作分隔，將每個花色作排序。以黑桃花色為例子，在雙方的手牌中，黑桃由大到小分別為 A、K、J、9、7、5、3，持有的玩家分別為 A、B、A、A、B、A、A，我們以 1 代表玩家 A，0 代表玩家 B，可以得到一組 26 bits 表示的玩家手牌的排序。但是只有這樣還不夠，我們需要更明確的指出花色的分布情況，也就是我們必須記錄各種花色各有幾張。在這個例子中，黑桃有 7 張，紅心有 7 張，方塊及梅花各有 6 張。由於每個花色最多不會超過 13 張，所以如下圖每個花色只需 4 bits 即可以記錄所有情況。

花色 1 張數	花色 2 張數	花色 3 張數	花色 4 張數
Bit12~15	Bit8~11	Bit4~7	Bit0~3

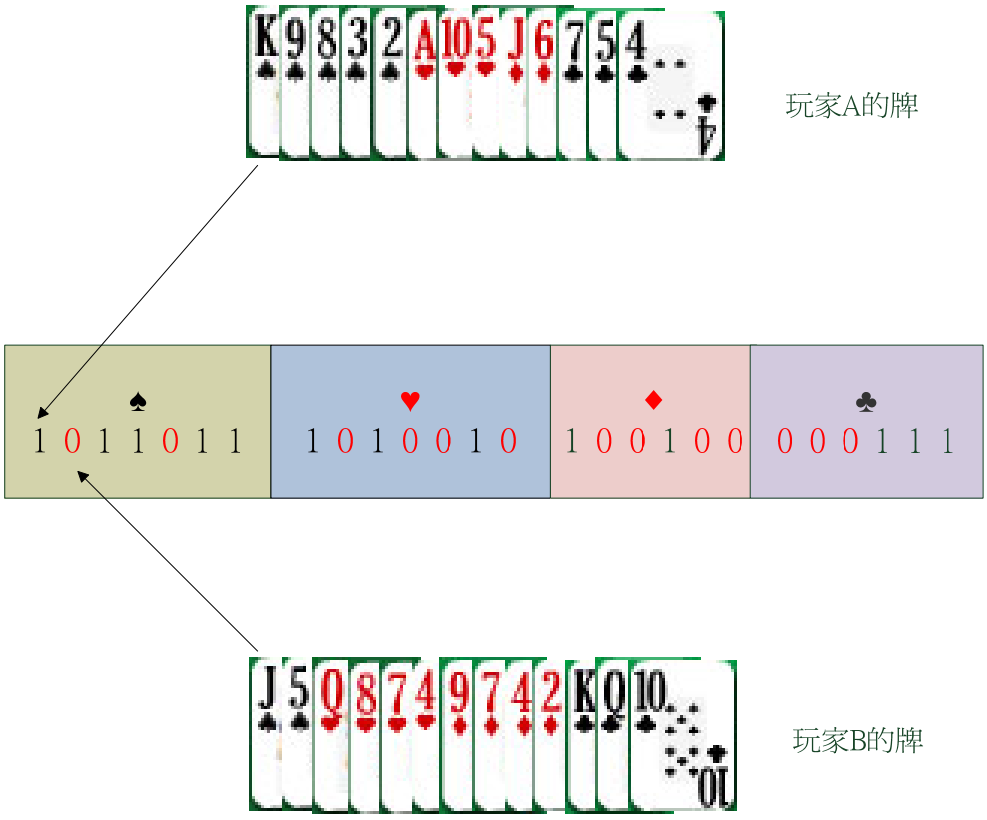


圖 3-8 玩家手牌排序圖 2

在圖 3-7 及圖 3-8 的兩副牌中，發出的手牌不同，但經過花色排序後得到了相同的排序結果和花色分佈，因此經過 Min-Max 演算法的展開搜尋後，這兩副牌最後的得分會相同。在這種資料結構中，只需要兩個整數(int)大小的參數，就能藉由程式得到打牌的結果。

3.3.2 演算法設計

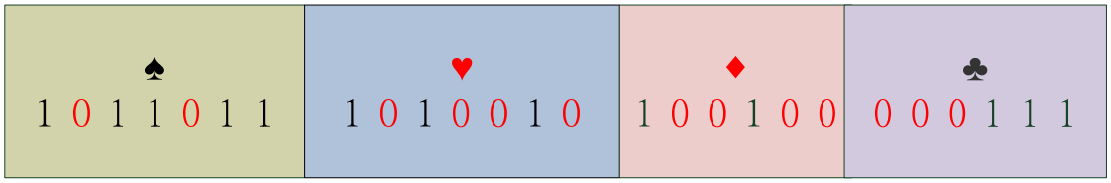
在第二章曾經提過，叫牌的難度在於如何讓電腦能分辨牌的好壞價值，在這裡我提出一個評估的方法。在一般情況下，基於蜜月橋牌的遊戲玩法，雙方玩家一開始的手牌經過**換牌**的步驟後，雙方手牌的好壞對比會與原來相差不多，所以玩家才能由一開始的手牌來進行**叫牌**的動作。於是當我的程式拿到了一開始的十

三張牌，會採用蒙地卡羅模擬法(Monte Carlo Simulation)使用亂數隨機發給對方十三張牌，雙方直接以這些牌模擬**打牌**的步驟，以 Min-Max 演算法展開並記錄結果，經過大量的模擬後，將所有結果取平均，可以得到一個分數，程式會以這些分數當基礎來做**叫牌**的評估依據。

延續上一小節，每一種排序及花色分佈經過 Min-Max 法展開搜尋而得到一個相對應的得分，我們可以將這些排序、花色分佈及得分預先計算好，建立一個資料庫，採用 hash table 的方式，在程式開始時載入記憶體，這樣就可以節省遊戲進行中展開計算所花的時間，節省下來的時間可以增加模擬次數以增加準確度。

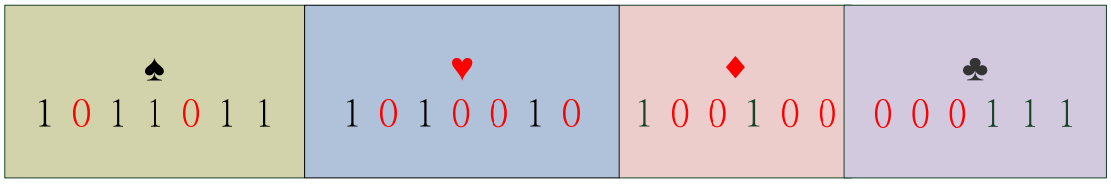
3.3.3 資料庫的建立:

在圖 3-8 中我們得到了排序如下:

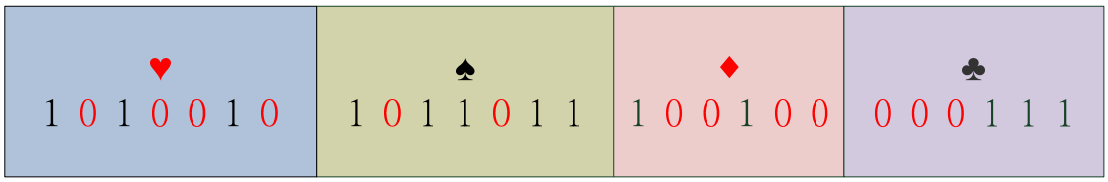


在蜜月橋牌中，有**王牌**這一個花色的存在，同樣的一局牌，以不同的花色作王牌來比賽，結果也會不同，所以我們還需額外指出王牌是哪一種花色，為了解決這個問題，我將王牌放在第一組花色的位置，根據王牌的不同，排序也會隨著改變，所以每一局牌，會計算分別以四種花色為王牌的四個分數。

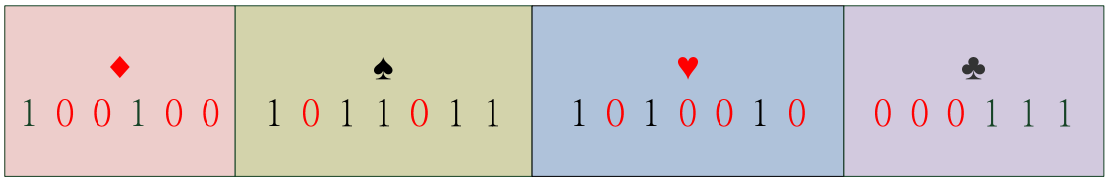
以黑桃為王牌:



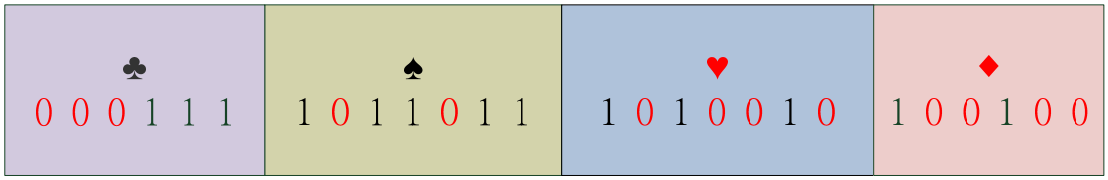
以紅心為王牌:



以方塊為王牌:



以梅花為王牌:



經過上面的排序，我們根據每個序列的分數，可以很明顯地看出選擇哪一種花色作為王牌較為有利，前提是我們必須能快速的得知這些序列的分數，所以這些序列分數資料必須事先計算好並載入記憶體才能夠快速存取，這也限制了資料庫的大小，為了減少一些相同情況的資料重覆載入記憶體，我又對序列作了一些調整，下面會進一步解釋說明。

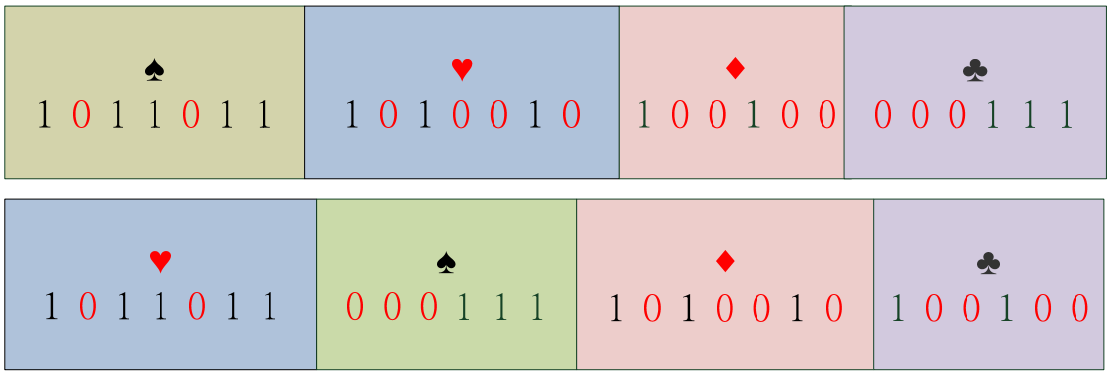
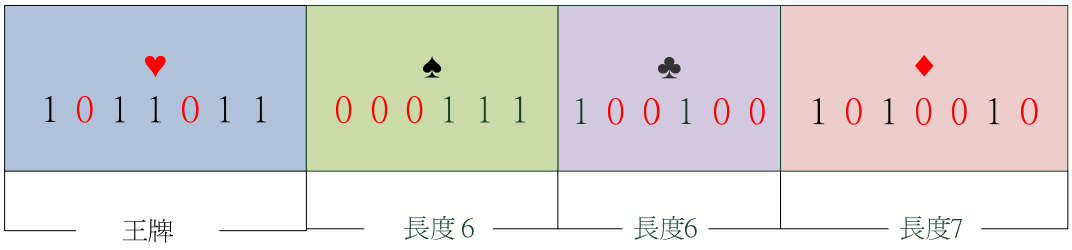


圖 3-9 序列比較圖

圖 3-9 的兩組序列，在王牌部分的順序都是 1011011，不同的只是花色，一副是以黑桃為王牌，另一副則是以紅心為王牌，在其他三個花色則分別由 1010010、100100、000111 三組序列組成，也就是說這兩組序列的不同只有花色分布的情況而已，兩組序列最後的得分應該會相同，我們在建立資料庫時應該把他們視為同一種序列。在這裡，我在建立資料庫時，對非王牌的三個花色排列順序作了調整，依照花色序列的長短(各花色的張數)來決定順序，由短到長排列，如下所示:



雖然還是會有相同長度造成的一些重複情況，但已經大幅的減少資料庫的儲存盤面數量。雖然如此，但在實做的過程中，發現資料量還是太過龐大，無法全部載入記憶體，必須改變做法。完全的 Min-Max 法展開計算時間太長，完整的資料庫又太佔空間，於是我採用了折衷的做法，依靠展開並搭配部分的資料庫來完成，如圖 3-10。

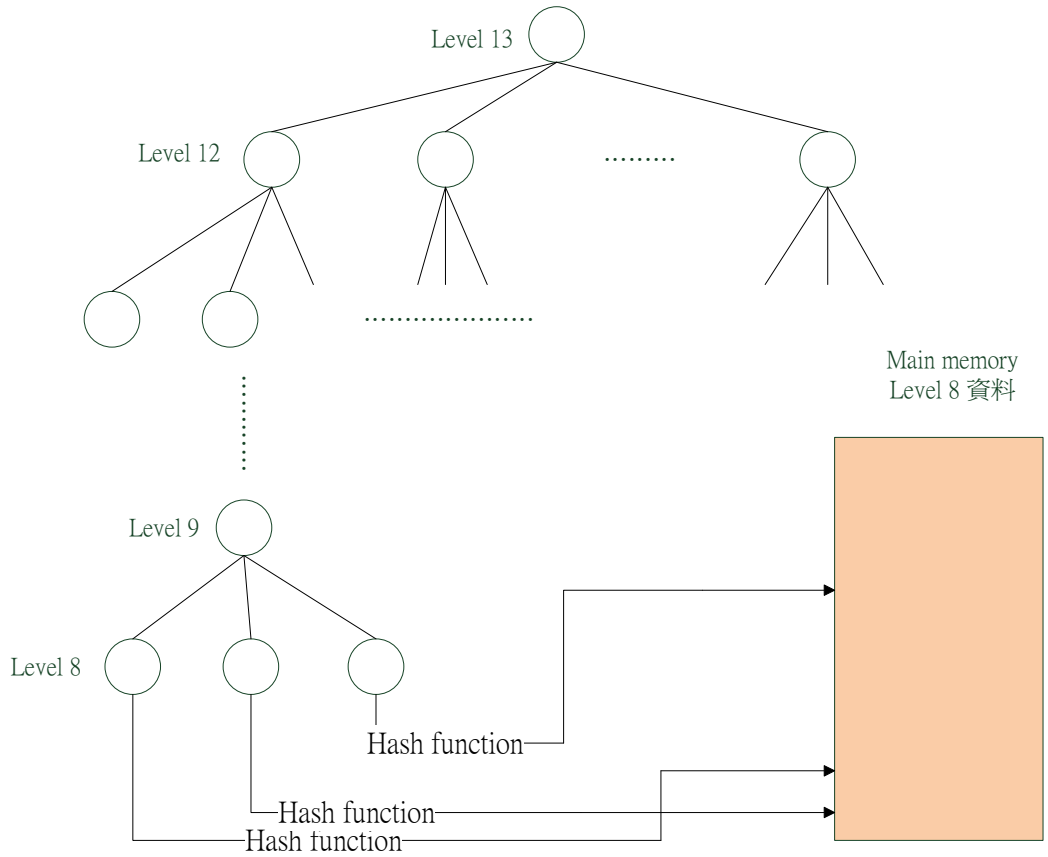


圖 3-10 Min-Max 演算法與資料庫架構圖

這邊我基於實作上的考量，最後選擇八層的資料庫(雙方都只剩八張牌)，在第八層以上使用 Min-Max 演算法作展開，展開到第八層再從記憶體讀取結果。

資料庫大小:

牌的張數	1	2	3	4	5	6	7	8
檔案大小	88B	748B	5.36KB	35.3KB	213KB	1.24MB	6.97MB	37.8MB

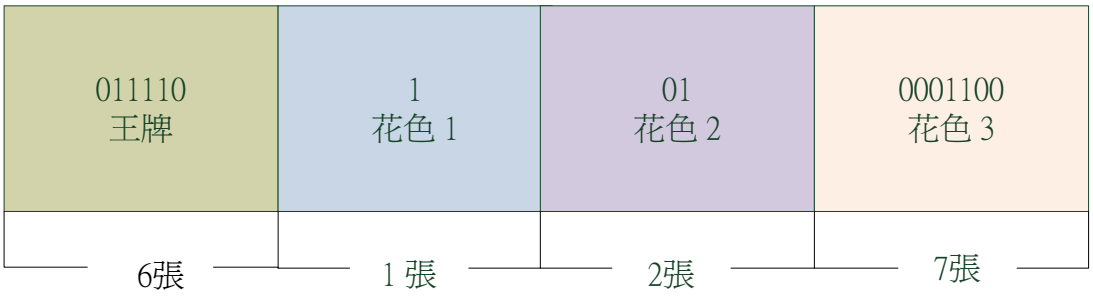
資料庫格式:

上圖是第八層資料庫的一部分，以圈選的部分為例做說明:

31372 表示雙方牌的大小順序，轉成二進制為 0111101010001100(請看下圖)，這是雙方各有八張牌的混合排序，1 表示這回合有出牌權的一方，0 是跟牌方的牌。

1、2、7 表是非王牌的各花色有幾張牌。此例王牌的張數有 16-1-2-7 = 6 張。

4 表示先出牌的玩家經過 Min-Max 法展開後能獲得的磴數為 4。



資料庫建立過程:

不同的兩副牌，在展開的過程中，卻可能產生相同盤面的子節點，如圖 3-11 所示。如果我們直接對每一種牌型做展開，其子節點盤面可能在許多不同副牌型都重新被展開，為了避免這種情形，資料庫必須由底層往上建立，將每一層所有可能結果都儲存起來，再由這些結果建立更高層的資料庫。

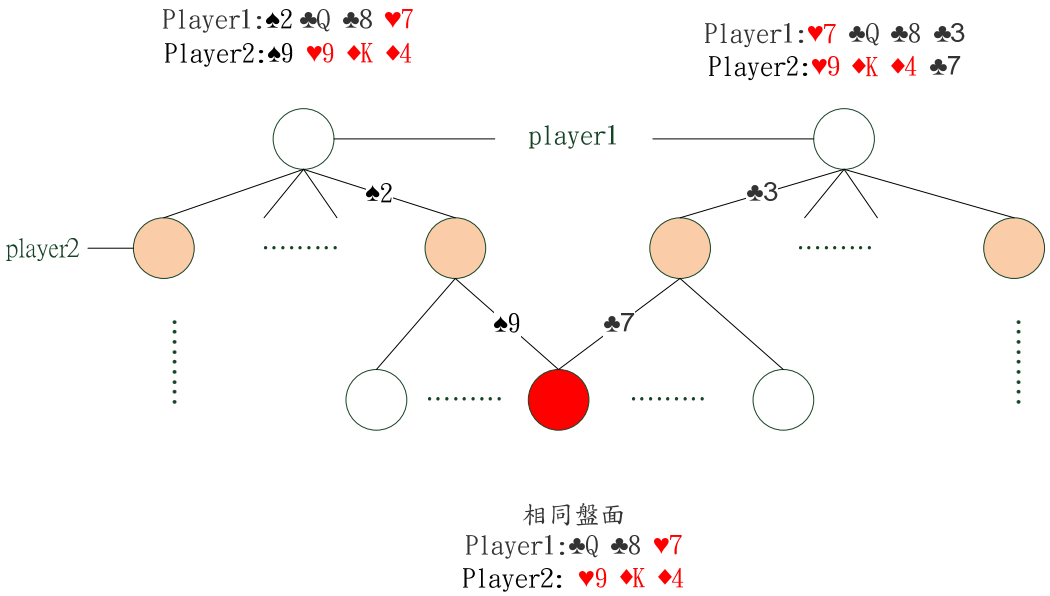


圖 3-11 相同盤面

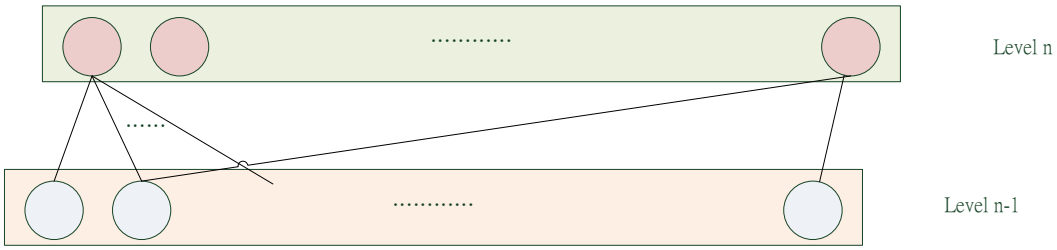


圖 3-12 資料庫建立過程示意圖

在打牌的過程中，隨著遊戲進行，玩家每回合的手牌張數都會減少一張，所以每一層展開的子節點盤面一定都存在下一層的資料庫中，如圖 3-12。例如，假設玩家目前手中還有六張牌，要選擇某一張來打，而我們已經有了所有五張牌時可能的情況及分數，此時我們只要對手中的六張牌做一層的展開搜尋就能知道應該要打哪一張。每一子節點盤面只需要一次的展開計算儲存後，就能被多次的讀取使用。

3.3.4 叫牌模擬過程


玩家喊 1 黑桃
5.52
9.7
9.04
6.4
7
8
8
8
Com : 2 heart
玩家喊 2 黑桃
5.38
10.22
9.12
7.18
6
9
9
9
Com : 3 heart
請玩家叫牌
3 0
玩家喊 3 黑桃
5.42
10.56
10.02
7.72
5
10
10
10
Com : 4 heart
請玩家叫牌
0
玩家pass

圖 3-13 叫牌模擬畫面

程式執行時，玩家的手牌會顯示在執行視窗中，電腦方的手牌會被記錄在文件檔裡，以上為程式執行畫面，每次輪到電腦方都亂數模擬五十副牌，在綠色框框中代表模擬計算出電腦方四個花色的分數，紅色框框中表示四個花色獲勝需贏得的磴數，在真正的比賽時玩家只能看到自己的手牌及對方的叫牌，其餘資訊則不會被玩家看到，為了說明程式流程，因此顯示在畫面上。

第一次模擬如下表：以紅心的模擬分數和獲勝需要磴數差距最大，因此選擇叫紅心為王牌，也就是叫 1 紅心。

	模擬分數	獲勝需要磴數	差距
黑桃	5.6	7	-1.4
紅心	9.26	7	2.26
方塊	8.16	7	1.16
梅花	6.8	7	-0.2

如圖 3-13 所示，這時人類玩家喊 1 黑桃。

第二次模擬如下表：以紅心的模擬分數和獲勝需要磴數差距最大，選擇叫紅心，也就是 2 紅心。

	模擬分數	獲勝需要磴數	差距
黑桃	5.52	7	-1.48
紅心	9.7	8	1.7
方塊	9.04	8	1.04
梅花	6.4	8	-1.6

如圖 3-13 所示，這時人類玩家喊 2 黑桃。

第三次模擬: 以紅心的模擬分數和獲勝需要磴數差距最大，選擇叫紅心為王牌，也就是叫 3 紅心。

	模擬分數	獲勝需要磴數	差距
黑桃	5.38	6	-0.62
紅心	10.22	9	1.22
方塊	9.12	9	0.12
梅花	7.18	9	-1.82

如圖 3-13 所示，這時人類玩家喊 3 黑桃。

第三次模擬: 以紅心的模擬分數和獲勝需要磴數差距最大，依然選擇叫紅心為王牌，也就是叫 4 紅心。但是在這一次模擬中，黑桃跟紅心的最後得分差距已經很接近，也就是電腦認為叫牌跟不叫牌的勝率是差不多的。

	模擬分數	獲勝需要磴數	差距
黑桃	5.42	5	0.42
紅心	10.56	10	0.56
方塊	10.02	10	0.02
梅花	7.72	10	-2.28

如圖 3-13 所示，這時人類玩家放棄叫牌。

在蜜月橋牌的叫牌過程中，玩家在叫牌的同時也透露了玩家手牌中各個花色的強弱，要做出正確的模擬，就必須把握住這些資訊。在一開始叫牌時，因為沒有這些資訊，所以模擬時只用亂數發出對手的牌，但是在競叫的過程中，當對手叫了一個花色，而模擬時卻仍然從剩下的 39 張牌隨機發給對手，這樣是不合理的，所以我們要對模擬作一些調整。在我的調整方法裡考慮的因素有兩個：(一)對手叫的線位，叫的線位愈高意謂著獲勝條件愈困難，也需要更好的手牌來支持

這個契約，從另一個角度看，對方獲勝條件高就是我方獲勝條件低，這也增加了我方 pass 的可能性，在一般情況下，玩家叫的線位愈高，其可信度也愈高。在叫牌剛開始時，有些玩家可能會隨便叫牌欺騙對手，隨著線位愈高，欺騙的可能性也愈低。考慮這些之後，在做模擬時，對手在其叫的花色會有更高的機率拿到好牌，隨著線位愈高，這些條件會愈明顯，對手叫的花色模擬分數通常會降低，其他花色會略微升高。(二)自己的手牌，撲克牌的牌組是固定的，所以在某個花色當一方拿到好牌時，另一方就會相對地弱勢，所以當己方手牌較弱時，對方的喊牌可信度較高，牌也可能愈好，反之，若己方手牌較強時，即使對方叫的線位較高，能拿到好牌的機會也較小，在模擬時線位的修正效果也會較小。結合以上兩點，這些就是我在叫牌部分的想法。

叫牌程式虛擬碼如下:

```
int bid () {
   int card[4][13];  //紀錄 52 張牌中哪些是自己的牌
   float score[4];      //記錄四個花色的得分
   enum flower {spade,heart,diamond,club};
   for (i = 0;i < simulation_times; i++) { //設定模擬次數
       random_card();     //隨機猜測對手的牌
     /*分別以四個花色為王牌做展開計分*/
       score[spade] += minmax(spade);
       score[heart] += minmax(heart);
       score[diamond] += minmax(diamond);
       score[club] += minmax(club);
       }
    for (i = 0; I < 4; i++) score[i] /= score[i]; //取平均分數
```

```
    make_choice(score);     //根據目前盤面及模擬分數做叫牌選擇
}
```

在上述程式碼中，一開始我們需要在記錄撲克牌的 4×13 二維陣列(card[4][13])中註記其中哪些是我方的牌，每次模擬時在剩下的牌中隨機發給對手，但是這裡所謂的隨機並不是完全的隨機，必須參考對手的叫牌來作修正，發完牌之後分別以四個花色為王牌來計分。當模擬次數達到程式的設定次數(simulation_times)後將各花色每次的模擬分數(score)取平均，審局函數會再根據這些分數做出叫牌或者不叫牌的判斷。

第四節 換牌演算法

3.4.1 換牌策略

換牌階段每回合都會有一張翻開的牌在檯面上(參照圖 2-3)，視雙方玩家出牌大小決定牌的歸屬，牌較大者可以拿走翻開的牌並且下一回合優先出牌，較小者拿走下一張牌。那我們要如何決定我們要出大牌還是小牌?假如我們每回合使用審局函數，除了計算翻開的牌的價值外，還要考慮接下來先出有利還是後出有利，綜合之後再決定要出大還是小。但是，我們如何知道下回合是先出有利還是後出有利，那還要再看下下一回合的需要來評估，這就產生了一個問題，假如我們第一回合最佳策略路徑需要考慮第二回合的需要才能決定，而第二回合又需要考慮第三回合的需要才能決定，……，以此類推，當我們在做決定時都必須考慮到往後整個遊戲樹的發展，這樣龐大的分支度很難去做模擬及搜尋，所以我將每一個回合作單獨的考慮，以貪婪演算法(Greedy algorithm)選取每回合各自的最佳策略。

貪婪演算法的精神如下:在每一步面臨抉擇時，都選擇目前最有利的選擇(局部最佳解)，不考慮將來的影響，並期望最後能得到全域的最佳解。但是最大的問題在於，每一步的最大利益，總合起來會是整個問題的最佳解嗎?事實上，大

部分時候, 眼前的利益與長遠的利益並不相等，例如 0/1 背包問題; 但有少數問題可以經由證明顯示 「每一步都選眼前最有利的部分解答, 則最後所得的結果正是全盤考慮下最有利的解答」, 如霍夫曼碼(Huffman code)的編碼方法。

0/1 背包問題:

背包問題是一種組合優化問題。它的名字來源在於給定一組物品時，如何選擇最合適的物品放置於給定背包中。相似問題經常出現在商業、組合數學，計算複雜性理論、密碼學和應用數學等領域中。

問題描述:有一小偷跑到一間商店偷東西，店裡有 n 個商品，第 i 個商品價值以 V_i 表示，重量以 W_i 表示，小偷的背包最多只能裝總重量 C，每一項商品都不能切割，只能選擇拿或不拿(1 或 0)，小偷應該怎麼取才會有最大利益?以數學式表示如下:

$$\sum_{i=0}^{n-1} W_i X_i \le C \quad , \quad X_i \in \{0,1\} \quad , \quad 0 \le i \le n-1 \quad ,$$

求 $\sum_{i=0}^{n-1} V_i X_i$ 的最大值 。

這個問題以動態規劃來解可以獲得最佳解，但是如果以貪婪演算法來做取捨，有兩種方法，小偷在背包還沒滿的時候，(1)每次都取最有價值(V_i最大)的商品，或者(2)每次都取單位價值最高 ($\frac{V_i}{W_i}$ 最大)的商品，直到背包裝不下其餘的物品為止，但是這兩種作法都不一定會得到最佳解。

例:現在有 4 樣商品 A,B,C,D 價值分別為 25,23,15,10，重量分別為 20,15,10,5，背包的最大載重量為 25。在這個例子裡，以第一種方法來取，第一次會先取 A，第二次只能取 D，總價值為 25+10=35，以第二種方法來取，第一次會取 D，第二次會取 B，總價值為 23+5=28。但是最好的選擇應該是選 B 和 C，總價值為 23+15=38 為最高。

雖然貪婪演算法對 0/1 背包問題不保證可以得到最佳解，但它卻是利用直覺方式計算出非常接近最佳解的答案。據研究，以貪婪演算法解 0/1 背包問題，大約有 40%的機率會獲得最佳解，97%的機率會得到與最佳解相差 10%利益的結果。

霍夫曼碼:

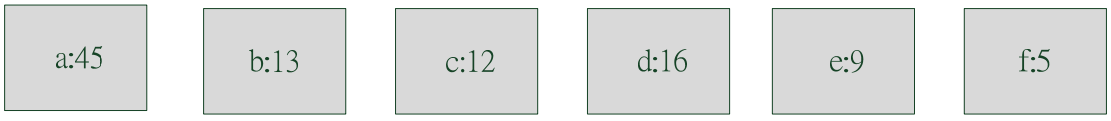
霍夫曼碼是一種具有可恢復性(沒有失真)的壓縮技術，以每個字元出現次數的多寡作為壓縮的依據，以不固定字元長度來重新編碼，給予較常出現字元較短的編碼，給予較少出現的字元較長的編碼，其儲存壓縮率大約介於 20%~90%。

霍夫曼碼是以建立霍夫曼二元樹的方式得到，每次選擇出現次數最低的字元做為新節點的左邊和右邊的子節點，並以兩個子節點的次數相加做為新節點的次數，重複同樣步驟，直到所有節點組合為一棵二元樹，如下圖所示。

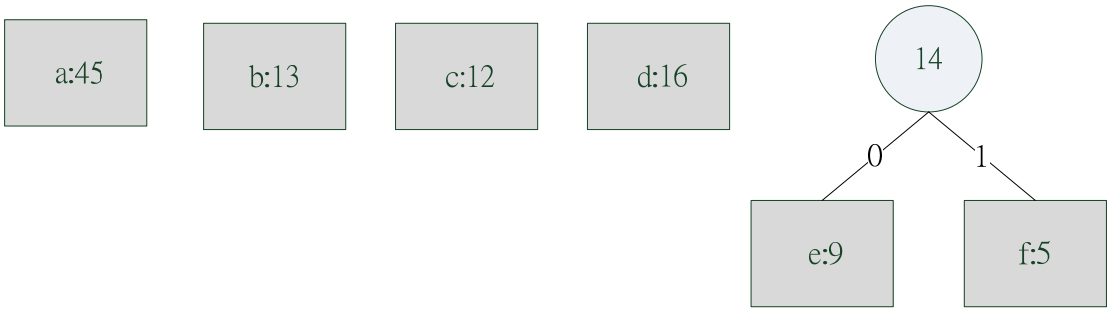
例:

	a	b	c	d	e	f
出現次數(千)	45	13	12	16	9	5
固定長度字碼	000	001	010	011	100	101
可變長度字碼	1	000	001	010	0110	0111

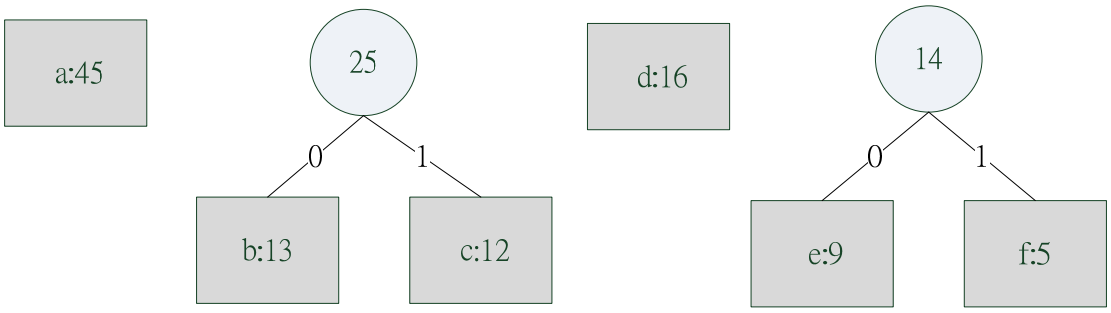
(1)



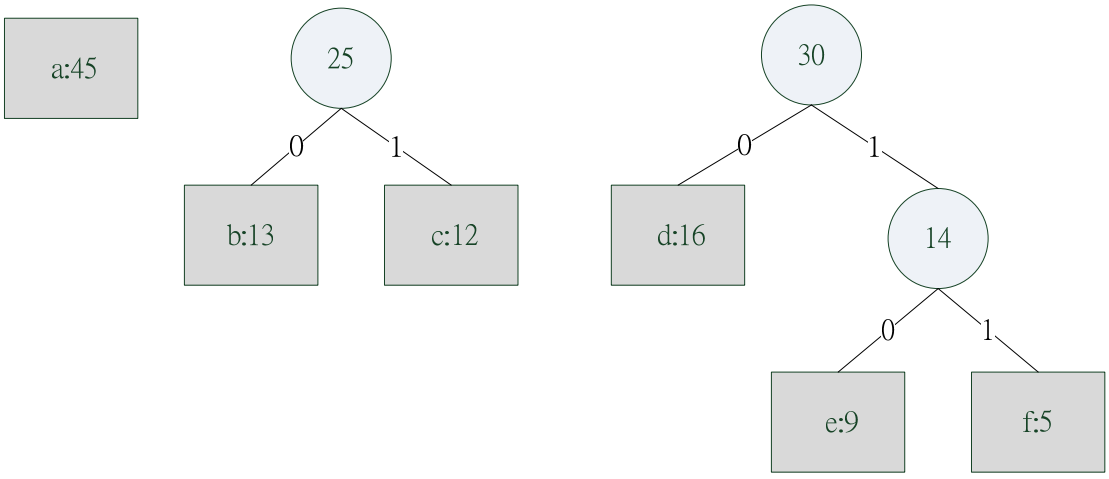
(2)


a:45
b:13
c:12
d:16
14
0
1
e:9
f:5

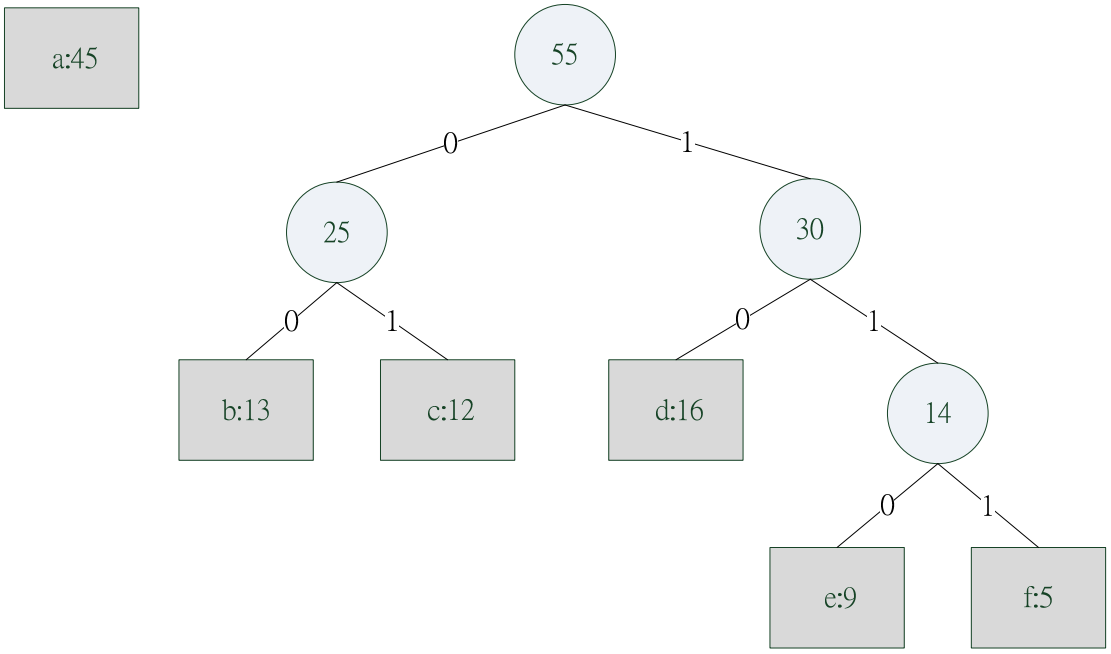
(3)


a:45
25
0
1
b:13
c:12
d:16
14
0
1
e:9
f:5

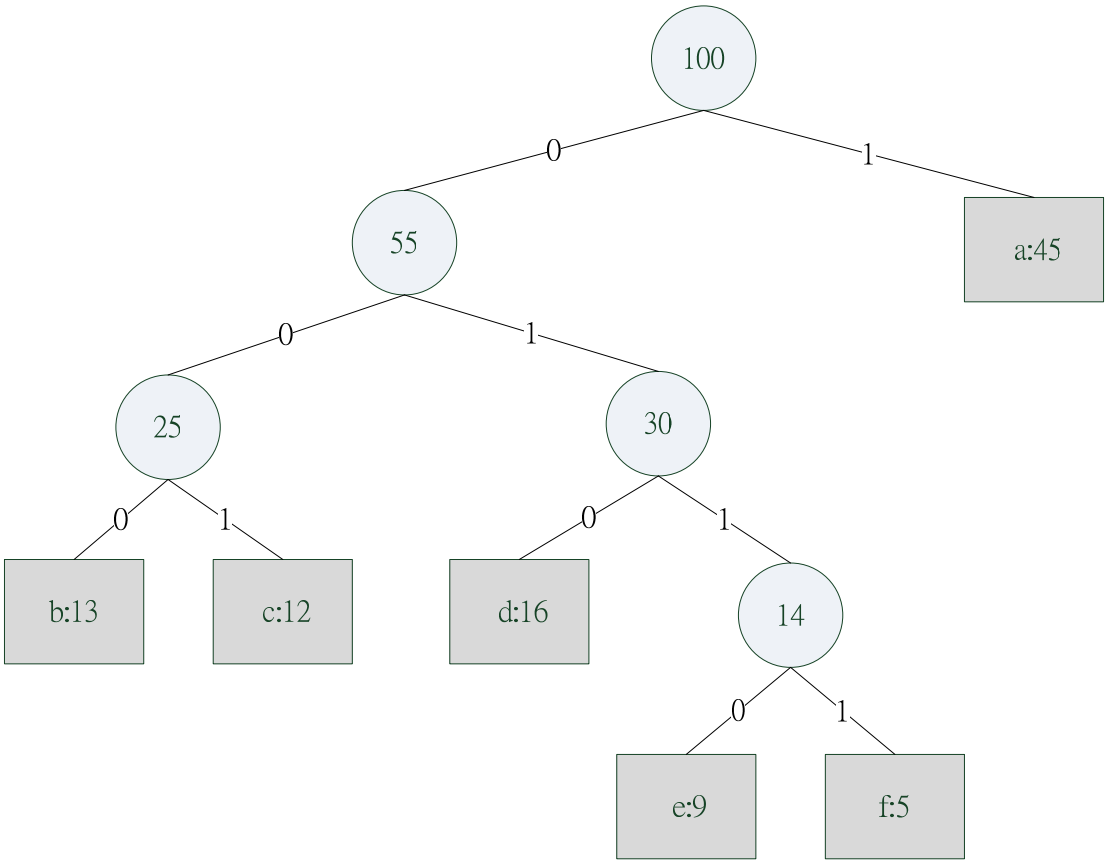
(4)


a:45
25
0
1
b:13
c:12
30
0
1
d:16
14
0
1
e:9
f:5

(5)



(6)



經過霍夫曼編碼後可得到新的字碼如下:

字元	a	b	c	d	e	f
編碼	1	000	001	010	0110	0111

在一般的遊戲賽局中，如象棋、圍棋，玩家每下一手棋後，盤面通常都會變得對自己比較有利，當對方回了一手棋，盤面局勢又會向對方傾斜一點，遊戲就是在這種互相拉鋸的情況下進行，這在電腦程式進行搜尋時會產水平效應(Horizontal Effect)，程式搜尋深度如果是固定的，搜尋的葉節點剛好都是其中一方玩家下完，那麼傳回的盤面的分數就會對其中一方較有利而不夠準確，也許輪另一玩家下完，分數會出現很大的改變。

但是在蜜月橋牌換牌時，水平效應並不明顯，因為在換牌時先出牌者未必有利，後出者也未必不利，先出牌的一方雖然可以視翻開的牌從自己的牌中任意選一張牌出，但這裡可以分成兩種情況，翻開的牌有價值或沒有價值。當翻開的牌有價值時，先出牌的玩家 A 會想要拿下這張牌，玩家 A 會選擇較大或較有可能獲勝的牌出，如果玩家 B 沒有更大的牌，會出一張沒有價值的小牌並且可以拿到一張未知的牌，玩家 A 付出一張大牌換來一張大牌，玩家 B 丟掉一張沒用的牌拿到一張未知的牌，雙方的差距並沒有太大的改變。假如翻開的牌沒有價值，玩家 A 會盡量丟一張小牌讓玩家 B 獲勝拿到這張沒價值的牌，這樣的結果是玩家 A 和玩家 B 都出掉一張小牌，玩家 A 獲得一張未知的牌，玩家 B 拿到一張沒價值的牌以及下回合先出牌的權利，所以雙方的差距也並不會有太大的改變。

經過上面的分析，可以看出遊戲中有很重的機率成分，水平效應的影響也不顯著，在這裡我認為貪婪演算法是較為直覺且有效率的辦法。

3.4.2 審局函數

選擇使用貪婪演算法來制定換牌的策略，會選擇眼前的最大利益，因此我們還需要審局函數將盤面轉為實際的分數，一般來說，審局函數需滿足兩個要求。

(一)　準確:審局函數必須能判斷一個盤面的好壞程度，愈準確的審局函數回傳的盤面分數也愈接近真實，根據這些分數程式愈不容易發生失誤。

(二)　通用性:一個好的審局函數必須能適用所有的盤面，不會在某些盤面做出錯誤判斷。

審局函數可分為兩種 : 靜態審局函數及動態審局函數。靜態審局函數是指在遊戲對奕的過程中，使用固定的參數或牌張價值來做盤面評估，在程式運作時較快且較易撰寫程式。動態審局函數，是指在遊戲過程中，會隨著盤面不同以及回合不同(開局或接近終盤)修正參數或牌張價值，這種作法較能貼近真實的盤面，但程式較為繁雜且需要驗證其正確性。為了程式方便撰寫，這裡採用靜態牌張價值評分，下面是各牌張的價值分數，這邊的分數是由我個人主觀定義出來的，當玩家每打出一張牌就會減去其相對應的分數。

王牌花色	非王牌花色
Ace : 40	Ace : 25
K : 35	K : 20
Q : 30	Q : 15
J : 26	J : 11
10 : 24	10 : 9
9 : 22	9 : 7
8 : 20	8 : 5
7 : 18	7 : 3
6 : 18	6 : 3
5 : 18.5	5 : 3.5
4 : 20.5	4 : 5.5

3　：22.5　　3　：7.5

2　：24.5　　2　：9.5

3.4.3 換牌情況分析

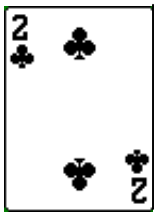
我以 10 分來當作牌張價值分數的分界，分數小於 10 的牌視為不想要的牌，根據翻開的牌分數(大於 10 或小於 10)，以及先後手的不同，程式共分四種不同情況設計審局函數:

(一)　持先手且翻開的牌分數小於 10 時 ：

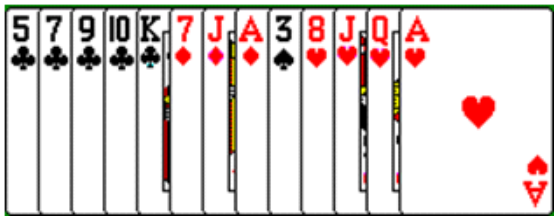
在這裡我們要考慮的是盡可能不拿到這張牌，選擇最有可能比對方小的牌，以例 3.1 的牌型為例。

例 3.1: 假設王牌為♥，我方為先手。

翻開的牌



目前手牌



牌組紀錄:

♠2	♠3	♠4	♠5	♠6	♠7	♠8	♠9	♠10	♠J	♠Q	♠K	♠A
●	○	?	?	?	?	◎	◎	?	?	?	?	●
♥2	♥3	♥4	♥5	♥6	♥7	♥8	♥9	♥10	♥J	♥Q	♥K	♥A
?	?	?	?	?	?	○	?	?	○	○	?	○
♦2	♦3	♦4	♦5	♦6	♦7	♦8	♦9	♦10	♦J	♦Q	♦K	♦A
?	?	?	?	?	○	?	?	?	○	?	?	○
♣2	♣3	♣4	♣5	♣6	♣7	♣8	♣9	♣10	♣J	♣Q	♣K	♣A
◎	?	?	○	◎	○	◎	○	○	◎	◎	○	?

●：在對方手上　○：在自己手上　◎：已經出過了　？：尚未出現

在換牌的階段裡，我們能掌握的資訊有(一)自己手上的牌，(二)雙方在之前的幾回合打過的牌，(三)對手在換牌時，出了較大的牌而拿走在牌桌中間翻開的牌。根據這些資訊，我們可以將牌張分為四類，在對方手上、在自己手上、已經被出過的牌、以及尚未出現(不能確定在牌堆或者對方手上)的牌這四種情況，上表就是在描述每一張牌的情形。

現在已知對方的其中 2 張牌，尚有 11 張未知，總共未知的牌有 30 張，每張未知牌在對方手上的機率為$\frac{11}{30}$，計算每個花色中最小的牌比對方小的機率:

♠3　0

♥8　$(1-\frac{11}{30})^6 = 0.0645$

♦7　$(1-\frac{11}{30})^5 = 0.1018$

♣5　$(1-\frac{11}{30})^2 = 0.4011$

以♣5 機率最高，在這回合會選擇出♣5。

(二)　持先手且翻開的牌分數大於 10:

盡力爭取翻開的牌，選擇較可能大過對方的牌，除此之外，還要計算自己出的牌是否值得去爭取翻開的牌，所以這裡並不是只以機率作為分數評估。我設計了一個審局函數，這個審局函數並沒有什麼依據，是由我嘗試後設計出的。

計算的式子為:

P：獲勝機率

W：翻開的牌牌張價值

'：我方打出的牌牌張價值

分數 = P * W - '

在這個式子裡，當 W 愈大時，P 對分數的影響也愈大，此時程式為了提高獲勝機率(P)，會選擇擲出牌張價值(W_{pay})更高但也更可能大過對方的牌，以確保能夠爭取到翻開的牌。

同樣以例 3.1 的牌型為例，我方仍為先手，王牌花色改為♣，則♣2 的牌張價值為 24.5。依照設計的審局函數每一張牌作評分:

♠3 : 0 - 7.5 = -7.5

♥A : 1 * 24.5 - 25 = -0.5

♥Q : $(1-\frac{11}{30})$ * 24.5 - 15 = 0.516

♥J : $(1-\frac{11}{30})$ * 24.5 - 11 = 4.516

♥8 : $(1-\frac{11}{30})^3$ * 24.5 - 5 = 1.223

♦A : 1 * 24.5 - 25 = -0.5

♦J : $(1-\frac{11}{30})^2$ * 24.5 - 11 = -1.172

♦7 : $(1-\frac{11}{30})^5$ * 24.5 - 3 = -0.503

♣K : $(1-\frac{11}{30})$ * 24.5 - 35 = -19.48

♣10: $(1-\frac{11}{30})$ * 24.5 - 24 = -8.483

♣9 : $(1-\frac{11}{30})$ * 24.5 - 22 = -6.483

♣7 : $(1-\frac{11}{30})$ * 24.5 - 18 = -4.483

♣5 : $(1-\frac{11}{30})$ * 24.5 - 18.5 + 18 = -4.983

經過審局函數計算，程式會選擇分數最高(4.516 分)的♥J 做為此回合的出牌來爭

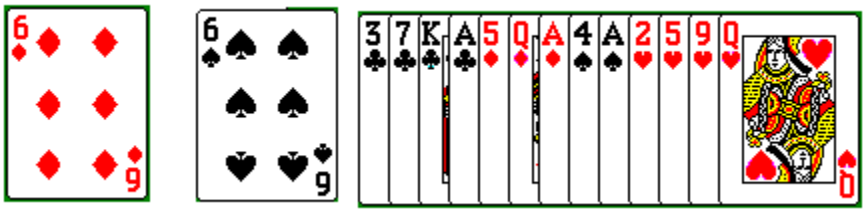
取牌張價值為 24.5 的♣2。

（三）　後手且翻開的牌分數小於 10:

在後手的情況下考慮的情況就簡單多了，因為已經看到對手的出牌，所以只要視對手的出牌來做策略選擇。如果手牌中有比對手小的牌，就從其中選一張牌張價值最低的出，同理，如果沒有，就從合法的出牌裡選一張牌張價值最低的出，以下以例 3.2 為例做說明。

例 3.2 :此局以♥為王牌

翻開的牌　　對方出牌　　我方的牌



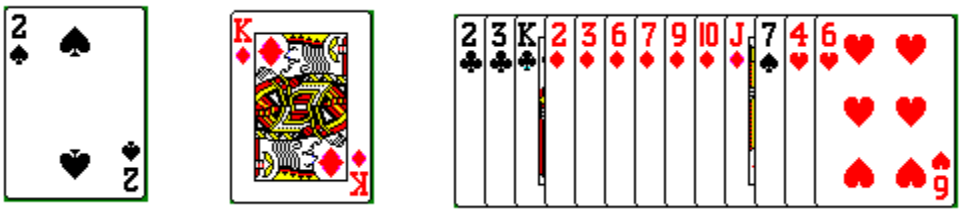
這個例子中♦6 牌張分數為 3，小於 10，不是我們想要的牌，故選擇出小牌，搜尋我方是否存在比♠6 更小的牌，找到♠4，所以程式會選擇出♠4。

（四）　後手且翻開的牌分數大於 10:

根據對方的出牌，我們會在牌裡尋找是否有比對方大的牌，如果有則從其中選擇牌張分數最低的出。如果沒有，則會從合法的牌裡選擇分數最低的牌出。

例 3.3: 此局以♠為王牌

翻開的牌　　對方出牌　　我方的牌



首先判斷♠2 分數為 24.5，大於 10，此時對方出♦K，搜尋牌組中是否有比♦K 更大的牌，由於不存在比♦K 更大的牌，所以從合法的牌♦2 ♦3 ♦6 ♦7 ♦9 ♦10 ♦J 選擇一張，其中以♦6 ♦7 最低分，在我的程式裡會選擇♦6 作為本回合的出牌。

換牌部分的虛擬碼:

```
Global float weight[2][13]    // 紀錄子力價值
int change (int up_card, int opp_card) { //紀錄翻開的牌和對手的牌
    float prob;
    int card;
    if (opp_card == Null) {    //表示我方是先手
       if (value(up_card) < 10) {
          card = compute_prob();  // 計算最可能小於對方的出牌
          return card;       //回傳要出的牌
          }
       else {                    //value > 10
          card = compute_gain(); //  計算分數最高的出牌
          return card;
          }
        }
```

```
    else {                          //我方是後手
        if (value(up_card) < 10) {
           card = search_less_than (opp_card); // 搜尋比對手更小的牌
           if (card != Null) return card;
           card = min_legal (opp_card); //搜尋合法出牌中子力最低的牌
           return card;
           }
        else {          // 翻開的牌子力分數大於10
           card = search_more_than (opp_card) //搜尋比對手更大的牌
           if (card != Null) return card;
           card = min_legal (opp_card);
           retrun card;
           }
       }
}
```

在換牌的函式 change 中會接收兩個參數 up_card(翻開的牌)以及 opp_card(對手出的牌)，當 opp_card 為 Null 時表示對手沒有出牌，也就是我方為先手，根據這兩個參數我們可以將情況分為審局函數中提過的四種組合(先後手及牌張分數是否大於 10)，再由審局函數對各種情況選擇出牌。

第五節 打牌演算法

3.5.1 先手轉換

在其他遊戲中，如圍棋、象棋…等，一旦先後手的關係決定，就會一直持續到遊戲結束，橋牌和這些遊戲最大的不同點就是沒有一個固定的先後手關係，無論是在換牌或者打牌步驟時，每一輪出牌較大者會獲得下一輪先出牌的權利，這使得我們在做 Min-Max 法展開時需要有一些改變。在一般的 Min-Max 法展開中，所有的節點分數都是站在我方的立場來看，我們在我方盤面時所有展開的子節點取 Max，對方盤面時對展開的子節點取 Min，如圖 3-14。

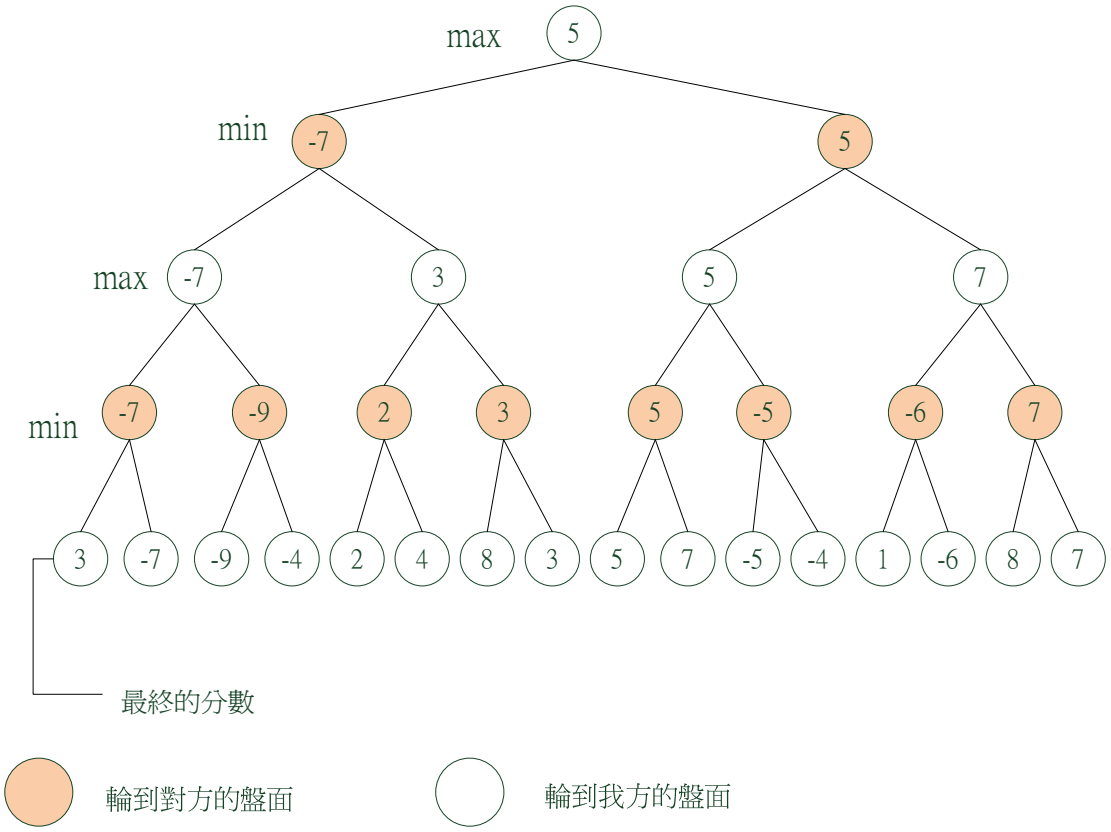


圖 3-14 Min-Max 法展開示意圖

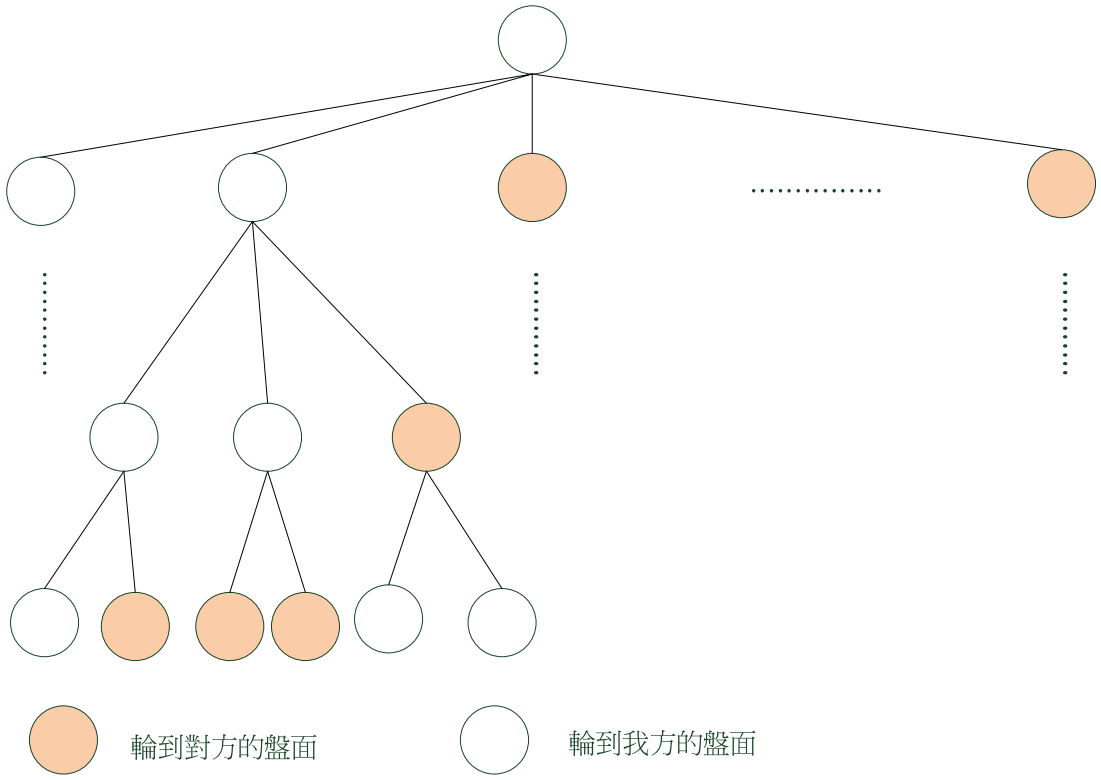
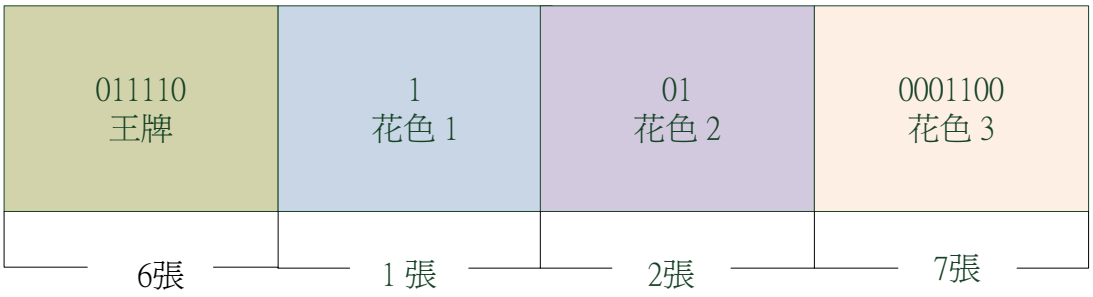


圖 3-15 蜜月橋牌可能出現的先後手轉換

蜜月橋牌遊戲中，很難會發生某一方從頭到尾都是先手(每一回合都大於對方)，或者是都是後手的情況(每一回合都小於對方)，所以勢必會出現 3-15 的情況，因為根據不同的出牌選擇，造成子節點會是由不同玩家先出牌的現象，這樣的情況無法直接取子節點分數最高或取子節點分數最低者來作為節點的分數。

在我設計的資料結構中，在展開的過程中不是以特定玩家來做盤面的評分，而是以「先手」的角度來看這一局牌，子節點回傳的分數都是在子節點盤面持先玩家的分數。例如:雙方各有八張牌時，排序為 0111101010001100，花色分佈為王牌 6 張，其餘花色為 1、2、7 張，先手(排序中 1 的玩家)可以在這八磴牌中贏得 4 磴，因此回傳分數為 4。我把其他的處理過程交給父節點來處理。



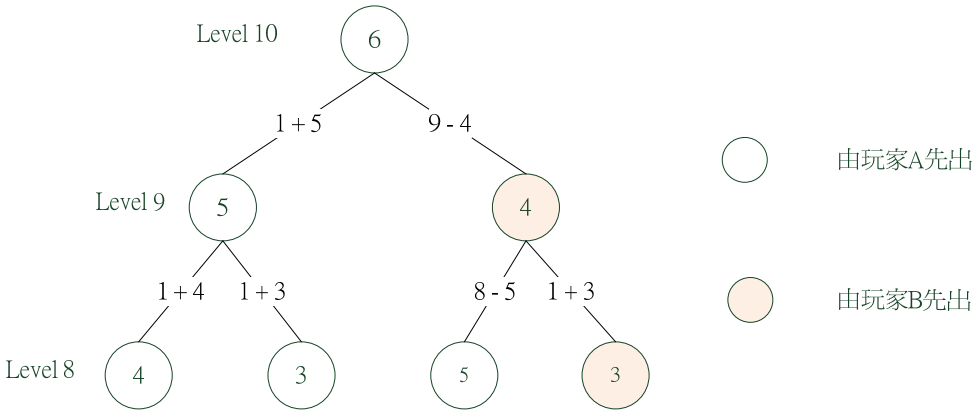


圖 3-16 橋牌應用 Min-Max 法展開示意圖

在圖 3-16 中，父節點分數由子節點回傳值決定，當子節點盤面仍是持先手，表示這一回合獲勝，故除子節點盤面回傳分數外，還要再加上 1 分才是真正的分數。若子節點盤面為對方先手，回傳分數為對方得分，因雙方總得分等於總牌磴數，level 減去對方得方即為己方得分。在這個架構下，不管是輪哪一方的盤面，都是取最大值。

3.5.2 盤面展開及分支裁減

在這裡我們要說明為何有些分支是不需要展開，而又要如何找出這些分支，以下面的實際例子來說明。

圖 3-17 是兩位玩家在打牌步驟時的牌以及排序組合，如同之前介紹過的資

料結構，其中 1 代表有出牌權的玩家擁有的牌在序列中的位置，0 代表另一位玩家的牌在序列中的位置。

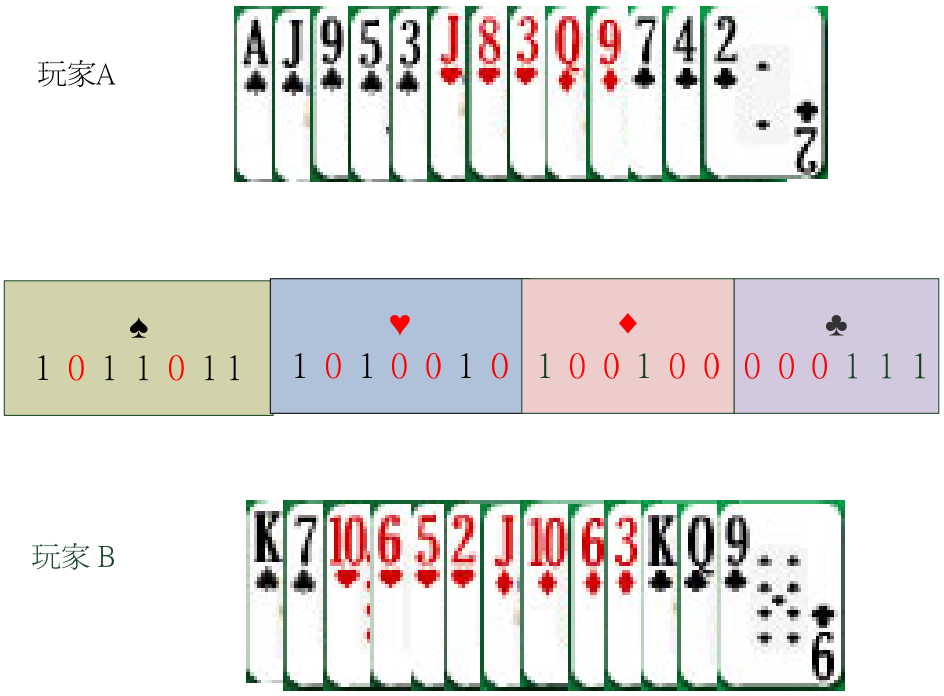


圖 3-17 雙方牌型組合

打牌時，先出牌的玩家 A 可以任意出牌，如果要求得最佳策略，就必須考慮所有路徑，對每一張牌作展開搜尋，如圖 3-18。

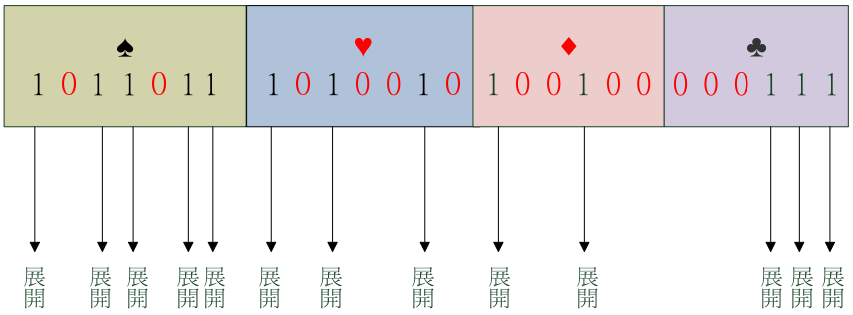


圖 3-18 分支展開圖

但是事實上有些分支是不需要被展開的，如圖 3-17，我們只看黑桃花色的部分，玩家 A 的牌有 A、J、9、5、3，玩家 B 的牌有 K、7，雙方的混合序列為

1011011。若玩家 A 選擇出♠J，玩家 B 可選擇的牌有♠K、♠7，出♠K 會使黑桃序列變為 11011，出♠7 會使序列變為 10111。而玩家 A 選擇♠9 時，玩家 B 可做的選擇仍然一樣，而最後會形成的序列也是一樣。因為在♠J、♠9 之間並沒有玩家 B 的牌，所以只要大於♠J 的牌，必然也會大於♠9，反之小於♠9，也會小於♠J，在玩家 B 看來，♠J、♠9 兩張牌是等價的。因此我們得到一個結論，在相同花色的兩張牌之間，若不存在對方的牌時，這兩張牌為等價，會有相同的展開結果，故不需要重複做展開，利用這一點，我們可以減少展開的分支度。圖 3-19 為經過分支裁減的結果，在這個例子裡我們減少了四個分支的展開。

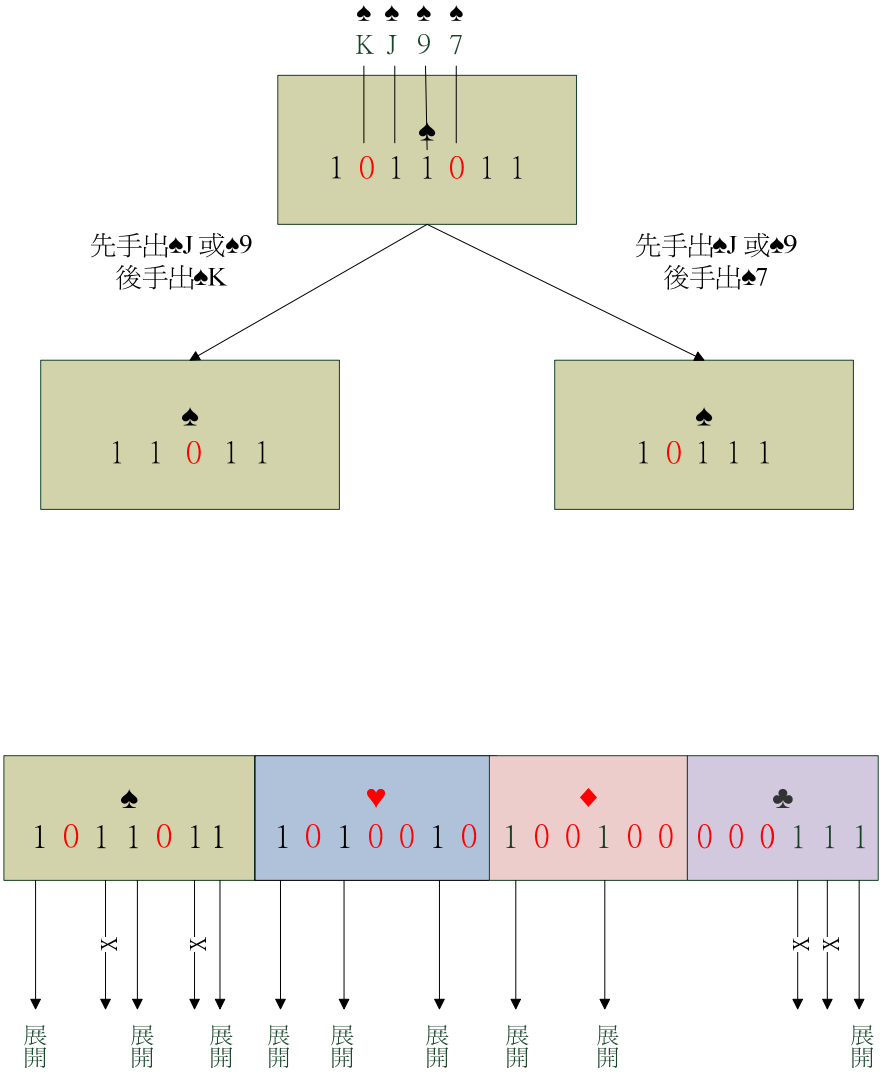
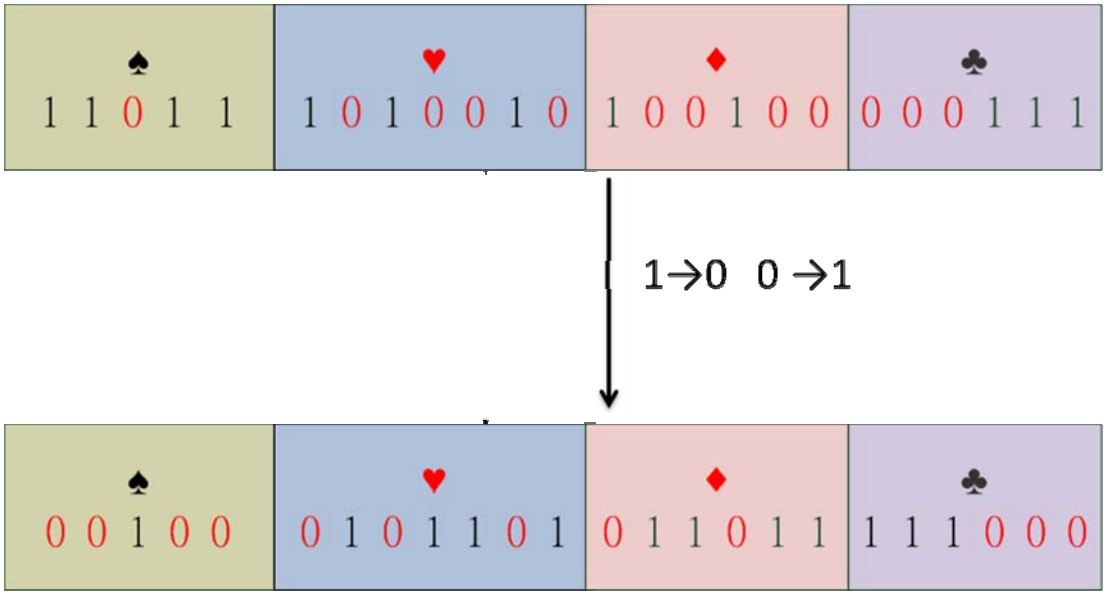


圖 3-19 分支裁減後展開

承上例，若玩家 A 出了♠9，玩家 B 出了♠K，則下一回合玩家 B 會先出牌，而原本的序列則是以玩家 A 為先手，這時只要對原本的序列取補數，就可直接將盤面修正為以玩家 B 為先手的正確盤面。在我的程式中，因為採用了序列的結構來記錄牌型，不僅在盤面資訊的傳遞上簡單，不需要複製整個牌組的陣列，只需要傳遞兩個整數，在運算上也只需要簡單的 AND、OR、XOR 等指令就可以完成牌組的展開。

最後，因為對手是人類，所不一定會照著電腦預計的最佳路徑來打牌，所以每一回合都需要重新做一次搜尋，針對對手的走步重新制定最佳策略。



先手轉換

打牌程式展開搜尋虛擬碼如下:

```
int search (int order, int flower, bool first, int depth) {
                                    //order 紀錄排序狀況
                                    //flower 為花色分佈  first 紀錄先手轉換
/*這裡採用了 min-max 的架構，所以在一開始時，min 要被初始化為無限大，max
```

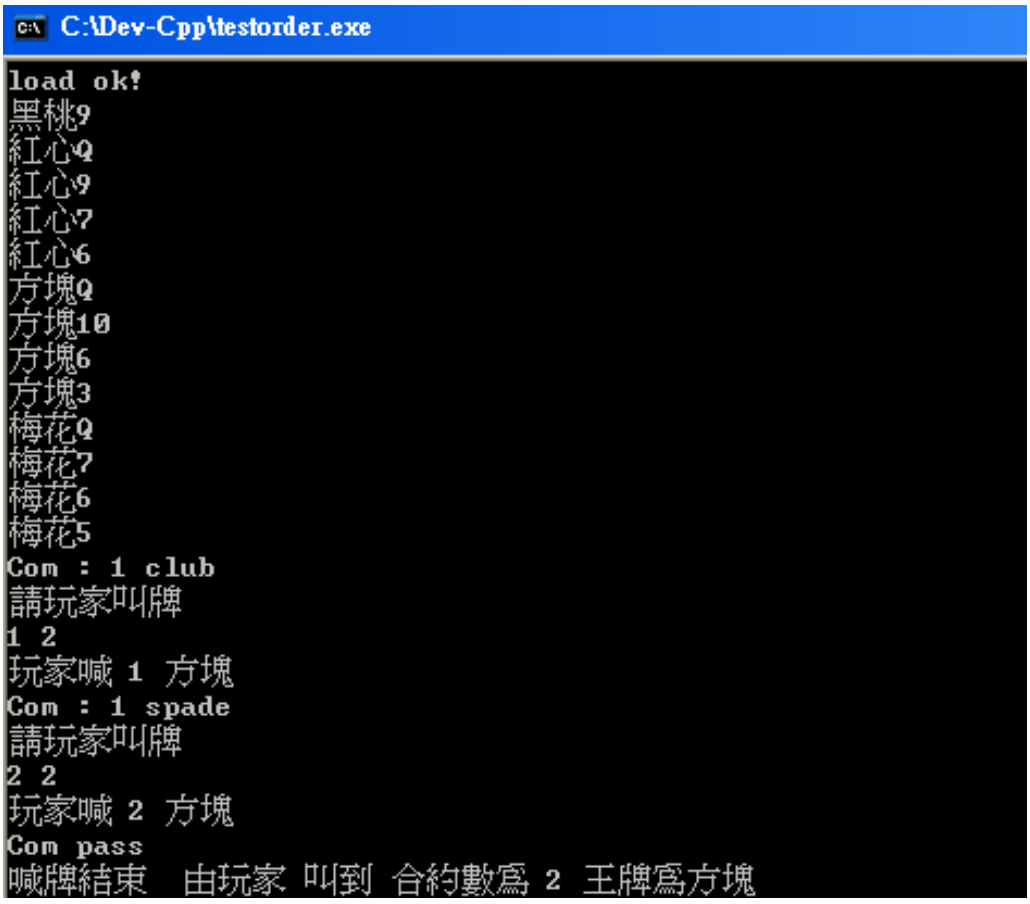
```
要被初始化為無限小*/
  int neworder;
  int min[depth] = verybig;
  int max = verysmall;
  int temp;
  if (depth == 0) return 0;
  if (firsr = 1) {          //先手轉換
      order = Not order    //排序取補數
      }
  for(i = 0; i < 2*depth; i++){
      if (testbit(order, i)) {     //測試order中第i個bit是否為1
         /* 檢查是否需要展開（分支裁減）*/
         if ( flower_diff(i,i-1) || testbit(order,i-1) == 0) {
            do {
               j = searchlegal(i); //尋找跟牌方合理出牌
               neworder = reorder(i,j);//產生新的序列
               newflower = reflower(i,j) //產生新的花色分佈
               if ( judge(i,j)) { //  i的牌型大於j的牌型
               /* 遞回呼叫 ，並且在各回傳值中選「最小」*/
                  min[temp] = minimum (min,
                     1+search(neworder,newflower,False,depth-1));
               else {
                  min[temp] = minimum (min,
                depth-1-search(neworder,newflower,Ture,depth-1));
               } while(j != Null);
```

```
            }
            max = maxmum (min[0],min[1],…);
        return max;
}
```

上述程式碼是以遞迴的架構來完成展開的動作，當深度(depth)為 0 為展開的最終節點，在打牌時如果有發生先手轉換(fiest == 1)的情況，則將序列取補數以符合盤面情況，函式 testbit()會檢查先手所有可能出牌，也就是在序列中為 1 的部份，再檢查後手所有合理的出牌，再以遞迴方式及 Min-Max 演算法找出最佳路徑。

第四章 測試結果及未來方向

本研究的蜜月橋牌程式是使用 C 語言寫成，開發軟體為 Dev C++ 4.9.9.2，個人電腦的硬體規格如下:CPU AMD 2500(約 1.86GHz)，記憶體 768MB，作業系統:Microsoft XP SP1，圖 4-1 為操作介面。



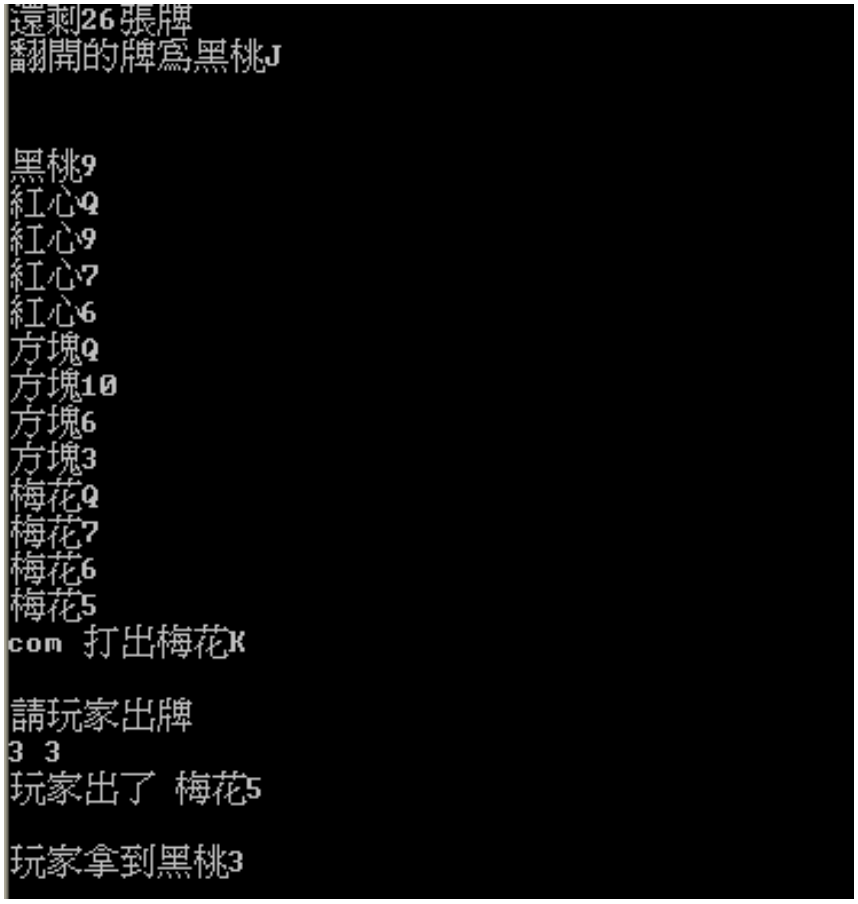


圖 4-1 操作介面

實驗安排如下:叫牌時以 50 副牌來做為我們的模擬局數，每次的模擬時間大約 20 秒，換牌及打牌的部份因為分別使用的是貪婪演算法及 Min-Max 演算法展開，計算時間都很短，程式幾乎是立刻做出決策。

第一節 測試結果及分析

以下為我的程式(com)和我(player)測試 19 場的數據及結果，設定蒙地卡羅模擬次數為 50 次。

合約	合約方	電腦評估	換牌結束	打牌結果	誤差	勝利方
2♦	com	7.92	6	7	-1.92	player
3♥	player	5.08	6	7	0.92	com
2♦	player	5.34	5	7	-0.34	com
1♥	player	6.02	4	4	-2.02	player

1♠	player	5.84	4	5	-1.84	player
2♣	com	7.42	7	7	-0.42	player
2♥	player	6.88	4	5	-2.88	player
4♠	player	4.5	4	4	-0.5	com
1♠	player	6.1	6	7	-0.1	com
2♥	com	9.25	10	10	0.75	com
3♠	player	2.95	3	3	0.05	player
3♦	player	3.56	3	5	-0.56	com
2♥	com	8.6	9	9	0.4	com
2♥	player	4.54	3	4	-1.54	player
2♠	player	3.72	5	5	1.28	player
3♦	player	4.15	3	3	-1.15	player
2♥	player	8.84	7	8	-1.84	com
1♣	com	5.18	6	6	0.92	player
1♠	player	5.76	7	7	1.24	com

在上表中誤差是指電腦評估磴數與換牌結束後的磴數差距，而不是與實際打牌結果的差距，因為人打牌時可能發生失誤，選取的並不是最佳路徑，這邊是以雙方都以最佳策略展開的結果來做評估。圖 4-2 是 19 盤的數據以折線圖來表示。

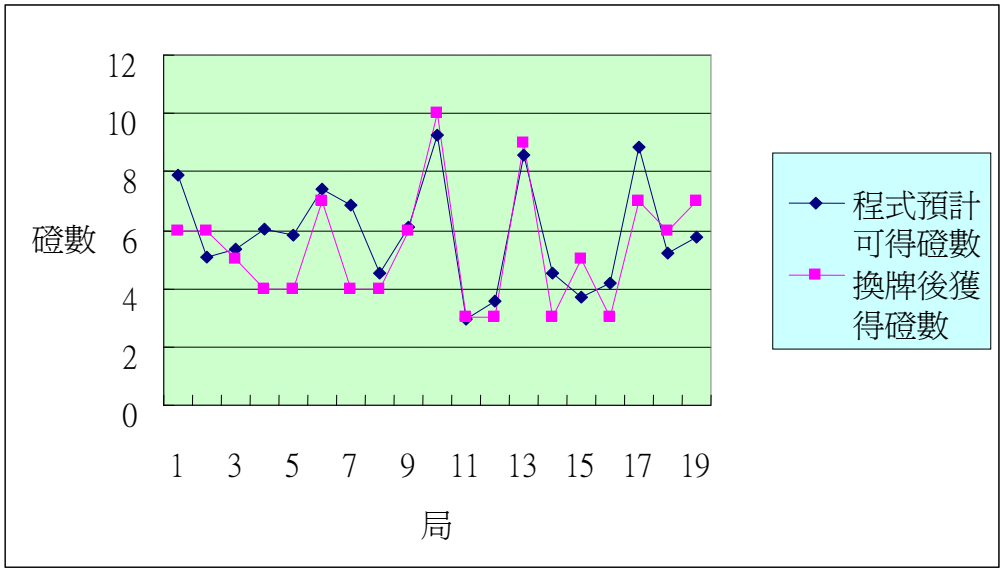


圖 4-2 程式猜測可得磴數與實際獲得磴數圖

在 19 局的比賽中，電腦贏了 9 局，勝率為 $\frac{9}{19}$ 約為 47%，但是這是以最後的結果來看，如果不考慮人為失誤的部分，電腦真正贏的場數只有 6 局，勝率為$\frac{6}{19}$約為 31%。

但是不能光以勝率來評估程式，蜜月橋牌是一種運氣成分很重的遊戲，玩家一開始的手牌會大大地影響之後遊戲的結果，所以我們還要從電腦對牌評估的正確性來看。圖 4-3 是誤差圖。

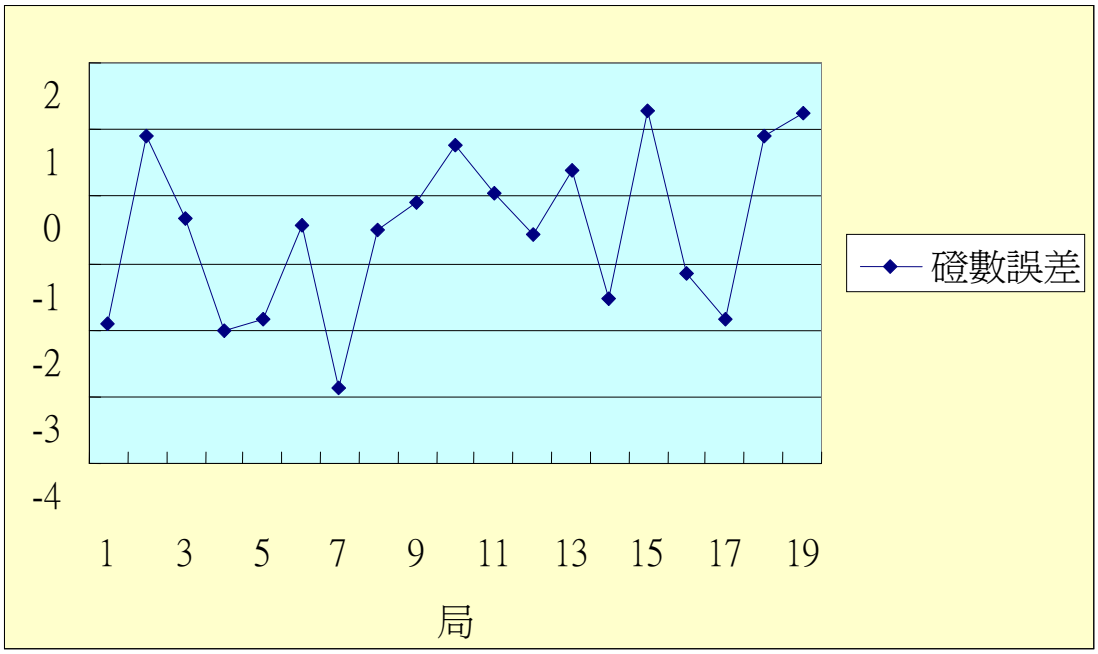


圖 4-3 可得墩數誤差圖

在 19 場的賽局中，程式有 12 局高估了自己的可得墩數，平均每一場高估自己 1.25 墩。另外 7 局低估了自己的可得墩數，平均低估了 0.794 墩，從圖 4-1 及圖 4-2 我們可以看出，預估的墩數還是有一定程度的準確性。再來我們從預估可得墩數來看程式預測勝負的準確度，以第一局為例，程式預計自己的可得墩數為 7.92，但是契約為線位為 2，表示電腦必須獲得 8 墩才能獲勝，但程式的預計可得墩數卻不到 8 墩，表示程式已經預計這一場比賽叫牌與否都會輸，只是選擇了一個勝率相對較高的叫牌策略。在玩家獲勝的 10 局中，其中有 9 局程式計算

出來的分數都不能阻止玩家獲勝，也就是受限於牌力而處於劣勢，並不完全是策略錯誤。

因為我們採用了蒙地卡羅模擬法來模擬計算各花色的分數，所以我們必須考慮為了配合遊戲時間所採用的模擬次數是否足夠準確。下表是分別以 10 次、50 次(遊戲中採用的模擬次數)、100 次及 200 次模擬計算得到的各花色分數，我們分別對四局牌做分數的模擬。在下表中顯示，我們程式中採用的 50 次模擬次數分數的精確度只有到小數後兩位，但與 100 次及 200 次模擬出來的分數之間的差距並不大，而這些微的差距隊也很少會影響到最後做出的叫牌決策。

牌局場次	模擬次數	黑桃分數	紅心分數	方塊分數	梅花分數
1	10	8.2	8.5	8.5	9.0
1	50	7.54	7.7	7.58	8.24
1	100	7.5	7.63	7.57	8.13
1	200	7.345	7.6	7.62	8.215
2	10	6.2	8.1	8.3	6.3
2	50	6	7.58	7.88	5.76
2	100	5.76	7.58	7.82	5.69
2	200	5.65	7.605	7.905	5.505
3	10	6.6	6.6	6.3	5.1
3	50	6.62	5.44	6.34	4.94
3	100	6.57	5.37	6.28	4.98
3	200	6.425	5.21	6.175	4.91
4	10	8.9	9.1	7.7	7.7
4	50	7.94	7.96	7.58	6.84
4	100	7.84	7.76	7.32	6.73
4	200	7.92	7.825	7.29	6.825

為了進一步驗證程式的牌力，我另外找了兩位人類玩家(高中同學)來與我的程式各比賽 10 場。為了公平起見，相同的一副牌，程式及玩家會交換角色，分別由程式及人類各持一次比賽看結果。但為避免人類玩家已看過牌，故交換角色時，改由另一玩家持交換的牌來比賽，也就是說，這裡為了避免玩家對牌組的印象影響結果，所以由兩個不同的玩家來進行交叉測試。在測試的過程如下表，程式在 20 局的比賽中共獲勝 12 局較多，但主要是由於在打牌部分能夠使用完全的展開搜尋因此能夠找到最佳策略，對人類玩家來說，除了某些牌型外，當手牌張數還很多時要找到最佳策略是相當耗時且不容易的。

場次	獲勝者	交換手牌獲勝者
1	com	player
2	com	com
3	player	com
4	player	com
5	com	com
6	com	com
7	player	com
8	com	player
9	player	player
10	com	player

第二節 結論及未來方向

在這之前，似乎沒有探討蜜月橋牌的論文，因此在開發過程，大多數的方法及資料結構都是作者對遊戲的心得，以及從其他遊戲借鏡得來的。其中有很多想法在其他遊戲可行，但在開始實作後卻遇到瓶頸，不少想法都放棄了再想新方

法，本著盡量不以遊戲經驗或專家法則建立 rule-based 的想法來做蜜月橋牌，但是在許多地方還是需要將自身的經驗帶入遊戲中，例如參數的設立以及換牌時的考量，這些是我覺得可惜沒有完全克服的地方。雖然測試的次數並不多，感覺目前程式的功力應該與一般玩家相去不遠，但我認為還是有很多可以改善的地方，未來可以再加以探討:

(一) 蜜月橋牌遊戲中雖然有著很高的機率性，但玩家的行為也是一種可加以利用的資訊。如何把握這些資訊，進一步改善程式，讓程式能掌握這些資訊，以建立更準確的審局函數，這也是一個有趣的議題。

(二) 在程式中使用了大量模擬的部分，可以透過平行化提高模擬次數，蒙地卡羅模擬法在程式平行化時很容易製作，只要確保各個程式之間亂數的產生不會同步互相影響即可。現在電腦大都具有多核心，也讓蒙地卡羅法在遊戲實做中更容易被大家所採用。

(三) 動態審局函數及牌張價值，在遊戲中審局函數及牌張價值可以根據目前盤面資訊，還有以前盤面的資訊來做動態的調整，牌張價值也可以參考基因演算法，透過子代突變競爭慢慢改善，藉由對奕過程慢慢往較準確的方向演化。

參考文獻

[1] 謝曜安，“電腦暗棋之設計及實作”，國立台灣師範大學資訊工程研究所碩士論文，2008。

[2] 張璨文，“「德州撲克」不完全資訊賽局之研究”，國立台灣師範大學資訊工程研究所碩士論文，2005。

[3] 旁士東“囚犯的兩難—賽局理論與數學天才馮紐曼的故事”，左岸文化出版社，2007。

[4] T. S. Ferguson ，“Gmae Theory”課程講義 ，http://www.math.ucla.edu/~tom/math167.html，2005。

[5] Wikipedia，http://zh.wikipedia.org/wiki/Wiki。

[6] J. v. Neumann， “Zur Theorie der Gesellschaftsspiele”， Math. Ann.，1928。

[7] J. Nash ，“Non-cooperative Games”，Math. Ann.，1951。

[8] Honeymoon Bridge，http://www.pagat.com/boston/honeymoon.html

[9] S. J. Russell , P. Norvig，Artificial Intelligence A Modern Approach，Pearson International Edition，2003。

[10] T. H. Cormen , C. E. Leiserson , R. L. Rivest , C. Stein，Introduction to Algorithms，MIT Press，2003。