

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

電腦象棋程式 Shark 的設計與實作

The Design and Implementation of the
Chinese Chess Program Shark

研究生： 劉孟謙 撰

中華民國 一百零三 年 七 月

摘要

電腦對局在人工智慧領域一直是一個非常吸引人的項目，而電腦象棋相關研究發展遠不及國外西洋棋來的豐富，而象棋是我國國粹，因此我們希望針對電腦象棋的領域進行深入的研究探討。本研究以網路上開源的象棋程式「象眼」為基礎，設計改良開發成象棋程式 Shark，以供象棋棋手精進棋力、參與各項電腦對局競賽，並期望能與人類大師抗衡。

下棋程式的棋力與搜尋演算法、審局函數息息相關，本研究主要針對這兩項進行改良。我們以軟體工程的概念設計加強演算法的可靠性；研發新的資料結構—BitBoard 大幅的增進了審局函數的效能，讓程式能搜尋得更深、且讓特殊棋型的偵測以及複雜的長捉盤面處理可以非常有效率的實現；另外增強程式對於兵卒的掌握以及使用 Material Table 建構更多程式對於中殘局的知識。

原始象眼估計其棋力約有六段，改良後的 Shark 與象眼對弈已有 80% 的勝率，並且於 2013 年 12 月參加 TAAI 電腦對局比賽獲得銅牌、經過再改良後 2014 年 6 月參加 TCGA 電腦對局比賽獲得銀牌，在眾多持續開發多年的象棋程式中擠進了一席之地，估計其棋力已晉升至七段。其後我們將繼續開發改良，期望能更精進棋力，向頂尖程式與人類大師看齊。

關鍵字：位棋盤、人工智慧、電腦象棋

ABSTRACT

Computer gaming is a very interesting subject in artificial intelligence. Compared with Chess, Chinese chess (XiangQi) has fewer researches. Since Chinese chess is the quintessence of Chinese culture, we decide to make a further study on the subject.

Our Chinese chess program "Shark" is improved from "ElephantEye" - an open source Chinese chess program on the Internet. Shark has participated in the TAAI 2013 and TCGA 2014 computer game tournaments.

The strength of a Chinese chess program depends on its search algorithms and evaluation functions. We attempt to improve the two parts. We apply software engineering principles to improve the reliability of the search algorithms, develop a new data structure - BitBoard to improve the performance of the evaluation functions, make our program able to search deeper, recognize patterns more efficient, and detect the perpetual chase well. Then we build Material Table to establish the knowledge of mid-to-end games.

ElephantEye is ranked about 6 dan. So far Shark has won 80% of the games against ElephantEye. And it won the bronze medal in TAAI 2013, the silver medal in TCGA 2014. It is ranked about 7 dan.

We will continuously improve our Shark program. The ultimate goal is to compete with the strongest program and the human masters.

Keywords: BitBoard, artificial intelligence, computer Chinese chess

致謝

感謝指導教授林順喜博士，在這幾年的研究以及論文寫作上耐心及熱忱的指導我，並且解決各式遇到的困難，教授總是有非常多的 Idea，並且不厭其煩的與我討論各項細節，讓我獲益良多。

特別感謝實驗室的徐大開同學，也是我大學時認識的象棋社社長，象棋棋力有三段，由於我本身象棋棋力不高，如果沒有他的大力幫助，我想我可能無法完成這個研究，在此獻上萬分感謝之意。

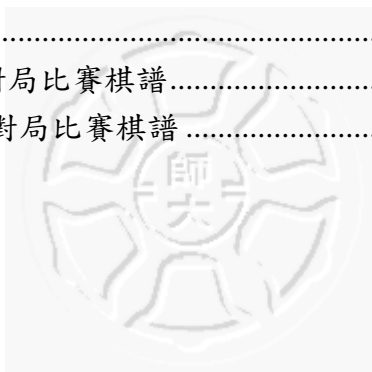
感謝實驗室的學長姐，在畢業後還是在我比賽前最辛苦的日子回來幫我加油打氣，給我心態上的調適，讓我能更專心投入在研發上。感謝陳志宏學長常常提供我研究上以及論文上的技術支援。感謝實驗室的同學與學弟們的支持與鼓勵。

最後感謝我的父母，提供我多年良好的求學環境，使我衣食無虞，並一路支持著我，讓我能順利完成學業，在此僅以本論文以及成果獻給我最敬愛的父母。

目錄

摘要.....	i
ABSTRACT.....	ii
致謝.....	iii
目錄.....	iv
圖目錄.....	vi
表目錄.....	viii
第 1 章 緒論.....	1
1.1 研究背景.....	1
1.2 研究目的.....	2
1.3 研究意義.....	2
第 2 章 文獻探討.....	4
2.1 相關論文及程式介紹.....	4
2.1.1 象眼.....	4
2.1.2 ELP.....	4
2.1.3 棋謀.....	5
2.1.4 象棋世家.....	5
2.1.5 BrainChess.....	5
2.2 電腦對局理論.....	6
2.3 演算法.....	8
2.3.1 Mini-Max Search.....	8
2.3.2 Alpha-Beta Search.....	10
2.3.3 PVS.....	13
2.3.4 水平線效應、寧靜搜尋.....	16
2.4 資料結構.....	18
2.4.1 棋盤表示與走步生成.....	18
2.4.2 Transposition Table.....	20
2.5 審局函數.....	22
2.6 後續發展.....	23
第 3 章 方法與步驟.....	24
3.1 資料結構的改進.....	24
3.1.1 適用於象棋的 BitBoard.....	24
3.1.2 BitBoard 的棋盤表示法.....	26
3.1.3 棋子表示法.....	27
3.1.4 Intrinsic 函數.....	29

3.1.5	BitBoard 的基本操作指令	31
3.1.6	以 BitBoard 加速走步產生	34
3.1.7	Magic Hash	39
3.2	演算法改進	41
3.2.1	走步排序策略	41
3.2.2	Late move reduction (LMR)	44
3.2.3	PVS 的改良	46
3.2.4	控制格計算	47
3.2.5	長捉的偵測	49
3.3	審局函數改進	56
3.3.1	子力價值	56
3.3.2	特殊棋型	58
3.3.3	自由度	63
3.3.4	Material Table	66
第 4 章	實驗結果與未來發展	69
參考文獻	73
附錄 A	– TAAI 2013 電腦對局比賽棋譜	74
附錄 B	– TCGA 2014 電腦對局比賽棋譜	79



圖目錄

圖 1	不同遊戲的狀態複雜度與遊戲樹複雜度【1】	3
圖 2	目前已知最少子困斃最多子盤面，黑方所有子都在卻無合法走步	6
圖 3	井字遊戲為例的遊戲狀態樹	7
圖 4	Mini-Max 樹示意圖	9
圖 5	Alpha-Beta 樹示意圖	11
圖 6	輪黑，紅方在 8 步後能將死黑方	18
圖 7	範例盤面	19
圖 8	BitFile 結構	20
圖 9	BitRank 結構	20
圖 10	Transposition Table 示意圖	22
圖 11	128 bits 象棋棋盤 BitBoard 示意圖	25
圖 12	初始盤面紅方各個棋子對應的 BitBoard 示意圖	27
圖 13	MS1B、LS1B 示意圖	30
圖 14	棋盤合法位置的 BitBoard，O 部分表示該 bit 為 1	32
圖 15	BitTable[43]的 BitBoard 示意圖	33
圖 16	Col[3]的 BitBoard 示意圖	33
圖 17	車的 Mask	35
圖 18	與 Occupied 做 AND 運算後的結果	35
圖 19	查表得到的走步資訊	35
圖 20	車的走子步	36
圖 21	車的吃子步	36
圖 22	馬與象其拐腳處的 Mask	37
圖 23	反向偶的 Mask	38
圖 24	要檢查是否有偶的位置	38
圖 25	AVX2 指令 PEXT 運作示意圖【11】	40
圖 26	Killer Heuristic (左)與 History Heuristic(右)比較	43
圖 27	水平線效應	44
圖 28	使用延伸策略後	44
圖 29	使用 LMR 後	45
圖 30	範例中局盤面	47
圖 31	紅方的控制點	48
圖 32	黑方的控制點	48
圖 33	紅方受到保護的棋子	49
圖 34	黑方受到紅方威脅的棋子	49

圖 35	紅俾長捉假根包，紅方不變作負	51
圖 36	紅偶長捉真根包，雙方不變作和	51
圖 37	黑方長捉無根兵，不變判負	52
圖 38	由於黑方車不能反吃，因此為紅方俾炮連捉黑方車，紅方不變作負	55
圖 39	TCGA 2014 Shark 對 BrainChess：雙方走閒著，不變作和	55
圖 40	合法的 From 與 To 組合	58
圖 41	不合法的 From 與 To 組合	58
圖 42	各種栓鍊	62
圖 43	車的自由度計算	64
圖 44	馬的自由度計算	65
圖 45	車兵勝士象全	68
圖 46	單車勝單缺士	68
圖 47	TAAI 2013 棋謀對 Shark 中局盤面	70
圖 48	陷阱佈局	71
圖 49	陷阱佈局結果	71



表目錄

表 1	NegaMax 演算法虛擬碼	10
表 2	Alpha-Beta 演算法的虛擬碼.....	12
表 3	PVS 演算法的虛擬碼.....	15
表 4	棋子編號意義，最右邊為 bit 0.....	28
表 5	棋子編號方式	28
表 6	Intrinsic 指令與傳統 C++指令比較：128 bits Shift 為例	29
表 7	Intrinsic 指令與傳統 C++指令比較：LS1B 為例	30
表 8	MSB 的 C++程式碼【9】	31
表 9	BitBoard 基本指令	32
表 10	檢查將軍的實作方法比較	39
表 11	不適合使用 LMR 的狀況定義	46
表 12	亞洲棋規長捉規則細節	50
表 13	判斷捉的虛擬碼	53
表 14	車馬炮在開中殘局的價值	56
表 15	仕相對於對方的威脅下降而降低其價值	57
表 16	各種狀態下兵的價值	57
表 17	Material Table 編碼表	67
表 18	TAAI 2013 電腦對局象棋比賽結果.....	69
表 19	TCGA 2014 電腦對局象棋比賽結果	71

第1章 緒論

1.1 研究背景

在人工智慧的研究領域之中，電腦對局一直是十分重要的一部分。除了各式對抗類遊戲的人工智慧開發，同時也有各類益智遊戲的解題研究，許多嶄新的演算法與資料結構也隨著這些研究不斷推出。並且隨著硬體效能與演算法的改良，許多遊戲的人工智慧已經開始達到了職業玩家的水準，有些遊戲甚至人類已經不是電腦的對手，相信未來這樣的趨勢將會越來越明顯。

象棋是中國的國粹之一，其歷史相當的悠久，從古至今棋手累積了大量的棋譜與開中殘局的豐富知識，例如《橘中秘》、《梅花譜》等等，也有些特殊棋型的衍生成語與棋諺，例如說馬後炮、飛象過河等，也顯示了象棋融入了華人的生活中。甚至有些國外的學者也熱衷於象棋研究。

象棋屬於完全資訊、無隨機性遊戲，遊戲中所有資訊對對弈雙方都是已知的，且沒有隨機成分，這類型的遊戲還有像是西洋棋、黑白棋、將棋等等，因為盤面資訊都已知，因此對遊戲的理解與掌握將主導棋局的進行，雙方的對弈比的就是硬實力了。目前國際電腦對局學會(ICGA, International Computer Game Association)、台灣電腦對局學會(TCGA, Taiwan Computer Game Association)以及中華民國人工智慧學會(TAAI, Taiwanese Association for Artificial Intelligence)也都將象棋列為正式的比赛項目之一了。

1.2 研究目的

象棋一直是華人愛玩的一種棋類，個個棋手不斷互相對弈，突破自我，追求更強的棋力，而我們的目的便是希望能設計出強大的象棋程式與人類頂尖棋手抗衡，甚至超越人類冠軍。因為程式是不會老化的，希望人類能與強大的程式對弈並且不斷精進棋力。

目前象棋的研究比起西洋棋來說發展不算完善，相關技術論文相對較少，特別在專家知識的建構上比較欠缺，因此電腦象棋算是很有研究價值的題目，我們嘗試改進電腦象棋在開中局的布局以及殘局觀念，並改善程式的效能以求更強的棋力。



1.3 研究意義

以完全資訊雙人對弈遊戲的人工智慧來說，不外乎都是以搜尋樹的方式來展開遊戲狀態，程式從中選擇最佳的走步行之。隨著搜尋層數加深，其遊戲樹搜尋節點數是以指數暴漲。圖 1 列出了各種遊戲其狀態複雜度以及遊戲樹複雜度，其中象棋的遊戲樹複雜度為 10^{150} ，可說是相當高。

設計者不斷的利用各種演算法以及分支切捨技術試圖加速以求更深的搜尋，各種新演算法也不斷的被提出。但一旦搜尋的層數到一定的深度，再增加深度對於棋力的提升便十分有限，且所花的計算成本卻大幅上升，因此勢必要將研發的重心轉向搜尋樹底層的審局函數，因為審局函數才是程式知識的核心，引領程式走向勝利。我們將試圖改進審局函數的專業知識，配合資

料結構設計高效益的審局函數。

Id.	Game	State-space compl.	Game-tree compl.
1	Awari	10^{12}	10^{32}
2	Checkers	10^{21}	10^{31}
3	Chess	10^{46}	10^{123}
4	Chinese Chess	10^{48}	10^{150}
5	Connect-Four	10^{14}	10^{21}
6	Dakon-6	10^{15}	10^{33}
7	Domineering (8×8)	10^{15}	10^{27}
8	Draughts	10^{30}	10^{54}
9	Go (19×19)	10^{172}	10^{360}
10	Go-Moku (15×15)	10^{105}	10^{70}
11	Hex (11×11)	10^{57}	10^{98}
12	Kalah(6,4)	10^{13}	10^{18}
13	Nine Men's Morris	10^{10}	10^{50}
14	Othello	10^{28}	10^{58}
15	Pentominoes	10^{12}	10^{18}
16	Qubic	10^{30}	10^{34}
17	Renju (15×15)	10^{105}	10^{70}
18	Shogi	10^{71}	10^{226}

圖 1 不同遊戲的狀態複雜度與遊戲樹複雜度【1】

第2章 文獻探討

本章將介紹電腦對局的基礎知識與本研究參考的主要資源—象眼，以及其特色。

2.1 相關論文及程式介紹

2.1.1 象眼

電腦象棋在大陸的研究較台灣來的多且完整，因此本論文主要是參考塗志堅撰寫的「電腦象棋的設計與實現」【2】以及開源的象棋程式「ElephantEye (象眼)」【3】來進行改良。

象眼由象棋程式「夢入神蛋」參照塗志堅的論文改進研發而成。象眼在大陸的聯眾、弈天等象棋對弈網站上作過測試，用等級分來衡量，聯眾網的戰績在 2500 分左右，弈天網快棋的戰績在 2000 分左右，慢棋在 1500 分左右。

2005 年 9 月象眼參加在台北舉行的第 10 屆 ICGA 電腦奧林匹克大賽中國象棋組比賽，戰績是 7 勝 5 和 14 負，在 14 個程式中排名第 11，目前在開源的程式中象眼在知識部分稍有不足，因此還沒有辦法向頂尖的程式看齊。

2.1.2 ELP

ELP 初名為「象棋明星」，由許舜欽教授領導的團隊研發，主要以組合語言寫成，因此效能上非常好，是屬於審局簡單、偏向遊戲樹展開的程式，在早期(2000 年左右)是國內首屈一指的程式。

2.1.3 棋謀

棋謀為交通大學吳毅成教授與其博士生曾汶傑研發，棋謀總共開發了約 10 年左右。不斷經由分析棋譜中的關鍵走步改良審局以特殊棋型偵測，並且完全平行化，在開中局可以搜尋至 20 層左右，在攻殺方面十分擅長。

2.1.4 象棋世家

象棋世家為鄭明政所開發，鄭明政有象棋五段的棋力，因此在審局精確度以及專家知識建構上可說是數一數二。

象棋世家由 2000 年開始研發，前後與許多棋手不斷測試改良，也參與了各種比賽，近期(約 2006 年至今)的比賽幾乎都是以全勝之姿蟬連冠軍寶座，在國內是無人能敵。現在象棋世家由東華大學顏士淨教授接手繼續研發改良。

2.1.5 BrainChess

BrainChess 由中研院博士後研究員陳柏年所開發，近期針對開局庫加入了大量陷阱布局，使得棋力大幅上升。其未來發展不容小覷。

2.2 電腦對局理論

象棋由棋手雙方輪流走子，至一方王被吃掉或是無合法走步——無法解將或因斃(如圖 2)即結束棋局，因此遊戲狀態可以以雙方就當前盤面可能的走子步一層一層的展開成一樹狀結構，樹上的每個節點代表著盤面狀態，根節點為當前的盤面。程式分析這棵龐大的對局樹並且從中選出一個最佳的走步作出行動，圖 3 是以井字遊戲中局盤面為例的遊戲狀態樹。

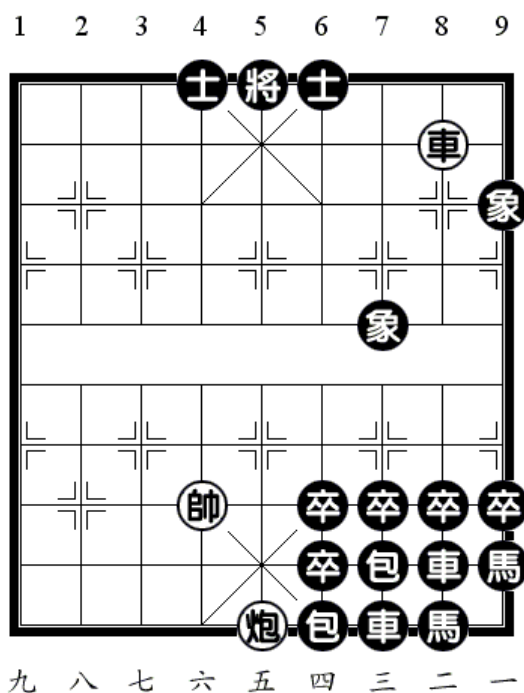


圖 2 目前已知最少子困斃最多子盤面，
黑方所有子都在卻無合法走步

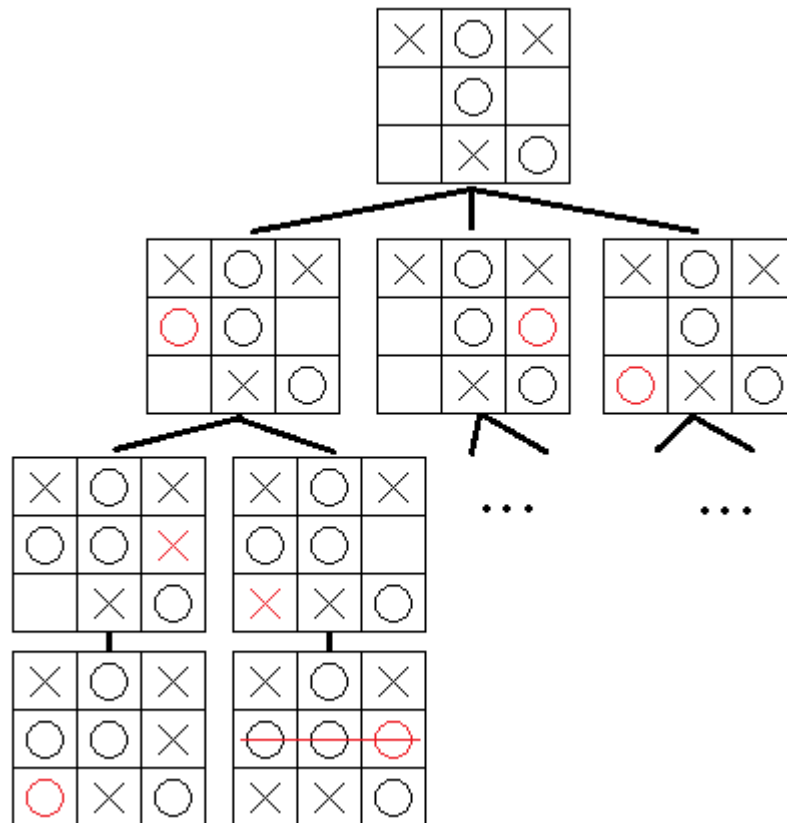


圖 3 井字遊戲為例的遊戲狀態樹

如果能完全展開整棵遊戲樹，則可以保證程式在所有盤面下都能做出最佳應對，換句話說就是破解了這個遊戲，但破解不代表能讓自己立於不敗之地，因為有可能在雙方都下出最佳走步的情況下有一方必敗，而井字遊戲的結果為必和。

由圖 1 可知，象棋的狀態複雜度有 10^{48} 、對局樹複雜度有 10^{150} ，要展開整棵樹在有限時間內是辦不到的，因此我們只能由根節點往下展開數層進行分析。

在展開到設定深度後，我們必須要對盤面進行分析，以一個大略的數值代表著盤面的局勢，即審局函數。審局函數的意義代表著假設這個狀態發展到遊戲結束的話能獲得的分數近似值，也可以稱作是獲勝的可能性分數，其

值越高代表著獲勝的可能性越高。

2.3 演算法

本節將詳細介紹對局程式常用的演算法以及其原理。

2.3.1 Mini-Max Search

由有限深度展開的遊戲樹中，我們必須要在葉節點(也就是有限深度搜尋樹中的最底層)呼叫審局函數進行局勢的評分，以評估剩餘至棋局結束大概的局勢。以零和遊戲來說，分數對於我方正值代表優勢，負值代表劣勢。如果雙方都是以最佳的走法下棋，則我方會想辦法讓局勢分提高，也就是 Max 方；敵方會想辦法讓局勢分降低，也就是 Min 方。

Mini-Max Search(最小-最大搜尋法)就是基於這個準則模擬雙方最佳的走步，也就是依據 Min 與 Max 的選擇走步策略不斷的遞迴由下往上回傳結果，圖 4 為 Mini-Max Search 的示意圖，Min 節點會選擇局勢分最低的子節點、Max 節點會選擇局勢分最高的子節點，由最底層的審局分數不斷經由 Min 與 Max 選擇並往上回傳分數，至根節點(當前盤面)時，此時的 Max 方選擇的分支就是當前搜尋樹下得到的最佳走步，程式就會輸出此步。

Mini-Max Search 是深度優先的演算法，複雜度為指數 k^d ， k 為分支因數，也就是每一手的平均合法走步數目， d 為搜尋樹展開的深度，是一個暴力的搜尋演算法。

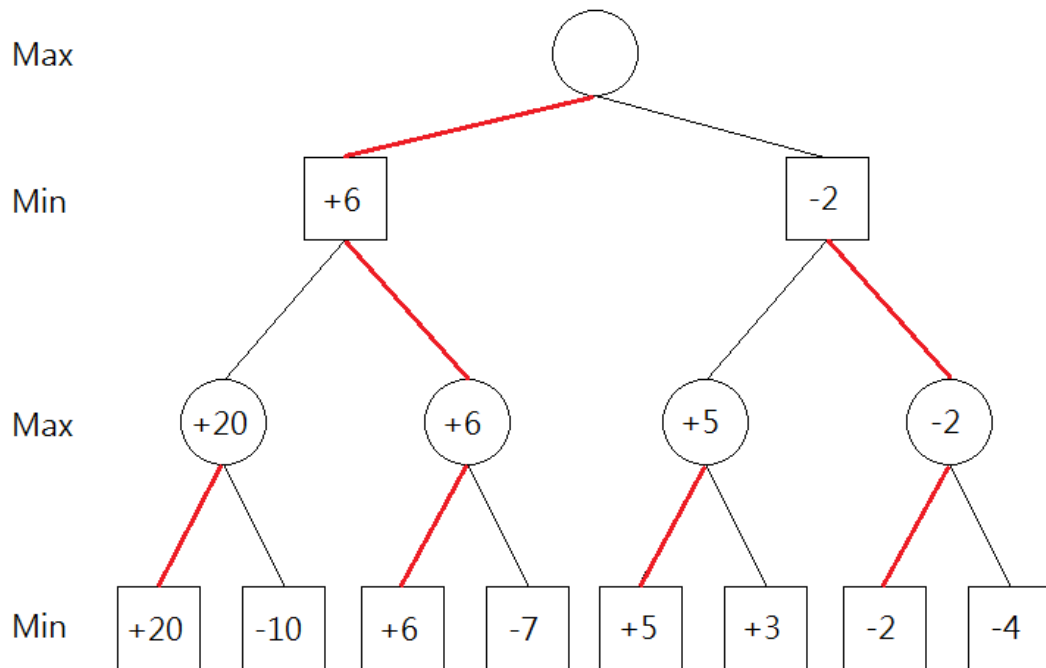


圖 4 Mini-Max 樹示意圖

而實際上對於雙方棋手的策略都是為了要提高己方的分數，並壓低對方的分數，因此我們可以在搜尋樹的分數往上回傳時都做變號處理，把看分數的角度變成當前節點的出手方，因而所有的節點都取 Max，這樣可以省去一半的程式碼，提升可維護性，此類的變種稱作 NegaMax Search。

表 1 是 NegaMax 演算法虛擬碼，注意 **Evaluate()** 永遠使用正號，表示局勢分數以當前走子方的角度計算，而在遞迴呼叫的部分前面加了一個負號，因為對於當前盤面來說，對方得正分就是我方得負分。

```

int NegaMax(int depth)
{
    if (depth == 0)
        return Evaluate(); //到達限制搜尋深度，回傳審局分數
    int best_score = -INF; //最佳分數設為負無限
    while (moves left) //嘗試當前盤面所有可能著法
    {
        MakeMove(move); //執行著法
        //減少一層的深度，遞迴演算盤面
        int score = -NegaMax(depth-1);
        UndoMakeMove(move);

        if (score > best_score)
            best_score = score; //更新最佳分數
    }
    return best_score;
}

```

表 1 NegaMax 演算法虛擬碼

2.3.2 Alpha-Beta Search

Alpha-Beta Search【4】【5】是 Mini-Max Search 的改良，概念在於由於對弈雙方都是選擇對自己分數最高的分支，基於這個特性，當搜尋中發現有一些盤面回傳的分數過高時(也就是對於上一層對手來說過低時)，便不需要再看其它分支，因為上一層對手必定不會選擇這個走步。

舉圖 5 的搜尋樹為例，當 E 節點得到+5 的分數時，由於 B 節點是 Min 節點，因此可以保證 B 節點回傳的分數絕對不會大於+5，又 A 節點已經得到+6 的分數，在根節點是 Max 節點的情況下，可以確保根節點絕對不會選擇 B 節點，因此 B 節點其下的子樹(F 節點開始)就都沒有必要搜尋了，這個切斷後續搜尋的動作叫做 Cutoff。

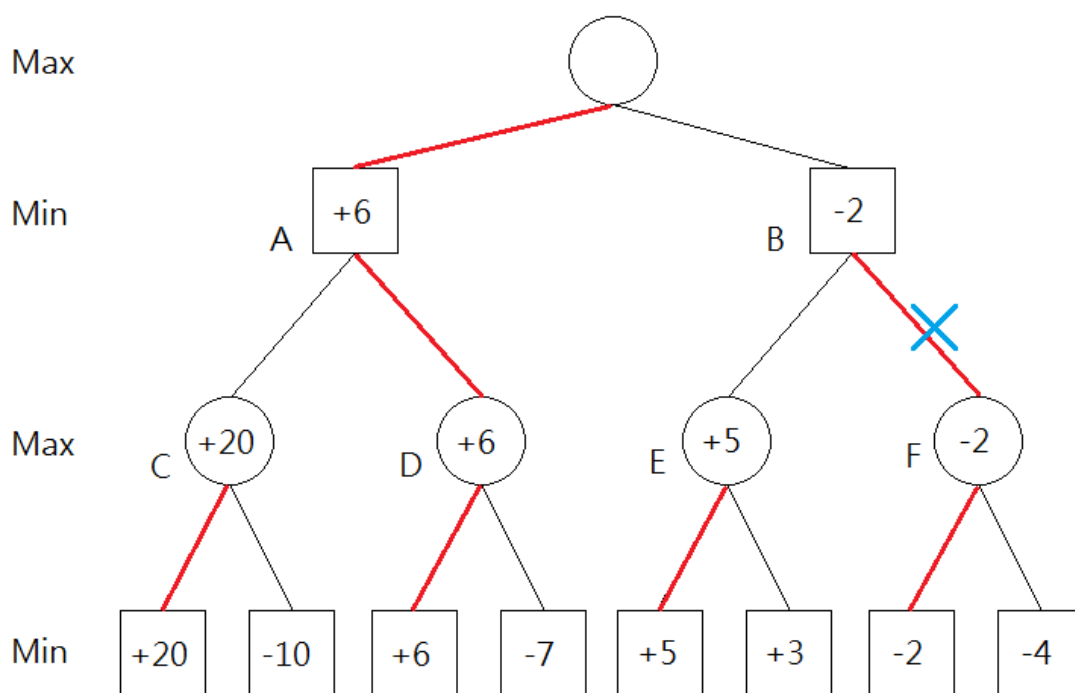


圖 5 Alpha-Beta 樹示意圖

我們在原始 Mini-Max Search 演算法中增加了 Alpha、Beta 兩個參數，分別表示目前我方、敵方已經找到的最佳走步分數，修改後的虛擬碼如表 2 所示。注意在遞迴呼叫的部分把 Alpha 與 Beta 做變號交換位置以適應 NegaMax 永遠以當前走子方為視角的原理。隨著搜尋的進行，Alpha 值可能越來越高，Alpha 與 Beta 所含括的範圍稱作 Alpha-Beta Window，這個 Window 限定了節點的分數必須落在這個區間內，若有分數超過 Beta，則發生 Beta Cut，其後所有分支都不用搜尋；如果所有分支分數都小於 Alpha，則會回傳小於 Alpha 的值(也就是大於上一層的 Beta)，因此上一層便會發生 Beta Cut。

```

int AlphaBeta(int depth, int alpha, int beta)
{
    if (depth == 0)
        return Evaluate(); //到達限制搜尋深度，回傳審局分數
    int best_score = -INF; //最佳分數設為負無限
    while (moves left) //嘗試當前盤面所有可能著法
    {
        MakeMove(move); //執行著法
        //減少一層的深度，遞迴演算盤面
        int score = -AlphaBeta(depth-1, -beta, -alpha);
        UndoMakeMove(move);

        if (score > best_score)
        {
            best_score = score; //更新最佳分數
            if (score > beta)
                return score; //Beta Cut
            alpha = MAX(alpha, score); //更新 Alpha bound
        }
    }
    return best_score;
}

```

表 2 Alpha-Beta 演算法的虛擬碼

仔細觀察圖 5 可以發現，我們是在確認 E 節點分數比 A 節點小才能捨去其後的節點，因此像 A 節點這樣擁有較高分數的節點越早被發現則越有機會砍掉更多的分支。整個 Alpha-Beta Search 運作的過程中便是使 Alpha 值不斷提升、縮小 Window 的過程，因此越早發現最佳走步可以越早讓 Alpha-Beta Window 縮到最小，讓其後的節點更容易發生 Beta Cut，達到更高的效能。因此 Alpha-Beta Search 比起 Mini-Max Search 的效能提升非常仰賴走步生成的排序機制。

Alpha-Beta Search 在最佳狀況(即所有節點展開的第一個走步便是該節點的最佳走步，其後的節點的第一個子節點搜尋完後便 Cut 掉後續所有節點)，其時間複雜度比起 Nega-Max Search 可以從 k^d 降為 $k^{d/2}$ ，因而大幅提升效能，而實際狀況則介於兩者之間，畢竟電腦下棋就是一個排序走步的過程，如果走步排序機制完美，便不需要搜尋了。

2.3.3 PVS

在 Alpha-Beta 搜尋樹中，若有一個路線從搜尋樹的葉節點到根節點都被選為最佳的分支，則這條路線就被稱為 PV 路線。PVS (Principal Variation Search) 演算法【6】，是經由 Alpha-Beta 演算法選擇 PV 路線的特性改良而來。

在 Alpha-Beta 搜尋樹中，會有三種類型的節點，如下做介紹：

1. Alpha 節點：Alpha 節點其下面的子節點回傳的分數都小於 Alpha 值，表示這個節點對於要走子的一方局勢十分壞，該節點回傳的分數必定會超過上一層節點的 Beta 值，並發生 Beta Cut，即對於要走子的一方有其它更好的選擇而絕對不會讓盤面演變至此。又因為此節點必須搜尋其下所有子節點以確認分數都小於 Alpha，因此又稱 All 節點。
2. Beta 節點：Beta 節點其子節點至少會有一個分支回傳大於 Beta 的分數，表示盤面對要走棋的一方十分好，但上一層的對手有其它走步可以避開這樣的盤面，因此進行 Beta Cut，其後的走步都不需要檢查了。又因為此節點只要找到一個子節點回傳大於 Beta 的值即可發生 Beta Cut，因此又稱 Cut 節點。

3. PV 節點：PV 節點的子節點至少有一個分數會大於 Alpha，並且沒有分支的分數會大過 Beta，意即 PV 節點為程式預期雙方會出走的走步。PV 節點必須搜尋所有的子節點，Root 必為 PV 節點。

圖 5 中的 A、C、D 為 PV 節點、E 為 Alpha 節點、B 為 Beta 節點，實際上當 F 節點被 Cut 掉後，B 節點會回傳+5，然而這不影響整體搜尋的結果，Beta 節點表示的是一個 Upper Bound，也就是說 B 節點的子節點如果全部搜尋完的話分數必定不會大於+5。

PVS 便是假設每個節點的第一個搜尋到的子節點必定是 PV 節點，意即假設其後的子節點分數都不會比第一個好，因此對第一個子節點使用傳統的 Alpha-Beta Search，然後利用得到的分數 value 對其後的子節點使用 (value, value+1) 的 Window 參數進行搜尋，這是一種「猜測」，對於其後的節點只關心分數是否會大於第一個分支，而不需要知道實際分數，這個 Window 又稱為 Zero Window。如果測試後得到的分數小於 value+1，則表示猜對了，該分支的分數不會比第一個分支好，且由於 Alpha-Beta Window 極小，因而 Cut 掉了大量的節點而大幅提升效能；如果得到的分數大於等於 value+1，則表示猜錯了，該子節點的分數大於第一個子節點，因此必須花費額外的成本重新以正常的 Window 搜尋這個子節點，以得到準確的分數，然後再度對其後的子節點猜測分數，PVS 的虛擬碼如表 3 所示。


```

int PVS(int depth, int alpha, int beta)
{
    if (depth == 0)
        return Evaluate(); //到達限制搜尋深度，回傳審局分數
    int best_score = -INF; //最佳分數設為負無限
    while (moves left) //嘗試當前盤面所有可能著法
    {
        MakeMove(move); //執行著法
        int score;
        if (is first move) //如果是第一個著法，正常計算
            score = -PVS(depth-1, -beta, -alpha);
        else
        { //使用 (Alpha, Alpha+1) 的 Zero Window
            score = -PVS(depth-1, -alpha-1, -alpha);
            //結果若大於 Alpha，則表示猜錯必須重新搜尋
            //若大於 Beta 則發生 Beta Cut 不必再重新搜尋
            if (score > alpha && score < beta)
                score = -PVS(depth-1, -beta, -alpha);
        }
        UndoMakeMove(move);

        if (score > best_score)
        {
            best_score = score; //更新最佳分數
            if (score > beta)
                return score; //Beta Cut
            alpha = MAX(alpha, score); //更新 Alpha bound
        }
    }
    return best_score;
}

```

表 3 PVS 演算法的虛擬碼

與 Alpha-Beta Search 相同，如果走步排序完美(特別重要的是最佳走步必須排在第一個)，可以最大化的提升效能，因為當最高分的節點先被計算出後，

其後的節點可以用 Zero Window 快速驗證搜尋過。通常來說，PVS 在同樣的搜尋時間下可以提升 2~3 層搜尋深度，可說是非常可觀。

另外 PVS 也是非常適合平行化的搜尋演算法。在搜尋完第一個分支得到分數後，便可以對其後的所有分支平行搜尋，因為其後分支都是使用 Zero Window，因此平行運算不會因為 Alpha 值不斷上升的影響而失去 Window 縮小帶來的速度提升，只有在其中一個分支傳回大於等於 value+1 的分數(猜錯)時，必須停止所有的 Thread 重新搜尋。本研究沒有對平行化部分進行更深的探討。

2.3.4 水平線效應、寧靜搜尋

如果在搜尋樹到達設定的層數便停止搜尋傳回審局分數，則在葉節點的地方常常有可能是尚有吃子或被將軍(即明顯有對策)的盤面，而我們卻因為設定的搜尋層數到了而被迫停止推演接下來的發展因而錯估情勢。例如說在葉節點我方吃了對方一隻馬，因為到達搜尋設定層數因而沒有發現在下一步我方反而有一隻車會被吃掉，程式因而認為這步非常好而錯判情勢。這種情形稱作水平線效應 (Horizontal Effect)。

解決水平線效應的方法必須於一些特定盤面在基礎的層數外做額外的延伸。最常見的延伸策略為「寧靜搜尋」，此法為展開盤面上所有的吃子走步至盤面穩定才回傳審局分數，如此一來可以在複雜的連續吃子中準確的評估形勢。

另外考慮一種情況是可能的對策為唯一或是走步自由度大減的情形，例

如說解將步通常只會有少數幾步；又如換子走步，為走步選擇極少而且必須應對的盤面，由於這類型的盤面通常其應對方式十分單調(意即任何對於象棋稍微有點概念的人或程式看到這樣的盤面通常都會做出正確且單調的回應)，因此實際上在這個盤面的這一層的搜尋是沒有得到任何資訊的，此狀況會讓搜尋樹展開相同層數下實際得到的資訊下降。在實際的搜尋樹中，Alpha-Beta 演算法會嘗試著拖延不好的事件至搜尋限制層數之外而回傳一個不是太差的分數，這也是水平線效應的一種。舉例來說，如果有一個著法會導致你失去一隻馬，Alpha-Beta 演算法會試圖找到一系列迫著迫使對方必須立刻應對(比方說將軍或是捉車)，進而把真正失去馬的盤面拖延至搜尋限制層數外而認為盤面狀況不差，即使最終還是會失去馬。

圖 6 舉出了另一種緩著(當自己快要被將死時不斷的送子來阻擋)，圖中黑方已無力阻擋紅方車的殺棋，但在到達這個盤面之前的情況，在較淺層的搜尋下卻有可能無法發現殺棋。

上述兩種類型的水平效應是比較不容易被發現的，也可以說是 Alpha-Beta 演算法的一大缺失，它會導致在某些路線上獲得的實際資訊量大減，因此必須要針對這類型的分支單獨增加搜尋深度，以避免搜尋樹退化。

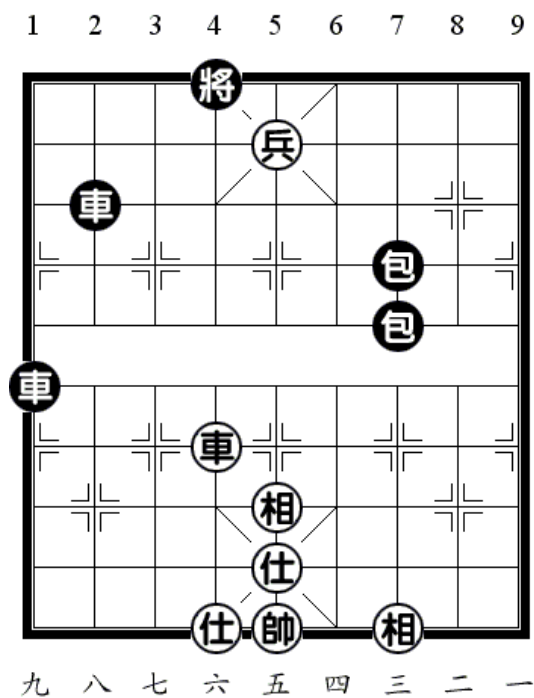


圖 6 輪黑，紅方在 8 步後能將死黑方

2.4 資料結構

本章將介紹象棋常用的棋盤表示法以及 Transposition Table。

2.4.1 棋盤表示與走步生成


棋盤表示通常以雙向的 Piece-Square Table 來記錄，該 Table 是兩個一維陣列，分別記錄棋盤上的格子裡有什麼棋子以及各個棋子在棋盤上的位置，象眼使用了 16x16 的 Square-Piece Table (Mail Box) 記錄棋盤，Piece-Square Table 以長度為 48 的一維陣列使用其中 16~47 記錄各個棋子所在位置，相同的兵種有不一樣的 Index：例如兩隻紅仕擁有不同的 Index。

走步產生時可使用 Piece-Square Table 查出棋子的位置，再以 Square-Piece

Table 檢查棋盤狀態以決定每種棋子的合法走步。合法走步的計算通常是在程式啟動時預先生成棋盤狀態對應棋子的走步並建表預存，往後計算時只須依據棋子所在位置查詢相對應的表然後考慮盤面資訊如棋盤的空格以及吃子與否即可省去大量計算時間。

而車炮長距離移動的走法需要循序的檢查棋盤被阻擋的狀態，炮要檢查炮架、馬與象則需要檢查馬腿與象眼，都是比較花時間的檢查，而吃子必須要判斷目標是否是對方的子，因此走步產生是程式中相當花時間的部分。

對於車與炮的走步產生，象眼使用了 BitFile、BitRank 的結構，如圖 7 的範例盤面，以及圖 8、圖 9 的 BitFile、BitRank 結構，每一縱列與橫列都是一個 2 進位資料，每個格子代表一個 bit，bit 為 1 代表裡面佔有棋子，圖中以黑色部分表示該 bit 為 1。



		相		帥	仕	相		
俥				仕				
			炮				炮	
		兵	傜					兵
兵			車			馬		
				兵		卒		
卒								卒
			士	象				
車				將		象		

圖 7 範例盤面

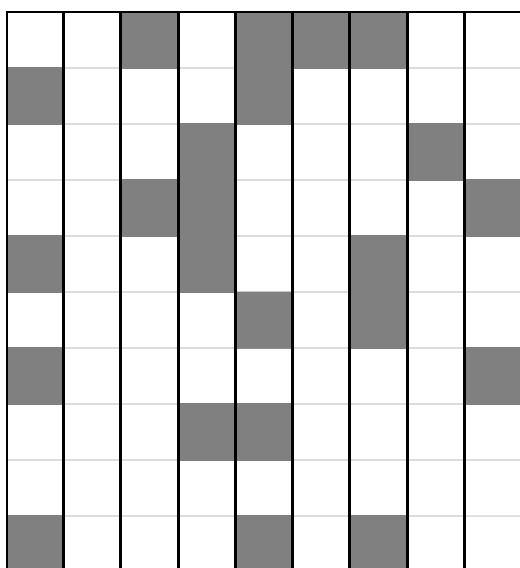


圖 8 BitFile 結構

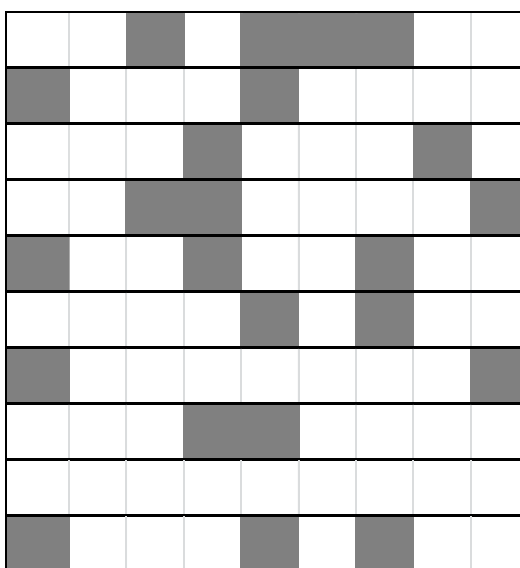


圖 9 BitRank 結構

當要產生車或炮的合法走步時，取出棋子所在位置所對應的 BitFile 與 BitRank，然後以此 2 進位資料為 Index 直接去查詢走步預生成表格，可以得到走子步資訊，吃子步則需要檢查目標點是否為敵方棋子。

2.4.2 Transposition Table

Transposition Table (同形表)是用來記錄已經搜尋過的盤面資訊，包含審局分以及最佳走步等，可以在以後遇到相同盤面時直接查詢而省去重複搜尋相同盤面的時間。

Transposition Table 一般是把盤面資訊經過 Zobrist Hash 得到一個 index 對應到紀錄區。Zobrist Hash 是一種快速產生 Hash Index 的算法，對於每一個棋子在棋盤上的每一個位置都有一個代表的亂數與之對應。當給定一個盤面時，把這些棋子一位置所對應的亂數全部經過 XOR 運算得到的結果即為其 Hash Index。Zobrist Hash 的優點是：利用 XOR 運算的可逆性，當盤面發生小

變化時(走子、吃子)，只需把走動的棋子其 From 與 To 兩個位置的亂數對當前的 Hash Index 各做一次 XOR 運算即可得到新的 Hash Index，而吃子只需對被吃子在原位的亂數多做一次 XOR 即可得到結果(即提取棋子)，因為盤面發生變化時不需要重新計算所有棋子，而且以 Index 查表是 $O(1)$ 的複雜度，因此 Zobrist Hash 法效能非常高，特別適合下棋程式。

而如果有兩個不同盤面剛好擁有相同的 Index，會被分配到同一個 Table 的位置，稱作 Collision (碰撞)。我們因為沒有紀錄完整盤面資訊，因此會從中取出錯誤的資料進而導致搜尋結果錯誤。改良的方法是加長 Zobrist Hash 中棋子一位置亂數的長度以降低 Collision Rate，多出來長度的部分當作盤面資訊存入 Transposition Table 中做為驗證碼 (Verification code)，當查詢其中資料時必須比對驗證碼是否正確，如果驗證通過則假設其資訊正確，由於此時 Collision Rate 已經非常低，因此若還是 Collision 則忽略其造成影響，如果驗證碼是 32 位元，則 Collision Rate 可以降為原來的 $\frac{1}{2^{32}}$ ；若驗證失敗則對於撞到的位置必須選擇取代的策略，常用的策略有深度優先以及始終覆蓋兩種，深度優先可以確保較準確的資訊會留下來；始終覆蓋的想法來自於相同的盤面常常是近期搜尋過的盤面，因而希望保留最近期的資料。

而我們使用兩層的結構分別做深度優先的取代策略以及始終覆蓋的策略【7】，這樣可以同時兼顧其準確度(深度優先)以及命中率(始終覆蓋)，圖 10 為 Transposition Table 結構示意圖。

另外利用 Transposition Table 也可以檢測重複盤面，以利用長將或長捉判負規則迫使對方產生禁著而必須變著，因而創造優勢，相關方法將在 3.2.5 節

說明。

Index	分數	深度	驗證碼
0			
1			
2			
...			
n			

圖 10 Transposition Table 示意圖

2.5 審局函數

審局函數通常包含了子力價值、位置分、自由度、棋盤控制以及特殊棋型、王的安全性等等，把上述所有項目做線性結合所得的分數就是審局分數。

審局亦有靜態審局以及動態審局，靜態審局就是各個項目的評分標準在棋局的進行都是固定的；動態審局則是評分標準會隨著棋局的局勢而有所調整。例如說兵卒在過河前的價值是很低的，但過河後威力會大增，並且配合其它攻擊子力可以造成更大的攻勢；炮在開中局的影響力會大過殘局等等。很顯然動態審局是更準確的，但相對的設計上以及計算成本也高得多。

審局是在搜尋樹的最底層執行的，因此是下棋程式運行最花時間的部分，審局函數的準確度與運算成本無法同時兼顧，審局函數運行過久會導致搜尋層數的降低，如何取捨審局準度與搜尋層數也是一個困難的議題。

要加速審局函數必須要有好的資料結構配合，並且配合著專家知識結合程式設計能力。審局函數涵蓋了下棋程式所有的知識，可以說是程式的核心，

是影響程式棋力最關鍵的部分，好的審局函數配合搜尋演算法才能引領程式走向勝利。

象眼的審局函數包含了子力價值、位置分數、特殊棋型以及自由度，並且在開始搜尋前預先判斷盤面情勢，包含開中殘局程度、雙方攻勢等，並對上述評分項目的參數相應做調整，但因為攻勢與開中殘局程度會隨著搜尋進行而變化，因此參數調整動作應該在葉節點審局時做調整，但因為調整成本很高，因此在搜尋深度不深的情況下可以免強接受。

2.6 後續發展

與西洋棋發展多年的大量研究比較起來，象棋研究可以說是鳳毛麟角，因此除了參考西洋棋的發展外，象棋獨有的部分可以說是還有很多研究空間的。例如說象棋特有的將帥九宮格、士象的防守、炮與馬的獨特性，以及開中殘局的策略，與人類比較起來，電腦不容易判斷「勢」與「先」，人類玩家常常會使出棄子求先的妙著，也就是長遠的布局。另外在殘局的策略對於不同的棋型其目標都不相同，例如優勢方要盡早破士象、劣勢方要盡力守和而不要一味地謀求無意義的進攻而失去良好的防守陣型，一般通用的審局函數很難適用於所有殘局狀況。這些問題目前象棋研究尚未對此有所發展。

第3章 方法與步驟

程式的棋力主要由兩個因素控制：搜尋深度與審局準確度。加強這兩個項目便可以提升程式的棋力。

搜尋深度的提升可以使用 Pruning 策略(砍搜尋樹分支或減少部分分支搜尋層數)來減少搜尋的節點數或單純改善程式效能以求相同時間下能搜尋得更深。審局準確度則需要象棋的專家知識與程式結合。我們將針對這兩個部分進行改良以加強棋力，本章將詳述相關改進方法。

我們的實驗測試平台為 Intel Core i7 3970X @3.5GHz、DDR3-1600 64G。

3.1 資料結構的改進

首先是針對速度上的提升，程式的效能瓶頸通常發生在走步產生以及審局函數，我們必須要有相對應的資料結構搭配以改善效能。

3.1.1 適用於象棋的 BitBoard

與迴圈處理一維陣列比較起來，電腦更善於一次進行大量的位元運算，即邏輯指令。傳統產生一個盤面走子方的合法走步需要循序讀取與判讀棋盤上的一些位置，十分的花時間。現在在西洋棋上已經被大量使用的一種新資料結構叫做 BitBoard (位棋盤)【8】，以二進位資料中 bit 的 0 或 1 來表示棋盤上的格子有無棋子，西洋棋棋盤有 64 個格子，正好與現今電腦能一次處理的

單位資料 64 bits 相符，因此每個棋子以一個 64 bits 的資料表示其位置，並相互進行邏輯運算可以快速檢查棋盤以產生合法走步或其它應用。

我們將此概念應用在象棋棋盤上，但因為象棋棋盤有 90 個格子，無法直接像西洋棋一樣簡單表示，象眼的解決方式為使用 BitFile 與 BitRank，對於每一行與每一列都使用條狀的 bit 結構表示，可用於加速產生炮與車的走步(參閱 2.4.1 節)，但功能也僅限於此，其它棋種並不適用，無法加速審局函數計算，而且必須隨著搜尋樹展開更新多筆資料，無法完全發揮效能。

以象棋 90 個棋盤位置以楚河漢界分邊，我們使用兩個 64 bits 共 128 bits 的結構來表示棋盤，如圖 11 所示。

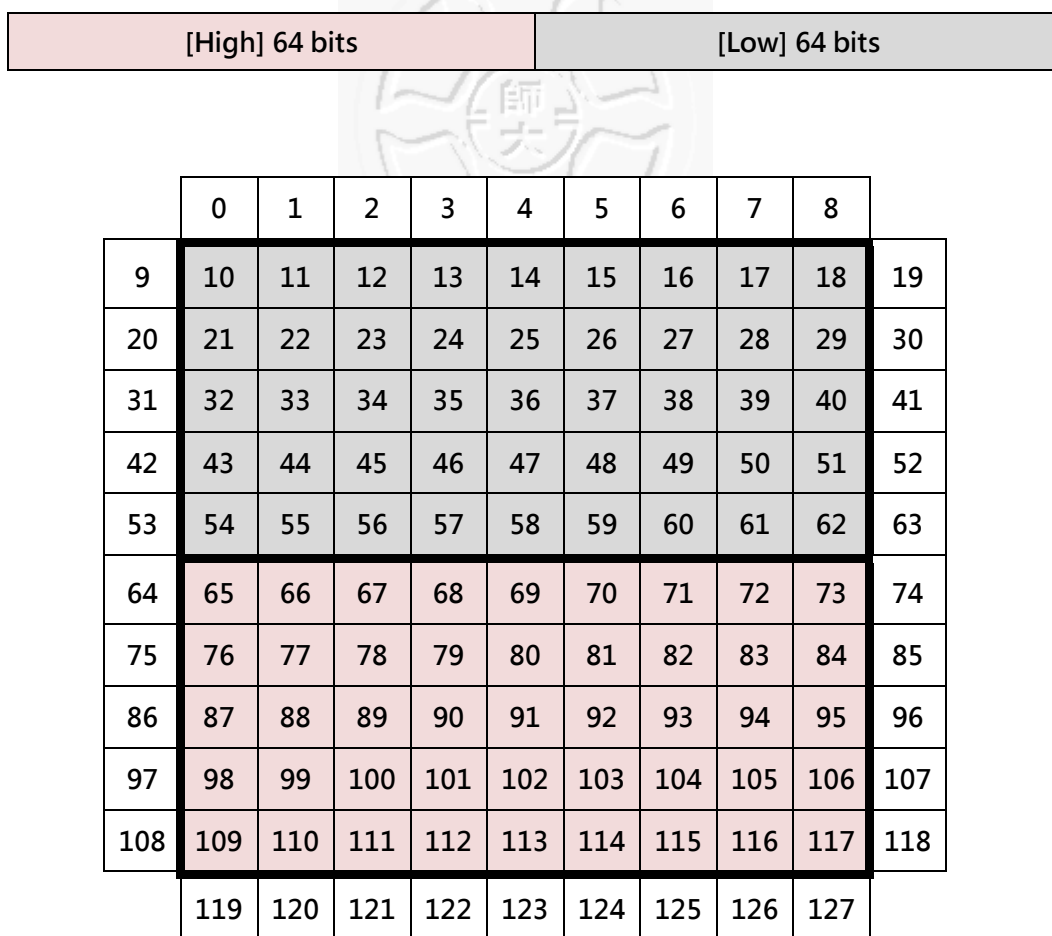


圖 11 128 bits 象棋棋盤 BitBoard 示意圖

由高低位組成的 128 bits 結構使用其中 90 個 bits 表示棋盤 90 個位置。當棋盤的格子中有棋子時，該結構對應的 bit 就會被設為 1，圖中也表示棋盤格子的編號，0~63 為 Low、64~127 為 High。外圍為牆壁，用以快速處理邊界。

分成 Low 與 High 可以快速檢查棋子是否過河，如果是棋子位置編號只需檢查第 6 個 bit 是否為 1；如果是 BitBoard 則只需檢查 Low 或 High 是否不為零即可。

3.1.2 BitBoard 的棋盤表示法

圖 12 為初始盤面代表紅方所有棋種及 Occupied 的 BitBoard 表示，Occupied 為所有棋種的聯集，是走步生成的依據，將在 3.1.6 介紹。

這些 BitBoard 資訊將隨著搜尋演算法對棋局的演示而不斷的更新維護。使用了此結構即取代 Piece-Square Table，但仍然保留 Square-Piece Table，因此更新成本的增加只有在 Occupied 的部分。

在許多應用上我們需要得到棋子位置編號的訊息(原始 Piece-Square Table 的功能)，因此必須把 BitBoard 結構中的 bit 轉換成位置編號，需要使用 LS1B 指令，該指令將在 3.1.4 節介紹。

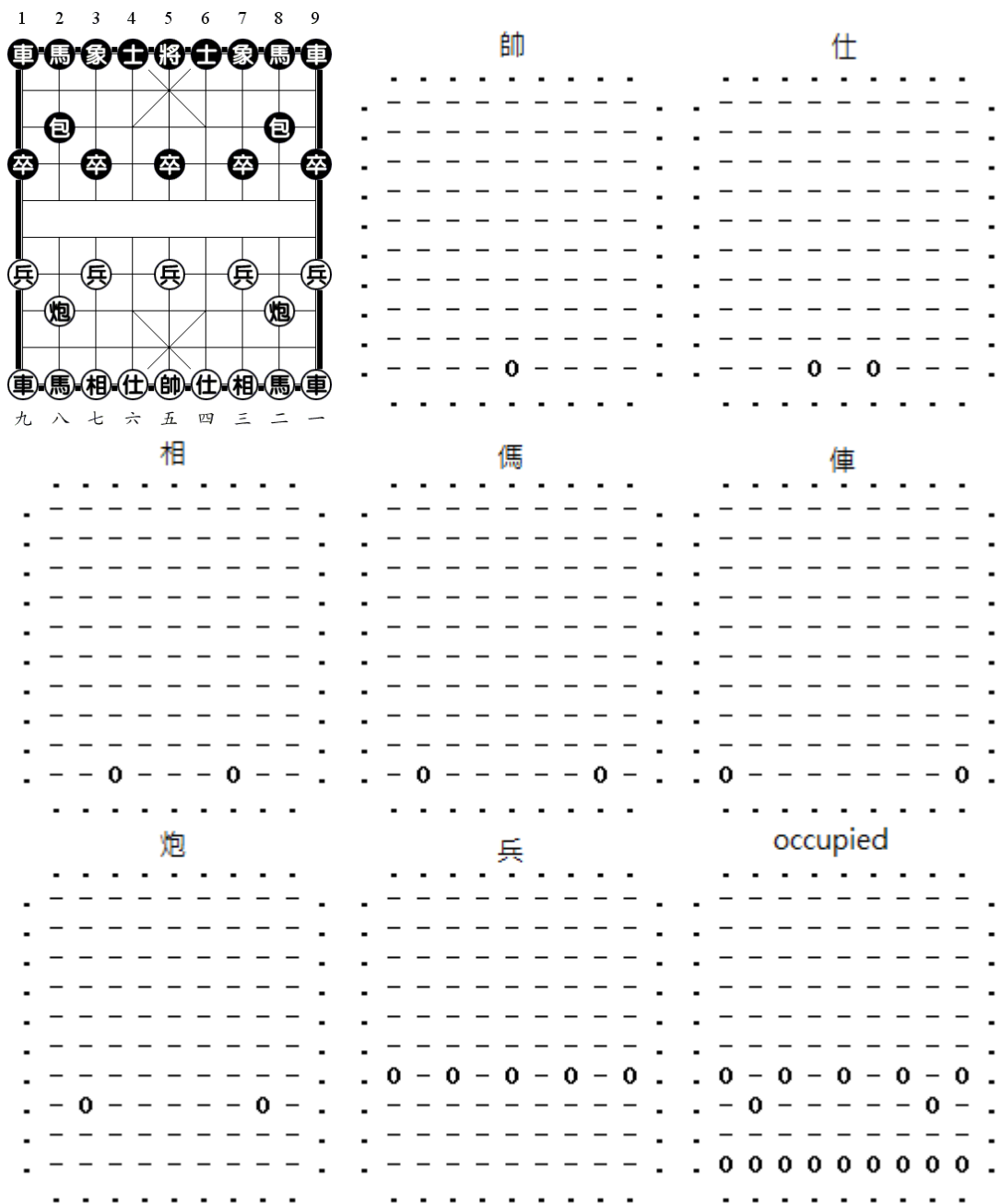


圖 12 初始盤面紅方各個棋子對應的 BitBoard 示意圖

3.1.3 棋子表示法

表 4 表示棋子編號的表示法，0~2 bit 為棋子種類(Type)，3~4 表示棋子所屬方，01 表示紅方、10 表示黑方，00 表示空白。

表 5 列出了各個棋種對應的編號，其中 0~7 不使用，輪紅下棋時對應的值(side)為 8，輪黑為 16。舉例來說，黑方炮的編號二進位表示為 10110，前面 10 表示黑方，後面二進位 110 是十進位的 6—炮的編號。

其中棋子類型為 0 的部分為代表該方所有棋子聯集(Occupied)的編號，在走步產生以及棋型偵測上佔有十分重要的地位，在後面章節會詳細介紹。

黑	紅		Type	
---	---	--	------	--

表 4 棋子編號意義，最右邊為 bit 0

棋子類型	0	1	2	3	4	5	6	7
輪紅=8	All	帥	仕	相	傴	俥	炮	兵
編號	8	9	10	11	12	13	14	15
輪黑=16	All	將	士	象	馬	車	包	卒
編號	16	17	18	19	20	21	22	23

表 5 棋子編號方式


如此編號方式特別便於取出以及判斷棋種，例如要取出我方的炮只要使用 **side+6** 即可，不必去判斷我方是紅方還是黑方；走步產生時也只需要使用 **side+kind** 的組合便可以讓紅黑雙方共用走步產生程式碼，提升程式可維護性；要取出對方的王時使用 **(side^0x18)+1**，使用 XOR 翻轉代表 side 的 3~4 bit 並加上棋種編號即可快速取得；要檢查一個棋子 pc 是否是我方的只要使用 **(side&pc) != 0** 即可；檢查某一方 side 的棋子於棋盤位置 sq 是否過河則使用 **(sq^(sd<<2)) & 0x40** 快速測試。

3.1.4 Intrinsic 函數

如果要在兩個 64 位元的資料間進行操作可以使用 C++ 的傳統語法來達成，但需要多個步驟因而導致效能的下降，這也是象棋完全使用 BitBoard 困難的原因之一。而我們使用了 intrinsic 函數解決這個問題。

intrinsic 函數是編譯器提供的 CPU 低階指令呼叫方式。CPU 包含了多種指令集，除了常用的 x86、x64 外還有 MMX、SSE、AVX 等。一般在編譯 C++ 的程式時，編譯器通常只會使用到極少種的 CPU 指令，實際上 CPU 提供了大量的原生指令，通常可以做到一般 C++ 語法很難表達的操作，進而大量提升效能。使用 intrinsic 函數便能以函數呼叫的方式使用這些指令，但必須 Compiler 要支援。

如果要在兩個 64 bits 的資料間做 shift 的指令，以 C++ 的語法要實現是十分複雜的，表 6 表示從 B 往左 shift 40 位元到 A 所需要的指令比較。



一般 C++ 語法	使用 intrinsic 函數
<pre>A = (A << 40) (B >> 64-40); B <<= 40;</pre>	<pre>A = __shiftleft128(B, A, 40); B <<= 40;</pre>

表 6 Intrinsic 指令與傳統 C++ 指令比較：128 bits Shift 為例

一般 C++ 語法至少會產生 4 條 CPU 指令，而使用 intrinsic 卻只需要 2 條 CPU 指令，在指令延遲減少、資料相依性降低的情況下可以有效增加整體速

度。

另外要從 BitBoard 的「佔有」意義轉換成位置編號的訊息必須使用 MS1B、LS1B (Most/Last Significant 1 bit—取得最高、最低位元為 1 的 Index)，也是 BitBoard 最常使用的操作。以 LS1B 為例，傳統作法必須獨立出最低位元為 1 的資料，然後對該資料做 \log_2 運算以取得該 bit 代表的編號，當然 Log 運算十分慢，亦有作法使用 Magic Hash 查表以加速【9】。但 LS1B 以及 MS1B 都有對應的原生指令可以使用：_BitScanReverse64 與 _BitScanForward64，如圖 13 所示。此例中 MS1B 為 60、LS1B 會得到 1。



圖 13 MS1B、LS1B 示意圖

使用 Magic Hash 的 C++ 語法	使用 intrinsic 函數
<pre>t = x & -x; //取出 LSB //計算 Magic Hash 的 Index hIndex = t * Magic Number >> S; //查表取得結果 result = HashTable[hIndex];</pre>	<pre>//使用 BSF 指令，一步完成 _BitScanForward64(&result, x);</pre>

表 7 Intrinsic 指令與傳統 C++ 指令比較：LS1B 為例

表 7 表示 LS1B 所需要的指令比較，使用 Magic Hash 的 C++ 語法至少會產生 5 條 CPU 指令，且需要查詢記憶體資料，而使用 intrinsic 卻只需要 1 條 CPU 指令，而 BSF (BitScanForward) 指令只需要 3 個 CPU 週期，與一個乘法運算相當，速度比傳統做法快上不少。再者這是 LS1B 的例子，取出 LSB 有

較快的做法 $x \& -x$ ，若是 MSB 則運算更加繁雜，如表 8 所示，更別說還有後面 Magic Hash 的計算，而使用 intrinsic 函數仍然是只要一個指令解決。

```
U64 MSB(U64 x)
{
    x |= x >> 32; x |= x >> 16; x |= x >> 8;
    x |= x >> 4; x |= x >> 2; x |= x >> 1;
    return (x >> 1) + 1;
}
```

表 8 MSB 的 C++ 程式碼【9】

如果能善用 intrinsic 函數，可以有效的利用原生硬體指令的高效能，而且程式碼也較為簡潔易懂。而由於我們 BitBoard 中 High 與 Low 大部分的運算都是獨立不相關的，因此另外尚有許多 SIMD (Single Instruction Multiple Data) 指令集可以利用，本篇不再多介紹。

3.1.5 BitBoard 的基本操作指令

BitBoard 中常用的基本指令除了上一節介紹的 Shift、MS1B、LS1B 外，還有 AND、OR、NOT、XOR 等邏輯指令以及 Population count、BitSet、BitClr 等。

邏輯指令 AND、OR、NOT、XOR 只需對 High 與 Low 分別作對應的運算即可，BitSet、BitClr 則分別使用 OR 及 AND 指令進行；Population count 則使用 intrinsic 指令 `__popcnt64`，分別對 High 與 Low 計算後相加，表 9 列出了 BitBoard 基本指令與實際 C++ 程式碼對應，其中因為棋盤只使用了中間 90 bits，因此 NOT 指令使用棋盤中 90 bits 為 1 的數字做 XOR 運算，如圖

14 所示。而 BitTable 記錄了只有一個 bit 為 1 的陣列，於 BitSet、BitClr 使用，

圖 15 為 BitTable[43]的 BitBoard 示意圖。

在本資料結構的設計中，BitBoard 相關的運算於 High 與 Low 的部分幾乎沒有資料相依性，可以充分利用 CPU 的 Pipeline 發揮性能。

基本指令	C++ 程式碼
AND、OR、XOR 以 AND 為例	A.High & B.High A.Low & B.Low
NOT	X.High ^= ~0xFFC0180300600C01 X.Low ^= ~0x80300600C01803FF
Population count	__popcnt64(X.High) + __popcnt64(X.Low)
BitSet(n)、BitClr(n) 以 BitSet 為例	X.High = BitTable[n].High X.Low = BitTable[n].Low

表 9 BitBoard 基本指令

```

0x3fe7fcff9ff3fe 7fcff9ff3fe7fc000
. . . . .
. 0 0 0 0 0 0 0 0 0 0 .
. 0 0 0 0 0 0 0 0 0 0 .
. 0 0 0 0 0 0 0 0 0 0 .
. 0 0 0 0 0 0 0 0 0 0 .
. 0 0 0 0 0 0 0 0 0 0 .
. 0 0 0 0 0 0 0 0 0 0 .
. 0 0 0 0 0 0 0 0 0 0 .
. 0 0 0 0 0 0 0 0 0 0 .
. 0 0 0 0 0 0 0 0 0 0 .
. 0 0 0 0 0 0 0 0 0 0 .
. 0 0 0 0 0 0 0 0 0 0 .
. 0 0 0 0 0 0 0 0 0 0 .
. . . . .

```

圖 14 棋盤合法位置的 BitBoard，O 部分表示該 bit 為 1

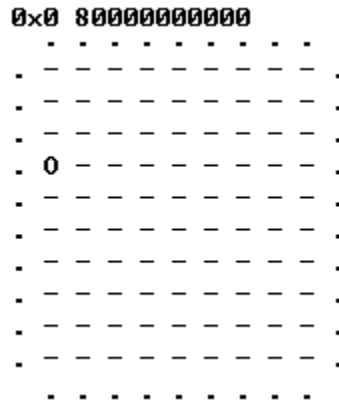


圖 15 BitTable[43]的 BitBoard 示意圖

另外在使用 BitBoard 於象棋中之前必須預先建立 BitTable、BitRow、BitCol 陣列以利運行，其分別代表棋盤中每個格子、Row 以及 Column 的 Mask，如圖 16 為 Col[3]之 BitBoard 示意圖。



圖 16 Col[3]的 BitBoard 示意圖

截至目前，我們已經準備好象棋 BitBoard 的基本操作與資料了，接下來便可以開始進行實際的應用，發揮其強大的威力。

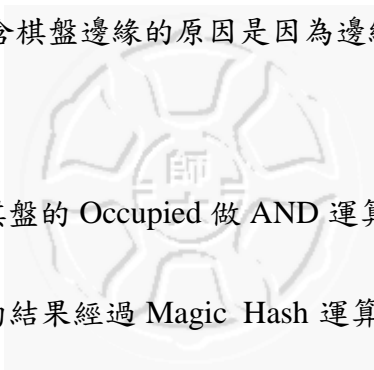
3.1.6 以 BitBoard 加速走步產生

走步產生也是下棋程式中效能瓶頸的一部分，如果使用一維陣列則必須循序地去檢查棋盤格子，相當花時間，而使用 BitBoard 便可以快速的檢查。

以圖 17 的範例盤面為例，若要產生車的走步，必須檢查其四個方向第一個能碰到的棋子的位置，BitBoard 的使用方法為：

1. LS1B 取出黑車的位置編號。
2. 查表取得該車十字方向的 Bit Mask。(圖 17)

十字 Mask 不包含棋盤邊緣的原因是因為邊緣是否有子不會影響以下流程計算的結果。



3. 將這個 Mask 與棋盤的 Occupied 做 AND 運算。(圖 18)
4. 做完 AND 運算的結果經過 Magic Hash 運算後得到一個 index，查表便可以取得車能走到的位置。(圖 19)
5. 其結果與「空白」的 BitBoard (即 Occupied 做 NOT 運算)做 AND 運算可得走子步(圖 20)
6. 其結果與對手的 Occupied 做 AND 運算可得吃子步(圖 21)
7. 5.與 6.得到的結果以 LS1B 配合 BitClr 便可以取出合法走子的目標。

		相		帥	仕	相		
俥				仕				
			炮				炮	
		兵	偶					兵
兵			車			馬		
				兵		卒		
卒								卒
			士	象				
車				將		象		

圖 17 車的 Mask

		相		帥	仕	相		
俥				仕				
			炮				炮	
		兵	偶					兵
兵			車			馬		
				兵		卒		
卒								卒
			士	象				
車				將		象		

圖 18 與 Occupied 做 AND 運算後的結果

		相		帥	仕	相		
俥				仕				
			炮				炮	
		兵	偶					兵
兵			車			馬		
				兵		卒		
卒								卒
			士	象				
車				將		象		

圖 19 查表得到的走步資訊

		相		帥	仕	相		
俥				仕				
			炮				炮	
		兵	傜					兵
兵			車			馬		
				兵		卒		
卒								卒
			士	象				
車				將		象		

圖 20 車的走子步

		相		帥	仕	相		
俥				仕				
			炮				炮	
		兵	傜					兵
兵			車			馬		
				兵		卒		
卒								卒
			士	象				
車				將		象		

圖 21 車的吃子步

與傳統的走步生成比較起來，使用 BitBoard 可以化簡一切循序繁瑣的檢查，比方說循序檢查馬腳的位置、炮架、是否是空格、吃到的是否是對方棋子等等，只需要以 Mask 取出棋盤的部分資訊查表並作 AND 運算即可，AND 運算可以平行的大量處理逐格檢查的動作，而且全部都是邏輯運算完成，可以省去大量的循序判斷式增進效能。

車炮的走子步規則完全相同，因此可以共用一張表，而吃子走步則必須要查不一樣的表，但上述的流程是完全一樣的。

將帥、兵卒、士的走步規則只依據自身所在位置，與棋盤狀態無關，因此只需根據自己所在位置查對應的表並跳至步驟 5 即可完成。

馬、象則需要檢查其馬腳處以及象眼處是否有棋子，因此在步驟 2 的部分使用不同的 Mask，其餘步驟均相同，圖 22 為紅方傜與黑方象的 Mask 示意圖。

		相		帥	仕	相		
俥				仕				
			炮				炮	
		兵	偶					兵
兵			車			馬		
				兵		卒		
卒								卒
			士	象				
車				將		象		

圖 22 馬與象其拐腳處的 Mask

另外在搜尋過程中經常需要判斷是否被將軍，用來檢查走步合法性，因為一個著法導致自己呈現被將軍狀態是不允許的。象棋中，若一方沒有合法著法則做負，包含棋子全部被堵死(圖 2)或無法解將。傳統一維陣列必須一一檢查對方合法著法中是否有一個著法的目標點為我方王，以及是否將帥對臉，若對方合法著法有 40 步則需要做 40 次檢查，十分費時。而我們可以利用 BitBoard 快速做檢查。

比較基本的 BitBoard 用法是產生對方所有子的合法走步 BitBoard (如圖 21)，然後將它與王的 BitBoard 做 AND 運算，若結果不為零，則表示被將軍，注意這邊不需要把圖 21 的吃子步 BitBoard 以 LS1B 拆出來，因為我們不需要知道目標位置的 Index，只需要知道它是否涵蓋了王的位置即可，因此 AND 運算可以快速達成此目標。此作法需要對對方能過河的兵種各做一次檢查以及檢查將帥對臉。

進階的做法是反向檢查，我們可以把目標王假設成各種兵種，然後產生

該兵種反向的吃子著法，若該反向吃子著法能吃到該兵種的棋子，則表示檢查成立。如圖 23 所示，若要反向檢查將是否被紅偶將軍，我們可以

1. 取得將在該位置反向偶的 Mask (圖 23)。
2. 以 Magic Hash 查表取得要檢查是否有偶的位置(圖 24)
3. 與對方偶的 BitBoard 做 AND 運算，若不為零則表示檢查成立。

如此做法對於紅方的兩隻偶從原本需要分別檢查兩隻偶變成只需要反向檢查一次，因為是反向檢查，第 1 步驟偶的 Mask 會與正常偶的 Mask 不相同。而相應地，對於炮、兵都是一樣的處理方式，對於兵原本要做五次檢查也是可以一次就完成，加速不少。

		相		帥	仕	相		
俥				仕				
							炮	
								兵
兵						馬		
				兵				
						偶		
			偶	象				
炮	兵			將	士	象		

圖 23 反向偶的 Mask

		相		帥	仕	相		
俥				仕				
							炮	
								兵
兵						馬		
				兵				
						偶		
			偶	象				
炮	兵			將	士	象		

圖 24 要檢查是否有偶的位置

比較特別的是，將帥對臉的檢查可以與車的檢查同時進行，因為其本質是完全相同的，步驟如上所示，在步驟 3 的部分改成對紅俥與帥 OR 在一起的 BitBoard 做一次 AND 運算即完成。

表 10 比較各種做法需要的檢查運算次數，上述進階作法能最大化發揮 BitBoard 的優勢。

	檢查運算次數	說明
傳統作法	約 40 次	相當於一手的平均分支因數
基本作法	12 次	帥、車馬炮各 2、兵 5 隻
進階作法	4 次	帥車、馬、炮、兵各 1 次

表 10 檢查將軍的實作方法比較

3.1.7 Magic Hash

3.1.6 節中我們提到了各種 Mask，Mask 與各種 BitBoard 做 AND 運算的實質意義是「取得 Mask 對應在 BitBoard 上的資訊」，這些取得的資訊將用於走步產生等用途。BitBoard 與 Mask 做 AND 運算的結果會讓我們想要取得的資訊分布在 Mask 對應的一些 bits 上，我們必須把這些 bits (部分棋盤資訊)「濃縮」成一個唯一的 index 以查表，這個動作稱作 Magic Hash，其公式為

$$(\text{Value.H} * \text{Magic Number1} \wedge \text{Value.L} * \text{Magic Number2}) \gg \text{Constant}$$

其中 Value 為 Mask 與棋盤資訊做 AND 運算的結果，兩個 Magic Number 與 Constant 為 64 bits 的數值，其配對可以把在定義域範圍內的所有 Value 唯一的對應到一個 index，其中 Mask 的 Population count 以 64 減去後即為 Constant 的值，以圖 17 的車在該位置為例，其 Mask 的 Population count 為 13 (代表我們想取得棋盤中 13 個位置的資訊)、Constant 值為 51，因此對應的 Table 大小為 2^{13} 個元素。以車為例，車在棋盤上不同位置都有不同的 Magic Number 與 Constant 的配對。

Magic Number 必須由試誤法不斷尋找，Constant 決定了對應域的 Table 要開多大，由於只要 Value 唯一的對應到 Table 上即可，因此可以開比較大的 Table，以減少對應域碰撞的機率。雖然比較浪費空間，但相對的 Magic Number 會比較容易找到。

而近期 Intel 發展了 AVX2 指令集，其中包含了 PEXT (位抽取)指令，其作用如圖 25 所示，利用原始資料與要抽取 bits 的 Mask，可以直接將其進行濃縮，產生最後的結果。這個動作與本節講的 Magic Hash 不謀而合，把盤面資訊與十字的 Mask 做 PEXT 運算可以直接得到最終的 index，省去了 AND 運算以及 Magic Hash 公式運算的步驟，如此更能提升效能，並且省去了尋找 Magic Number 的麻煩。但由於需要最新一代的 CPU 才有 AVX2 指令集，我們為了讓程式維持在不同電腦硬體間的良好相容性，因此沒有採用 PEXT 指令。

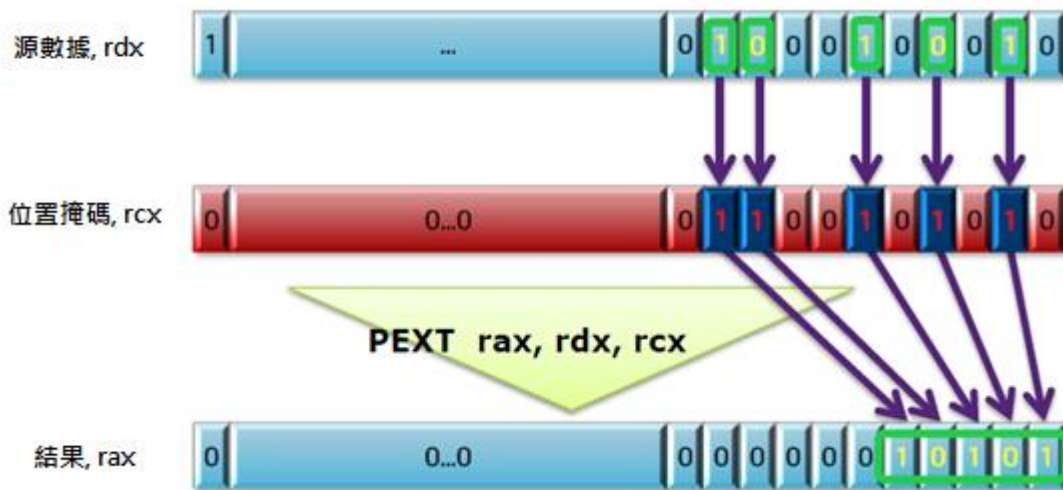


圖 25 AVX2 指令 PEXT 運作示意圖【11】

3.2 演算法改進

演算法是下棋程式中運作最為複雜的部份，雖然各種新算法與資料結構不斷被提出來，但在各種算法與資料結構間的複雜交互作用下，時常會藏匿一些錯誤或非預期的情況，我們到目前為止還很難深入瞭解所有細節(2.3.4節提到的水平線效應就是一例；與 Transposition Table 的交互作用產生的錯誤更是不好解決)，因此我們針對一些觀察出來的問題嘗試尋找改進方案。

3.2.1 走步排序策略

以 Alpha-Beta 為基礎的演算法十分仰賴好的走步排序策略來提升砍分支的效率(參閱 2.3.2)，象眼實作的走步排序策略共有 4 個，以下按照順位表示：

1. 記錄在 Transposition Table 中的最佳著法
2. 優良的吃子走步，以 MvvLva 排序
3. 2 個殺手著法 (Killer Heuristic)
4. 非吃子步與較差的吃子走步，以 History Heuristic 排序

Transposition Table 中的最佳著法是之前搜尋過當前盤面時記錄下的最佳著法，當對同一個盤面有更深的搜尋需求時，該著法理論上有非常大的機率仍然是最佳著法，因此優先順位最高，並且 PVS 的運作(2.3.3 節)非常仰賴這個策略。

MvvLva 是 Most valuable victim / Last valuable attacker 的縮寫，表示用最

小價值的子吃到最大價值的子應該要最優先，並同時考慮了如果被吃的目標是有保護的，則應該還要多考慮攻擊子會被反吃的因素。另外我們實驗發現，Mvv 是比 Lva 重要的，因此先以 Mvv 做排序，內部再以 Lva 排序，以此策略改良後，在實際對戰盤面上約可以減少 10% 以上的搜尋節點數。而一些乍看之下不好的吃子著法(例如用車吃一個有保護的馬)則留至走子步一起考慮。

接下來是殺手著法，也就是 Killer Heuristic，該結構記錄了在同樣深度的兄弟節點中造成 Beta Cut 的走子步，殺手的意義即在此。這個策略假設在相同搜尋深度下盤面通常是比較接近的，因此如果有一個著法在其兄弟盤面中造成 Beta Cut，則在當前盤面也造成 Beta Cut 的機率是比較高的，因此也要優先嘗試。但由於記錄這個著法的兄弟盤面不會完全與當前盤面一樣，因此著法有可能是非法步，需要先檢查著法是否合法。而殺手著法只記錄走子步，因為吃子走步會讓盤面發生巨大變化，這與 Killer Heuristic 的最初概念是相違背的。

最後是走子步與不好的吃子步，這部分以 History Heuristic 排序，History Heuristic 假設在一定的層數內、棋盤的「勢」不會發生太大的變化，因此在不同的深度下，其造成 Beta Cut 的著法仍然有可能是當前盤面好的著法，並以此依據進行排序剩下的走步。一般 History Heuristic 記錄的是一個著法 (From、To) 造成 Beta Cut 的次數，深度越深的記錄權重越高，然後在其它節點產生出的走步便參照該權重表進行排序。而基於與 Killer Heuristic 相似的前提假設，History Heuristic 一樣只記錄走子步。

圖 26 是 Killer Heuristic 與 History Heuristic 示意圖，Killer Heuristic 運作

在相同深度的兄弟節點中；History Heuristic 則運作在不同深度的節點中。該例中，若[馬 2 進 3]造成 Beta Cut，則在兄弟節點與不同深度的節點中該走步排序的優先權會提高。

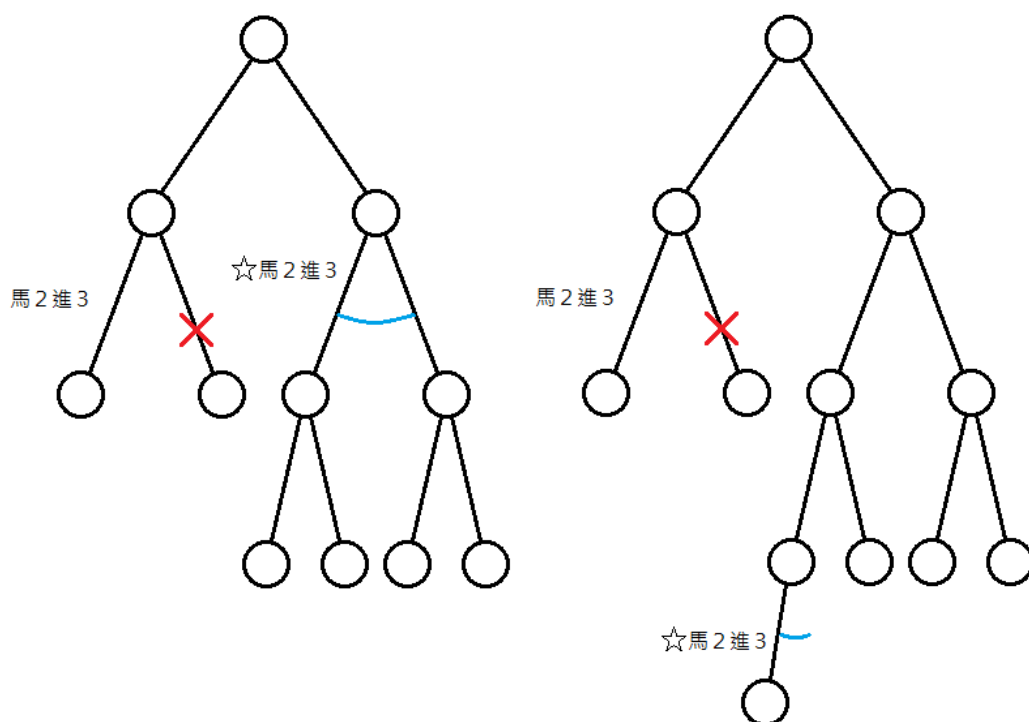


圖 26 Killer Heuristic (左)與 History Heuristic(右)比較

而我們發現，象棋的棋子在著法的 From、To 之間沒有絕對的關聯性。例如說車與炮的著法在只有 From 與 To (棋子位置變化)的資訊下是無法分辨的，顯然的這兩個兵種走一樣的著法造成的影響相差甚遠。

因此我們改良了 History Heuristic 的策略，從 From、To 改成 Pieces、To，記錄了什麼棋子移動到了什麼位置，如此一來便可以精確的記錄該走步對盤面的「勢」造成了什麼影響，更符合 History Heuristic 的精神，以此策略在實戰盤面中能減少約 7% 搜尋節點數，因為 History Heuristic 是順位比較低的走步排序策略因此效能增加沒有 MvvLva 來的多。

3.2.2 Late move reduction (LMR)

相較於吃子步，走子步對於盤面的影響是比較沒這麼劇烈的，在 2.3.4 節中，我們討論到了水平線效應會讓搜尋樹在含有迫著或緩著的分支路線得到的資訊量下降，因而造成棋力不穩定，通常配合延伸搜尋深度的作法以讓含有迫著或緩著的分支能得到相對應的資訊補償，然而延伸深度會大幅增加計算成本，且延伸的量不足的情況下無法完全補足減少的資訊量。

圖 27、圖 28 為水平效應以及使用延伸策略後的影響示意圖，左三角形為搜尋樹、右三角形為獲得的資訊量，水平線效應就是在某些含有迫著或緩著的盤面上實際獲得的資訊量是比較少的，而圖 28 使用了延伸策略後可以緩解部分分支資訊量的不足，但會造成搜尋樹的暴漲。

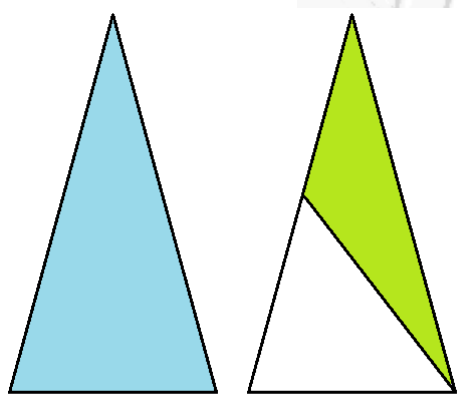


圖 27 水平線效應

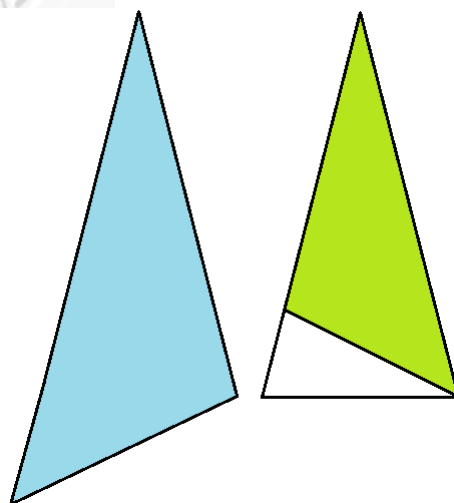


圖 28 使用延伸策略後

因此，我們再使用了與延伸深度相反的作法：減少較平靜的盤面搜尋深度，目的是要盡量維持搜尋樹中所有路線獲得的資訊量一致，並以此空出來的計算時間再使整體搜尋樹的平均深度加深以提升整體棋力。

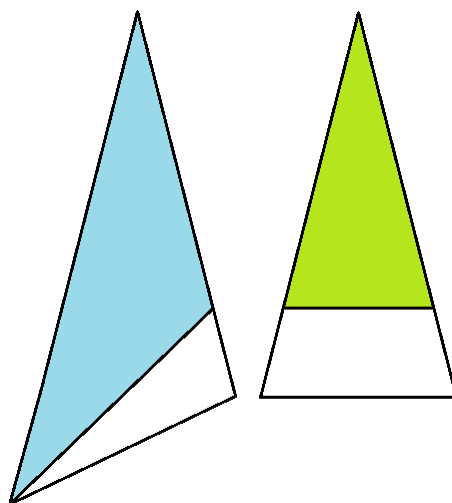


圖 29 使用 LMR 後

較平靜的盤面通常是走子步所造成，而走子步通常在走步排序順位是比較後面的，這也是 Late move reduction【10】名稱的由來。

圖 29 為使用 LMR 後的影響示意圖，對於較平靜的盤面減少搜尋深度後，盡可能的讓搜尋樹在各個分支獲得的資訊量是接近的，相對的整體資訊量會下降，但因為 LMR 減少了部分分支的搜尋深度，因此實驗顯示在相同的時間下可以加深約 2 層的整體搜尋深度，補足整體資訊量下降的問題。

LMR 另一個層面的意思是，對於走步排序較後面的分支認為比較不重要（不太可能會成為 PV 路線，即不太可能大於當前的 Alpha 值），因此如果發現被 LMR 減少搜尋深度的分支成為 PV 路線時應該要以完整的深度重新搜尋以得到更準確的分數。

LMR 並不是所有情況下都適用，為了維持各個分支獲得資訊量的平衡，我們必須根據盤面狀況或走步類型作出選擇，表 11 定義了不適合使用 LMR 的走步類型：

規則	說明
吃子步	吃子步屬於迫著，因為對方通常必須反吃回來，選擇單純，資訊量不足
將軍步或解將步	屬於迫著與緩著，資訊量不足，不宜使用
剩餘搜尋深度小於 3	剩餘資訊量已經很低，不宜再刪減
兵卒的走步	兵卒走步屬於比較緩慢的走步，常常容易得不到資訊，因此不使用 LMR
盤面狀態相當危險	若王的安全性危急，則不能使用 LMR，因為容易錯判情勢

表 11 不適合使用 LMR 的狀況定義

3.2.3 PVS 的改良

承 2.3.3 節，PVS 演算法假設每個節點的第一個子節點一定是 PV 節點，因此對第一個子節點總是使用完整的 Alpha-Beta Window 搜尋。然而我們觀察發現，在 PVS 樹中，Alpha 節點與 Beta 節點是交互存在的，意即 Alpha 節點的父節點一定是 Beta 節點，因此 Alpha 節點的第一個子節點一定會回傳小於 Alpha 值的分數，因此我們對於 Alpha 節點的第一個子節點便可以使用 (Alpha, Alpha+1) 的 Zero Window 去猜測並驗證他不會超過 Alpha，而不需要用完整的 Alpha-Beta Window，從而加速其運行。

參閱 2.3.3 節的 PVS 虛擬碼，改良後的修正只要把斷走步是否是排序中第一個順位的判斷刪除掉即可。

然而在搜尋完這個節點之前，我們無從得知當前節點的類型，因此我們

永遠猜測當前節點是 Alpha 節點，始終以 Zero Window 搜尋第一個子節點。

由於 Alpha 節點在搜尋樹中佔了絕大部分的比例，因此永遠都猜 Alpha 節點

其命中率是高的，實驗顯示，使用此方法可以減少約 5% 的搜尋節點數。

3.2.4 控制格計算

在象棋的審局函數中，棋子的控制格是一項非常重要的資訊，其中可以延伸包括保護與威脅關係、自由度計算、王的安全性等等，這些應用將在 3.3 節做介紹，我們這節介紹控制格如何產生。

有了 BitBoard 的資料結構後，我們對於棋盤的資訊處理從循序處理變成單步處理，利用了平行位元指令的優勢，這些方法對於控制格計算更是再適合不過了。

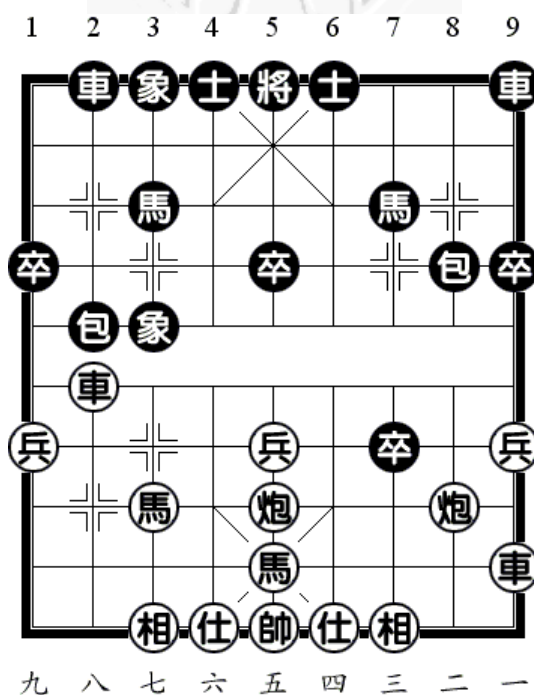


圖 30 範例中局盤面

而控制格的產生十分簡單，只要對於所有棋子產生其吃子範圍的 BitBoard (不需要用 LS1B 拆出來)，然後通通用 OR 運算結合即可，而且這些中間產生吃子步的 BitBoard 資訊更可以直接利用來產生 From、To 配對的實際走子表示，當然其中只有炮的走子步需要重新計算，因為炮的走子與吃子是不同的。如圖 30 是一個中局盤面，分析其黑方的控制格與紅方的控制格可得到如圖 31 與圖 32 的 BitBoard 資訊。

	車	象	士	將	士			車
		馬				馬		
卒				卒			包	卒
	包	象						
	俥							
兵				兵		卒		兵
		偶		炮			炮	
				偶				俥
		相	仕	帥	仕	相		

圖 31 紅方的控制點

	車	象	士	將	士			車
		馬				馬		
卒				卒			包	卒
	包	象						
	俥							
兵				兵		卒		兵
		偶		炮			炮	
				偶				俥
		相	仕	帥	仕	相		

圖 32 黑方的控制點

有了這些資訊後，要分析諸如黑方 2 路線的包被紅方俥拴住或是紅方窩心馬的出口被黑方過河卒控制住都變得容易。而要檢查一個棋子或區域是否被保護或威脅，則只要把棋子或一個區域的 BitBoard 對己方(保護)或敵方(威脅)控制點的 BitBoard 做 AND 運算即可立即得到結果，如圖 33、圖 34 所示。

	車	象	士	將	士			車
		馬				馬		
卒				卒			包	卒
	包	象						
	俥							
兵				兵		卒		兵
		偶		炮			炮	
				偶				俥
		相	仕	帥	仕	相		

圖 33 紅方受到保護的棋子

	車	象	士	將	士			車
		馬				馬		
卒				卒			包	卒
	包	象						
	俥							
兵				兵		卒		兵
		偶		炮			炮	
				偶				俥
		相	仕	帥	仕	相		

圖 34 黑方受到紅方威脅的棋子

當然這些只是初步的資料，以圖 32 為例，黑方包在 2 路線紅方陣地上有控制區域，但實際上如果黑包打出去的話黑車會被紅俥吃掉，這類情況都還是需要仰賴搜尋演算法去進一步分析。

由於控制格需要產生雙方的吃子走步 BitBoard，而只有當前走子方的吃子步資訊資訊可以利用於走步產生，因此計算控制格的成本稍高，但其應用價值可以彌補這一點。

3.2.5 長捉的偵測

在象棋的對戰中，棋規也是一個相當重要的因素，在高手過招時，如果能善用棋規去限制對手，便有可能達到意想不到的效果。

在台灣的象棋比賽使用的是亞洲棋規，包含了長將、長捉等禁手，並且其中長將嚴重性大於長捉、在雙方不違例或雙方違反同一棋規而盤面循環時

判和。長將包含一子連將、多子連將、兩將還一將，但一將一捉、一將一閒不在此例，而所有棋種長將都算違例。長捉定義相當複雜，如表 12 所示，長捉必須被捉的一方下一手有做任何逃避的動做，包含閃躲、阻擋、保護、破壞砲架等，但不含獻子(移動棋子致對方下一手可以吃掉你)，而捉子方不斷地繼續捉同一子，則長捉始成立。亞洲棋規中，長捉不包含捉未過河的兵卒，因此長捉未過河的兵卒是允許的。另外長捉需要判斷被捉子是否有真根，真根表示被捉子有保護，而且被捉子被吃掉時，保護子確實可以反吃；而假根表示被捉子乍看之下有保護，但實際上被捉子被吃掉時，保護子不能反吃。捉真根子不算長捉，但被捉子是車的話不管有無真根都算長捉。若車馬炮長捉同類子則視為長獻(前提是被捉子可以吃掉你，對方馬被拐馬腳無法反吃則仍然視為捉)，但若被捉子吃掉你的著法是非法步(導致王對臉或是對方下一手可以馬上吃掉王時)則仍然視為長捉。圖 35 為假根的範例，黑方 8 路包乍看之下有保護，但實際上 5 路包無法反吃，因此屬於假根；圖 36 為真根範例，黑包乍看之下有保護，但若黑車離開左路線，紅方[車六進六]將直接殺棋，但此例為情勢所逼而非黑車反吃為禁著，因此棋規上不屬於長捉，雙方不變作和。

兵種	規則
將帥	可以長捉任何子，不變判和
兵卒	可以長捉將帥以外任何子，不變判和
車	不能長捉無根子，捉車不在此例
炮	不能長捉無根子或車，捉炮不在此例
馬	不能長捉無根子或車，捉馬若對方被拐馬腳則算捉

表 12 亞洲棋規長捉規則細節

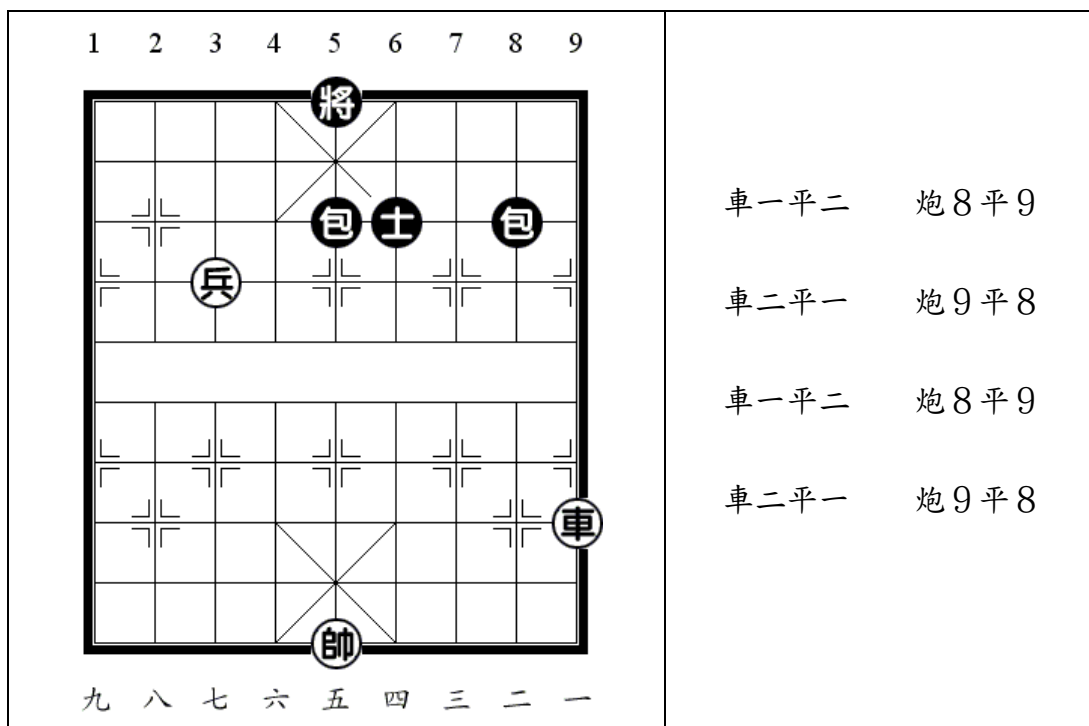


圖 35 紅俾長捉假根包，紅方不變作負

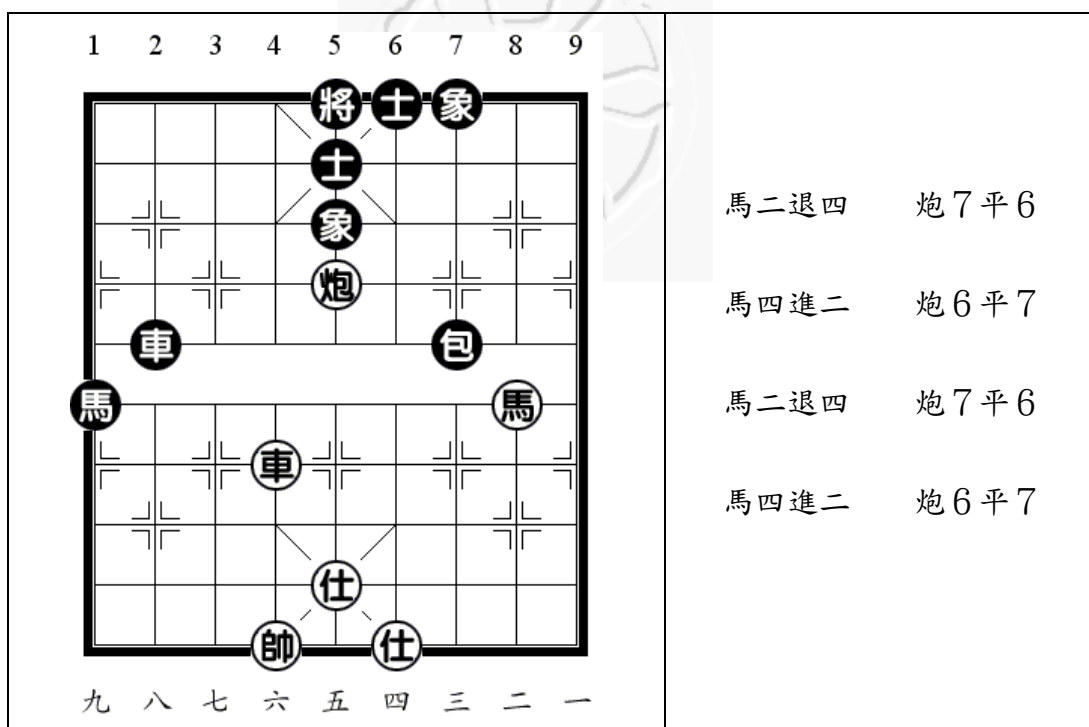


圖 36 紅偽長捉真根包，雙方不變作和

對於程式來說，長將是比較好判斷的，而且長將規則相對簡單。象眼使

九 八 七 六 五 四 三 二 一

52

我們在 3.2.4 節中，實作了控制格的計算，因此使用控制格的概念來改進長捉的判斷。依據象眼的結構，在發現長捉之前，必須先測試當前走步是否為捉，然後在該盤面重複時判斷為長捉，我們便依據之前說明的長捉定義做檢查。

在產生控制格的同時，我們於其中加入了如表 13 的程式碼，計算完控制格後我們可以發現對方受到我方威脅的棋子，也可以找出對方受到保護的棋子，並且在產生控制格時同時知道威脅的來源，並檢查捉同類子或捉車不考慮有無真根的問題。我們觀察發現所謂長捉便是「捉—逃—捉—逃」的循環，因此當搜尋路線上發現重複的盤面後，我們會往前檢查盤面是否「被威脅子的位置」由不受威脅狀態轉變為受威脅狀態，若是，則判斷為長捉；而「被威脅子的位置」如果原本就是受威脅的狀態代表被捉子沒有做逃避的動作(即獻子)則不算長捉。

```
//產生對方控制格
略
//產生我方控制格
//將帥(不做檢查)
//兵卒(不做捉的檢查，長將另外處理了)
//車
ctrlRook & oppOccupied & ~oppProtected & ~oppRook
//馬
ctrlKnight & ((oppOccupied & ~oppProtected) | oppRook) &
~oppKnight
//炮
ctrlCannon & ((oppOccupied & ~oppProtected) | oppRook) &
~oppCannon
```

表 13 判斷捉的虛擬碼

至於為什麼是判斷「被威脅子的位置」，因為被捉子可能非常多，不像長將的判斷，我們無法同時追蹤多個被捉子的動向(至少 BitBoard 不允許，舉例來說，兩隻車在 BitBoard 的表示法中是無法區分的)，但我們可以記錄所有被威脅子當前的位置，並以 BitBoard 表示，由於搜尋演算法使用演示盤面的方式計算，因此每次執行 Make Move 時我們知道前後的盤面變化，而且移動的只有走子方的棋子，這點相當重要，表示被威脅子在走子方行動時確實留在原位，我們才能確保是捉同一子。

在如此改進下我們可以檢查出大部分的情況了，但目前我們尚無法判斷假根與捉同類子但對方反吃你是禁手的問題，如圖 38 所示，由於黑方車反吃紅方俥為禁手(導致將曝於紅方馬的攻擊下)，因此屬於紅方俥炮連捉黑方車，這類的判斷花費成本較高；而多子連捉一子與一子連捉多子尚無法判斷，因為此類型要依序檢查兩次重複盤面間的所有過程，且必須確實追蹤所有被捉子的動向以及保護關係，相當複雜，花費成本更高，因此我們目前尚無法找到解決辦法。

在改進了長捉的判斷後，Shark 參加於 2014 年 6 月 28 舉辦的 TCGA 電腦對局比賽時便沒有再違反任何棋規，反而在一局對上 BrainChess 時，對方(黑方)使用了陷阱佈局導致我方程式掉了一隻馬，在我方陷於極大的不利時對方卻陷入循環盤面而被我方求和，如圖 39 所示，因此可以想見處理好棋規對於比賽的幫助有多大。

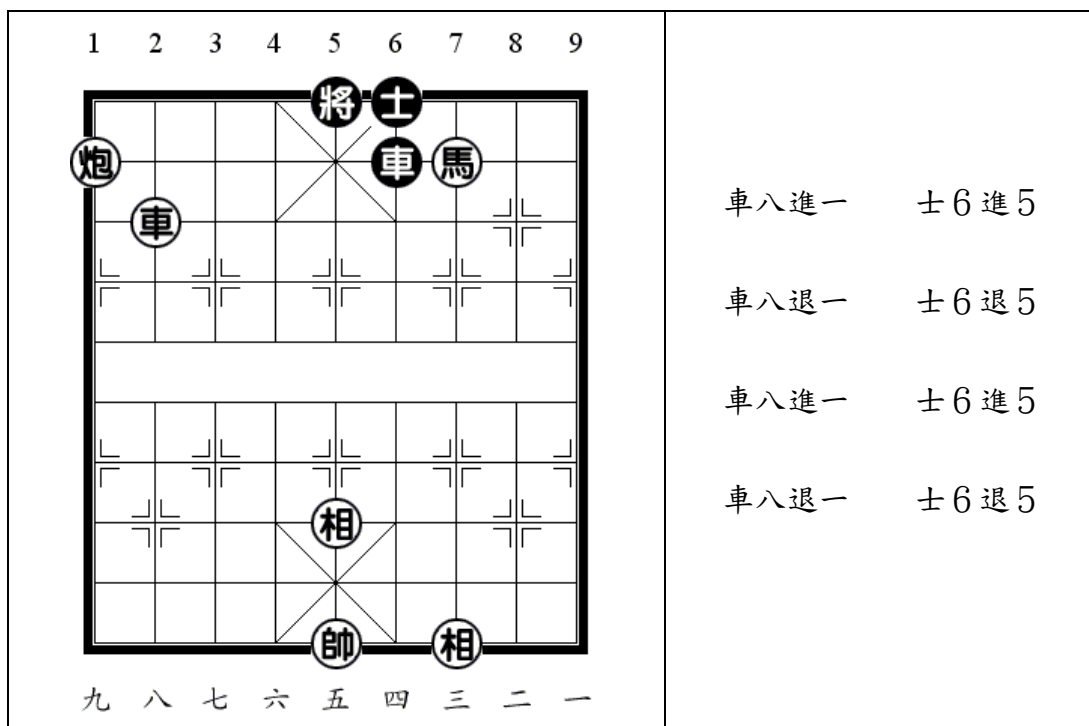


圖 38 由於黑方車不能反吃，因此為紅方俾炮連捉黑方車，紅方不變作負

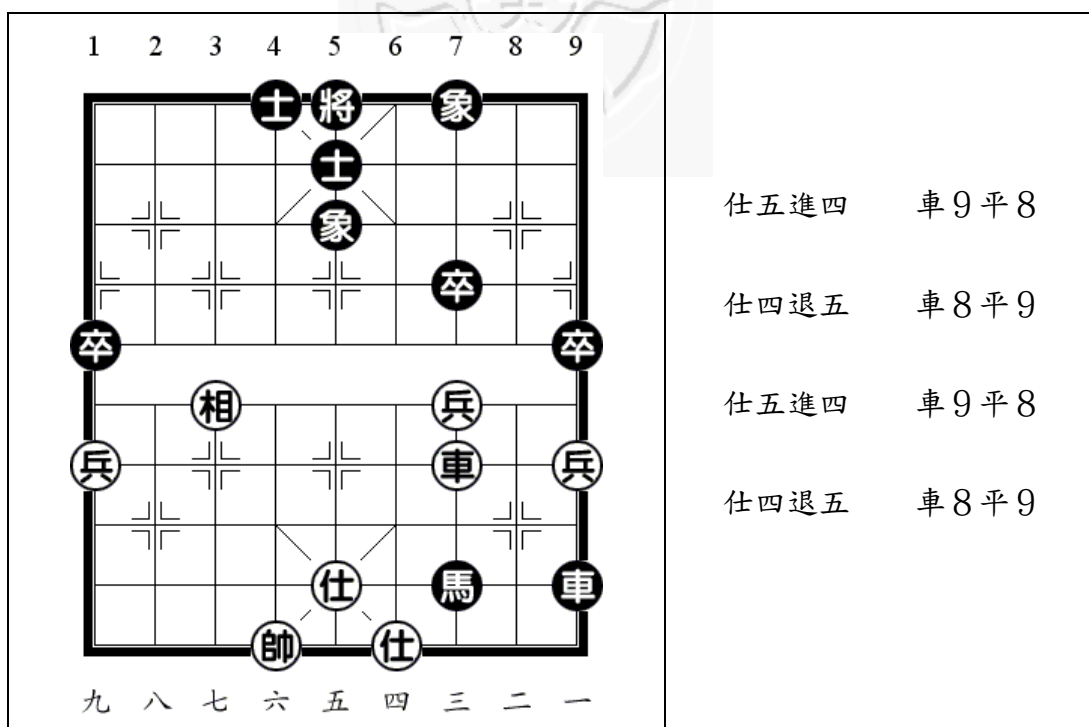


圖 39 TCGA 2014 Shark 對 BrainChess：雙方走閒著，不變作和

3.3 審局函數改進

審局函數是象棋程式的知識核心，也是最重要的部份，一隻程式甚至只靠審局函數便能下棋了，然而好的審局函數搭配有效率的演算法更能發揮其威力，搜尋演算法能補足審局函數無法推演棋局進行的缺陷，而審局函數則彌補搜尋演算法的知識成分以及水平線效應。我們針對審局函數改良其演算效能以及部分結構。

3.3.1 子力價值

審局函數部分我們調整了子力在開中殘局的價值。以下列出調整後的各個子力分數，而左側項目列出的盤面狀態如開中局或殘局、有無進攻機會不是一個絕對的值，是一個模糊化表示「程度」的概念，進而內插調整其分數，這是象眼的一個特色，但其缺點是這些內插調整由於計算成本高，因此這類盤面分析只在根節點計算，其後的搜尋都採用預先計算好的結果，因此若有一分支發生了猛烈的局勢變化則無法反應出精確的審局分，我們目前針對這點尚無比較好的改進方案。

	車	馬	炮
開中局	200	98	100
殘局	180	98	90

表 14 車馬炮在開中殘局的價值

	仕	相
受威脅	40	40
不受威脅	20	21

表 15 仕相對於對方的威脅下降而降低其價值

兵卒的特點在於行動緩慢、過河後威力強、進逼九宮格時能造成更大威脅，而一般過河兵單獨作戰能力不高，僅能做牽制之用，但是要搭配其它攻擊子力其威力則可以接近一顆大子，因此兵卒的價值算是象棋中最難衡量的兵種。兵卒在優勢時要往前進逼，劣勢時要留在高位用以遮對方王以及牽制對方棋子，基於這些特性，我們調整兵卒的價值如表 16，其中過河列出的是兵卒剛過河時的價值。

	邊兵	3、7 路兵	中兵
未過河	15	17	19
過河、有進攻機會	33	40	41
過河、無進攻機會	37	44	46

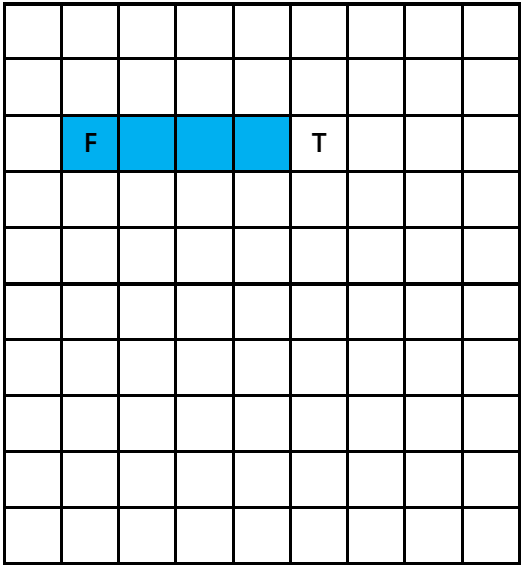
表 16 各種狀態下兵的價值

而另外與兵卒過河後位置相關的分數不在此列，大體上越進逼九宮格的分數越高，而佔花心有高過一隻馬的價值，但己方劣勢時則讓兵卒盡量保持在高位用以牽制與遮王；因為進攻速度的關係，中兵價值大於 3、7 路兵再大於邊兵；另外開中局時的兵卒過河威力要大過殘局時的，而殘局未過河兵價值則稍高(殘局宜留兵)。兵卒的掌握是象棋相當困難的地方，特別是開中局過河兵的利用以及殘局時過河兵如何搭配其它子力作攻勢等等，兵卒由於移動緩慢，因此以搜尋樹為主的演算法常常看不見兵卒所造成的威力，因此若能掌握好兵卒，可以大幅的提升棋力。

3.3.2 特殊棋型

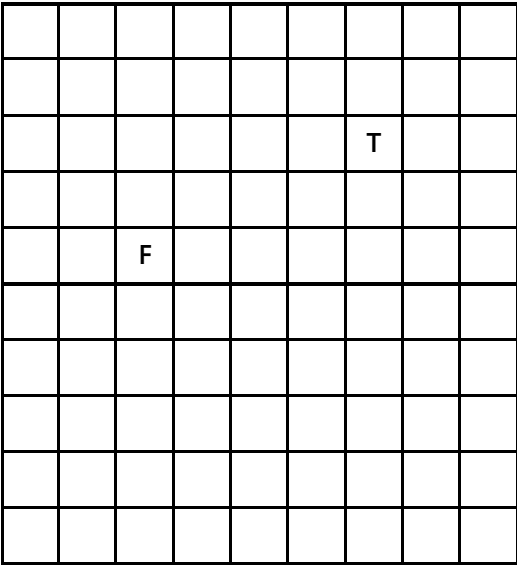
特殊棋型是象棋局勢中相當重要的因素，只看子力通常沒有辦法準確反應真正的局勢，特殊棋型對於大子的限制或對王的威脅其重要性通常不亞於子力，也是影響所謂的「勢」與「先」重要的指標。

對於象眼原本的空頭炮、中炮鐵門門、栓鍊、炮鎮窩心馬等我們調整了其分數，並且以 BitBoard 重新實作了這些特殊棋型偵測。在開始前，先介紹一個用於偵測棋型的新結構—RookSpanMask，顧名思義，它是一個紀錄著由 From 到 To 所構成的一直線 BitMask 的 BitBoard 表格。如圖 40 所示，當輸入的 From 與 To 是在同一行或列上時，會輸出由 From 開始直線延伸到 To 的 BitMask，From 的位置 bit 設為 1 代表該 From、To 的組合合法；圖 41 則為不合法的 From、To 組合範例，由於 From、To 不屬於同一行或同一列，因此 From 的位置 bit 設為 0 (實際上整張 BitBoard 都為 0)，代表該 From、To 的組合不合法。



	F					T			

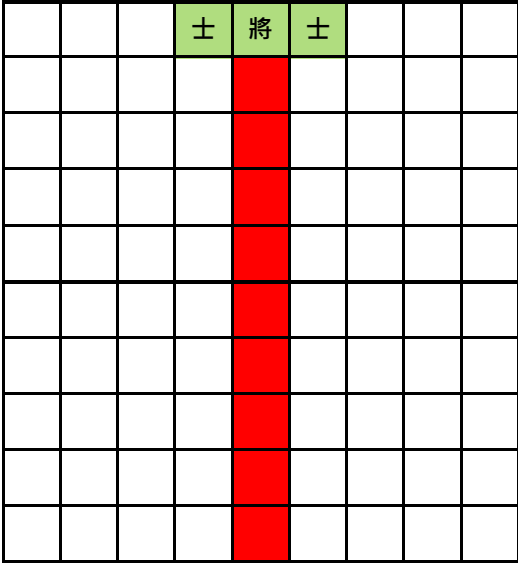
圖 40 合法的 From 與 To 組合

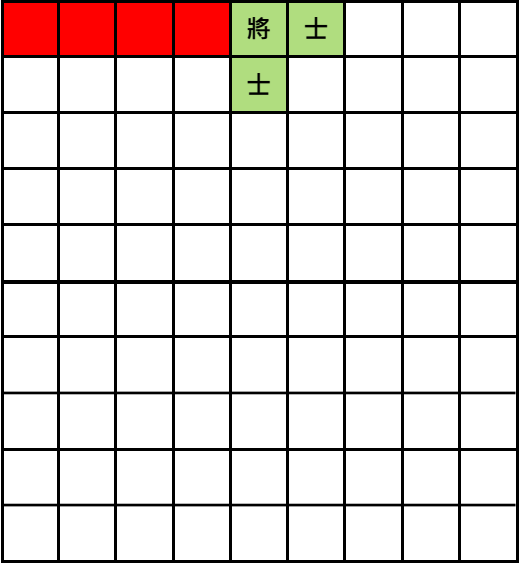
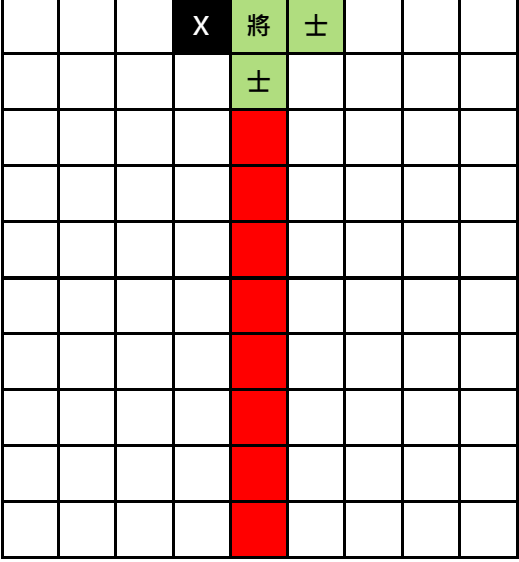


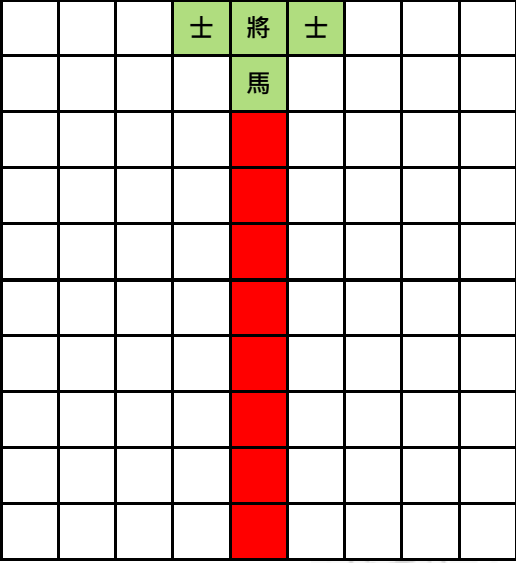
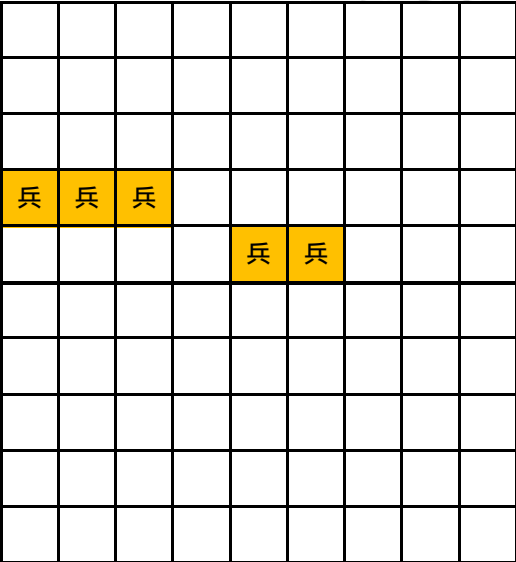
						T			
		F							

圖 41 不合法的 From 與 To 組合

接下來我們便應用 RookSpanMask 結構於棋型的偵測，下表列出一些特殊棋型與其實作方式：

棋型	說明
<div data-bbox="341 607 861 1167">  </div> <p data-bbox="555 1182 651 1216">空頭炮</p>	<p data-bbox="895 618 1410 734">首先比對將與士的位置，語法為 (Advisor King)==constant</p> <p data-bbox="895 790 1410 1160">使用 BitBoard 表示法後便不需要再處理兩隻士 Index 不一樣的問題。接下來取出紅方五路炮的位置，以下 From 表示紅方炮的位置、To 表示將的位置，BitBoard 語法為：</p> <p data-bbox="895 1216 1410 1332">(RookSpanMask (From, To) & Occupied).PopCnt() == 1</p> <p data-bbox="895 1384 1410 1921">PopCnt 即 Population Count，代表著炮到將經過了幾個棋子，由於包含 From 的位置，因此若為 1 代表中間沒有棋子，若為 2 代表中間有一個棋子，依此類推。空頭炮配合攻擊子極容易抽子，最高有 80 分的價值，炮的高度越低價值隨之下降。</p>

	<p>同空頭炮，先確定將士的型，然後取出底線的炮，並使用 RookSpanMask 計算炮到將中間間格的棋子。沉底炮亦屬於容易抽子的型，最高有 40 分的價值，炮離將越近分數隨之下降。</p>
沉底炮	
	<p>此例略不同於空頭炮，這邊中炮到將的間格棋子數量要為 2，即語法為：</p> <pre>(RookSpanMask (From, To) & Occupied).PopCnt() == 3</pre> <p>然後要檢查 X 處是否被紅方所控制，只需檢查 3.2.4 節所述之控制格於 X 處的 bit 是否為 1 即可，而特別的是由於這個是針對王的安全性，因此 X 的控制要另外考慮紅方帥照向 X 處的情形，語法如下：</p> <pre>(RookSpanMask (帥的位置, X) & Occupied).PopCnt() == 1</pre> <p>一樣是使用 RookSpanMask 檢查中間</p>
中炮鐵門門	

	<p>間隔的棋子，若 PopCnt 為 1 則表示帥直接照向 X，若為 0 則表示帥不在 X 那列上。中炮鐵門門屬於容易殺棋的型，有 55 分的價值。</p>
 <p>炮鎮窩心馬</p>	<p>此例接續空頭炮的方法，另外要檢查花心是否為馬，而 RookSpanMask 則判斷炮到將之間間隔 2 子始成立(一為馬、二任意子均可)。炮鎮窩心馬由於封死了將於九宮格的移動，被殺的機會增加，並且也封住了馬的行動，因此有 70 分的價值。</p>
 <p>牽手兵</p>	<p>牽手兵計算極為簡單，BitBoard 語法如下：</p> <p>(Pawn & (Pawn>>1)).PopCnt()</p> <p>把兵的位置 Mask 橫向平移後與原來的 Mask 作 AND 運算，最後做 Population Count 即可得知牽手兵有幾組，左圖中有 3 組牽手兵，而由於我們的 BitBoard 有外加牆壁，因此無須擔心兵的位置被 Shift 到下一行造</p>

	<p>成誤判，且兵在未過河時不可能形成牽手兵，因此也不必檢查是否過河。</p> <p>一組牽手兵有 10 分的價值，而多組的牽手兵會有額外的加分。</p>
--	---

最後是栓鍊，栓鍊的種類很多，如圖 42，包含牽制方的俥與炮、牽制目標的車與將、被牽制子的馬與包這些的組合。

				將				車
				馬				
	車							
								馬
	包							
				俥				兵
								炮
	俥							
				帥				

圖 42 各種栓鍊

要偵測栓鍊首先必須先找出牽制方俥與炮對被牽制方車與將的配對，並使用 RookSpanMask 檢測是否位於同一行或列、中間是否間隔一子(炮的話為兩子)，語法如下，

```
(RookSpanMask(From, To) & Occupied).PopCnt() == 2 (炮為 3)
```


這邊利用了前述 RookSpanMask 以 From 的 bit 位置紀錄其與 To 的組合是否合法的特性，因此上式的 PopCnt 若得到 0 則表示 From 與 To 不在同一行或同一列，栓鍊自然不成立；若 PopCnt 不等於 2 (炮則不等於 3)，則表示並沒有確實牽制到目標，栓鍊亦不成立，因此只需檢查 PopCnt 是否正好等於 2 (炮為 3) 即可。最後以 LS1B 或 MS1B 取出被牽制子的位置，並且檢查被牽制子的屬性。栓鍊的目的是限制對方大子的行動，包含牽制目標與被牽制子，因此若被牽制子距離牽制目標越近，則價值越高，檢測距離為對 RookSpanMask 作 PopCnt 運算，栓鍊最高有 40 分的價值。

在實現了 BitBoard 後，我們不必再處理相同棋子有不同的 Index 的問題，只需要對對應的位置做 AND 運算即可。BitBoard 的優勢便在棋型檢測上，可以在幾乎不影響效能的情況下增加判斷項目，而一些較花時間的判斷(如栓鍊)應該仍然有優化加速的空間，而且還有更多的 Pattern 可以利用 BitBoard 快速判讀，這邊就留給後面的研究去開發了。

3.3.3 自由度

審局中還有一個相當重要的因素—自由度，通常一方的棋子靈活度高比較不會受制於對方，並且有比較大的機會壓迫對方以及發現攻勢。實際上 3.3.2 節介紹的特殊棋型大部分都是壓迫著對方的王或是對方的大子，利用己方少量的子力牽制著對方大子迫使對方必須要用更多大子做防守，進而削弱對方大子的作戰能力，在對方大子自由度極度受限的情況下我方便可以積極進攻

獲得優勢。

我們考慮了馬與車的自由度，車的部分計算其走子步的自由度，如圖 43 所示，取得車的走子步 BitBoard 後做 Population Count 即可得到自由度，一個自由度計算 0.5 分做累計。

		相		帥	仕	相		
俥				仕				
			炮				炮	
		兵	傜					兵
兵			車			馬		
				兵		卒		
卒								卒
			士	象				
車				將		象		

圖 43 車的自由度計算

馬的部分比較特別，不像西洋棋，象棋中的馬容易被拐馬腳並且當其自由度非常低時極度容易被吃掉，因此我們考慮了馬受到限制與威脅的來源，半忽略己方棋子的阻擋，如圖 44 所示，在考慮黑方馬(反白顯示)的自由度時，我們把馬能走到的目標格分成三種類型：

1. 完全自由：綠色○標示，馬移動到目標點不受阻礙，並且目標點不受對方控制。
2. 半自由：藍色△表示，馬移動到目標點受己方棋子阻礙，並且目標點不受對方控制。這類型受限於己方，嚴重性不高。
3. 受限制：紅色×表示，馬移動到目標點受對方棋子阻礙，或是目標

點受對方控制。這類型受限於對方，因此屬於受到比較嚴重的限制。

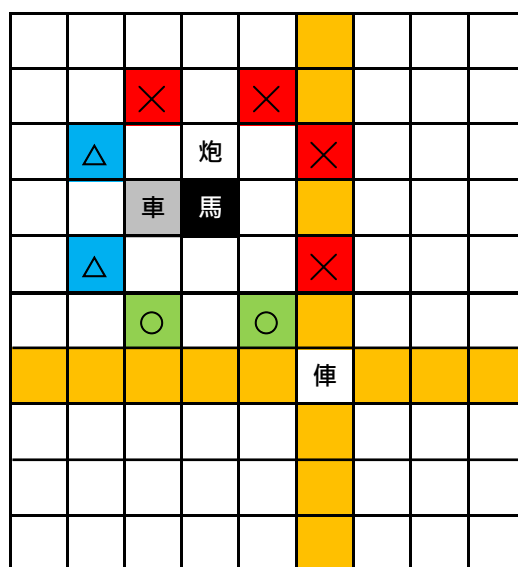


圖 44 馬的自由度計算

當○的數量過少時會略為扣分(5~10)，而×的數量過多時會扣 20 分，如此改良的程式會盡量的壓制對方馬的自由度，進而容易發現能抽吃對方馬的方法，而不會因為搜尋看不到能吃掉對方馬而沒有方向，以經驗來說，馬非常怕過河兵或車，這兩個兵種只要佔在馬跳象田或緊鄰的位置即能有效限制馬的行動。

目前我們考慮的方式比較簡易，事實上象棋中很多特殊棋型的共同點都是壓制對方的自由度。人類高手在看一個局面的優劣時，對於處處受限的一方通常是會覺得盤面較不利的，然而搜尋演算法卻會陷於水平線效應而無法察覺到自由度降低帶來的潛在威脅，因此如果我們在搜尋樹中容易得不到資訊(即搜尋出來的走步分數非常接近)的情況下，逐步壓制對方的自由度較容易在接下來的搜尋樹中發現進攻的手段，意即壓制對方的自由度可以限制對方獲得優勢的機會，並增加我方發現攻勢的機會。而王的安全性又是自由度策

略的最高境界，因為在象棋中王就是一切，對於對方的王做威脅可以最大化的限制對方的自由度，進而容易發現抽子或擴大優勢的手段，不過這個部分由於相當複雜，我們對此尚無更深入的探討。自由度策略在很多對弈類遊戲中都適用，如黑白棋、五子棋等，是一個很有研究潛力的項目。

3.3.4 Material Table

另外我們規劃建立 Material Table，該知識庫記錄了雙方不同棋種組合的優劣性，以分數的方式呈現，屬於全方位的知識庫，可以引導開中殘局的換子策略以及攻殺方式，引導棋局進行至有利的棋子組合，甚至進入一些勝和定式。

勝和定式不同於殘局庫，殘局庫需要記錄大量的盤面資訊，而且象棋即使到殘局，其子力仍然非常多，特別是在含有車或炮這種高移動性的兵種時，要窮舉所有情況十分困難，而且會耗用大量儲存空間，必須要動用硬碟作動態存取，效能堪憂。而且在真正進入少子的殘局時通常大勢已定，只有較少數的殘局需要依賴殘局庫才能解，因此想辦法由 Material Table 引導至較好的殘局會比在殘局庫尋求妙著來的重要得多。

Material Table 中包含勝和定式以及其衍生，讓程式知道什麼樣的子力對上什麼樣的子力例勝、例敗、例和等等，也記錄一些棋子組合的優劣性，例如馬炮要優於雙馬或雙炮、缺士怕雙車、缺象怕炮等等，讓程式在中殘局會有大局觀念：在盤面優勢時的時候盡量的做有利的換子或破士象以加速棋局進行至例勝的棋種組合以免發生其它變數；在盤面劣勢時的時候亦嘗試換掉

對方的關鍵子力或保護士象想辦法進入例和的組合，而不是一味的作無意義的進攻導致挫敗。

而建立 Material Table 必須要對棋種組合進行編碼，因為紅黑雙方是相對的，因此只需記錄一方。表 17 列出了我們使用的編碼方式，該編碼記錄各種兵種的數量，並乘上編碼遞增值，不考慮無將帥的情況，對於一方共有 1458 種組合，雙方共 2125764 種組合，這屬於最壓縮的方式。而這個編碼並不是在審局時才做計算，而是隨著盤面演練時作逐步更新，當盤面發生吃子時減去其兵種對應的編碼遞增值即可。

兵種	仕	相	馬	車	炮	兵
可能的值	0~2	0~2	0~2	0~2	0~2	0~5
編碼遞增值	1	3	9	27	81	243

表 17 Material Table 編碼表

而因為勝和定式以及棋子組合優劣性多屬於人類高手多年來代代累積下來的經驗，我們較難完全讓機器自動生成該組合的解，但仍然有部分可以讓機器自動推演，例如孤兵擒王，可以往前推演：馬兵擒王、雙兵勝單王...等等。

而這類的 Material Table 即使建構不完整也不會有任何副作用，沒有記載的棋種組合會保持程式原先的判斷，但唯一要注意的是前後一致的問題，若前後不一致則程式便不會推進棋局進行下去，舉例來說：車兵破士象全(圖 45)，但大部分的情況下解法是用兵去換一士或一象，然後演變成單車破單缺士或單缺象(圖 46)，若 Material Table 中車兵對士象全的分數若高過車對單缺士或單缺象，則在都屬於例勝組合的情況下，程式便不會再繼續往前推進棋局，

這是較困難必須解決的問題。而另外還有考慮高低兵的組合，由於兵不能後退，因此高兵能成為低兵，低兵無法變為高兵，高低兵也是影響勝和定式的重要因素，舉例如高低兵勝雙士、孤兵擒王(底兵除外)，這些需要做額外判斷。

目前由於本程式開發時間有限，因此 Material Table 尚未完成，但提供此想法做為參考。

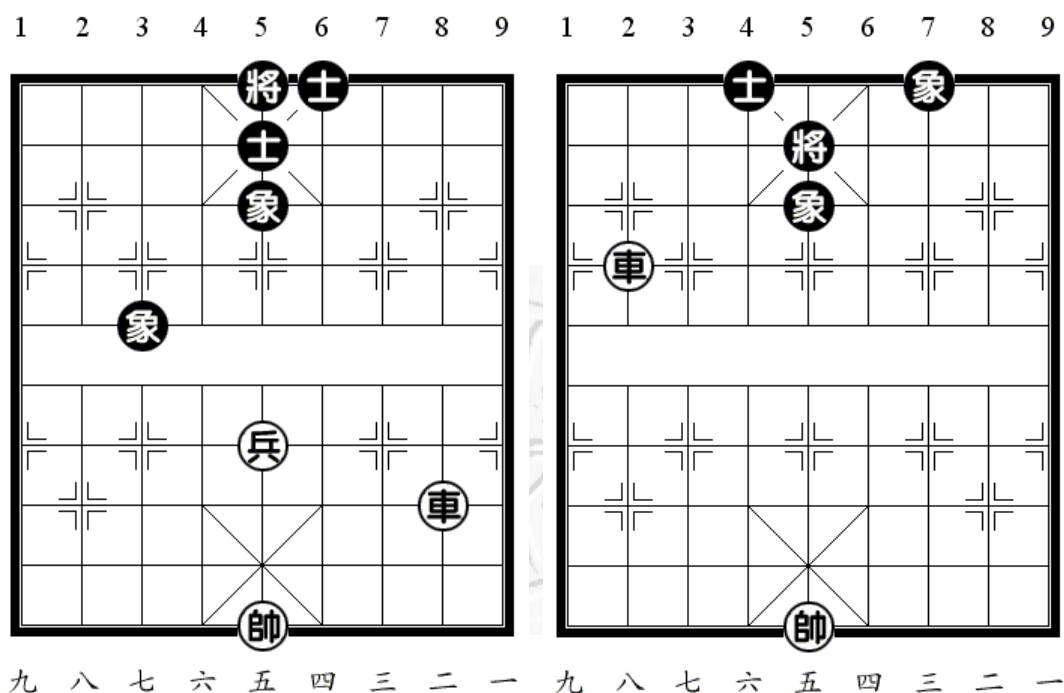


圖 45 車兵勝士象全

圖 46 單車勝單缺士

第4章 實驗結果與未來發展

隨者種種的改良，Shark 首次於 2013 年 12 月 7 日參加了於政大辦的 TAAI 2013 電腦對局比賽，並於 2014 年 6 月 28 日參加於台師大辦的 TCGA 2014 電腦對局比賽，對手分別是象棋世家、棋謀、BrainChess，相關介紹請參閱 2.1 節。TAAI 2013 Shark 獲得銅牌，比賽結果如下表所示，比賽棋譜請參閱附錄。

Shark 與象棋世家有一段明顯的實力差距，常常走出一些不好的走步讓象棋世家輸出的審局分數跳躍式的上升，自己卻渾然未覺；而與棋謀則棋力尚接近，Shark 利用了判斷特殊棋型的優勢贏了一局，但在棋謀因搜尋深度而非常強的中局攻殺能力下亦輸了一局；而對於 BrainChess 則略勝一籌，但當時 BrainChess 使用筆記型電腦做為比賽平台，並且開中殘局都設定固定搜尋 12 層，因此硬體上較吃虧，其真實棋力尚未知。

對手	象棋世家	棋謀	BrainChess
Shark 執先	負	勝	勝
Shark 執後	負	負	勝

表 18 TAAI 2013 電腦對局象棋比賽結果

圖 47 為 TAAI 2013 棋謀對 Shark 的中局盤面，Shark 執黑，並且在中局前半時棄了一隻馬而形成中炮的棋型，配合著 7 路的车對於紅方帥造成了一定的威脅，而棋謀對於此情勢的處理似乎並不理想，後來在黑方左側馬的搭配卒進攻下，Shark 便吃回紅方馬而近逼進攻得勝。

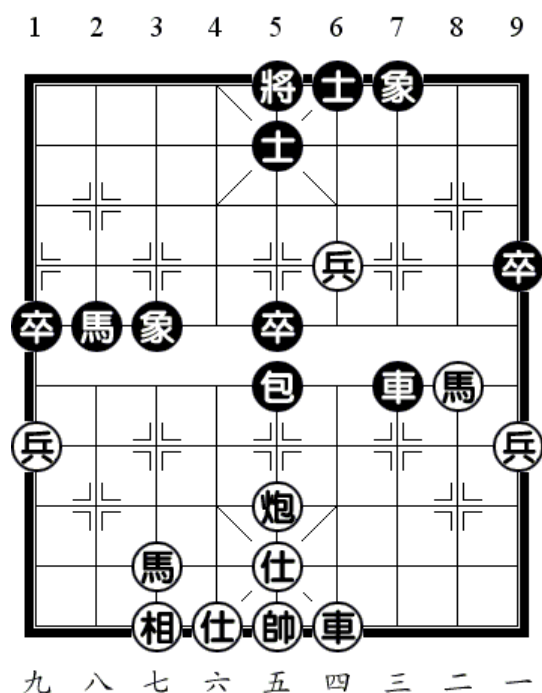


圖 47 TAAI 2013 棋謀對 Shark 中局盤面

Shark 再經過一連串針對資料結構以及兵卒的掌握改良後，與象眼大量對戰勝率已高達八成、與 TAAI 2013 時的 Shark 比賽版大量對戰勝率達七成以上，其在開中局的棋力大幅上升。以一局雙方各 30 分鐘為基準，象眼中局約能搜尋 11 層，Shark 經過諸多改良後，中局可以搜尋約 14 層。

Shark 於 TCGA 2014 獲得銀牌，比賽結果如表 19 所示，比賽棋譜請參閱附錄。Shark 與象棋世家仍然有差距，但比起上一次比賽屢屢讓對方審局分數跳階，這次比較不容易看出有失誤的部分，但還是慢慢地被奪去優勢而致負，同時我們也觀察到了象棋世家對於殘局知識以及兵卒的掌握十分好，換子非常果斷而且在殘局非常有方向，而因為完全平行化的關係，搜尋深度頗高。大體上 Shark 在開中局輸出的審局分數趨勢和象棋世家接近，而殘局時審局分數對局勢反應來的慢，顯示 Shark 在殘局較無方向，功力尚不足；而與棋謀則棋力仍然接近，棋謀由於也是完全平行化，因此對於搜尋深度上仍然頗

占優勢；而 BrainChess 針對開局庫作了大幅度的改良，包含了多種陷阱布局，並且使用較好的平台比賽，使棋力大幅上升，未來可看性非常高。

對手	象棋世家	棋謀	BrainChess
Shark 執先	負	和	和
Shark 執後	負	和	勝

表 19 TCGA 2014 電腦對局象棋比賽結果

圖 48 為 TCGA 2014 BrainChess 使用的陷阱布局，黑方[包 5 退 1]讓紅方[車五平八]認為可以抽車，而下一手黑方[包 8 平 5]棄車後形成「雙炮鎮窩心馬」的棋型，該布局陷阱在約 34 步後生效，情勢大逆轉，形成圖 49 的結果，紅方少一偶而大劣，雖然 BrainChess 最後陷於循環盤面遭求和(55 頁圖 39)，然而其陷阱佈局對於大部分搜尋深度不夠深(34 層)或審局不夠精確的程式來說，其威力相當強大。

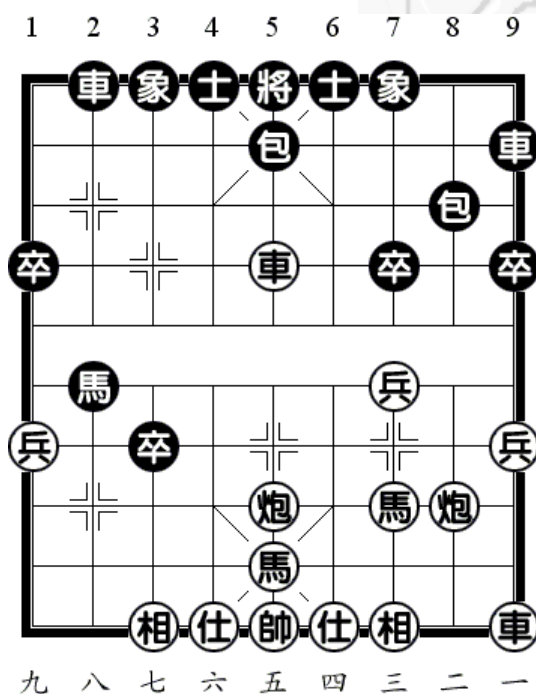


圖 48 陷阱佈局

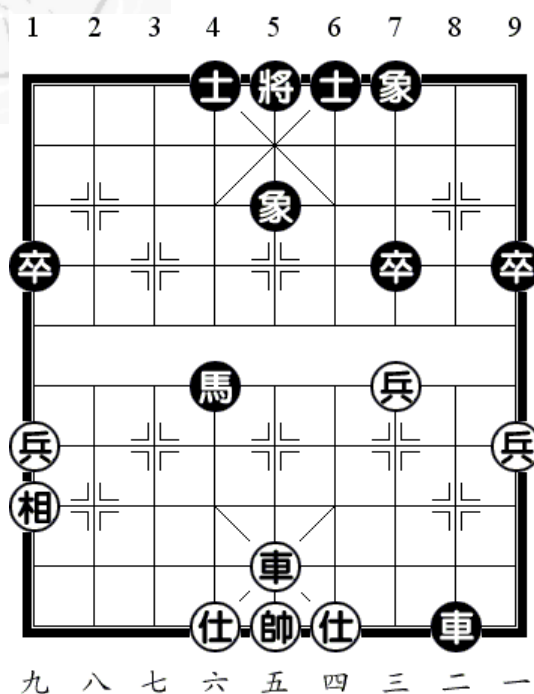


圖 49 陷阱佈局結果

目前 Shark 在殘局攻殺方面尚有不足、並且 BitBoard 的技術不如西洋棋來的成熟，以及目前尚無平行化，這些都是須改進的地方。而 2.3.4 與 3.2.2 節提到的搜尋演算法的資訊量平衡也是潛在的增強了不少棋力，讓 Shark 即使未平行化，搜尋深度輸其它程式一大截卻仍然能在競爭激烈的電腦象棋上占有一席之地，這點是非常有趣的討論，相信對於高知識量、高運算量的人工智慧代理人程式可以經由此分析獲得助益。

我們接下來也將繼續發展程式並進行更多的實驗，期待程式能夠更加的完美，並在之後的比賽能有更突出的表現，同時將各項成果寫成完善的研究與教學文獻，並期望本論文對於象棋以及象棋程式愛好者能有所幫助。



參考文獻

- 【1】 Jaap van den Herik, Jos Uiterwijk, Jack van Rijswijk, “Games Solved: Now and in the Future. Artificial Intelligence”, Vol. 134, Nos.1-2, pp. 277-311, (2002)
- 【2】 涂志堅, “電腦象棋的設計與實現”, 中山大學碩士論文, (2004)。
- 【3】 象棋巫師, <http://www.xqbase.com/index.htm>
- 【4】 Alexander Brudno, “Bounds and valuations for shortening the search of estimates”, Problemy Kibernetiki (10) 141–150 and Problems of Cybernetics (10) 225–241, (1963)
- 【5】 D. E. Knuth and R. W. Moore, “An Analysis of Alpha-Beta Pruning”, Artificial Intelligence 6(4), 293-326, (1975)
- 【6】 Alexander Reinefeld, “An improvement to the Scout tree search algorithm”, ICCA Journal, (1983)
- 【7】 Breuker DM, Uiterwijk JWHM, and Herik HJ van den, “Replacement Schemes for Transposition Tables”, ICGA Journal, (1994)
- 【8】 Chess Programming WIKI, <https://chessprogramming.wikispaces.com/>
- 【9】 施宣丞, 電腦暗棋程式 DarkCraft 的設計與實作, 國立臺灣師範大學資工所碩士論文, (2012)
- 【10】 David Levy, David Broughton, Mark Taylor, “The SEX Algorithm in Computer Chess”, ICCA Journal, Vol. 12, No. 1, (1989)
- 【11】 Intel Instruction Set Architecture Extensions
<https://software.intel.com/en-us/intel-isa-extensions>

附錄 A – TAAI 2013 電腦對局比賽棋譜

Shark vs 象棋世家 象棋世家後勝		
1. 炮二平五 馬8進7	33. 兵七進一 卒9進1	67. 馬七進八 炮5退3
2. 馬二進三 馬2進3	34. 兵七平六 卒1進1	68. 馬八進七 將5進1
3. 車一平二 車9平8	35. 馬六退四 象5進7	69. 馬七退六 將5進1
4. 馬八進九 卒7進1	36. 馬四進三 卒9進1	70. 仕六進五 炮5平1
5. 炮八平七 車1平2	37. 馬三退五 象7退5	71. 仕五進六 炮1平3
6. 車九平八 炮8進4	38. 馬五退四 將6平5	72. 馬六進四 炮3平6
7. 車八進六 炮2平1	39. 馬四進三 炮9平7	73. 馬四退五 炮6退5
8. 車八平七 車2進2	40. 相五退三 象5退7	74. 帥四退一 炮6進3
9. 車七退二 象3進5	41. 馬三退五 炮7退5	75. 馬五進七 將5平4
10. 兵三進一 馬3進2	42. 馬五進六 炮7平8	76. 相三進五 士6進5
11. 兵三進一 馬2進1	43. 兵六進一 卒9平8	77. 帥四進一 炮6退4
12. 車七平二 車8進5	44. 馬六退四 將5退1	78. 仕六退五 炮6平5
13. 馬三進二 馬1進3	45. 馬四進三 將5進1	79. 馬七退六 象7進9
14. 車二進三 象5進7	46. 相三進五 卒8進1	80. 仕五進四 象9進7
15. 炮五平三 象7退5	47. 馬三退二 卒8平7	81. 仕四退五 士5進6
16. 馬二進三 車2進2	48. 兵六平五 卒7進1	82. 仕五進四 卒3平2
17. 車二平三 馬3退5	49. 兵五平四 卒1進1	83. 帥四退一 象7退9
18. 車三平五 車2平7	50. 兵四進一 卒1平2	84. 帥四進一 炮5平6
19. 馬三進五 象7進5	51. 兵四平三 卒2平3	85. 馬六進七 士6退5
20. 炮三進五 炮1平7	52. 兵三進一 炮8進1	86. 仕四退五 炮6平5
21. 相七進五 車7平5	53. 馬二退三 將5退1	87. 馬七退六 象9進7
22. 車五進二 卒5進1	54. 馬三進四 炮8進7	88. 仕五進四 炮5平7
23. 馬九進八 卒5進1	55. 相五退三 卒3進1	89. 馬六進五 炮7平6
24. 馬八進六 象5退7	56. 馬四進六 將5進1	90. 仕四退五 士5進6
25. 馬六進四 卒5進1	57. 馬六進八 卒3進1	91. 仕五進四 象7退5
26. 馬四進六 將5進1	58. 兵三平四 將5平6	92. 馬五退六 卒2平3
27. 仕六進五 卒1進1	59. 馬八退六 將6平5	93. 馬六進七 炮6平5
28. 兵七進一 象7進5	60. 馬六退四 將5退1	94. 馬七退六 士6退5
29. 馬六退八 卒5進1	61. 馬四退六 卒7平6	95. 馬六進八 象5進3
30. 相三進五 炮7平9	62. 馬六退五 卒6平5	96. 馬八退七 象3退1
31. 馬八進七 炮9進4	63. 仕五退六 卒5平4	97. 仕四退五 象1退3
32. 馬七退六 將5平6	64. 帥五進一 炮8平6	98. 仕五進四 將4平5
	65. 帥五平四 炮6平5	99. 仕四退五 士5進6
	66. 馬五進七 卒4進1	100. 仕五進四 將5退1

101. 仕四退五 象3進1	16. 兵三進一 炮7退5	53. 相一進三 象7退9
102. 仕五進四 將5平6	17. 兵九平八 卒3進1	54. 相三退一 象9進7
103. 馬七進六 士6退5	18. 兵八平七 炮7進2	55. 相一進三 象7退9
104. 馬六退四 將6進1	19. 車八平三 馬8進6	56. 相三退一 象9進7
105. 馬四退二 炮5進7	20. 炮六進四 車6進1	57. 相一進三
106. 馬二進三 將6退1	21. 兵五進一 炮7平4	
107. 馬三進五 將6進1	22. 車三平四 車6進1	Shark vs 棋謀
108. 馬五退四 炮5進2	23. 馬三進四 炮3進2	棋謀後勝
109. 帥四退一 炮5退5	24. 炮五進一 炮4平3	1. 相三進五 炮8平3
110. 馬四退六 炮5平6	25. 相七進五 前炮平4	2. 車一進一 馬8進7
111. 帥四平五 卒3進1	26. 仕五進四 炮4平1	3. 車一平六 象3進5
112. 馬六進五 將6平5	27. 馬四進三 炮1進1	4. 馬八進九 馬2進1
113. 仕四退五 炮6退4	28. 帥五進一 炮3平1	5. 炮八平七 車1平2
114. 仕五進六 卒3平4	29. 馬三進四 前炮退6	6. 車九平八 炮2進4
115. 帥五平四 士5進6	30. 馬四退六 將5進1	7. 馬二進四 士6進5
116. 馬五退四 炮6進6	31. 馬六進八 馬1退3	8. 兵七進一 車9平8
117. 帥四進一 士6退5	32. 炮六退一 後炮平4	9. 車六進二 炮2進1
118. 帥四退一 炮6退6	33. 炮五進三 將5平6	10. 車六進五 炮3退1
119. 仕六退五 後卒平5	34. 兵七進一 炮4退2	11. 炮七退一 炮2退6
	35. 馬八退九 炮4平5	12. 車六退三 炮2進7
	36. 炮五進二 將6平5	13. 仕四進五 車8進6
象棋世家 vs Shark	37. 兵七進一 馬3退5	14. 兵三進一 車8平9
象棋世家先勝	38. 兵五進一 將5平6	15. 馬九進七 車9進3
1. 兵七進一 炮2平3	39. 兵五進一 馬5進6	16. 炮二退二 炮2退1
2. 炮二平五 象7進5	40. 兵五平四 馬6進8	17. 馬七進五 卒3進1
3. 馬八進九 馬2進1	41. 兵四平三 馬8退7	18. 馬五進四 士5進6
4. 車九平八 車1進1	42. 馬九進八 炮1平4	19. 車六進一 炮3進2
5. 馬二進三 車1平6	43. 兵三進一 馬7退9	20. 前馬退五 炮2退4
6. 車一平二 車6進2	44. 炮六平二 士6進5	21. 車六進一 卒3進1
7. 兵九進一 馬8進6	45. 炮二退四 將6退1	22. 炮七平九 炮2進5
8. 馬九進八 車9平8	46. 炮二平四 將6平5	23. 仕五退四 炮2平3
9. 馬八進九 炮3進3	47. 炮四平一 象5進7	24. 車八進九 馬1退2
10. 炮八平六 炮8進4	48. 炮一進五 將5平6	25. 相五進七 士6退5
11. 兵三進一 炮8平7	49. 炮一平四 炮4平7	26. 車六退一 後炮退2
12. 車二進九 炮7進3	50. 相五進三 將6進1	27. 炮九進五 車9退1
13. 仕四進五 馬6退8	51. 馬八退七 象7退9	28. 馬五退三 馬2進3
14. 車八進四 炮3進1	52. 相三退一 象9進7	29. 炮九進三 後炮退1
15. 兵九進一 卒7進1		

30. 相七退五 馬3進2
 31. 車六退三 前炮退4
 32. 炮二進五 馬2退3
 33. 車六進三 前炮平6
 34. 車六平七 馬3退1
 35. 兵九進一 炮6退1
 36. 車七退二 馬1進2
 37. 車七平八 馬2進4
 38. 車八平六 馬4退3
 39. 車六平七 車9退4
 40. 炮二退五 炮6退1
 41. 炮二平三 車9平2
 42. 車七平六 車2退1
 43. 兵九進一 車2退2
 44. 炮九退三 馬3進1
 45. 兵九進一 車2進3
 46. 馬三進五 車2平6
 47. 馬四進六 炮6進7
 48. 馬五進六 士5進4
 49. 前馬進八 炮3進8
 50. 車六進三 士4進5
 51. 車六退三 炮3平9
 52. 兵三進一 象5進7
 53. 馬八退六 炮9進1
 54. 仕六進五 將5平6
 55. 車六平一 炮9平8
 56. 車一退四 車6進1
 57. 兵九平八 象7退5
 58. 兵五進一 卒7進1
 59. 後馬進七 車6進3
 60. 兵八平七 將6進1
 61. 相七進九 馬7進6
 62. 車一進六 馬6進4
 63. 車一退四 馬4退3
 64. 車一平二 士5進4
 65. 馬七進八 馬3進4
 66. 馬六退七 卒5進1

67. 馬八進七 卒5進1
 68. 車二進六 將6退1
 69. 車二進一 卒5進1
 70. 前馬退五 將6平5
 71. 相五退七 將5進1
 72. 馬五退三 馬4進6
 73. 仕五進四 卒5進1
 74. 相七進五 車6退1
 75. 車二退一 將5退1
 76. 相九退七 馬6進8
 77. 車二退六 車6平8
 78. 馬三退五 車8平7
 79. 馬五進六 將5平6
 80. 馬七進五 炮6退2
 81. 炮三進一 車7進1
 82. 帥五平四 車7平4
 83. 馬五進七 車4退6
 84. 相五進七 車4進7
 85. 帥四進一 車4退1
 86. 帥四退一 炮6進1
 87. 馬七進六 車4退7
 88. 帥四平五 車4進7
 89. 相七退五 炮6平9
 90. 相七進九 炮9進1

棋謀 vs Shark

Shark 後勝

1. 炮二平五 馬2進3
 2. 馬二進三 馬8進7
 3. 車一平二 車9平8
 4. 兵三進一 卒3進1
 5. 馬八進九 卒1進1
 6. 炮八平七 馬3進2
 7. 馬三進四 車1進3
 8. 車九進一 車1平4
 9. 炮七進三 炮8進2
 10. 車九平四 炮2平5

11. 馬四進三 炮5進4
 12. 仕四進五 車4進1
 13. 炮七平二 車8進4
 14. 車二進五 車4平8
 15. 車四進二 炮5退1
 16. 車四進四 車8退2
 17. 兵七進一 象3進5
 18. 車四退三 炮5進1
 19. 車四退一 炮5退1
 20. 兵三進一 士4進5
 21. 兵三平四 車8進5
 22. 車四平三 車8退3
 23. 兵四進一 卒5進1
 24. 馬九退七 車8平6
 25. 兵七進一 象5進3
 26. 馬三退二 車6進1
 27. 馬二退三 馬7進6
 28. 車三進二 車6進1
 29. 馬三進二 車6平7
 30. 車三平四 車7進3
 31. 車四退五 車7退4
 32. 馬二進一 炮5進1
 33. 馬七進八 象7進5
 34. 馬八退九 馬2進1
 35. 馬九進八 馬1進2
 36. 車四進二 車7進4
 37. 車四退二 車7退3
 38. 馬一退二 車7退1
 39. 車四平二 卒1進1
 40. 馬八退七 馬2退3
 41. 兵四平五 馬3進4
 42. 兵五平六 卒5進1
 43. 馬七進八 卒1平2
 44. 馬八退九 卒5平4
 45. 馬九進七 炮5平3
 46. 相七進九 卒2進1
 47. 馬七退九 卒2進1

48. 相九退七 炮3平5
49. 馬九進八 卒4進1
50. 馬八進六 炮5退1
51. 馬六退八 象3退1
52. 兵六平五 象1退3
53. 兵五平六 炮5進1
54. 馬八進六 車7平4
55. 馬二進四 炮5退2
56. 車二進四 車4退2
57. 車二進二 車4進2
58. 車二退二 車4平8
59. 馬四退二 馬4退3
60. 帥五平四 炮5平3
61. 相七進九 卒2平1
62. 炮五平二 卒1平2
63. 馬二進三 卒4平5
64. 馬三進四 炮3平7
65. 炮二進七 炮7退4
66. 兵一進一 卒2平3
67. 兵一進一 卒3進1
68. 馬四退五 卒3平4
69. 馬五進七 卒5進1
70. 兵一平二 卒4平5
71. 馬七退五 前卒平4
72. 馬五退四 卒5平6
73. 馬四退六 卒6平7
74. 炮二退三 炮7進3
75. 炮二進三 象5退7
76. 炮二平一 卒7進1
77. 仕六進五 馬3退4
78. 兵二平三 炮7平4
79. 炮一退八 炮4進3
80. 兵三平二 馬4進5
81. 帥四平五 炮4退4
82. 炮一平二 卒7平6
83. 炮二進二 卒4平5
84. 帥五平六 馬5進3

Shark vs BrainChess	
Shark 先勝	
1. 炮八平五 炮2平5	
2. 馬八進七 車1進1	
3. 車九平八 馬2進3	
4. 馬二進三 車1平6	
5. 兵三進一 馬8進7	
6. 車八進四 卒3進1	
7. 兵七進一 卒3進1	
8. 車八平七 炮5退1	
9. 馬三進四 炮5平3	
10. 車七平六 馬3進2	
11. 車六平八 炮3進8	
12. 仕六進五 馬2退3	
13. 炮二平四 車6平4	
14. 車一平二 車9平8	
15. 車二進六 車4進1	
16. 車八退四 炮3退1	
17. 馬四進五 馬7進5	
18. 炮五進四 車4進2	
19. 炮五退二 車4平6	
20. 炮四平二 炮8退1	
21. 車八進八 車6平2	
22. 車二平三 車2退3	
23. 炮二進三 車2進8	
24. 仕五退六 車2退4	
25. 馬七進八 炮3平5	
26. 仕六進五 將5進1	
27. 炮二平五 將5平6	
28. 車三平四	
BrainChess vs Shark	
Shark 後勝	
1. 相三進五 炮2平4	
2. 馬八進九 卒1進1	
3. 兵三進一 炮8平5	

4. 馬二進四 馬8進7
5. 車一平三 車9平8
6. 車九進一 馬2進1
7. 車九平六 士4進5
8. 兵三進一 卒7進1
9. 車三進五 車1平2
10. 車三進一 象7進9
11. 車六進二 車2進4
12. 炮八平六 炮4平3
13. 炮二進四 車2平8
14. 炮二平五 前車平5
15. 炮五平一 車8進8
16. 馬四進二 馬7進9
17. 車三平一 車8平4
18. 馬二進三 車5平7
19. 車一平五 象9退7
20. 兵五進一 炮5平7
21. 馬三退二 象3進5
22. 馬二進一 車7平8
23. 兵五進一 炮3平2
24. 兵五平四 車8平6
25. 馬一進二 車6進4
26. 馬二進三 車6退7
27. 車五進一 車6平7
28. 車五平八 炮7平5
29. 車六平五 車4退1
30. 車八平九 車7進3
31. 仕六進五 車4進1
32. 兵七進一 車7平2
33. 車九進二 士5退4
34. 馬九進七 車4平3
35. 馬七退九 車3平4
36. 馬九進七 車4平3
37. 車九退一 車2進2
38. 車九平四 炮5退1
39. 車四退五 象7進5
40. 車五平六 炮5平1

41. 兵九進一 炮1進4
42. 馬七退六 車2平4
43. 車四平六 車3進1
44. 仕五退六 車3退2
45. 帥五進一 車3進1
46. 車六平七 車3平2
47. 車七平九 士4進5
48. 相五退三 卒3進1
49. 兵七進一 象5進3
50. 兵一進一 炮1平4
51. 相三進五 炮4退5
52. 車九平六 卒1進1
53. 兵一進一 卒1進1
54. 兵一平二 卒1平2
55. 兵二平三 炮4平1
56. 兵三進一 炮1進8
57. 車六退一 卒2平3
58. 車六平八 車2平3
59. 相五進七 象3退5
60. 車八進七 士5退4
61. 車八退二 炮1平4
62. 車八平五 士4進5
63. 帥五退一 炮4退8
64. 相七退五 卒3平4

65. 仕四進五 車3退5
66. 兵三進一 車3平7
67. 兵三平四 車7平8
68. 相五退三 車8平6
69. 兵四平三 炮4進3
70. 相三進一 炮4平1
71. 仕五退四 炮1平5
72. 相一進三 卒4進1
73. 帥五進一 車6平8
74. 帥五退一 卒4進1
75. 兵三進一 將5平4
76. 車五平二 車8退1
77. 兵三平四 車8進6
78. 兵四平五 卒4進1



附錄 B – TCGA 2014 電腦對局比賽棋譜

Shark vs 象棋世家 象棋世家後勝		
1. 馬二進三 卒7進1	32. 帥四平五 卒8平7	65. 馬三退四 士5進6
2. 兵七進一 馬8進7	33. 仕五進六 卒7平6	66. 炮八進四 馬7進9
3. 馬八進七 車9進1	34. 炮七退五 炮6平8	67. 帥五平六 炮8退8
4. 炮八平九 車9平3	35. 馬六進七 將5平6	68. 相三退五 卒7平6
5. 車九平八 卒3進1	36. 炮七平四 士5進6	69. 仕六退五 卒6平5
6. 兵七進一 車3進3	37. 炮四進六 馬3進4	70. 相五進七 馬9進7
7. 炮九退一 炮2平3	38. 炮四平一 馬4進3	71. 炮八進二 炮8進7
8. 炮九平七 車3平6	39. 仕六進五 卒6進1	72. 仕五進六 馬7進6
9. 車一進一 馬2進1	40. 炮一退一 炮8平5	73. 仕六退五 馬6退5
10. 車八進八 士6進5	41. 帥五平四 馬3退1	74. 帥六進一 馬5退3
11. 炮二平一 炮8平9	42. 炮一平九 馬1退3	75. 帥六平五 象5退7
12. 相三進五 炮3進6	43. 炮九退三 炮5退2	76. 馬四退五 象3退1
13. 車一平七 車6平3	44. 馬七退八 炮5平6	77. 馬五進三 後卒進1
14. 兵三進一 卒7進1	45. 帥四平五 炮6退3	78. 馬三進四 後卒進1
15. 相五進三 車1平2	46. 馬八退七 象3進1	79. 仕五進四 炮8退4
16. 車八進一 馬1退2	47. 炮九平八 將6平5	80. 帥五退一 炮8平5
17. 炮一退一 馬7進8	48. 炮八進三 士4進5	81. 馬四退五 馬3退5
18. 馬七退五 車3進4	49. 馬七進九 炮6進3	82. 帥五退一 卒5進1
19. 炮一平七 馬8進9	50. 馬九退七 士5進4	83. 仕四退五 卒5進1
20. 馬三進一 炮9進4	51. 相五退三 炮6進1	84. 仕五退四 馬5進6
21. 馬五進七 象7進5	52. 相七進五 士4退5	85. 炮八退六 將6平5
22. 馬七進六 馬2進3	53. 炮八退五 炮6退2	86. 炮八平九 將5平4
23. 炮七進五 卒9進1	54. 仕五退四 馬3進5	87. 炮九平六 馬6進7
24. 馬六進八 馬3退1	55. 炮八平五 馬5退7	88. 帥五平六 將4進1
25. 仕四進五 炮9平8	56. 馬七退八 炮6退2	89. 炮六進三 馬7退8
26. 馬八進六 馬1進3	57. 馬八進九 象1進3	90. 帥六進一 馬8進6
27. 兵九進一 卒9進1	58. 馬九進八 炮6平8	91. 仕四進五 馬6退7
28. 相三退五 炮8退1	59. 馬八退六 炮8進8	92. 炮六進二 馬7進5
29. 相七進九 炮8平6	60. 馬六進四 卒6進1	93. 炮六退三 馬5進7
30. 帥五平四 卒9平8	61. 相五進三 卒6進1	94. 仕五進六 馬7進6
31. 相九退七 炮6進1	62. 炮五平八 卒6進1	95. 帥六退一 卒5平4
	63. 帥五進一 卒6平7	96. 帥六平五 馬6退7
	64. 馬四進三 將5平6	97. 帥五平六 卒4平5

98. 帥六進一 馬7進6	30. 兵七進一 士5退6	67. 馬四進二 後炮平4
99. 帥六退一 卒5進1	31. 兵七平八 炮1進5	68. 兵四進一 士5進6
100. 炮六進三 馬6退8	32. 炮九進三 士4進5	69. 馬二進四 將5平6
101. 炮六退四 馬8退6	33. 車七進三 車4退6	70. 兵七平六 炮4平6
102. 炮六進四 卒5進1	34. 炮九平六 象5退3	71. 炮六進一 炮5進1
象棋世家 vs Shark		
象棋世家先勝		
1. 炮二平五 馬8進7	35. 炮六退五 象3進5	72. 馬四進二 將6平5
2. 兵三進一 車9平8	36. 馬九進七 炮1平3	73. 炮六平五 象5退3
3. 馬二進三 炮8平9	37. 帥五平六 炮5平4	74. 馬二退四 將5平6
4. 馬八進九 車8進6	38. 兵八進一 士5進6	75. 馬四進三 炮6退3
5. 車一平二 車8平7	39. 馬七進九 炮3平2	76. 炮五平一 炮6平7
6. 炮八平六 車1進1	40. 兵八平七 炮4平2	77. 炮一進四 將6進1
7. 車九平八 車7退1	41. 炮六平五 士6進5	78. 炮一平七 炮5平9
8. 仕六進五 車7進1	42. 炮五進二 後炮平4	79. 炮七退五 炮9平5
9. 兵五進一 士6進5	43. 馬九進八 將5平4	80. 兵六平五 炮5退1
10. 車八進五 車1進1	44. 兵七進一 象5進3	81. 炮七退三 炮5退1
11. 兵五進一 卒5進1	45. 兵一進一 炮2平3	82. 帥五平六 炮5進1
12. 車二進八 卒5進1	46. 兵七平八 炮3平4	83. 仕五進四 炮5平6
13. 車八平五 象3進5	47. 帥六平五 後炮平5	84. 炮七平二 炮7平8
14. 車五退一 馬2進3	48. 炮五平六 將4平5	85. 馬三退二 將6退1
15. 車二平三 馬3進5	49. 兵一進一 象7進5	86. 炮二進七 炮6退4
16. 炮五進四 馬7進5	50. 炮六退二 炮4平8	87. 馬二進四 將6進1
17. 車五進二 車7進1	51. 帥五平六 炮8進3	88. 帥六進一 將6退1
18. 炮六平五 炮9平6	52. 相五退三 炮8退6	89. 兵五進一
19. 車五平七 將5平6	53. 兵八平七 炮8平4	
20. 車七平三 車7退4	54. 炮六平七 將5平4	
21. 車三退二 炮2平3	55. 相七進五 炮5退2	
22. 炮五平四 將6平5	56. 馬八進七 炮4進2	
23. 相三進五 炮6進4	57. 帥六平五 象5進7	
24. 兵七進一 車1平2	58. 兵一平二 象7退5	
25. 兵九進一 炮6平5	59. 兵二進一 炮5進3	
26. 車三平七 炮3退2	60. 兵二平三 象5退7	
27. 炮四平一 車2進4	61. 兵七平八 象3退5	
28. 炮一進四 車2平4	62. 馬七退六 炮5平4	
29. 炮一平九 炮3平1	63. 馬六退四 後炮平5	
	64. 兵三平四 炮4退2	
	65. 兵八平七 炮4平5	
	66. 炮七平六 將4平5	

Shark vs 棋謀		
和棋		
1. 炮二平五 馬8進7		
2. 馬二進三 車9平8		
3. 車一平二 馬2進3		
4. 馬八進九 卒7進1		
5. 炮八平七 車1平2		
6. 車九平八 炮2進4		
7. 車二進四 馬7進6		
8. 兵九進一 象7進5		
9. 馬九進八 卒7進1		
10. 車二平三 炮2平5		
11. 仕六進五 炮5退2		

12. 車三平四 馬6退7
 13. 炮七平八 炮5平2
 14. 炮八進三 車2進4
 15. 兵一進一 車2平3
 16. 馬三進五 馬7進8
 17. 車四平三 車3進2
 18. 馬八進六 車3平4
 19. 車八進六 卒3進1
 20. 車八平七 士6進5
 21. 炮五進四 馬3進5
 22. 車七平五 炮8平7
 23. 車五平一 車8進2
 24. 相三進五 馬8進9
 25. 車三平四 炮7平6
 26. 兵一進一 炮6退2
 27. 兵三進一 士5進6
 28. 馬六進四 士4進5
 29. 馬五進六 車8進4
 30. 兵三進一 馬9進8
 31. 兵三進一 車4進2
 32. 車一平二 車8平2
 33. 車二退二 馬8進6
 34. 車四平六 馬6退7
 35. 車六退三 馬7退8
 36. 兵一平二 馬8進6
 37. 仕五退四 車2進3
 38. 車六平一 卒3進1
 39. 仕四進五 車2退5
 40. 相五進七 炮6進3
 41. 馬六進四 車2平8
 42. 相七退五 象5退7
 43. 車一平四 馬6退5
 44. 車四平三 馬5進4
 45. 仕五退四 象7進5
 46. 車三平一 車8平7
 47. 兵三平二 馬4退3
 48. 車一進八 象5退7

49. 車一退五 車7平6
 50. 車一平七 車6退1
 51. 車七進一 車6平8
 52. 車七進四 士5退4
 53. 車七退二 士4進5
 54. 相五進七 車8平5
 55. 仕四進五 象7進5
 56. 車七平八 士5退4
 57. 車八退二 士6退5
 58. 相七進五 士5退6
 59. 車八退二 象5退3
 60. 車八進六 象3進5
 61. 車八退四 士6進5
 62. 車八進三 車5平8
 63. 仕五退四 車8平4
 64. 車八退一 車4平5
 65. 仕四進五 士5退6
 66. 車八平六 象5進7
 67. 車六平九 象7退5
 68. 車九平七 象5退7
 69. 車七平三 象7進5
 70. 車三退三 象5退3
 71. 車三平六 象3進1
 72. 車六進一 象1退3
 73. 車六進三

棋謀 vs Shark
和棋

1. 兵七進一 炮2平3
 2. 炮二平五 象7進5
 3. 馬八進九 馬2進1
 4. 車九平八 車1進1
 5. 馬二進三 車1平6
 6. 炮五進四 士6進5
 7. 炮五平九 車6進5
 8. 兵三進一 馬8進6
 9. 相三進五 車9平7

10. 兵九進一 卒7進1
 11. 兵三進一 車7進4
 12. 車一平二 車6平7
 13. 炮八平六 卒3進1
 14. 兵七進一 後車平3
 15. 車八進六 炮8平7
 16. 車八平四 車3退1
 17. 車四進二 車3平1
 18. 車二進九 炮7退2
 19. 仕六進五 象5進3
 20. 兵五進一 車1進2
 21. 兵五進一 車1平5
 22. 兵五平六 象3退5
 23. 車四退二 卒9進1
 24. 馬三退一 車7平9
 25. 馬一進二 車5進1
 26. 馬二進三 車9平7
 27. 車四平五 馬1進2
 28. 炮六平七 炮3平2
 29. 馬三退五 馬2進4
 30. 炮七進三 卒9進1
 31. 馬五退三 車5退3
 32. 馬三進四 馬4退6
 33. 炮七平四 士5退6
 34. 炮四進三 炮7進6
 35. 車二退六 炮7平5
 36. 炮四平一 炮2進4
 37. 車二進二 車5進2
 38. 兵六進一 炮5平9
 39. 炮一退五 卒9進1
 40. 馬九進七 炮2進3
 41. 相七進九 車5平4
 42. 車二平八 炮2平1
 43. 馬七進六 象3進1
 44. 兵六平五 象5退3
 45. 馬六進八 士4進5
 46. 相九進七 卒9平8

47. 車八平九 炮1平2	84. 馬五退三 炮5平9	25. 仕五退四 卒3進1
48. 相五退三 卒8平7	85. 兵六平五 士6進5	26. 相七進九 馬4退6
49. 車九平六 車4退1	86. 馬三進四 炮9平6	27. 車五進一 象3進5
50. 馬八退六 象1進3	87. 馬四退五 士5進4	28. 車一平七 馬6進5
51. 兵五平六 卒7平6	88. 馬五進六 象5進3	29. 車七平五 馬5退4
52. 馬六退四 象3進5	89. 兵五平四 士4退5	30. 車五平六 馬4進6
53. 馬四進三 象5退7	90. 相五退七 將5平6	31. 仕六進五 馬6進7
54. 馬三退五 卒6平5	91. 相一退三 象3退1	32. 帥五平六 士6進5
55. 馬五退三 卒5平6	92. 帥五進一 炮6平7	33. 車六進一 車8平9
56. 相七退五 炮2退4	93. 相三進五	34. 車六平三 車9退1
57. 馬三進四 炮2平6		35. 相九進七 卒1進1
58. 馬四退二 炮6平5	Shark vs BrainChess	36. 車三進一 卒9進1
59. 帥五平六 炮5平1	和棋	37. 仕五進四 車9平8
60. 馬二進四 炮1退2	1. 炮八平五 馬2進3	38. 仕四退五 車8平9
61. 馬四退三 象7進5	2. 馬八進七 卒3進1	39. 仕五進四 車9平8
62. 兵六平七 象3退1	3. 車九平八 車1平2	40. 仕四退五 車8平9
63. 相三進一 象1退3	4. 兵三進一 馬8進7	41. 仕五進四 車9平8
64. 馬三進五 卒6平5	5. 車八進六 馬3進4	42. 仕四退五 車8平9
65. 相一退三 象5進7	6. 馬二進三 車9進1	
66. 馬五退七 卒5平4	7. 兵五進一 卒3進1	BrainChess vs Shark
67. 帥六平五 象3進5	8. 車八平六 卒3進1	Shark 後勝
68. 馬七退八 卒4平5	9. 兵五進一 炮2平5	1. 炮二平五 馬8進7
69. 兵七平八 炮1進5	10. 馬七退五 馬4進2	2. 馬二進三 馬2進3
70. 馬八進七 卒5平6	11. 兵五進一 馬7進5	3. 車一平二 車9平8
71. 馬七進五 卒6平5	12. 車六平五 炮5退1	4. 兵七進一 卒7進1
72. 馬五退三 卒5平6	13. 車五平八 炮8平5	5. 馬八進七 炮2進4
73. 兵八平七 炮1退3	14. 車八進三 馬2進4	6. 兵五進一 炮8進4
74. 馬三進五 炮1平5	15. 車八退八 車9平8	7. 車九進一 炮2平3
75. 兵七平六 卒6進1	16. 炮五進一 車8進6	8. 相七進九 車1平2
76. 帥五平六 卒6進1	17. 相三進五 車8退1	9. 車九平六 炮3平6
77. 帥六平五 卒6進1	18. 馬五退三 後炮進5	10. 車六進六 炮6進1
78. 帥五平四 炮5進3	19. 前馬進五 卒3進1	11. 兵五進一 炮6平3
79. 馬五退三 炮5平8	20. 車一進一 車8平5	12. 兵五進一 士4進5
80. 馬三退四 炮8進1	21. 仕四進五 車5平7	13. 車六平七 馬7進6
81. 相三進一 炮8退8	22. 馬三進二 車7進1	14. 車二進一 馬6進7
82. 馬四進五 士5進6	23. 車八進五 車7平8	15. 炮五進三 炮3平1
83. 帥四平五 炮8平5	24. 車八平五 車8進2	16. 車二平八 車8進5

17. 炮八平五 馬7進5
18. 馬三進二 炮8平5
19. 仕四進五 馬5退3
20. 帥五平四 馬3進2
21. 馬二進四 炮1進2
22. 帥四進一 炮5平3
23. 兵五平四 象7進5
24. 兵四進一 炮3進2
25. 仕五進六 炮1退1
26. 兵四平五 炮3退1
27. 仕六退五 馬2退1
28. 帥四進一 馬1退3
29. 兵五進一 將5進1
30. 馬四進三 將5退1
31. 車七平五 士6進5
32. 車五平八 馬3退5
33. 車八進二 馬5進4
34. 帥四平五 炮1退1
35. 車八退七 馬4進2
36. 仕五退四 炮3退3
37. 帥五退一 馬2退4
38. 帥五平四 炮3進4
39. 馬三退一 炮1進1

