國立交通大學

資訊工程學系

碩士論文

五子棋相關棋類人工智慧之研究

The Study of Artificial Intelligence Programming for Gobang-like Games

研 究 生：黃德彥

指導教授：吳毅成　教授

中 華 民 國 九 十 四 年 六 月

五子棋相關棋類人工智慧之研究

The Study of Artificial Intelligence Programming for Gobang-like Games.

研 究 生：黃德彥　　Student：Dei-Yen Huang

指導教授：吳毅成　　Advisor：I-Chen Wu

國 立 交 通 大 學

資 訊 工 程 學 系

碩 士 論 文

A Thesis

Submitted to Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science and Information Engineering

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

五子棋相關棋類人工智慧之研究

研究生：黃德彥　　指導教授：吳毅成

國立交通大學 資訊工程學系

摘要

五子棋是一個相當易學，而且相當受歡迎的遊戲，隨著時代的變遷還有人類智慧的累積，現今總是在舊有的規則上面做一些新的變化。如果能針對五子棋相關棋類的 AI 程式設計做探討，使 AI 能夠提高棋力，並且協助檢驗各種相似遊戲的公平性，對於棋與資訊界都將是很大幫助。

本篇論文研究五子棋相關棋類的 AI，把用在五子棋上的人工智慧技術，做些探討與改善應用在五子棋相關棋類上面，並提出緩著評估(Null-move heuristic)擷取有效著手的方式證明 Connect(6,2,3)是一個黑先必勝的遊戲。

The Study of Artificial Intelligence Programming for Gobang-like Games.

Student: Dei-Yen Huang Advisor: I-Chen Wu

Department of Computer Science and Information Engineering

National Chiao Tung University

ABSTRACT

Gobang (Five-in-a-row) is a game which is very easy to learn and so popular. According to the progress of the Artificial Intelligence technology and the accumulation of Human Intelligence, the rules of Gobang are changing based on the original ones. If we can design the AI focus on Gobang-like game, it will increase the score of the AI program and help to check the fairness of Gobang-like game.

This thesis studies AI Programming for Gobang-like Games and improves the original technology build on Gobang in order to apply it on Gobang-like game. It also propose a new null-move search method to prove *Connect(6,2,3)* is a Black-Win game.

誌謝

首先要感謝我的指導教授，吳毅成博士，由於他不厭其煩的細心指導，這篇論文才得以順利完成。

此外特別要感謝汪益賢學長和徐健智學長，他們在研究的過程中給予我許多寶貴的意見和指導，還有家齊學姊之前的研究給我了很大的幫助，以及同實驗室裡一起奮鬥的夥伴志祥、仕全和學弟俊彬、育嘉、怡良、承翰的協助。當然，還有在我遇到瓶頸時給我鼓勵的朋友小涵、表弟宏文、麻吉聖皓、韻如、學弟妹等。還有很多其他我忘了提到的同學及朋友，在我的研究期間，給了我的關懷與鼓勵，陪我度過這段最值得回憶的學生生活。

最後，我要感謝我的父母、姊姊、哥哥，在我的求學生涯中給了我最大的支持和照顧。還有，感謝我自己，你終於辦到了。謹以此論文，獻給我最摯愛的家人。

目錄

圖表目錄

第一章　緒論

1.1 研究目標

五子棋 (Gobang)是一個規則簡單、複雜度極高，相當受歡迎的遊戲。隨著時代的變遷，五子棋也曾因為電腦科技的進步，被完全解出來(基本規則)，但該愛好者卻也不停的演進五子棋的規則，從最早的基本規則到現在國際規則，基本的玩法沒有太大的改變，但是進階的規則讓更多人愛不釋手，迄今亦有像西洋棋、圍棋一樣有國際性職業棋會與比賽，國外更有一句名言：“5 MINUTES TO LEARN, A LIFETIME TO MASTER.”。

棋類人工智慧一向都是電腦科技發展的重要指標，圍棋、西洋棋、象棋，各式各樣的棋類人工智慧發展時有所聞，不同的棋類 AI 依據個別的遊戲性質與規則發展各自不同形式的搜尋技術(Game Tree Search)、評估函數(Heuristic Function)等技術。

因此，本篇論文的目標在於五子棋依循著舊有的玩法且不停的演進的特性，以此五子棋相關遊戲為人工智慧程式的研究範疇，發展相關遊戲的人工智慧演算法，並提出可解決部分遊戲的相關技術。

1.2 基本定義

1.2.1 五子棋

五子棋(Gobang)，是一個在二維棋盤上雙方輪流落子的遊戲，在任一方向(橫向、縱向、左斜、右斜)連成五子者獲勝，所以部份英譯為 Five-in-a-row。它起源於中國古代的黑白傳統棋種之一。相傳早在

堯造圍棋之前，五子棋遊戲在民間已經相當盛行了。在中國的文化裡，倍受人們的青睞[19]。沒有任何禁著的規則限定的五子棋，我們稱之為基本規則五子棋(Gomoku)。

日式規則 Renju：五子棋於何時傳到國外，已無可考，但隱約是在南北朝時期傳入朝鮮、日本等地方，而後由日本改良。在日本五子棋被稱之為五目碰，直至明治 32 年，在報上公開徵名，其中有連五、連五子、五丁、格五、五連棋等等的名稱、最後採用『連珠』(Renju)的名稱，並統一競賽規則。並不斷的改良，陸續出現了各式禁手等(如圖 1-1)的規則，並於 1931 年把棋盤也改成了 15×15 的專用棋盤[12][19]。

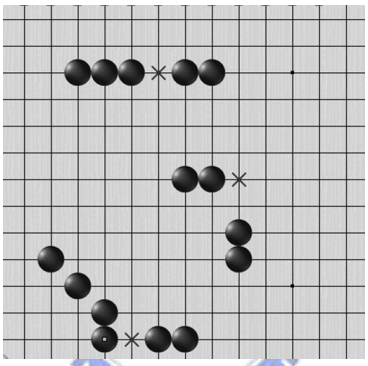
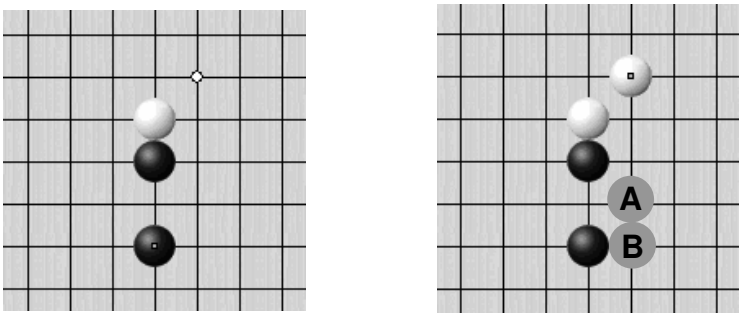


圖 1-1 五子棋日式規則(Renju)的禁手



(a) (b)

圖 1-2 五子棋 RIF 開局規則

RIF 規則：公元 1988 年 8 月 8 日，由日本、俄羅斯、瑞典、亞美尼亞、亞塞拜然、愛沙尼亞、法國、拉脫維亞、白俄羅斯九個成員

國在瑞典成立國際連珠聯盟，簡稱 RIF[16]。並且在日式規則上訂立 RIF 開局規則[17]：

1. 先由假先手挑選 26 棋形當中其中一種。(如圖 1-2 (a))
2. 由假後手決定雙方何者執黑，何者執白。自此黑白確立。
3. 在黑方第五手，必須要下兩個非對稱的點，供白方挑其中一。(如圖 1-2 (b))
4. 此後規則同日式規則。

1.2.2 五子棋相關棋類遊戲

後來更有許多數學家或者是資訊界的前輩們，把五子棋略做延伸，發展出以下遊戲跟規則，並從事相關的研究。

1. k 子棋(k-in-a-row)：將五子棋當中雙方獲勝的規則提升至要連成 k 顆子才能夠獲勝[14]。
2. (m,n,k) Game：在 m × n 的盤面上，雙方輪流下一顆子，先連成 k 顆子者獲勝。例如：Tic-Tac-Toe 就是(3,3,3) Game、Gomoku 五子棋遊戲是 (15,15,5) Game、k-in-a-row 可以視為 (∞,∞,k)Game[11][13][21]。
3. $A_k(p_1,p_2)$：一個回合當中，黑方(先手)可以下 p_1 顆子，白方(後手)可以下 p_2 顆子，先連成 k 顆子者獲勝[14][15]。
4. Connect(m,n,k,p,q)：在 m × n 的盤面上，除了第一回合先手僅能下 q 顆子以外，雙方輪流下 p 顆子，先連成 k 連續子者獲勝。我們在本論文中將把，Connect(∞,∞,k,p,q) 簡稱為 Ren(k,p,q)，Connect(6,2,1)又簡稱為 Connect6[23][24]。(如圖 1-3 是 Ren(6,2,1) 的範例棋局)

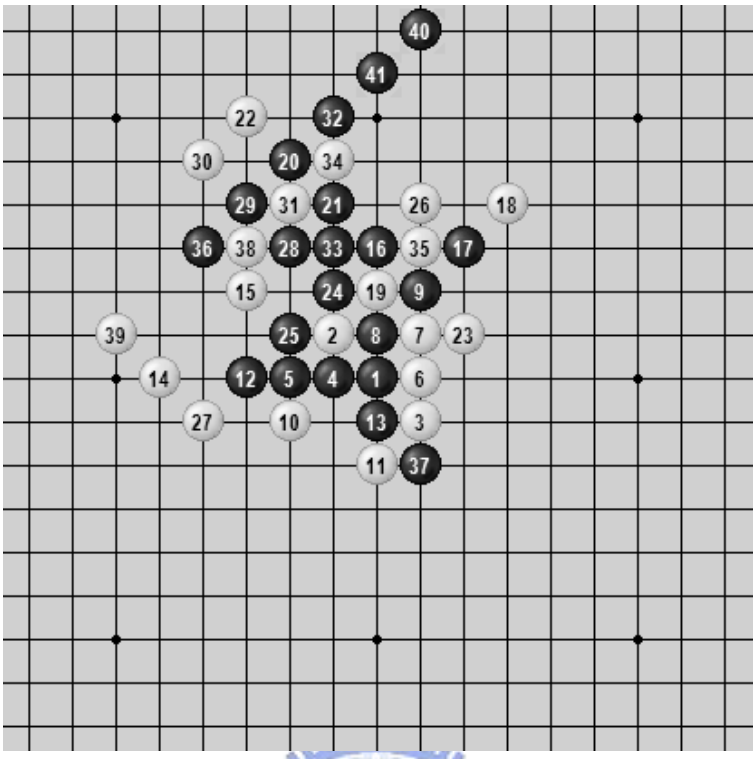


圖 1-3 Connect6 的範例棋局

但是，為了探討這些棋類的方便性起見，研究這些棋類的人又把這些棋類個別規劃成兩種版本[15]：。

1. Maker-Maker Version：是標準的版本，不論先手或是後手都是連成 k 顆子獲勝。
2. Maker-Breaker Version：連成 k 子獲勝的條件，僅適用於先手(Maker)，後手(Breaker)就算連成 k 顆子也不算獲勝。

這兩個版本的不同，我們以圖 1-4 作為說明，Maker-Maker Version 的 Tic-Tac-Toe 是一個和局的遊戲，但 Maker-Breaker Version 卻是一個先下必勝的遊戲。

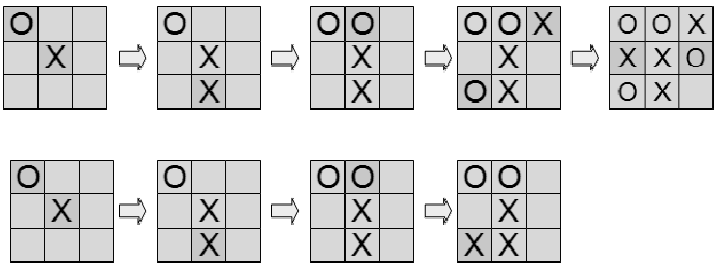


圖 1-4 Maker-Maker(a) and Maker-Breaker(b) Tic-Tac-Toe

會分成這兩種版本的主要意義在於：若是我們僅僅對於 Maker-Breaker Version 做討論的話，就不用多考慮後手會連成 k 顆子獲勝的可能性，整個遊戲就變成考量先手會不會獲勝即可，若是在 Maker-Breaker Version 可以後手白子可以阻止先手獲勝的話，應用相同的策略，在 Maker-Maker Version 的狀況底下，後手依舊可以和棋，甚至是贏該盤棋。

所以，在理論上研究遊戲性質，研究學者們才定義出 Maker-Breaker Version 便於討論，但若針對遊戲的公平性而言，Maker-Maker Version 才是對先後手雙方比較公平的遊戲模式。

1.3 論文大綱

在本論文第二章的背景說明裡，介紹一些前人的研究與各種五子棋相關遊戲的研究現況。第三章介紹我們所提出或改良一些五子棋及相關棋類的棋類 AI 演算法。第四章介紹我們所提出在五子棋及相關棋類可以使用的緩著評估(Null-move heuristic)概念如何篩選出有效的著手(moves)並完整的證明出 Ren(6,2,3) 黑先必勝。第五章是結論與未來的展望。最後在本篇論文的附錄中，附上一些 AI vs Human 與 AI self play 的棋局。

第二章　背景說明

本章介紹五子棋(Gobang)和五子棋相關棋類(Gobang-like Game)的發展背景。在五子棋人工智慧的發展過程當中，有兩個主要的方向，一者是由數學定理的方式去解決無窮大盤面的問題，再者是利用電腦的相關技術去解決有限盤面的問題。以下第 2.1 節介紹目前被解決的 k 子棋遊戲。第 2.2 節五子棋基本規則與日式規則的解法。第 2.3 節介紹五子棋相關棋類的發展現況。

2.1 被解決的 k 子棋遊戲

在一開始的時候，由於 k 子棋(k-in-a-row)沒有禁著，所以對於先下的人，在盤面上具有一定的優勢，並且利用，“模仿對手下棋策略”(Strategy-stealing argument)[21]可以證明這一類型遊戲，後下的不能獲勝。

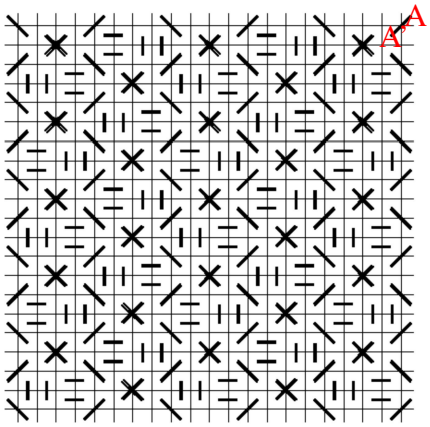


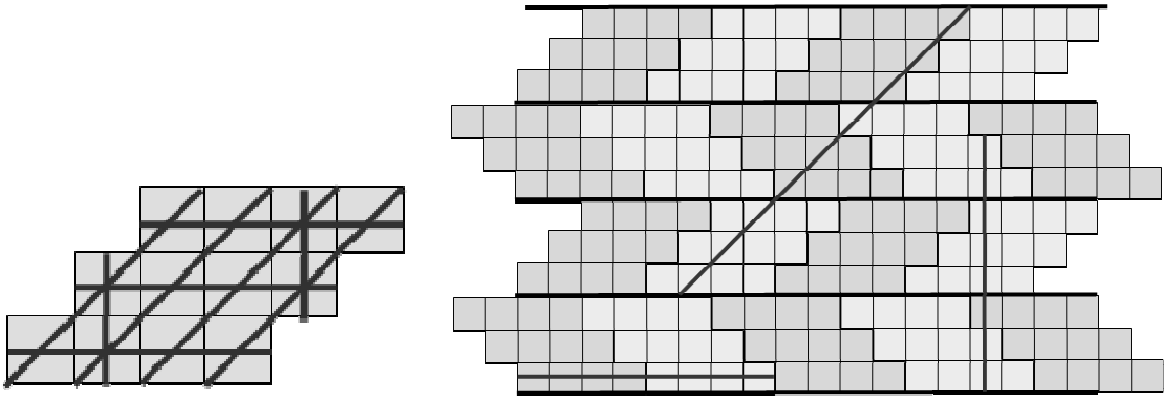
圖 2-1 9-in-a-row Hales-Jewett pairing strategy

(Courtesy of Hales, Jewett[10])

接著由於 5-in-a-row 不是一個這麼好解的問題，大家紛紛把矛頭指向 k-in-a-row 的遊戲，由於當 k 越大的時候，連成 k 顆子的目標就越難到達，Jewett 跟 Hales 等人利用 Hales-Jewett pairing strategy(圖 2-1)證明 9-in-a-row 是一個和局的遊戲[10]：將無限

大的棋盤按照該 pattern 下子，如果你的對手下在 A 處，那麼就下在相對應的 A’ 處，如此一來雙方都可以避免對方獲勝。

在 8-in-a-row 的部份，也是利用類似的技巧，Zetter 首先證明小區塊(如圖 2-2(a)) 不能連成那些作記號的進攻線，組合成大的盤面的時候(如圖 2-2(b))也就沒辦法構成八子以上的連線[26]。



(a) (b)

圖 2-2 Pattern for proving 8-in-a-row

而七子棋和六子棋一直到目前為止，一直到現在為止都還沒有被證明是黑勝或者是和局[13]。

2.2 五子棋基本規則與日式規則的解法

在五子棋的職業棋界，早在許多年前認為基本規則(Gomoku)應該是一個黑先必勝的遊戲，不過一直無法證明，一直到了 1994 年 Allis 才將利用“迫著搜尋”(Threat Space Search) (我們將在第 3.3 節作詳盡的介紹) 將 15×15 還有 19×19 的棋盤的 Gomoku 給予證明[1][2] 。2001 年，接著由 Janos Wagner、Istvan Virag 兩位應用職業棋士下開局與迫著搜尋接著下中盤的方式，證明有禁著的日式規則 Renju 依舊是個黑先必勝的遊戲[22]。

2.3 介紹五子棋相關棋類的發展現況

在最近幾年,先後由Pluhar、Wu提出雙方下多顆子的k-in-a-row遊戲，並提出一些相關的定理。

首先是有關於 Pluhar 在 2002 年所提出的 The accelerated k-in-a-row 遊戲(以下我們稱之為 $A_k(p_1, p_2)$)[14][15]。Pluhar 為了討論的方便性起見，僅僅討論 Maker-Breaker Version 的定理結果，以下列出幾個 Pluhar 提出相關的定理：

定理一：若 $k \geqq n+80\log_2 n+160$ *且* $n \geqq 1000$ *則* $A_k(n,n)$ *後下的可以逼和棋局。*▌

定理二：若 $k \leqq n+\log_2 n/\log_2(\log_2 n)-1$ *則* $A_k(n,n)$ *先下可必勝。*▌

但是 Maker-Breaker Version 判別獲勝的方式，在遊戲性上對於後下的人是相當不公平的，除了利用有限的部分 Maker-Breaker 的結果來探求 Maker-Maker Version 的遊戲結果外，直接研究 Maker-Maker 的遊戲結果、或是設計 AI 仍是有其必要性。另外，比較可惜的是在以上定理中，雙方每回合所下的子都要上千顆，對於遊戲的即時娛樂性就低了許多，所以研究每回合落子數少些的研究，是值得探討的。

在 2005 年時，Wu 提出 Ren(k,p,q)系列[23][24]，除了雙方每個回合都下 p 顆子之外，並且利用 q 這個參數，限定先手第一個回合僅能下 q 個。若是當 q 比 p 小的時候，便閃避了 Strategy-stealing argument，先下的不可以採用 Strategy-stealing 的方式獲取優勢，後者也是有可能獲勝的。如 Ren(6,3,1)、Connect(6,3,2)。

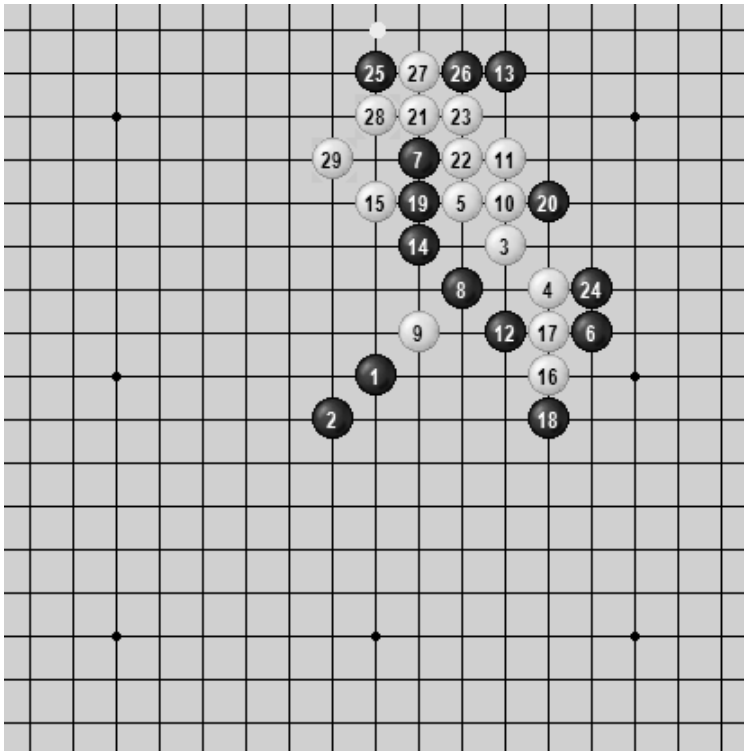


圖 2-3 Connect(6,3,2) White win

現況，五子棋相關遊戲的發展到雙方都可以下多顆子，如何將舊有的五子棋 AI 雙方僅僅只能下一顆子的概念，推廣到在棋盤上選擇多顆子就是我們本篇論文所要探討的核心。

第三章 五子棋相關棋類的 AI 程式設計

在這一章中，主要介紹我們提出改善現有的 AI 演算法，推廣原有的五子棋 AI 的想法到所有五子棋相關棋類遊戲上面。首先在第 3.1 節中介紹迫著策略(Threatened-based strategy)。接下來在第 3.2 節講解著手生成器(Move generator)。第 3.4 節介紹評估函數(Heuristic Function)。第 3.5 節介紹迫著搜尋(Threat Space Search)。第 3.6 節提出兩階段搜尋樹的架構。

3.1 迫著策略(Threatened-based strategy)

以五子棋而言，往往在比賽的後段其中一方以連續進攻的方式獲得比賽的勝利，有以下兩個專有名詞[19]：

1. VCT :所謂 VCT 就是連續活三取勝，它是英文 Victory by Continuous Threes 的縮寫，連續活三(包括連三跟跳三)。但是，每一著活三都是先手，都迫使對方防守，最後形成取勝棋形。
2. VCF: 所謂 VCF 是英文 Victory of Continuous Four 的縮寫，即以連續不斷的衝四取得勝利。

在五子棋這個遊戲中，活三、與死四屬於進攻者的強勢進攻，防禦者僅有幾種方式可以防守，在此我們先簡單稱之為“迫著”(Threats) [1][2][8][9]，在第 3.1.1 小節會有更嚴謹的定義介紹。

在其餘跟五子棋相似的棋類也有類似的特性，所以實作出一套系統，可以在盤面上發現:下棋應該要下在哪個位置有 Threats 出現，或者下在哪可以避免對手有足夠多的 Threats 可以獲勝，都能夠大幅的提高估算著手(move)分數的精準度。在此節當中，將會詳盡的介紹。第 3.1.1 小節 迫著的定義與演算法。第 3.1.2 小節 迫著前置棋型的定

義與演算法。第 3.1.3 小節 介紹線狀棋型資訊的產生的方式與如何應用在 AI 程式設計上。

3.1.1 迫著的定義與演算法

在五子棋的研究範疇中，迫著(Threats)一詞首先由 Allis 所提出，並且利用 Threats 的概念與 Threat Space Search 的方式來證明 Gomoku 是一個黑子先下必勝的遊戲[1][2][3]。以下我們將 Threats 的概念做數量化的延伸，並且定義出五子棋相關棋類的 Threats 計算方式。

定義一：我們稱一個盤面某方有 t 個 Threats iff 另外一方沒有辦法馬上獲勝，而且至少需要花 t 顆子阻止某方再下一個回合當中獲勝。▌

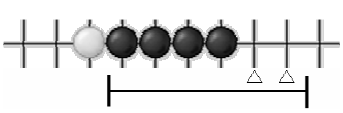
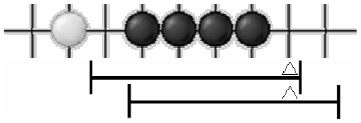
在 Allis 証明 Gomoku 是黑方必勝的文中[2]，他把活三、死四當成 Threat 並且利用 Threats Space Search 來搜尋進攻方是否存在有 Threats 連續進攻，一直到達進攻方有 2 Threats (四四、四三、三三、活四)為止。符合以上的定義：死四是 1 Threat 因為在防守方必須花一顆子要防守。活三是 Threat 是比較特殊一點的，因為若是再一個回合之後，進攻方可以一次產生兩個 Threats，此時要防守，在 Gomoku 的遊戲規則下，雙方僅僅只能下一顆子，於是就會來不及防守，此處我們將其另給一個名稱叫做 Delayed Threat，但是在雙方都可以下多子的狀況底下，防守方可以兩面夾殺一個 Double Threats 所以 Delayed-Threat 便比較不重要了。

以下我們列出如何計算一個線狀棋型(line-pattern)的 Threats 有幾個的演算法：

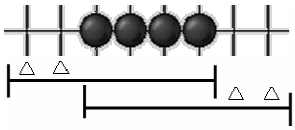
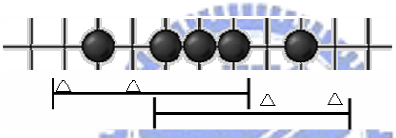
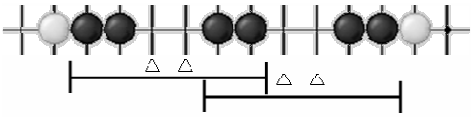
1. 針對 line-pattern 由左自右，我們把連續 k 個子視為同一個切割(Sliding Window)

2. 針對每個 Sliding Window，我們做以下的步驟
3. 若是 Sliding Window 內如果沒有任何對方的子、沒有子有特殊記號、而且該 Sliding Window 內超過 k-p 顆己方子的話，我們把該 line-pattern 的 Threats 數加 1 並且將該 Sliding Window 裡面所有的空點作特殊記號。

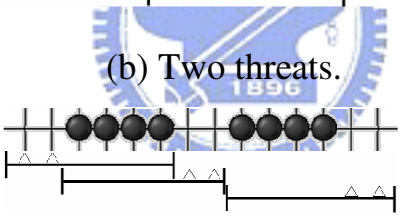
如圖 3-1，我們列出一些 Connect6 的 Threat，下方是計算 Threats 的 Sliding window 和被做特殊記號(△)的空點。

(a) One threat.

(b) Two threats.



(c) Three threats.

圖 3-1 Connect6 的 Threats 範例

3.1.2 迫著的前置棋型定義與演算法

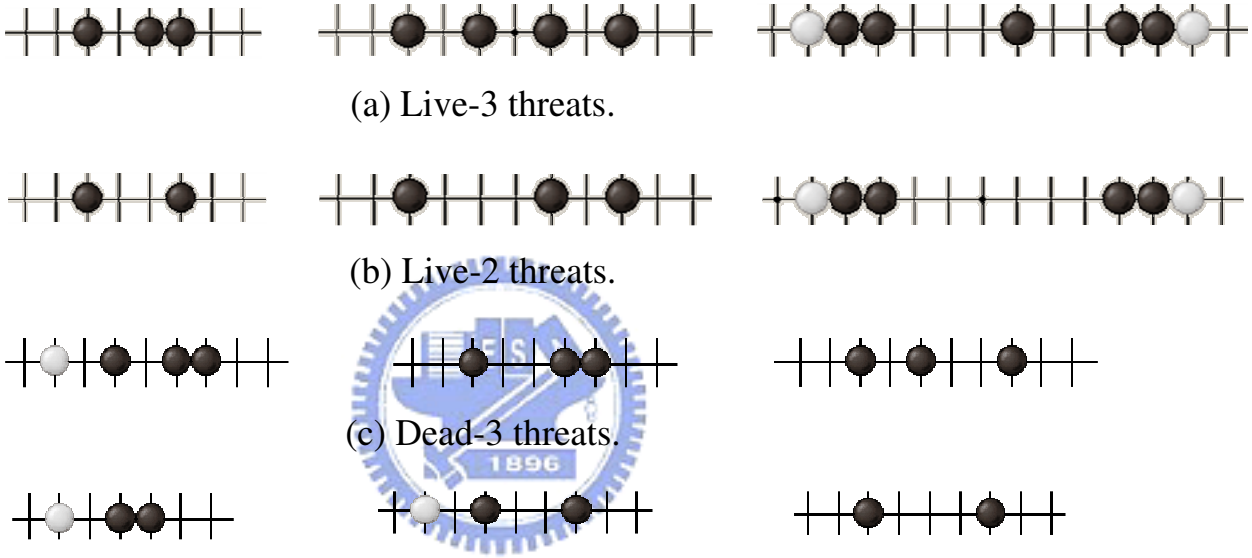
在上一小節當中，我們已經可以清楚的判斷一個 line-pattern 有幾個 Threat。但像是五子棋當中一些 Threats 的前置棋型也是相當重要的，像是活三接著可以形成活四(2 Threats)，死三可以形成死四(1 Threats)，在五子棋相關的遊戲當中，類似的棋型也是重要的，我們清楚的定義如下：

定義二：假設在一個 line-pattern 上，如果可再多額外下 $k - p - l$

顆子，就可以造成有 1 個 Threat 的棋型，我們稱之為“死 l”。

定義三: 假設在一個 line-pattern 上，如果可再多額外下 k – p –l 顆子，就可以造成有 2 個 Threats 的棋型，我們稱之為“活 l”。

我們舉 Connect6 為例，活三、死三、活二、死二(圖 3-2)，都是相當重要的棋型，因為再過一個回合，就可以創造出防守方不得不擋 Threats。



(a) Live-3 threats.

(b) Live-2 threats.

(c) Dead-3 threats.

(d) Dead-2 threats.

圖 3-2 Connect6 Threat 前置棋型

3.1.3 線狀棋型資訊

在此節當中我們討論，如何把以上兩個小節的 Threatened-based strategy 實作到 AI 當中。我們舉 Connect6 為例。首先，在針對盤面上的空點，我們需要下子下在該空點後的盤面結果，用以判斷下子下在該點的功用(圖 3-3)，我們稱之為線狀棋型資訊(Line-pattern information)。方法有二：1.在 AI 執行的過程中，設計一個函式自動做計算。2.我們預先運算所有的線狀棋型(line-pattern)上面的空點可以造成的效果，然後把它記在表裡，等到 AI 需要使用的時候，

在由 Table 上直接抄出。在實作這兩種方法的時候，我們發現，第一種方法，很難正確的分辨死活，如圖 3-4 該點是一個活三點，卻很難判斷出來。因此我們採用第二種方式來為 AI 產生棋型資訊。

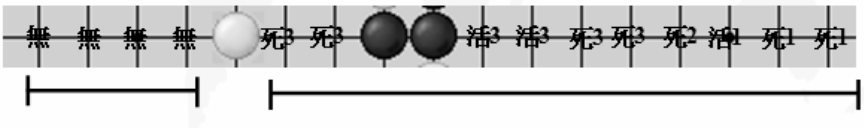


圖 3-3 line-pattern Information

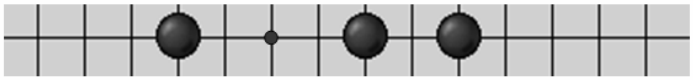


圖 3-4 比較難判斷的活三點

首先我先把所有的棋形所代表的意義先按照上兩小節的定義產生出來。然後針對每一個 line-pattern 上面的空點，做判讀，若是原本的 line-pattern 沒有活三，下在該空點之後，有活三產生，我們成為該空點為活三點。若是該 line-pattern 上面已經有活三，下了某個空點，依舊還是只有活三(沒有產生 Threat)，該空點我們視為一般點。演算法如下：

1. 針對每一個棋型上面的每個空點作以下的步驟
2. 先找出原有 line-pattern 所代表的訊息(info 1)。
3. 再取針對空點下子過後 line-pattern 所代表的訊息(info2)。
4. 該空點所代表的訊息及為將 info 1 提升到 info 2 的價值。

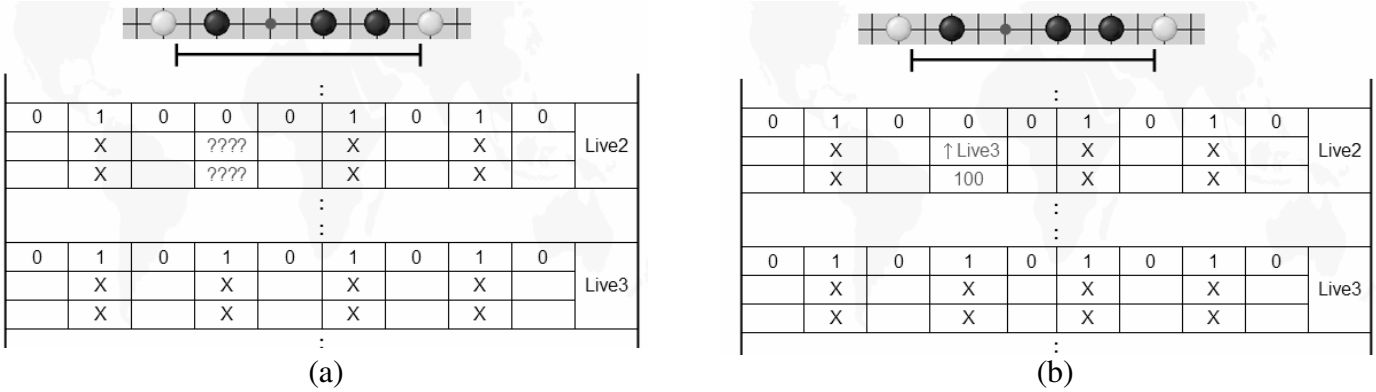


圖 3-5 Line-Pattern Table 生成點資訊

我們採用圖 3-5 的例子來做解說，若是我們已經建立好每一個棋形所代表的意義之後，接著我們要建立 line-pattern table 以供我們查詢空點資訊和分數。如圖 3-5(a)我們所希望知道小圓點的分數，於是我們就先去查詢若是該點下過之後會是什麼情形，如 3-5(a)下方圖我們可以得知該點下過之後是一個活三的棋型，則該紅點所代表的意義即為“上升到活三”，我們就可以給予他一個分數跟資訊(如圖 3-5(b))。

以上的方法亦可用來做出 line-pattern 防守的資訊另外做成一張表提供所有 line-pattern 的防守資訊。

在產生 line-pattern table 之後，在盤面上的判讀上，只需要合併該點四個方向的資訊，就可以設計評估函數(Evaluation Function) 對該點評分。

但是產生所有的 line-pattern table 需要花的記憶體約需要 80MB(其計算式子如以下)：

- 所有的格點數：$1\times2 + 2\times2^2 + 3\times2^3 + \ldots + 19\times2^{19} = 18\times2^{20} + 2$。
- 所需要的記憶體量：$(18\times2^{20} + 2) \times 2 \text{ bytes} \times 2 \sim 80\text{MB}$。

若是棋盤太大的話，對於 Memory 將會是個很大的問題，所以如何實作更大棋盤，甚至是無限大棋盤的 Line-pattern Table，都是將來可以探討的議題。

3.2 著手生成器(Move generator)

在五子棋相關棋類另外有個共通特色，就在雙方在每個回合是下多顆子的方式在進行遊戲，每個回合落下的子我們稱之為著手(move)，原本盤面上的點已經很多了，現在又要在眾多的點當中挑取 p 個，複雜度是 $O(n^p)$，我們提出以下的著手生成器來挑選出我們所

需要的 move。我們以 Connect6 為例，提出以下的演算法：

1. 先在盤面上選擇 w 分數比較高的空點。
2. 針對每一個我們剛選出來的空點 g_i 作以下的動兩個步驟。
3. 在 g_i 下子後，更新盤面資訊。
4. 選擇盤面上最高分的 w_i 個空點，並針對此步驟選出的空點 g，與 g_i 組成一組 move 放置進候選的清單之中。(若是已在清單之中的話，可以另選空點後補)。
5. 經過了所有以上的步驟之後，從候選清單之中選擇 w' 個最好的著手。

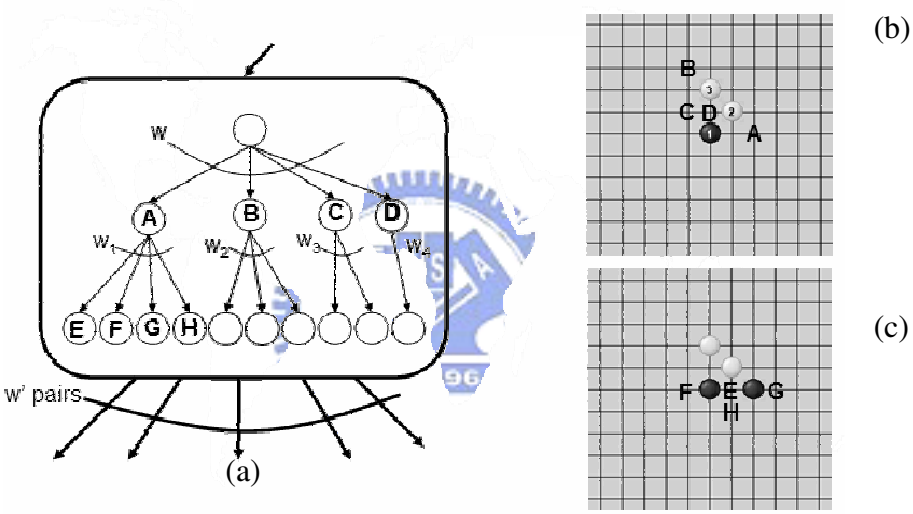


圖 3-6 Connect6 Move Generator

我們舉個例子來說明以上演算法，目前盤面是(b)圖，我們要產生合適的著手進入 AI 搜尋，第一步驟：先找出盤面上分數最高的 4 個點(如何評定分數，我們在下個小節當中做介紹)，然後接下來依次下子這 4 個點上，首先我們下在 A 點上的話(圖(c))，更新棋盤盤面的資訊後，可以發現 EFGH 是分數最高的四個點，所以(A,E) (A,F) (A,G) (A,H)都是我們的候選 moves。如同我們處理 A 點的方式接著處理 BCD 分別找出 w_2, w_3, w_4 個候選 moves，接著我們在針對這些候選 moves 盤面，找出造成盤面最高的 w' 個 moves，作為我們搜尋的分支所用。

此一方式亦可以用於其餘的五子棋相關遊戲，若是同一個著手中可以允許下 p 顆子的話，那麼像圖 3-6(a) 節點內部的樹狀結構，高度就會有 p 層。

如此樹狀的著手生成器找尋合適的著手，複雜度比 $O(n^p)$小，但是選出的著手優劣，僅比窮舉方式略差一些。不過，若是那些一個著手可下的 p 顆子，p 很大的遊戲，將會對於 Tree-based Move Generator 造成負荷。

3.3 評估函數

在本節當中將會列出兩個層面的評估函數，一個是針對盤面上的空點作分析用，另外一個是針對整個盤面評估出一個總分給搜尋使用。第 3.3.1 小節介紹空點評估函數。第 3.3.2 小節介紹盤面評估函數。

3.3.1 空點的評估函數

空點的評估函數其作用在於幫助著手生成器可以選擇出比較好空點位置去組合出一個著手出來，棋盤上每個空點上的資訊應該都會有四個方向的線狀棋型資訊，如圖，在該點位置對於黑子而言有進攻資訊{活三，活二，活一，死一}四個方向的資訊，防守亦有{無，無，無，守活一}，我們要依據這些資訊來為這個點評分，採用以下表格的分數。

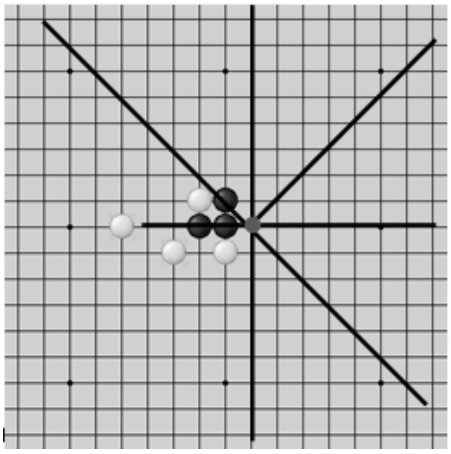


圖 3-7 每個點四個方向訊息的示意圖

進攻分數表	一	二	三	四
活	125	500	1200	6000
死	25	100	400	2000

表 3-8 空點評估的進攻分數表

防守分數表	一	二	三	四
活	250	1800	2400	15000
死	50	200	800	15000

表 3-9 空點評估的防守分數表

- 例如圖上該點的分數就應該為

  進攻分數：1200+500+125+25=1850

  防守分數：0+0+0+250=250

  一共有：2100 分

3.3.2 盤面的評估函數

盤面的評估函數應用在 Alpha-Beta 搜尋對局樹當中的葉節點 (leaf nodes) 去評估該點的雙方優勢的分數，對己方有利的話為正分，對對方有利的話負分，並且分數要落在最大值跟最小值之內，若是已經篤定獲勝的盤面的話，就給予最大值，若是篤定失敗的盤面，就給最小值。

在我們設計的六子棋的 AI 內，以自己下過棋後，要輪到對方下的時機點上評估盤面，並將演算法與給分規則，列出如下：

條件(依判定的優先次序列出)		給分
1.我方連六		1000000
2.對方有 1 個 Threat 以上		-1000000
3.我方有 3 個 Threat 以上		20000
4.我方有 2 個 Threat		5000
	另外每多 1 個活三	+2000
	另外每多 1 個活二	+1000
	另外每多 1 個死三	+1000
5.我方有 1 個 Threat		2500
	對方有活三	跳到 6.判斷
	對方有活二	-2000
	對方有死三	-2500
	自己每多 1 個活三	+1000
	自己每多 1 個死三	+500
	自己每多 1 個活三	+500
6.對方有活二或活三或兩個死三		-5000
	對方每多 1 個活三	-2000
	對方每多 1 個活二	-1000
	對方每多 1 個死三	-1000
7.其他		
	對方有死 3	-2000
	對方有死 2	-800
	對方有活 1	-1000
	我方有活三	3500
	我方有死三	600
	我方有活二	600
	我方有死二	120
	我方有活一	150

表 3-10 計算盤面的分數表

3.4 迫著搜尋

在本節當中將會介紹將五子棋當中找 VCT 與 VCF 的 AI-Search 迫著搜尋方法(Threat Space Search，後面我們簡稱為 TSS)，第 3.4.1 小節介紹基本的演算法。第 3.4.2 小節介紹保守防禦的加速 TSS。第 3.4.3 小節列出相關的比較結果。

3.4.1 迫著搜尋的基本演算法

TSS 是由 Allis 在證明 Gomoku 遊戲是黑子必勝時一併提出：在一般的 Search 當中，只留下與 Threats 相關的分支(包含進攻、防守與反攻)做展開，利用迫著之間的相關性和細而長的搜尋樹找尋進攻方連續迫著獲勝的進攻方式(Winning Threats Sequence)，若是失敗的話回傳沒有找到(Fail) 。

在此另有一個重點需要提出，以往因為五子棋的概念雙方每個回合僅僅只能下一顆子，所以一旦進攻方有 Threat 出現，防守方僅能下的那顆子就肯定被牽制下在少數可以防守 Threat 的點上，一旦進攻方達成盤面上有 2 個以上的 Threats 的話，進攻方即可宣告獲勝。而五子棋相關棋類遊戲則不然，若是雙方都可以下 p 顆子的狀況的話，進攻方要能完整牽制防守方的防禦點就必須要在盤面上下出 p 個 Threats，如此一來防守方的 move 才有辦法被預測。一旦進攻方達成在盤面上有 p+1 個(含)以上的 Threats 才可宣告獲勝。

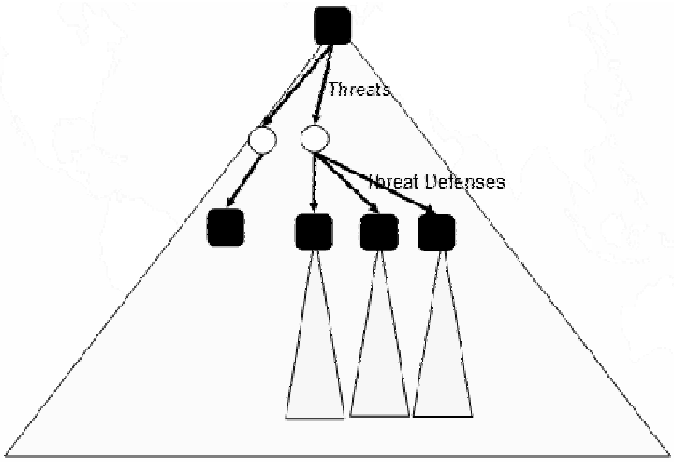


圖 3-11 TSS 示意圖

3.4.2 保守防禦

TSS 在進攻方下子過後，白方防守往往不會僅有一種擋法，(如圖 3-12，{A,C},{B,C},{B,D})，在搜尋樹(Game-Tree)上頭就會產生數棵子樹(sub-tree)並在搜尋過三棵子樹都回傳是進攻方必勝的話，那麼我們就可以說，進攻方可以獲勝。

圖 3-12 保守防禦示意圖

但如果我們採取保守一點的想法：假設黑方此時允許白方把所有可以防禦的點({A,B,C,D})通通下滿(我們稱之為保守防禦 Conservative Defense)的話，進攻方仍就可以找到一組 Winning Threats Sequence 必勝，其實運用同樣的 Winning Threats Sequence 在三組防守點是拆開防禦的狀況底下，依舊是可以獲勝的。但此時，搜尋樹就不必搜尋三棵子樹而僅僅做一個 Conservative Defense 的 sub-tree 就好，藉此我們希望藉此可以提高 TSS 的搜尋效率。

依據 Conservative Defense 的特性我們可以規劃以下三種 TSS：

1. 傳統型 TSS (Traditional TSS)：僅依據原本 TSS 做搜尋的工作搜尋典型的分支，不包含 Conservative Defense 的分支。
2. 保守型 TSS (Conservative TSS)：所有的 TSS 的防守分支當中，僅僅執行 Conservative Defense，不執行所有的分支情況。(圖 3-13(a))
3. 混合型 TSS (Hybrid TSS)：所有的 TSS 的防守分支，首先執行 Conservative Defense，若是沒有辦法回傳進攻方必勝，再執行點典型的防守分支。(圖 3-13(b))

但是使用保守型 TSS 的狀況底下，若是回傳出找尋失敗(Fail)，其實並不代表進攻方就沒有必勝路線，在下一小節我們提出一個例子，在傳統型 TSS 下是可以取得 Winning Threats Sequence 而在 保守型 TSS 是找不到的，並且在下一節以傳統型 TSS、保守型 TSS、混合型 TSS 做實驗比較。

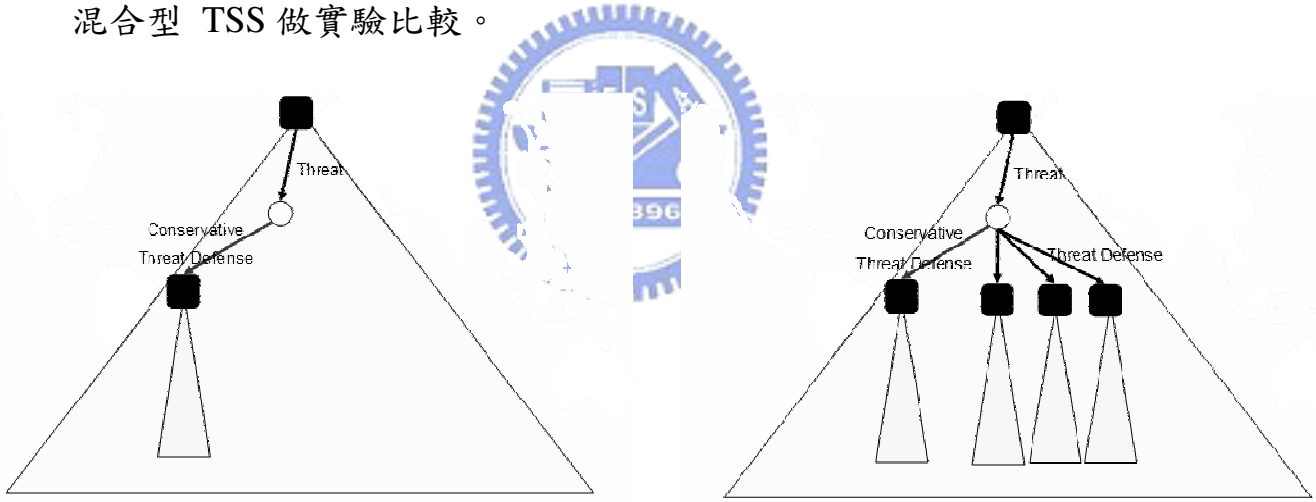


圖 3-13 非典型 TSS 示意圖 (a) 保守型 TSS (b) 混合型式 TSS

3.4.3 實驗比較結果

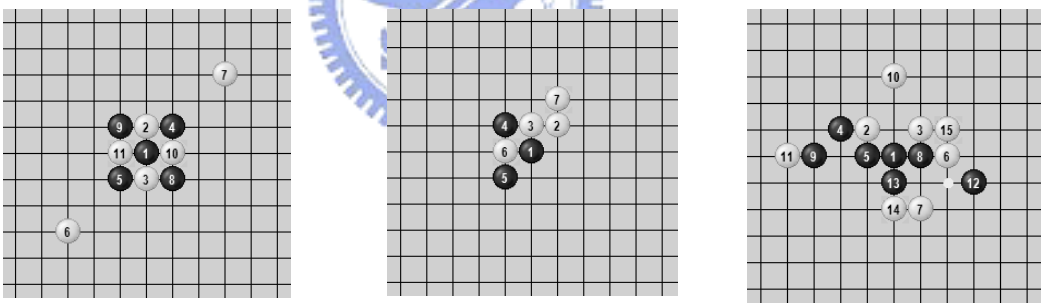
在做三種 TSS 的數據比較實驗之前，由於五子棋相關棋類的遊戲人類對局的棋譜並不多，我們採用 AI 自動對局的棋譜來作測試。首先我們讓 AI 對局的時候，每一次都展開最好的 10 個 move 紀錄完整的三層 Mini-Max Game Tree，扣除盤面重複與對稱的一共有 683 局，以這些對局，讓我們設計的 AI 自動對局到遊戲結束，一共會產生

12743 個盤面，平均每局約 18 個著手盤面。這些盤面我們可以拿來測試三種 TSS 的效能，並且可以把這三種盤面分類如下：

1. 每一種 TSS 都可以找出進攻方獲勝的盤面。此種狀況共有 3054 個盤面，例如圖 3-15(a)即為其中一個例子。
2. 保守型 TSS 找不出進攻方獲勝的盤面。此種狀況共有 77 個盤面，例如圖 3-15(b)即為其中一個例子。
3. 三種 TTS 都沒有辦法找出進攻方獲勝的盤面。此種狀況共有 9612 個盤面，例如圖 3-15(c)即為其中一個例子

Case I	Case II	Case III	Total
3054	77	9612	12743
23.9%	0.6%	75.4%	100%

圖 3-14 每種討論情況佔總盤面的比例



(a) (b) (c)

圖 3-15 TSS 盤面分類的實例

我們以上述的 12743 個盤面來測試三種 TSS 找出 Winning Threats Sequence 的成功率與效能，我們針對每一個不同的分類三種 TSS 作測試，得到的平均數據結果列在表 3-12 中。

Case 1:

Search	Search nodes	Sub-nodes	Return
Traditional TSS	1213	7131	Success!
Conservative TSS	267	2249	Success!
Hybrid TSS	1868	9659	Success!

(a)

Search	Search nodes	Sub-nodes	Return
Traditional TSS	16,873	114,461	Success!
Conservative TSS	1,867	30,372	Fail
Hybrid TSS	40,242	168,093	Success!

(b)

Search	Search nodes	Sub-nodes	Return
Traditional TSS	341	2,158	Fail
Conservative TSS	165	661	Fail
Hybrid TSS	674	3,387	Fail

(c)

表 3-16 三種 TSS 在三種盤面狀況的效能數據表

從上面三個表得知，三種 TSS 的方式在效率方面以 Conservative TSS 的效能最為優越，唯獨美中不足的狀況是，若是一個具有 Winning Threats Sequence 的盤面，Conservative TSS 沒辦法百分百的確定可以找出來(如圖 3-15(b))。在上述的實例中，所有能找到 Winning Threats Sequence 的共有 3131 個盤面，發現保守型 TSS 可以解決當中的 3054 個盤面，成功率高達 98%。而混合型 TSS，雖然可以找出必勝的 100%成功率，但所需要的搜尋節點數，卻是比傳統型多，甚至更大於傳統型 TSS 與保守型 TSS 的總和。

因此，在要求 100%成功率的時候，我們應該採用傳統型的 TSS，在比較在乎效率的時候，我們可以採用保守型的 TSS。

3.5 兩階段搜尋樹

在本章的前三個小節中，所介紹的方法，我將其整合在棋類 Alpha-Beta Search 上面。當進入一個新的 Node 的時候我們先嘗試使用 TSS，看該執方是否可以獲勝，若是可以獲勝的話，就可以直接回傳一個最高分的值，不必再進行其餘的 Alpha Beta Search。

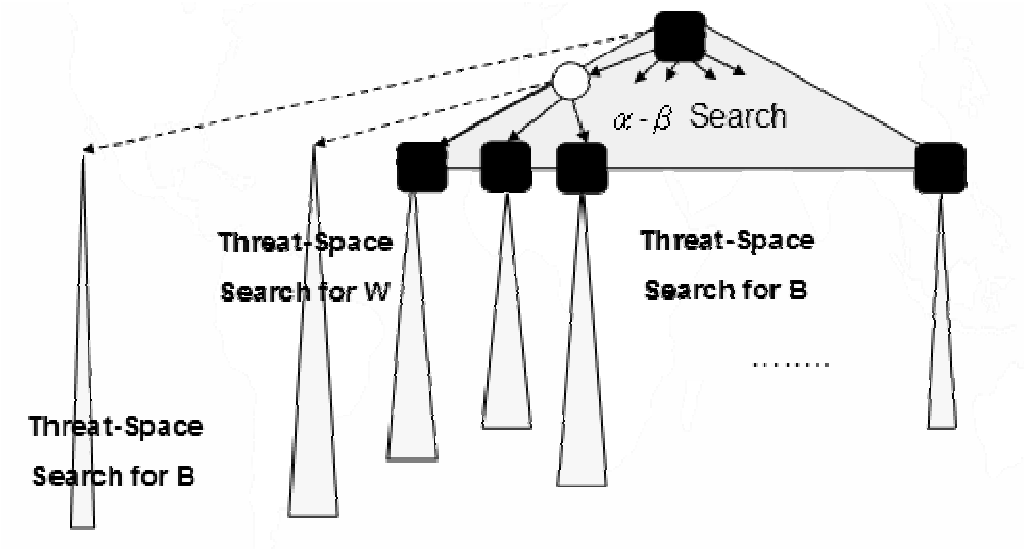


圖 3-17 Two-Level Tree Search

由於上節所探討的結果，在此每一個節點裡面我們所採用的 TSS，我們的作法是

1. 首先先以效能較好的保守型 TSS 搜尋。
2. 若是沒有辦法搜尋到，在思考時間的允許之下，我們繼續以 iterative deepening 傳統型 TSS 搜尋。也就是說，我們搜尋深度由淺至深，從 D=1 開始，如果思考間允許的話，一直搜尋到 D=19 為止，採用傳統型 TSS 做必勝的搜尋。一方面能控制搜尋時間，另外一方面也可以提升 TSS 搜尋的成功率。

採行以上的方法，設計 Connect6 的人工智慧程式，發現該程式棋力能與人類在伯仲之間。並在判別盤面是不是具有必勝走法的時候，能利用 TSS 做出最佳著法。

第四章　緩著評估

在這一章我們介紹緩著評估 (Null-move heuristic) [4][20]應用在五子棋類型的遊戲當中擷取出有效著手(moves)的幫助，並採用該方法證明 Ren(6,2,3)是一個黑子必勝的遊戲。

接下來分別在 4.1 節介紹整體概念，4.2 節介紹取出相關區域 (Relevancy Zone) 的演算法，4.3 節當中以證明 Ren(6,2,3)為黑先必勝為實例，說明 Null-move heuristic 的使用方式。

4.1 緩著評估的整體概念

在與五子棋類似的遊戲中，雙方唯一的目的在於如何連成 k 子獲勝，所以在棋盤上面下子佔有位置就是取得優勢，如果不下子的話，肯定會使得盤面對自己不利。可是往往在輪到己方的時候，在己方未下子之前，往往可以先借助自己先不下子，先行思考對方可以如何下，再來考量現在自己應該用什麼方式預先做些防守。

在上一章當中我們介紹了 TSS 去找出獲勝的 Continuous Threats Sequence ，若是一個盤面輪到白方下之時，白方沒有馬上可以下出 Threats 的地方，而黑方蓄勢待發，醞釀有連攻獲勝的方法，如果假設我們把白方的子下在遠離此棋團處，就像是沒有做任何動作一樣，接著黑方將有連續進攻的方法，可以在連續進攻數層後獲勝。如此一來，我們可以簡單的得到個概念：如果我們原本 null move 白方子應該要能阻止黑方接著連續的迫著進攻才有機會在之後的棋局獲勝，不然就是輸棋。

在下一節當中，我們將會清楚的定義白方子應該要下在何處才能防範黑子不會有連續的 Threats 進攻獲勝。

4.2 相關區域(Relevancy Zone)

針對一連串的可以導致連續進攻獲勝的迫著(Winning Threats Sequence) 進攻我們定義一個相關區域(Relevancy Zone[20]，簡稱RZone)，若是防守方沒有在這區域裏面及早佈子，則進攻可以依照原有的進攻策略強勢獲勝。

Relevancy Zone 包含的有下列三種點：

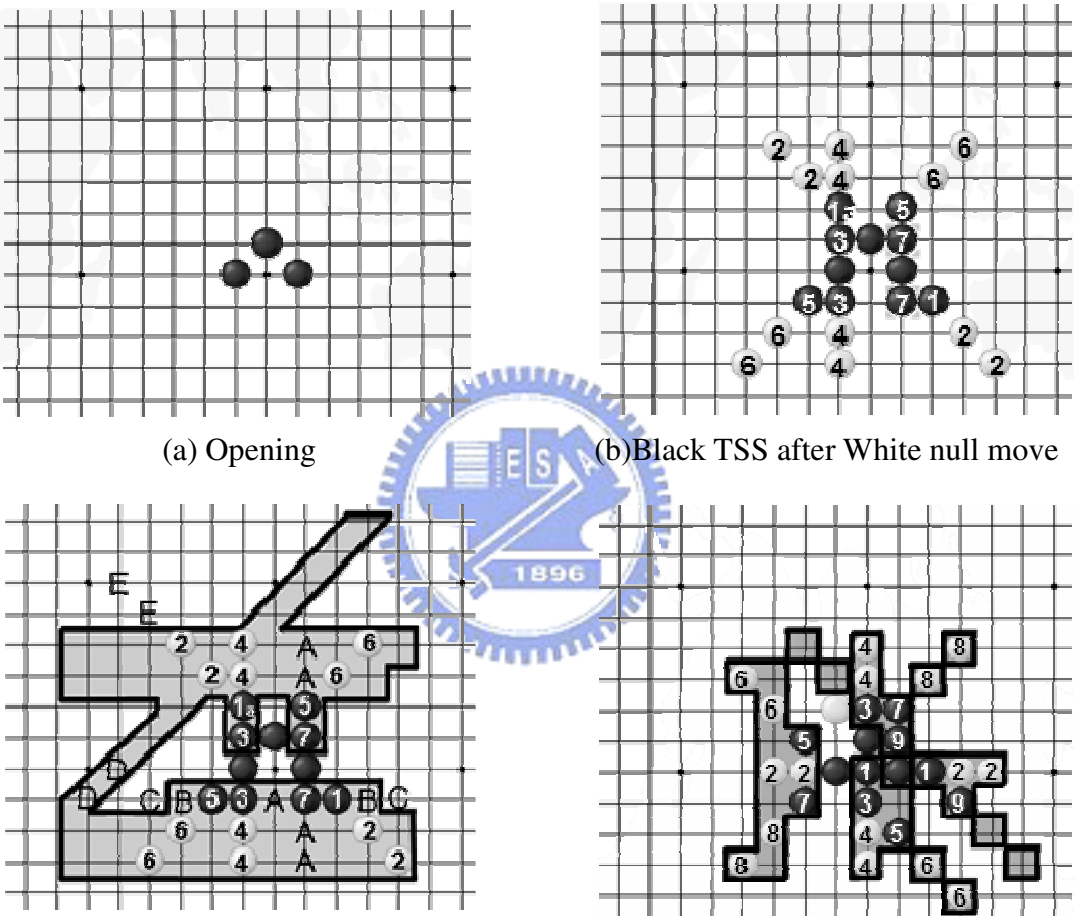
1. 進攻者獲勝 Threats 的防守點：Winning Threats Sequence 會停在 比 p 個大的 Threats 就代表進攻方已經獲勝了，若是防守方在最後的 Threats 的防守點預先佈子，那麼就會使 Winning Threats Sequence 失效了。
2. Winning Threats Sequence 的路線：若是防守方預先在，該 Winning Threats Sequence 的進攻、防守的路線上佈子的話，也會使得該 Winning Threats Sequence 無法順利進行。
3. 防守方的反攻點(inversion-threats point)：因為在進攻方的 Winning Threats Sequence 進行中，若是防守方先行佈子，然後在防守的過程，同時可以產生新的 Threat 的話，將會中斷進攻方的 Winning Threats Sequence。

在下一節當中我們將會用一個實例詳盡的解釋 Null-move heuristic 和 RZone 的應用方法。

4.3 實例解說

在此節當中我們嘗試證明 Ren(6,2,3)這個遊戲為黑子必勝的遊戲。Connect(6,2,3)遊戲的基本定義為：黑方在第一手的時候可以在盤面上面下三顆子，而接著白黑雙方在盤面上輪流落子，每次落子可以下兩顆子。

若要證明黑方必勝的話，在執方輪黑的時候，我們可以指定黑方的進攻模式，但是執方輪白的話，卻應該要對於所有的白方下法都給予證明，證明所有的白方下法都還是會導致黑方必勝。如果下子的盤面是無窮大的話，那麼無窮無盡的白方下法，將成為證明必須要克服的困難。因此，我們打算利用 Null-move Heuristic 的方式，來擷取出有效的白方下法。



(a) Opening (b)Black TSS after White null move

(c)RZone I (d)RZone II for 1a

圖 4-1 Solving Connect(6,2,3) by null-move and R-Zone

首先，我們先假設黑方在一開始在盤面上佈子為圖 4-1(a)的樣式，原本執方應該輪到白方下子，我們假設白方執行 Null-move(或者是想像成白方下在無窮遠處) 接著執方又輪回黑方，盤面已經可以利用 Threats Space Search 搜尋出 Winning Threats Sequence S1(如圖 4-1(b))，並依據 S_1 我們可以依上一節的 R-Zone 定義取出 R-Zone I(圖 4-1(c))，對於 R-Zone I 一些比較特別的點我們以下的解釋：

1. A:A 點是針對防守 Winning Threats Sequence 最後 Threats 的防守點，屬於我們上一節裡面所提到的第一類型的點。
2. B:B 點防守比較特殊。由於該橫向的 Threat 原本應該只需要一子就可以防守(防守在中間)，但是由於我們現在是雙方每個回合可以下兩顆子的遊戲，可以利用兩顆子守住 1 個 Threat，所以在考量第一類型點時，也必須要考慮像是 B 這樣子的點。
3. C:C 點已經距離 Threats 太遠了，沒有辦法有效防守住最後橫向的 Threat。
4. D:D 點是標準的屬於第三類型的點。因為現在是雙方每個回合可以下兩顆子的遊戲，所以只要下兩顆子之後可以構成 Threat(s)的點(活三點、死三點、會構成 Threat 的點)我們都需要考量。
5. E:E 點乍看之下是白子的死三點，但是實則不然，因為在盤面上的死二是由 Conservative Defense 所造成的，所以 R-Zone I 並不包含 E 點。

經過以上的判斷之後，整個 R-Zone I 共有 62 個點，並帶有以下的結論：白方至少有一顆子必須要落在 R-Zone I 當中，不然黑方就可以按照原有的 S_1 獲得勝利。

依據 R-Zone I 的分析，白方在落子之時，其中一顆子必須要落在 R-Zone I 當中，但是第二顆子仍然可以落在棋盤上任意處。因此我們打算再用一次相同的 null-move 策略來侷限住白子第二顆子的範圍。

為了找出第二顆子被拘限的範圍，我們先針對 R-Zone I 的 1a 點下白子，並做 semi-null-move(另外一顆白子先不下，或想成下在無窮遠處，如圖 4-1(d))，接著執方輪黑，透過 Threats Space Search 依舊可以找到 Winning Threats Sequence S_2 和與其對應的 R-Zone II for 1a，取得 R-Zone 的方式與上述方法相同，僅差別在於：現在白方只剩下一顆子可以下，所以上述的 2. 跟 5.的活三死三點不用出現在 R-Zone II for 1a 之中。這樣取出的 R-Zone II for 1a 一共有 35 個點，

搭配 1a 那個點，總共可行成白方有 35 種有效的防禦。

上述的 R-Zone II 侷限第二顆子的範圍我們必須依據每一個在 R-Zone I 上面的點都分別作一次找出白方的有效防禦各有哪些，我們可以整理出，白方有效的 moves 一共可能有 1713 組，我們只要證明，在 1713 組白方 moves 當中，黑方都必勝，我們就可以得到 Ren(6,2,3) 黑方必勝的結果。

我們利用黑方的 Threats Space Search 在個別驗證 1713 組白方防守的 moves 之後，黑方依舊可以獲勝(如圖 4-2)。

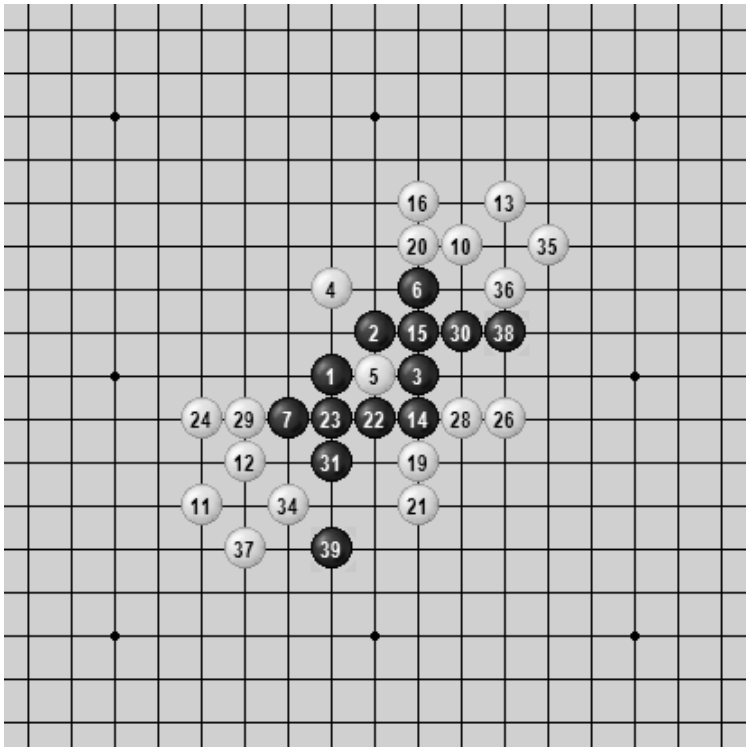


圖 4-2 Connect(6,2,3)白方有效 1713 防守之一，黑仍勝

第五章　結論與未來展望

本篇論文有兩個主要的研究成果：

一、 研究五子棋相關棋類的人工智慧程式設計的作法，提出並改良舊有的人工智慧演算法，並架構實做出五子棋相關棋類的下棋程式。

二、 提出緩著評估(Null-move heuristic)的概念，利用緩著與TSS的結果定義出相關區域(RZone)協助擷取有效的move。

在未來的發展上，有以下幾個方向可以持續進行：

1. 設計一個函式可以在 AI 運算的同時產生 line-pattern information。
2. 針對著手生成器(move generator)做加速
3. 證明出更多的Ren A_k系列遊戲的結果。
4. 改良緩著評估使其可以快速的融入AI的著手生成器當中。
5. 在原有的 Alpha Beta Search 當中架構 Threats-valued search(不足p個Threats的TSS)。

以五子棋相關遊戲作為研究背景，我們嘗試推廣原有的五子棋人工智慧技術應用在五子棋相關的遊戲上面，獲得不錯的結果，並且可以以這些技術直接或間接的了解遊戲的公平性。另外一方面，我們也期望能延伸類似的想法到其餘的棋類遊戲上，對人工智慧的研究上也有相當大的幫助。

參考文獻

[1] Allis, L. V. (1994). Searching for solutions in games and artificial intelligence, Ph.D. Thesis, University of Limburg,Maastricht.

[2] Allis, L. V., Herik, H. J. van den, and Huntjens, M. P. H. (1996). Go-Moku Solved by New Search Techniques. Computational Intelligence, Vol. 12, pp. 7–23.

[3] Allis, L.V., Meulen, M. van der, Herik, H.J. van den(1994). Proof-number search, Artificial Intelligence 66 (1) 91–124

[4] Beal, D.F. (1989). Experiments with the Null Move. *Advances in Computer Chess 5* (ed. D.F. Beal), pp.65-79. Elsevier Science Publishers B.V., Amsterdam, The Netherlands.

[5] Beck, J. (1981) On positional games. J. of Combinatorial Theory Series A 30 (1981), 117-133.

[6] Csirmaz, L., (1980). On a combinatorial game with an application to Go-Moku, Discrete Math. 29, 19-23.

[7] Cazenave, T. (2001a). Iterative Widening. Proceedings of IJCAI-01, Vol. 1, pp. 523–528, Seattle.

[8] Cazenave, T. (2001b). Abstract Proof Search. Computers and Games (eds. T. A. Marsland and I. Frank), Vol. 2063 of Lecture Notes in Computer Science, pp. 39–54, Springer. ISBN 3–540–43080–6.

[9] Cazenave, T. (2003). A Generalized Threats Search Algorithm. Computers and Games, Vol. 2883 of Lecture Notes in Computer Science, pp. 75–87.

[10] Hales, A.W., Jewett, R.I. (1963). Regularity and positional games. Transactions of the American Mathematical Society 106 222-229.

[11] Herik, H. J. van den, Uiterwijk, J.W.H.M., Rijswijck, J.V. (2002). Games solved: Now and in the future. Artificial Intelligence, Vol. 134, pp. 277-311.

[12] Japanese Professional Renju Association, (1903). History of Renju Rules, http://www.renjusha.net/database/oldrule.htm.

[13] Ma,W-J.(2005). Generalized Tic-tac-toe, http://www.klab.caltech.edu/~ma/tictactoe.html.

[14] Pluhar, A. (1994). Generalizations of the game k-in-a-row, Rutcor Res. Rep. 15-94.

[15] Pluhar, A. (2002). The accelerated k-in-a-row game, Theoretical Computer Science. 271 (1-2) 865-875.

[16] Renju International Federation. The rules and the history of Renju and other five-in-a-row games , http://www.renju.nu/r1rulhis.htm.

[17] Renju International Federation (1998). The International Rules of Renju, http://www.renju.nu/rifrules.htm.

[18] Renju International Federation (2003). MOM for the RIF General Assembly, http://www.renju.nu/wc2003/MOM_RIF_030805.htm.

[19] Sakata, G. and Ikawa, W. (1981). *Five-In-A-Row. Renju*. The Ishi Press, Inc., Tokyo, Japan.

[20] Thomsen, T. (2000). Lambda-search in game trees - with application to Go. ICGA Journal, Vol. 23(4), pp. 203–217.

[21] Uiterwijk, J.W.H.M., Herik, H.J. van den. (2000). The advantage of the initiative, Information Sciences 122 (1) 43–58.

[22] Wágner, J., Virág, I. (2001) Solving Renju, ICGA Journal, Vol. 24 (1) 30–34.

[23] Wu, I-C., Huang, D.-Y. (2005) A new family of k-in-a-row games, submitted to ICGA 2005.

[24] Wu, I-C., Huang, D.-Y. (2005) Web pages for Connect6. http://Connect6.csie.nctu.edu.tw.

[25] Wu, I-C., Huang, D.-Y. (2005) Null-move search for *Connect* Games. (in preparation).

[26] Zetters, T.G.L. (1980). Problem S.10 proposed by R.K. Guy and J.L. Selfridge, Amer. Math. Monthly 86 (1979), solution 87(1980) 575-576.

附錄一 AI 的對局

首先是 AI 對 AI 的對局，開局我們以兩種不同的模式開局至第 3 子為止。圖 A-1 AI 黑方獲勝，圖 A-2 為 AI 白方獲勝。

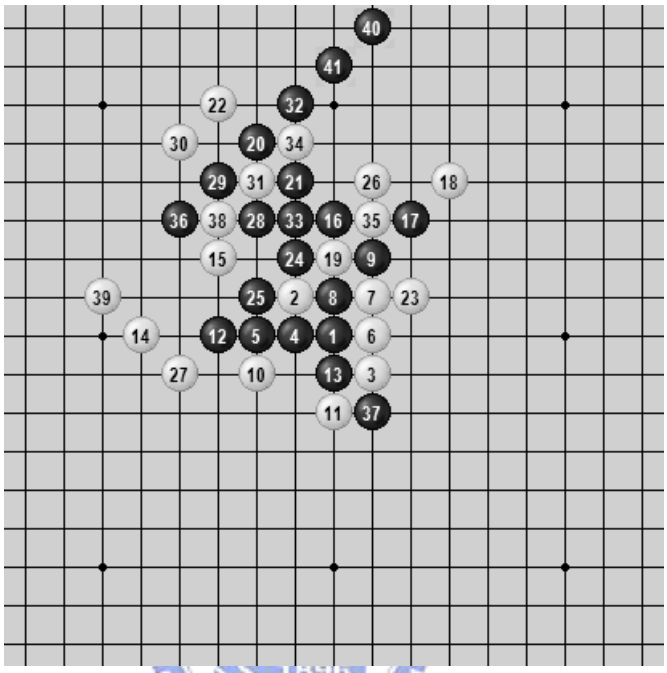


圖 A-1 Connect6 AI(B) vs AI(W) 黑方 AI 獲勝

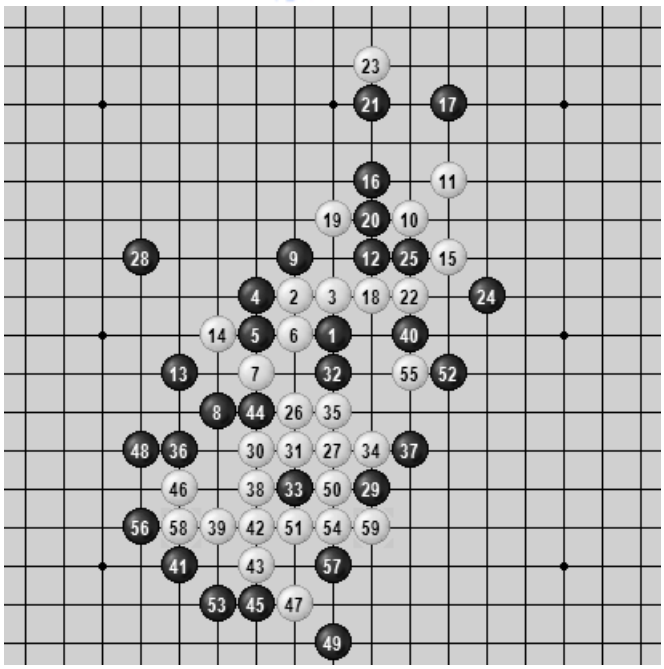


圖 A-2 Connect6 AI(B) vs AI(W) 白方 AI 獲勝

接著由我執黑先下子，AI 執白，開局至第三子依舊為不同模式的開局。圖 A-3 為 AI 白方獲勝，圖 A-4 為 AI 黑方獲勝。

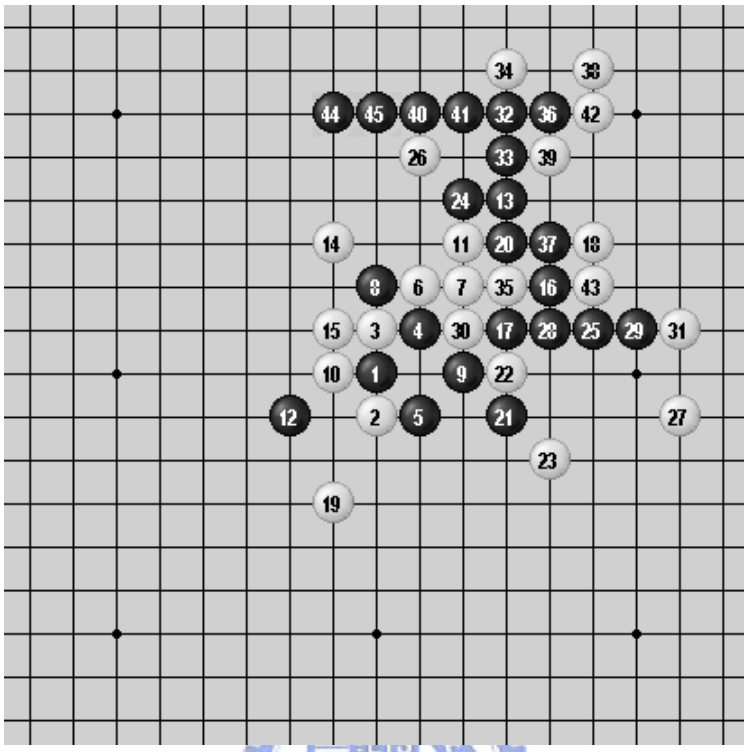


圖 A-3 Connect6 Huang(B) vs AI(W) 黑方獲勝

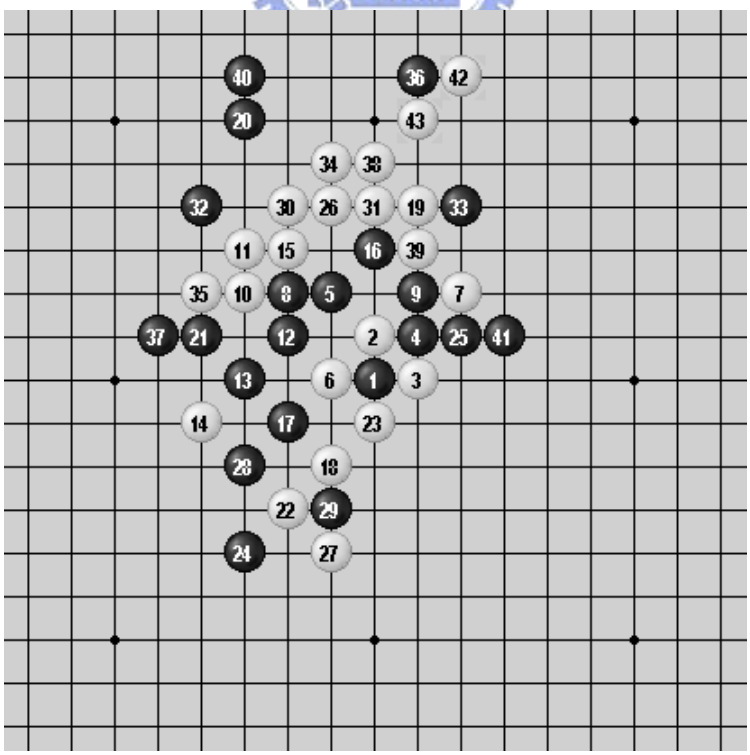
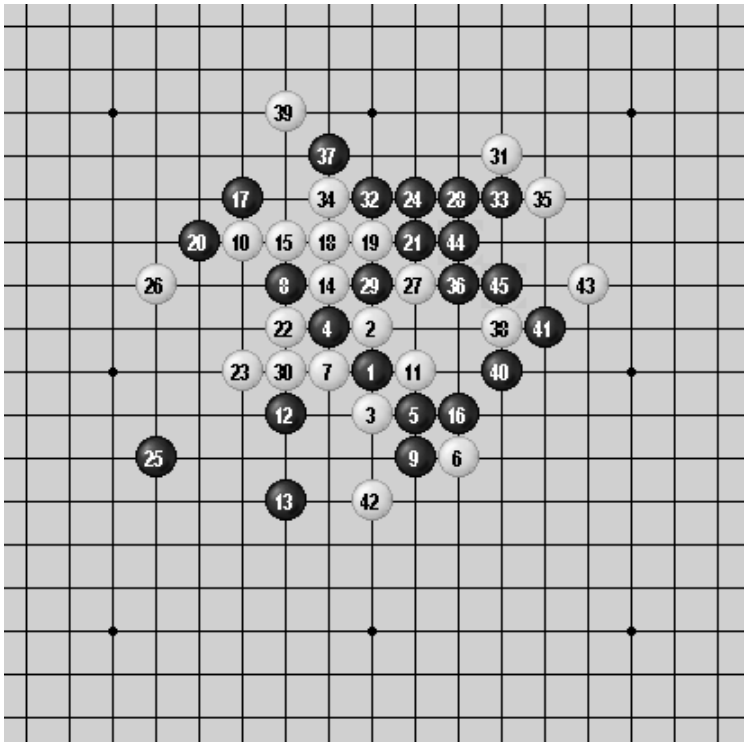
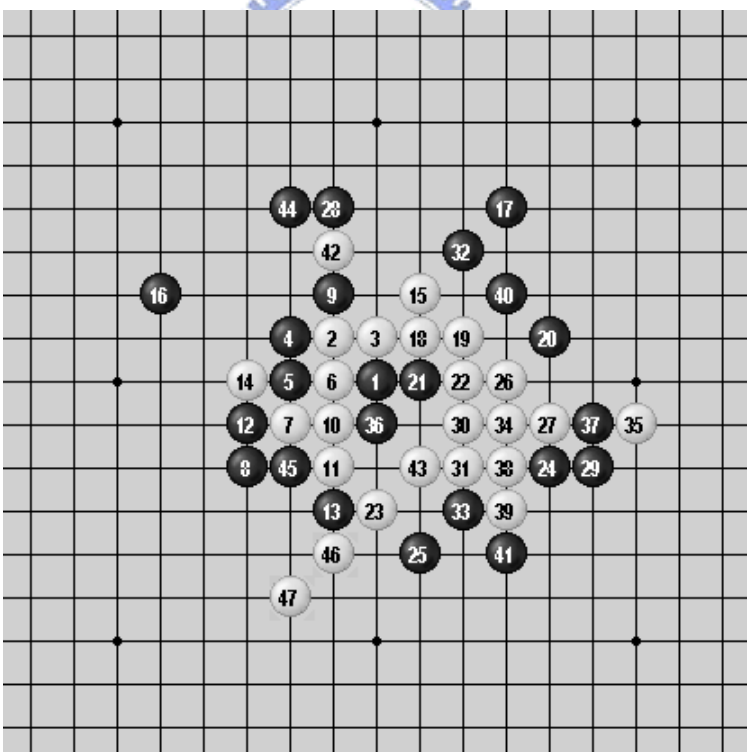


圖 A-4 Connect6 Huang(B) vs AI(W) 白方獲勝





最後由 AI 執黑先下子，我執白，開局至第三子依舊為不同模式的開局。圖 A-5 為 AI 執黑獲勝，圖 A-6 為我執白獲勝。

圖 A-5 Connect6 AI (B) vs Huang(W) 黑方獲勝

圖 A-6 Connect6 AI(B) vs Huang(W) 白方獲勝