國立交通大學

資訊學院　資訊學程

碩士論文

基於 UCT 之九路電腦圍棋程式 HappyGO 的設計與實作

Design and Implementation of

9x9 Computer Go Program HappyGO Base on UCT

研 究 生 ： 王永樂

指導教授 ： 吳毅成　教授

中華民國九十八年二月

基於 UCT 之九路電腦圍棋程式 HappyGO 的設計與實作

Design and Implementation of
9x9 Computer Go Program HappyGO Base on UCT

研 究 生 ： 王永樂　　Student : Yung-Le Wang

指導教授 ： 吳毅成　　Advisor : I-Chen Wu

國 立 交 通 大 學

資訊學院　資訊學程

碩 士 論 文

A Thesis

Submitted to College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Computer Science

February 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年二月

基於 UCT 之九路電腦圍棋程式 HappyGO 的設計與實作

研究生：王永樂　　指導教授：吳毅成

國立交通大學　資訊學院　資訊學程碩士班

摘要

UCT 演算法運用在電腦圍棋程式上只需要基本圍棋知識的特性，提供有別於傳統圍棋程式的另一種程式設計與實作的方法。使用 UCT 演算法實作圍棋程式已成目前最主要的方法之一。在本論文中，將說明如何基於 UCT 演算法實作九路圍棋程式 HappyGO 並加入不同的圍棋策略。

Design and Implementation of 9x9 Computer Go Program HappyGO Base on UCT

Student: Yung-Le Wang Advisor: Dr. I-Chen Wu

Degree Program of Computer Science
National Chiao Tung University

ABSTRACT

Only basic Go knowledge is needed to implement computer Go program based on UCT algorithm. This feature provides another method that is different to traditional Go program to design and implement Go program. Using UCT algorithm is one of major methods to implement Go program. In this paper, we give detailed information about how to implement a Go program, HappyGO, based on the UCT algorithm added with some Go strategies.

目錄

表目錄

圖目錄

第一章 介紹

1.1 圍棋

圍棋起源於中國，是一種圍地的遊戲，分成黑白兩方，由黑方先下，雙方輪流各下一子，圍地多者勝。圍棋為已知棋類遊戲中複雜度最高的遊戲。圍棋的所有可能變化大約是 10^{700}[1]。

1.2 名詞定義與說明

圍棋的棋盤由水平線及垂直線各十九條交叉組合而成，棋盤上共有三百六十一個棋點可以落子，稱為十九路棋盤。除了十九路棋盤之外，還有九路棋盤及十三路棋盤。本論文以九路棋盤為實作對象，因此，在本論文中所提及的圍棋相關事項皆以九路棋盤為對象。在此，先行定義在本論文中會使用到的名詞與用語：

1. 棋點

   如圖 1.1 棋盤上由水平線與垂直線各九條所形成的交接點，共有八十一個棋點。

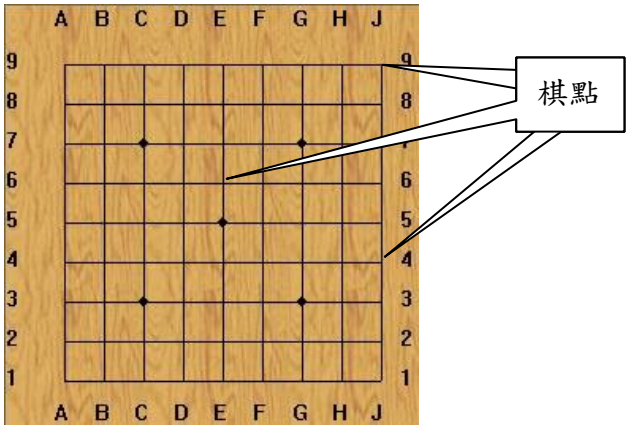
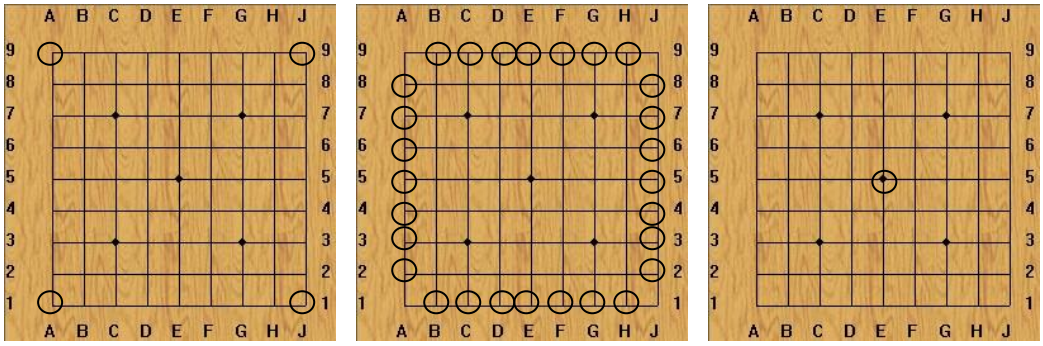


圖 1.1 棋點

2. 角、邊、中央

如圖 1.2(a)四個角落的四個棋點稱為角，圖 1.2(b)四條邊線上的棋點稱為邊，圖 1.2(c)位於棋盤正中央的棋點稱為中央。



(a)　(b)　(c)

圖 1.2 角、邊、中央

3. 棋子

下在棋點上的子稱為棋子，如圖 1.3 中有一顆黑子與一顆白子。

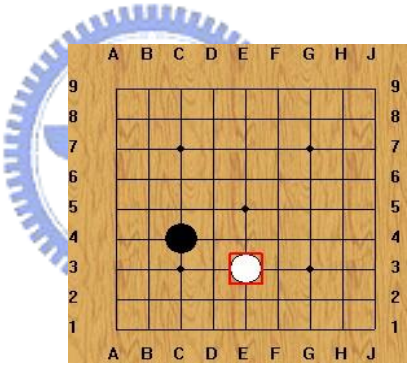


圖 1.3 一顆黑子與一顆白子

4. 空點

無棋子的棋點，如圖 1.4 中有八十一個空點。

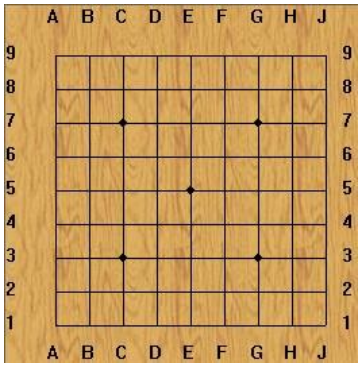


圖 1.4 八十一個空點

5. 線

棋子到邊的最短距離加一，九路棋盤最小為一線，最多為五線。

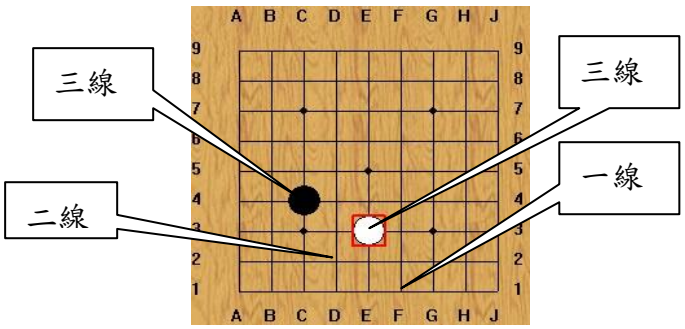


圖 1.5 線

6. 棋串

同色相連的棋子，無相連同色棋子的單一棋子也是棋串。

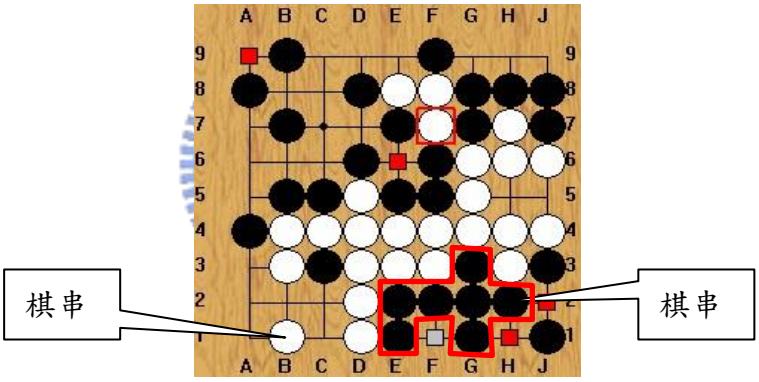


圖 1.6 棋串

7. 氣點

與棋子相連的空點稱為氣點。如圖 1.7 中的 H8 白棋有四個氣點。

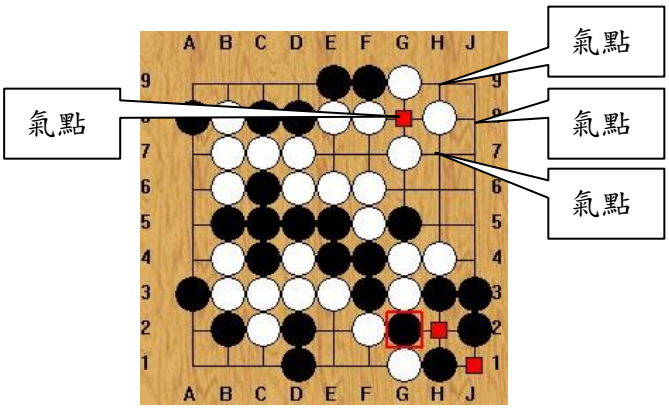


圖 1.7 氣點

8. 氣數

棋串的所有棋子相連之空點總數目。如圖 1.8 中的 B7 黑棋共有四個相連的空點，故其氣數為四。圖 1.8 中的 G1 所屬的黑棋棋串，與之相連的空點有 F1，H1，J2 三個，故其氣數為三。

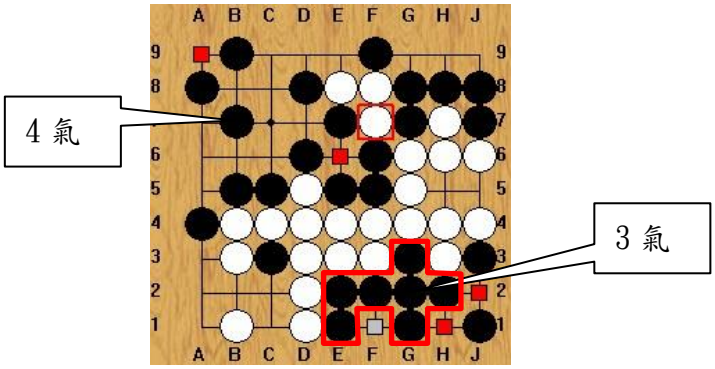


圖 1.8 氣數

9. 已活棋串

氣數不會成為零的棋串，如圖 1.9。而由一個已活棋串所圍成的封閉區域，如圖 1.9 中的 C4，C5，C6，D5 為白棋的已活棋串之封閉區域。為了避免誤判，我們增加一個限制，即封閉區域的棋點總數必須小於八，因為在角落要能成為已活棋串，最小區域為八。

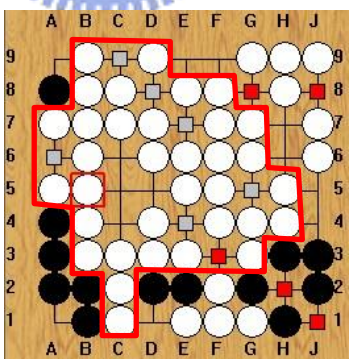


圖 1.9 已活棋串

10. 真眼

本身為空點，相連的棋點上為同色棋子而且屬於同一個棋串，且棋串的氣數大於一。如圖 1.10 中的 A6，C9，D8，E4 … 等空點。

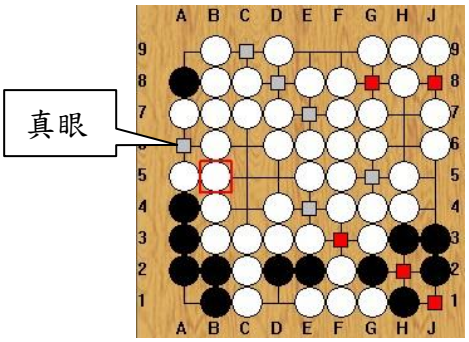


圖 1.10 真眼

11. 假眼

本身為空點，相連的棋點上為同色棋子但是不屬於同一個棋串。如圖 1.11 中的 F3，G8，H1，J1，J8 五個空點皆為假眼，而如圖 1.12(a)，當兩個棋串共同擁有兩個假眼，白棋能否在 D4 落子取決於 E3，而白棋能否在 E3 落子則取決於 D4，形成 D4 與 E3 彼此有依賴關係，讓白棋無法在此兩個假眼落子，兩個棋串共同擁有兩個不會被白棋消滅的氣點，同時成為氣數不會成為零的已活棋串，於是將這兩個棋串予以合併成為一個棋串，並把這兩個假眼設定成真眼。同樣地，如圖 1.12(b)， D5 依賴 E6，E6 依賴 D5 及 F7，F7 依賴 E6，因此，三個假眼成為真眼，四個黑棋棋串合併成一個棋串。但是如圖 1.11 中的 G8 與 J8 兩個假眼，無互相依賴，因此並不會將之設定成真眼。

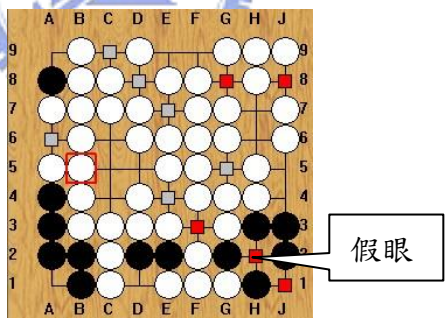
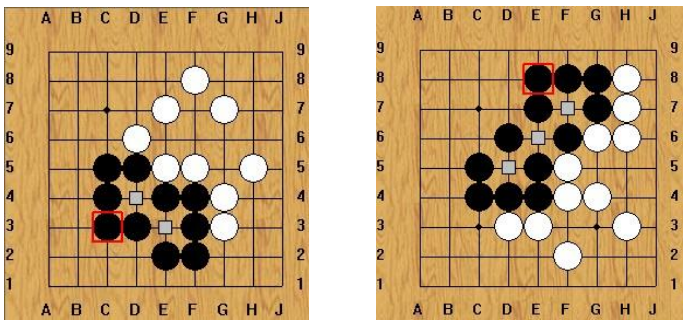


圖 1.11 假眼



(a) (b)

圖 1.12 假眼成真眼

12. 叫吃

讓對方棋串的氣數成為一的狀態。如圖 1.13 中白棋下 B6 讓黑棋棋串(B5,C4，C5，C6，D5，E4，E5，F3，F4)的氣數成為一。

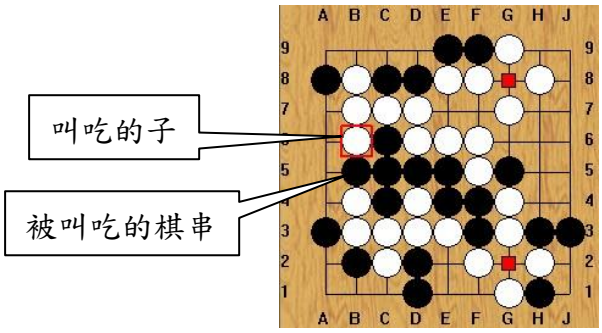
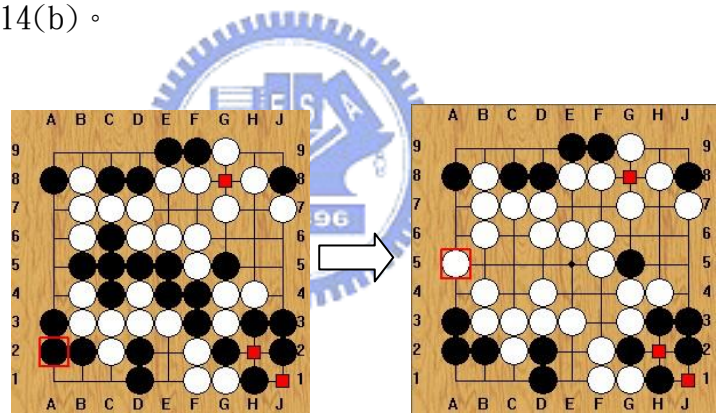


圖 1.13 叫吃

13. 吃子

讓對方棋串的氣數成為零的狀態。如圖 1.14(a)中白棋下 A5 吃掉黑棋棋串而成為圖 1.14(b)。

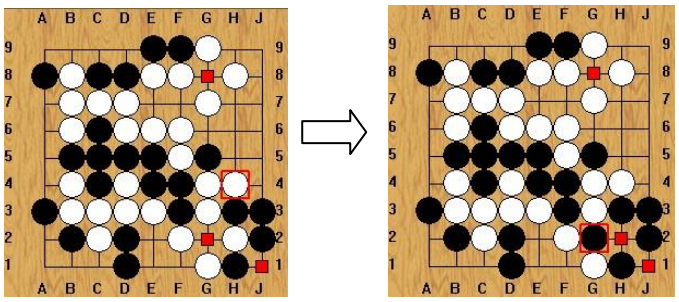


(a)　　(b)

圖 1.14 吃子

14. 劫

只吃一子，而且吃子後本身會只剩一氣而處於被叫吃的狀態。當黑棋由圖 1.15(a)下 G2 吃掉白棋的 H1 而成為圖 1.15(b)，此時白棋不能立刻下 H1 吃掉 G2 黑棋，否則會回到圖 1.15(a)，這樣會讓棋局一直循環而無法結束，因此，圍棋規則規定白棋必須先在別的地方下一子，如果黑棋沒有改變 G2 被叫吃的狀態而下在其它地方，則白棋就能下 H1 吃掉黑棋的 G2。這個過程稱為打劫。



(a) (b)

圖 1.15 劫

15. 馬步飛

與同色棋子距離為三但不在同一條線上,如圖 1.16 中的 C6 與 E5 即為馬步飛，另外，C4，D3，F3… 等棋步也是黑棋馬步飛的候選步。而 E7，E9，F6… 等棋步是白棋的馬步飛候選步。

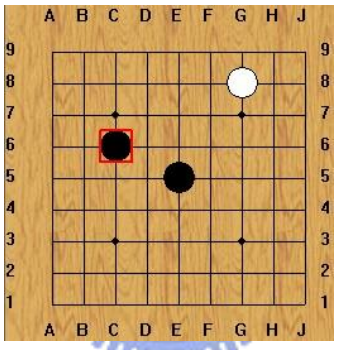


圖 1.16 馬步飛

16. 大馬步飛

與同色棋子距離為四但是不在同一條線上，且形成長方形而不是正方形，如圖 1.17 中的 B4 與 E5 即為大馬步飛，B6，D2，D8… 等棋步也是黑棋大馬步飛的候選步，但是 C3，C7…等與 E5 形成正方形的棋步則不是黑棋大馬步飛的候選步。

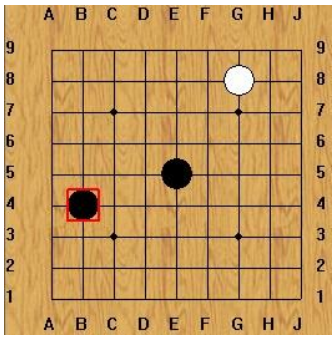
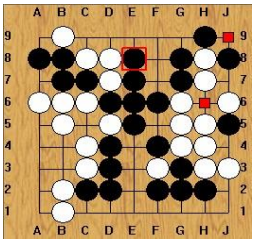


圖 1.17 大馬步飛

17. 棋形

由兩個及以上的棋子在棋盤上所形成的形狀，稱為棋形。上述的馬步飛及大馬步飛即是棋形。圖 1.18(a)中黑棋下 F8，而我們可以把 F8 及周圍八個棋點(D8，D9，E9，F9，F8，F7，E7，D7)設定成一個棋形如圖 1.18(b)以供快速比對尋找，圖 1.18(c)是圖 1.18(b)的示意圖，其中 P 是即將要落子的空點，W 表示白棋，B 表示黑棋，-表示空點。如果圖 1.19 是另一種棋形，則其中的 B 表示只有黑棋將要在此落子，此棋形才符合，即黑棋專用的棋形，而 w 表示是白棋或空點，b 表示是黑棋或空點，x 則表示無論是空點，黑棋，或白棋都可以。邊上的棋形，即一線上的棋形，如圖 1.20(a)及圖 1.20(b)，由於有些棋點並不存在，對此，將不存在的棋點以空點代替，以圖 1.20(c)表示，並輔以標記為一線棋形，只有在要比對的棋點為一線的棋點時才需要比對。大部份的棋形都是可以黑白互換，而且可以做鏡射或旋轉等轉換。



(a)

(b)

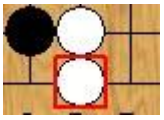
```
- - -
W P -
W B -
```

(c)

圖 1.18 棋形

```
- - -
w B x
W b b
```

圖 1.19 另一種棋形



(a)

(b)

```
B W -
- P -
- - -
```

(c)

圖 1.20　一線的棋形

圍棋的名詞眾多，以上是本論文中會使用到的名詞與其定義，而且這些屬於圍棋的基本知識，使用愈複雜的策略就需要愈豐富的圍棋知識。而相對地，使用適當的複雜策略，對程式棋力的提升也有助益。

1.3 電腦圍棋

Zobrist 於西元 1970 年設計出可與人對弈的電腦圍棋程式[2]，至今已三十多年。圍棋程式的棋力在八十年代以每年大約兩級的速度在進步，九十年代以每年大約一級的速度在進步[3]。西元 2000 年後，圍棋程式的棋力成長似乎進入緩慢近乎停滯的階段。其中主要的一項困難點是良好審局函數之製作[4]。

1.4 近幾年的圍棋程式發展

西元 2006 年，11th 電腦奧林匹克(Computer Olymiad)比賽，圍棋程式 CrazyStone 使用 Monte Carlo 方法於圍棋程式，獲得九路冠軍，十九路第五名。同年，Kocsis 及 Szepesvari 提出 UCT 演算法。

西元 2007 年，12th 電腦奧林匹克(Computer Olymiad)比賽，九路圍棋比賽前三名分別為：Steenvreter， MoGo，以及 CrazyStone。十九路圍棋比賽前三名分別為： MoGo， CrazyStone，以及 GnuGo。其中 MoGo 使用 UCT 演算法。

西元 2008 年，13th 電腦奧林匹克(Computer Olymiad)比賽，九路圍棋比賽前三名分別為：Many Faces of Go， Leela， 以及 MoGo。 十九路圍棋比賽前三名分別為： Many Faces of Go， MoGo， 以及 Leela。其中 Many Faces of Go 使用傳統圍棋人工智慧加上 Monte Carlo 方法，Leela 則使用類似 UCT 的演算法。

在這三年出現了很多基於 UCT 所發展的圍棋程式，如:Leela， Fudo Go， DMC， KK， MC_arc， … 等，而原本一些使用傳統圍棋人工智慧的程式也把 UCT/Monte Carlo 加入成為程式的一部份，如：Jimmy， Aya， Many Faces of Go， … 等。

UCT 演算法如今已是圍棋程式發展的主流。UCT 演算法具有幾項優點：第一、程式設計速度快，只需要基本圍棋知識。第二、可以隨時中斷搜尋並取得搜尋結果。第三、模擬棋局愈多且合理，程式棋力愈強。

1.5 研究目標

研究的目標為以 UCT 演算法為基礎，使用中國圍棋規則，實作出九路圍棋程式 HappyGO，研究不同的圍棋知識及策略，並逐一加入程式中。同時建立一個以 UCT 演算法為基礎的圍棋程式發展與測試平台，提供未來圍棋程式發展時測試的一個選擇。

1.6 論文架構

第一章主要是介紹圍棋及名詞定義與說明，以及電腦圍棋程式近幾年的發展，並概述研究目標。第二章則說明 UCT 演算法及相關的背景知識，包含 Monte Carlo，Bandit Problem，UCB1，與 UCB1-TUNED 等。同時介紹第一個使用 UCT 演算法的電腦圍棋程式 MoGo 及其相關加強方法和輔助演算法。第三章為本論文的主體，詳細說明 HappyGO 的程式架構及實作內容。第四章為實驗結果及分析，使用 GnuGo 為測試對象，測試不同的策略對程式棋力的影響。第五章結論，總結現階段的研究及實作成果，並列出未來可以持續發展的方向。

第二章　背景

2.1 Monte Carlo

Monte Carlo 方法由 Metropolis 及 Ulam 於西元 1949 年提出，原本是為了物理學研究而發展[5]。主要理論基礎為依據大數法則，在隨機取樣的情況下，可以獲得有誤差的評估值，取樣的數量愈大，誤差將愈小，評估值愈準確。

西元 1993 年，Brugmann 將 Monte Carlo 方法運用於圍棋程式中[6]，使用隨機的模擬棋局結果(勝/負，或目數)為[評估值]，經大量的模擬棋局後，具有最佳評估值的著手即為所選擇的棋步。

2.2 Bandit Problem（強盜問題）

強盜問題(bandit problem)是一種機器學習(machine learning)問題，可以使用吃角子老虎機(slot machine)的報酬率來類比說明：

1. 單一目標強盜問題(One-armed bandit problem)
   當只有一台機器時，每次拉下拉桿，都會有報酬，在多次的拉下拉桿後，可以得到平均的報酬，表示如下：

   $$\overline{X} = \frac{1}{S}\sum_{i=1}^{s} X_i \qquad (1)$$

   S 是測試的次數，X_i是第 i 次的報酬。

2. 多目標強盜問題(K-armed bandit problem)
   當有多台機器時，平均報酬可以表示如下：

   $$\overline{X}_{j,s} = \frac{1}{S}\sum_{i=1}^{s} X_{j,i} \quad (2)$$

   J 是機器的編號， $1 \le j \le K$。S 是編號 j 的機器被測試的次數。簡化後表示為： $\overline{X}_j = \overline{X}_{j,T_j(n)}$ ， $T_j(n)$是第 j 台機器被測試的次數。

2.3 UCB1

針對多目標強盜問題(K-armed bandit problem)在有限時間內的分析研究，Peter Auer 等人提出 UCB1[7]配置策略，公式如下：

$$\overline{X}_j + \sqrt{\frac{2 \log n}{T_j(n)}} \qquad (3)$$

$\overline{X}_j$是第 j 台機器的平均報酬，$T_j(n)$ 是第 j 台機器被測試的次數，n 是所有機器目前被測試的總次數。讓公式(3)的值最大的機器將是下一個被選擇來測試的機器。

2.4 UCB1-TUNED

Peter Auer 同時提出另一個實驗結果較佳的 UCB1-TUNED[7]配置策略，UCB1-TUNED 的公式如下：

$$V_j(s) = \left(\frac{1}{s}\sum_{\gamma=1}^{s} X_{j,\gamma}^2\right) - \overline{X}_{j,s}^2 + \sqrt{\frac{2 \log n}{s}} \qquad (4)$$

$$\overline{X}_j + \sqrt{\frac{\log n}{T_j(n)} \min\left\{\frac{1}{4}, V_j\left(T_j(n)\right)\right\}} \qquad (5)$$

讓公式(5)的值最大的機器將是下一個被選擇來測試的機器。

2.5 UCT 演算法

UCT(UCB1 for Tree Search)[8]演算法使用極小極大遊戲樹(minimax tree)搭配節點選擇公式(UCB1， UCB1-TUNED， … 等)，選擇樹節點，展開要測試的節點，然後使用 Monte Carlo 方法執行模擬棋局，模擬棋局的結果即是審局函數的結果，最後將模擬棋局的結果回饋更新。流程示意圖如圖 2.1。

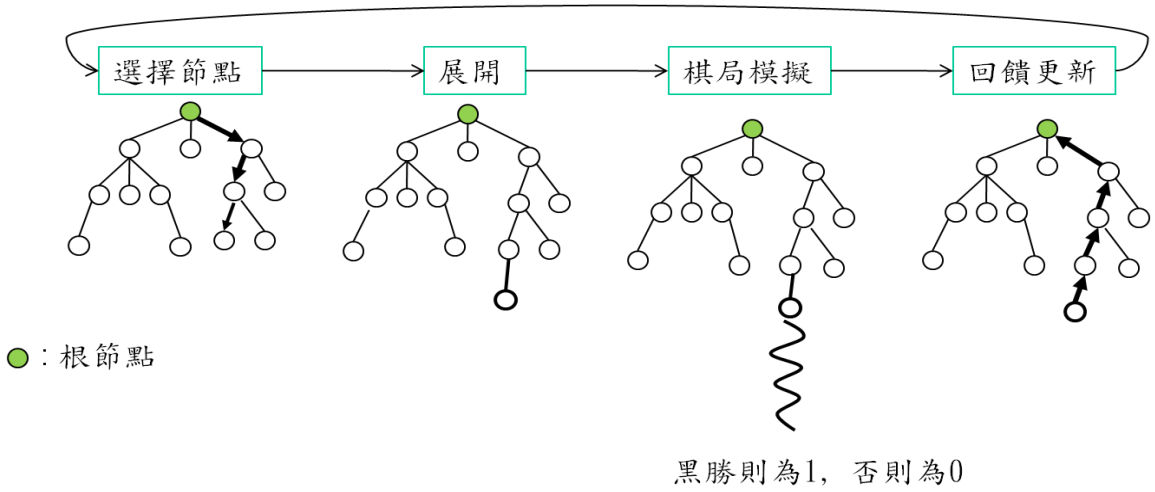


圖 2.1 UCT 演算法

2.6 MoGo

MoGo 為第一個發表使用 UCT 演算法的圍棋程式[9]。12th 電腦奧林匹克圍棋比賽獲得十九路冠軍，九路亞軍，13th 電腦奧林匹克圍棋比賽獲得十九路亞軍，九路季軍。MoGo 是世界第一個曾在持黑被讓子之棋局打敗職業圍棋棋士的電腦程式[10]。2008 年美國圍棋公開賽，MoGo 使用由 800 顆 CPU 所組成的超級電腦，曾在讓九子持黑的棋局打敗韓國八段職業棋士 Kim Myung Wan[10]。2009 年歐洲魔圍棋挑戰台灣職業棋士邀請賽，MoGo 使用由 640 顆 CPU 所組成的超級電腦，曾在讓七子持黑的棋局打敗世界棋王周俊勳[11]。

MoGo 除了使用 UCT 演算法以外，對模擬棋局的實作也有所加強[9]，而且也加入了一些策略，如 RAVE(Rapid Action Value Estimation) [12]。

2.7 MoGo 的模擬棋局加強方法

由於模擬棋局的結果是用來更新目前狀態與決定下一步搜尋與測試的重要依據，所以模擬棋局的合理性與正確性相當重要。MoGo 所使用的模擬棋局加強方法有：

1. 被叫吃棋串處理

當有棋串被叫吃時，嘗試尋找解救的方法：吃掉相連的棋串或者長氣。

2. 對手所下子周圍八點的棋形比對

在十一組棋形中尋找符合的棋形，若有兩個(含)以上的點符合任一棋形，則隨機選擇一個，十一組棋形的示意圖分成如圖 2.2 的非一線的棋形，圖 2.3 中的棋形只適合一線的棋步，最後一種是符合圖 2.4a 但是不能符合圖 2.4b 及不能符合圖 2.4c。

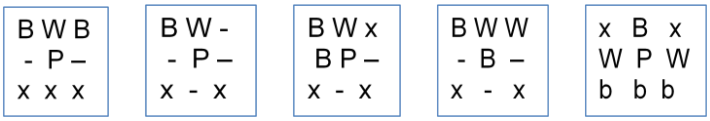


圖 2.2 非一線棋形

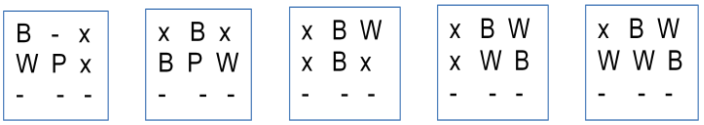


圖 2.3 一線棋形

(a) (b) (c)

圖 2.4 特殊棋形

3. 吃對方棋串

如果盤面上有可以提子的地方，則隨機選擇一個。

當上述條件皆不成立，則隨機選擇一個合法步。而且論文中也提及 MoGo 在實作上比上述的複雜許多，其中包含一些具備圍棋知識的小函式。

2.8 RAVE(Rapid Action Value Estimation)

RAVE 的主要觀念是所有著手如同第一手(all-move-as-first)[12]，對同一方而言，能夠贏棋的著手就是好的著手，在我方獲勝的模擬棋局中，出現愈多次的著手，無論是第幾手，都將之視同第一手，如此就能在少量的模擬棋局中，找

到好的著手。在圖 2.5 中，D4 是模擬棋局的起點，模擬的結果，除了更新到由 D4 到根節點的路徑上所有節點，同時也更新路徑上有 D4 白棋節點及 E4 黑棋節點的節點。

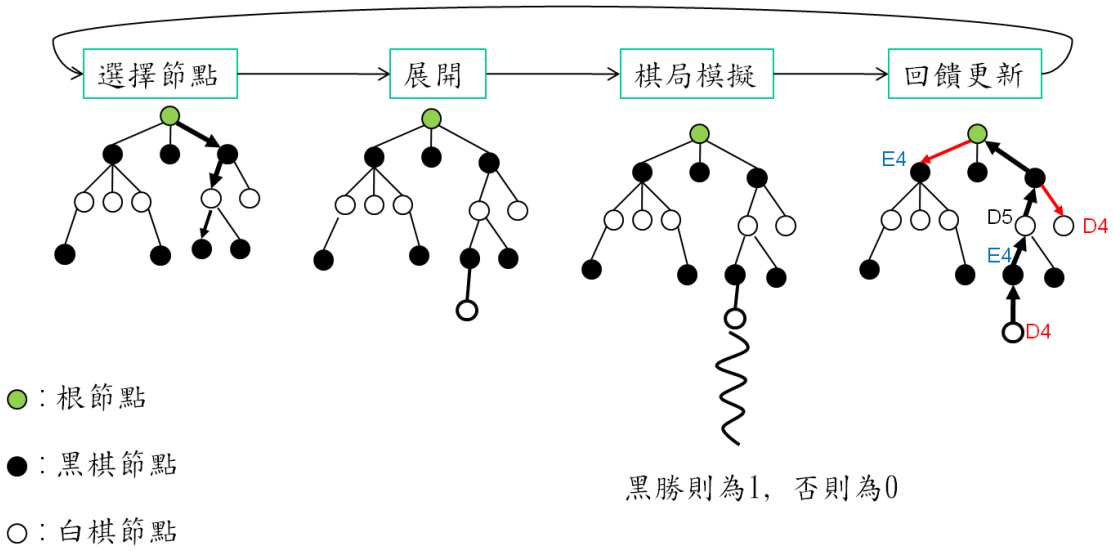


圖 2.5 RAVE

第三章 設計與實作

要從無到有設計並實作出一個可以與人對奕的電腦圍棋程式，需要的事項眾多，雖然有 UCT 演算法可以直接使用，但是仍然需要有可以相互配合的資料結構及程式運作流程。既然是電腦圍棋程式，基本的圍棋元素及知識是必備的。棋局的進行方式與規則是程式流程的核心部份。接下來將分成幾個小節分別說明 HappyGO 的架構設計及實作內容。

3.1 資料結構

在程式中使用的資料結構主要有三種：棋子，棋串，以及棋盤。棋子是最底層的資料結構，棋盤則是最上層，而棋串是程式中檢查與處理的主要物件。分別說明如下：

1. 棋子

   棋盤上的每個棋點，主要有三種狀態：沒有棋子，有黑棋，或者有白棋。因此可以視同三種棋子：空點，黑棋，及白棋。而三種棋子的差別只是顏色。因此，使用相同的資料結構即可。茲將棋子之資料結構的主要內容列出如下：

```
class GoStone {
    int m_color; // 0(空點) 1(黑棋) 2(白棋)
    int m_xIdx; // X 座標
    int m_yIdx; // Y 座標
    bool m_isBlackCandidate; // 是否為黑棋的候選步
    bool m_isWhiteCandidate; // 是否為白棋的後選步
    int m_eyeType; // 0(非眼) 1(真眼) 2(假眼)
    int m_lineNo; // 棋子所在的線
    class GoStone *m_sibling[8]; // 周圍 8 個棋子
    class GoStone *m_nextPtr; // 同一個棋串的下一個棋子
    class GoBlock *m_blockPtr; // 所屬棋串
};
```

2. 棋串

由同色相連的棋子所組成。死活檢查與處理的單位，其主要的資料結構如下：

```
class GoBlock {
    int m_liberty; // 氣數
    int m_nodeNum; // 棋子個數
    int m_eyeNum; // 眼位個數
    int m_color; // 顏色
    int m_closeAreaNum; // 封閉區域個數
    class GoStone *m_startNode; // 棋串的第一個棋子
    class GoStone *m_tailNode; // 棋串的最後一個棋子
};
```

3. 棋盤

棋盤是程式中最高層的物件，其中包含所有棋子及棋串，主要的資料結構內容如下：

```
class GoBoard {
    class GoStone m_board[9][9]; // 棋盤上所有棋子
    class GoBlock m_blockArr[81]; // 棋串陣列
    class GoStone *m_candidateArr[81]; // 候選步陣列
    int m_kmVal; // 貼目
    int m_blockNum; // 目前已使用的棋串個數
    int m_candidateNum; // 目前在候選步陣列中的候選步個數
    int m_curPlayNo; // 目前已下的著手計數
    int m_deadWhiteNum; // 已被提取的白棋個數
    int m_deadBlackNum; // 已被提取的黑棋個數
};
```

3.2 程式流程

基本的流程為 UCT 演算法，UCT 演算法的流程大致上分為四個部份，第一部份是選擇節點，在遊戲樹中選擇子節點。第二部份是展開，產生新的子節點。第三部份是棋局模擬，執行模擬的棋局。第四部份是回饋更新，將模擬棋局的結果以

回溯方式更新遊戲樹節點的資訊。在本論文中，針對前三部份加以修改及加入不同的策略。接下來將分成數節說明流程的各個部份。

3.2.1 流程總覽

UCT 演算法流程是程式中的基本流程，我們將由基本流程加以修改加強而成為程式中使用的流程，因此，先將使用 UCT 的基本流程與修改後的 HappyGO 流程作個概略的列表比較：

使用 UCT 之基本流程如下：

1. 選擇節點
   若有未測試的候選步，則優先以隨機方式選擇其中一個候選步將之展開，否則使用節點選擇公式 UCB1-TUNED 選擇子節點，重複此步驟直到無子節點或有未測試的候選步可以展開。

2. 展開
   隨機選擇候選步展開成子節點。

3. 棋局模擬
   由新展開的子節點開始執行模擬棋局，在模擬棋局中檢查是否有棋串被叫吃，若有則嘗試解救，對方著手的周圍八點，執行棋形比對。如果沒有符合棋形的空點，則嘗試吃對方棋串，若都無合適的棋步，最後使用隨機方式選擇合法棋步。

4. 回饋更新
   將模擬棋局的結果回溯更新遊戲樹節點的資訊。

修改後的 HappyGO 流程如下：

1. 選擇節點
   先檢查是否有[勝率百分之百的子節點]，稱為必勝節點，若有則優先選擇，否則檢查是否有未測試的候選步，若有則優先以隨機方式選擇其中一個候選步將之展開，否則使用選點公式 UCB1-TUNED 選擇子節點，重複步驟直到無子節點或有未測試的候選步可以展開。

2. 展開

如果是第一次展開，則對候選步作篩選，去除不合適的候選步，再對篩選後的候選步隨機選擇其一，展開成子節點。

3. 棋局模擬

由新展開的子節點開始執行模擬棋局，在模擬棋局中檢查我方與對方相連的棋串是否有危險或符合某些情況而需要處理，如果需要，則嘗試處理之，若無法處理，則將對方著手的周圍八點，執行棋形比對。如果沒有符合棋形的空點，則嘗試吃對方棋串，若依然沒有可以提取的對方棋串，則嘗試叫吃對方棋串，若都無合適的棋步，最後才使用隨機方式選擇合法棋步。

4. 回饋更新

將模擬棋局的結果回溯更新遊戲樹節點的資訊。

3.2.2 候選步的產生方式及管理機制

棋局一開始時，雖然棋盤上有八十一個空點，黑棋可以任意選擇其一，也就是說有八十一個候選步。如此一來，遊戲樹的樹分支將會緩慢遞減，但是，並不是每個候選步都是合適的，根據圍棋知識，選擇如圖 3.1 中的二十五個候選步為棋局開始的候選步。

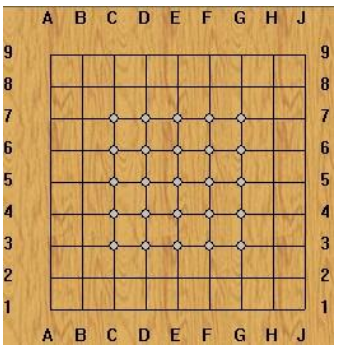


圖 3.1 二十五個候選步

如果一開始只有二十五個空點成為候選步，必須要在適當時機，把其它空點也加入候選步中，否則，有些空點將永遠不會被選擇，而造成錯誤。因此，需要一個候選步的產生方式及管理機制。

由於每下一子(稱為著手)，就會讓周圍附近的空點成為下一步合適的候選步。因此，產生候選步的方式就是每下一子時，根據所下子的位置，檢查附近的空點是否適合成為候選步。而檢查的方式及條件如下：

1. 二線著手

   距離一或距離二的空點，或者是馬步飛的空點，以及在一線上大馬步飛的空點。

2. 三線(及以上)著手

   距離一或距離二但不是一線的空點，或者是馬步飛且不是一線的空點。

圖 3.2 為候選步依著手的增加而變化的情形。

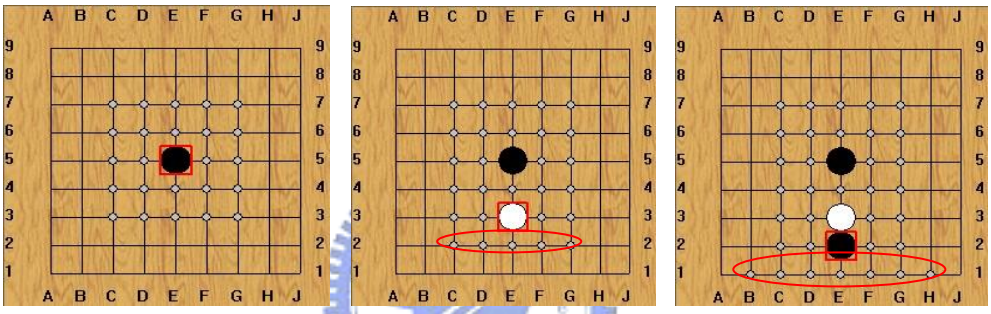


圖 3.2 候選步產生範例

由於圍棋棋盤上的空點數目是固定的，九路棋盤只有八十一個棋點，可以成為候選步的空點最多也只有八十一個，所以使用陣列來儲存與管理是適合的。配合棋點的資料結構，使用兩個欄位，一個黑棋使用，一個白棋使用，紀錄是由黑棋還是白棋導致此空點成為候選步，同時可以用來檢查是否已加入，避免同一個棋點被重複加入。在模擬棋局中隨機選點時更可以避免選到已有棋子之棋點的機率，省掉許多處理與判斷的時間。

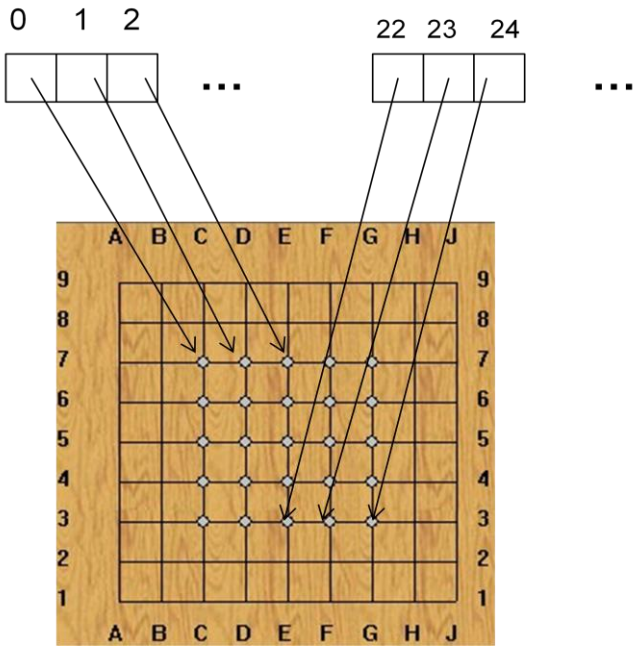


圖 3.3 候選步陣列

候選步的產生與管理是在每次落子時執行，也就是在程式的流程中只要有下子的動作，不論是選擇節點還是棋局模擬，皆會使用候選步的產生與管理。

3.2.3 選擇節點

由於搜尋的目的是為了尋找能獲勝的著手。因此，在選擇節點時，若有勝率百分之百的子節點(稱為必勝節點)時，直接對必勝節點做更深的搜尋以確認其是否是必勝的路徑。此項策略，稱為「贏著優先」(Winning Move First)。

運用「贏著優先」策略，可以加速確認是否已找到必勝的路徑。如圖 3.4，假設節點 J(黑方)第一次測試時得到勝利的結果(黑勝)，則節點 J 為其父節點(假設為 H)的必勝節點，當節點 H(白方)被選擇時，因為有必勝的子節點 J，所以節點 J 會被直接選擇而不會考慮其它未被測試的候選步，此時，節點 J 會嘗試展開子節點。

如果節點 J 的子節點(隨機選擇展開)不會讓黑棋輸(白棋勝)，則節點 J 依然是節點 H 的必勝節點。如此一來，白棋選擇節點 H 的可能性就會降低，黑棋選擇節點 H 的父節點(假設為 K)的機率提高。

相反地,如果節點 J 展開的子節點會讓黑棋輸(白棋勝),則節點 J 的勝率會立刻降為 50%,且節點 J 不再是必勝節點。如此一來,白棋選擇節點 H 的機率提高,而黑棋選擇節點 H 之父節點 K 的機率則降低。

如果沒有必勝節點存在,則依照原本節點選擇方式進行選擇與展開。

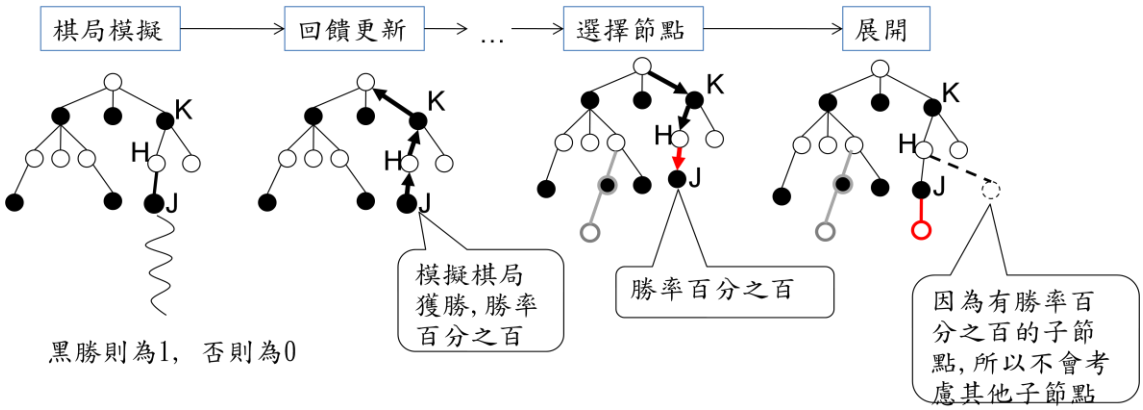


圖 3.4 贏著優先

3.2.4 展開

當節點要展開時,必須在許多候選步中選擇一個,但是並不是每個候選步都是合適的。如 3.2.2 節提到的,九路棋局一開始雖然有八十一個空點可以選擇,但是依據圍棋知識,並非八十一個空點都是適合的。因此,對於候選步有篩選的必要。將不合理或勝率過低的候選步予以去除,篩選的作用可以減少樹的分支,在相同的時間或相同的模擬棋局次數,可以增加搜尋的深度。

但是如果同一個節點每次展開時都要執行篩選的動作,將有不必要的重複動作產生,進而造成時間上的浪費。因此,當一個節點第一次要展開產生子節點時,先對候選步進行篩選,將篩選後的候選步紀錄在節點中,再由紀錄在節點中的候選步選擇一個來展開。而此篩選步驟稱為「節點第一次展開時的候選步篩選」。

棋局開始時,節點只有根節點,盤面上有八十一個空點可供選擇,但是只有二十五個空點為合適的候選步(圖 3.5),此為第一步粗略的篩選。再更進一步,由於對稱的點可以視為同一個,所以,僅七個候選步需要測試。如此一來,樹分支由原本的八十一縮減成七。

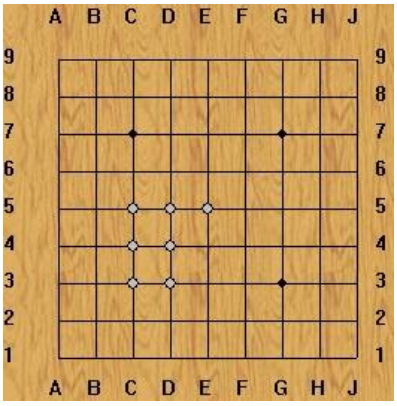


圖 3.5　七個需要測試的候選步

對於任何樹節點，在第一次展開子節點之前，皆執行候選步篩選，去除不合適的候選步，能減少樹分支，增加搜尋深度，進而提高搜尋結果的準確性。但是，篩選的進行需要圍棋知識的輔助，而且，不當的篩選有可能把合適甚至是最佳的候選步予以去除，造成無法挽救的錯誤。因此，對於候選步的篩選必須謹慎地進行，不僅篩選的條件必須嚴謹，更需要經過測試驗證以確認不會造成不必要的副作用。目前 HappyGO 中所使用的候選步篩選僅依據圍棋基本規則，將不合法及不適合的候選步予以去除。

目前歸納共有五種棋步,第一種如圖 3.6 的 A1，A9，… 等棋點對黑棋而言是自殺的著手，由於使用中國圍棋規則，因此自殺著手是禁著的一種。第二種是劫,如圖 3.7 的 H1，由於上一步黑棋下 G2 吃掉 H1，白棋不能下，這稱為打劫立即反提著手，也是禁著的一種，打劫立即反提著手是圍棋規則中避免棋局無法結束的一項規定，如果沒有這項規定,黑白雙方將會因為互相吃掉對方的子而不斷地循環而讓棋局無法結束。第三種是真眼，如圖 3.6 中的 A6，C9，D8，…等為白棋的真眼，是棋串死活重要的部份，白棋沒有理由把自己的真眼填掉，對白棋而言是不合適的著手，對黑棋而言則是禁著。第四種是已活棋串的封閉區域，如圖 3.6 中的 C4,C5,C6,D5 是白棋已活的封閉區域，已活棋串表示此棋串已是不可能被吃掉的，而其所圍住的封閉區域，也是屬於此棋串的領域，在此封閉區域下子，對此棋串並無任何影響，只是浪費一手而已，因此，在已活棋串的封閉區域落子是不合理而無意義的，是不合適的著手。第五種是迴圈著手，如圖 3.8 中(a)至(g)，如果不加以限制，將造成無窮迴圈而無法結束棋局，尤其是在棋局模擬中，將造成程式陷入無窮迴圈甚至造成錯誤而當掉。

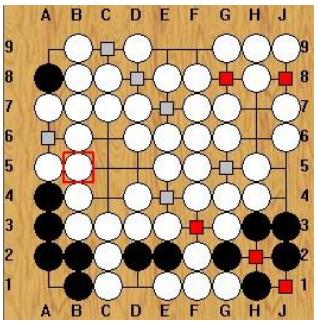


圖 3.6 禁著及已活區域

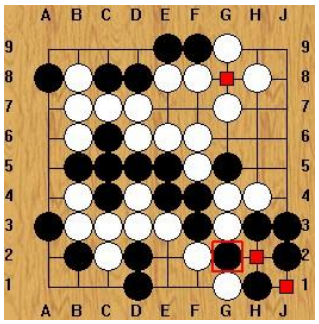
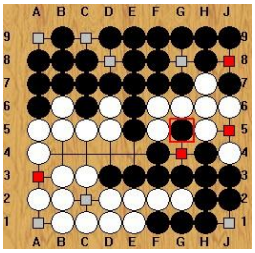
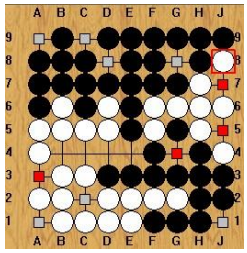


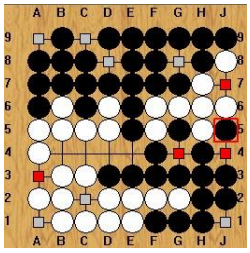
圖 3.7 打劫著手



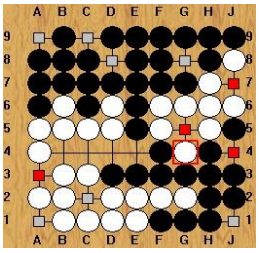
(a)



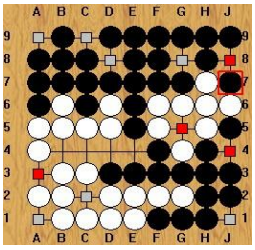
(b)



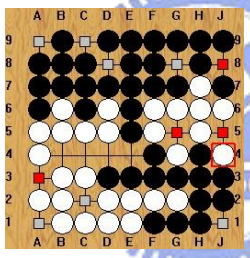
(c)



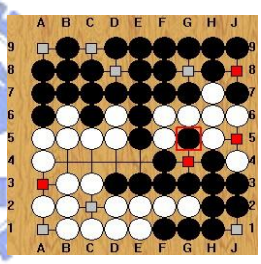
(d)



(e)



(f)



(g)

圖 3.8 循環棋局

以圖 3.9 為例，盤面上有二十四個空點都可以是黑棋的候選步，但是並不是每個候選步都是合法且合適的，使用上述的候選步篩選將發現 C1，C3，E5，F1…等是不合法或不合適的候選步，而剩下 A1，A2，A3，A5，A6…等十六個候選步是合適的。因此，節點在展開時將在篩選後的十六個候選步中選擇一個來展開。

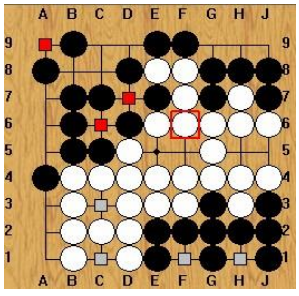


圖 3.9 候選步篩選例子

3.2.5 棋局模擬

模擬棋局結果是決定候選步好壞的一個重要依據，因此,，模擬棋局的合理性與正確性是相當重要的，雖然在大量的隨機取樣下，可以得到誤差小到可以忽略的結果，但是，一盤棋局的比賽時間是有限的，每一個著手更需要在更短的時間內獲得結果，所以，執行合理而正確的模擬棋局是有其必要性的。

模擬棋局的合理性與正確性是重要的，但是，要讓模擬棋局合理與正確，就不能只是單純地隨機選擇棋步，必須對棋步的選擇加以檢查及處理，至少不能是禁著或不合理的，對禁著及不合理棋步的檢查在此稱為「圍棋基本規則處理」。

另外，在一些常見的盤面情況，必須有特定的處理方式或者是人類棋士不會下的著手，這部份的加強就需要圍棋的知識。當愈多的檢查與處理被加入，模擬棋局的合理性與正確性也隨之加強，但是模擬棋局的所需時間也隨之增加，在相同時間內可以執行的模擬棋局的數量也就相對減少。

因此，如何讓模擬棋局足夠合理與正確，同時不會讓一盤模擬棋局的所需時間太長，將是一項挑戰。對此，我們使用的方式是先確認在相同的模擬棋局次數情況下，棋力是否有提升，再確認能在限定時間內完成棋局比賽。

在此,對模擬棋局的加強方法分別說明如下：

3.2.5.1 圍棋基本規則處理

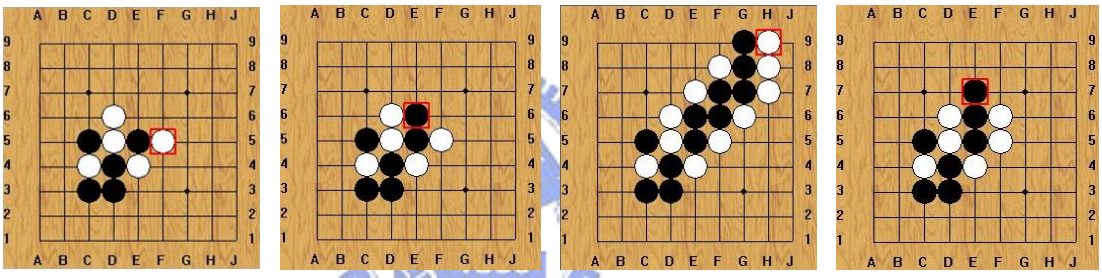
在模擬棋局中，有可能會選擇到非法或不合理的棋步，非法的棋步因為違反圍棋的基本規則，是一定要去除的，但是不合理的棋步，如果不去除，則會造成不合理的模擬棋局。

這部份使用 3.2.4 節中的候選步篩選方式處理，不再贅述，但是對於假眼，則必須確認是否能成為真眼，也就是檢查其斜角四個相鄰的棋點是否為空點，如果是空點，表示有機會直接形成真眼，此時就以斜角的相鄰空點任選其一為棋步，如果無空點，表示此假眼已成為劫，填此假眼只是粘劫，不屬於眼位。如果選擇

到的是對方的假眼，則要檢查是否此棋步會吃掉對方棋串，也就是對方有棋串只剩一氣，且其最後一氣就是此假眼。如果不是，則此對方的假眼是我方的禁著。

3.2.5.2 征子處理

征子是圍棋遊戲中特殊的情況，不僅常見且是必備的圍棋知識，如圖 3.10(a)，如果黑棋嘗試逃脫，會成為圖 3.10(b)，此時，白棋有兩個叫吃棋步可以選擇，但是正確的著手只有一個，最後會形成圖 3.10(c)，而另一個著手 F6 即是錯誤的著手，讓黑棋可以順利逃脫如圖 3.10(d)。如果使用隨機選擇的方式完成征子，因為每一手有二個選擇，假設要完成征子的動作需要 n 個著手，則正確完成的機率為 $(0.5)^n$， $n > 1$。



(a) (b) (c) (d)

圖 3.10 征子處理

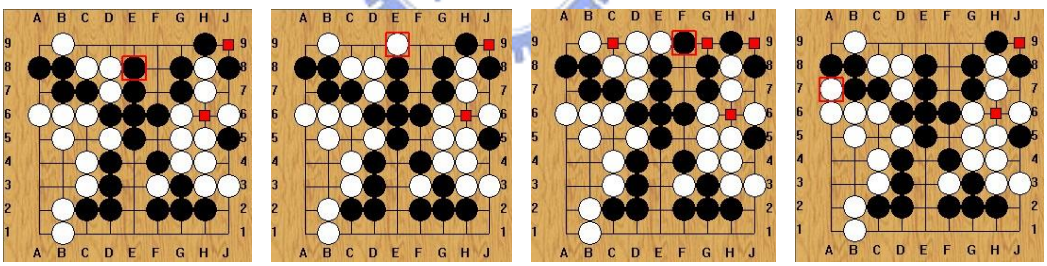
針對此種情形，當白棋發現可以下 F5 而形成征子時，便會執行征子而成為圖 3.10(a)，而黑棋發現嘗試如圖 3.10(b)的 E6 會落入征子的狀況而無法逃脫時，就不會在 E6 落子嘗試逃脫。如此一來，在對方嘗試逃脫征子時，我方就能正確地繼續進行征子，而當我方發現被對方征子時，就不會嘗試直接利用氣點逃脫，進而嘗試在別處落子爭取勝利。

目前只檢查及處理沒有引征的情況，也就是在征子會經過的路徑上沒有任何的被征子的同色棋子，如圖 3.10(a)及 3.10(b)。因為要處理引征的情況，必須檢查的情況多且複雜，例如引征的棋子與征子方棋子的關係為何，是否能讓被叫吃的棋串順利逃脫，而征子的一方能否在中途藉由其它子的輔助改變征子的方向，諸如此類的情況都必須列入檢查。但是，在模擬棋局中執行這方面的檢查，勢必會造成大量的時間耗費而降低模擬棋局的速度。因此，目前 HappyGO 並沒有對引征的情況加以進一步的檢查。

3.2.5.3　二氣棋串處理

當對方的著手造成我方棋串只剩下兩氣時，如果不立即處理，則對方叫吃我方棋串時，我方可能會無法解救。因此，我方必須檢查需要處理，如果需要處理，則確認是否能處理並找出用以處理的棋步。如圖 3.11(a)，左上方有三顆白棋的棋串，如果沒有處理而直接使用棋形比對或隨機方式尋找下一步，則可能會形成圖 3.11(b)，最後成為圖 3.11(c)，結果白棋被吃而輸掉這一盤棋。如果予以處理就會找到正確著手而形成圖 3.11(d)。

針對這種情況，我們設定的檢查是否成立的條件為：對方著手造成我方棋串的氣數只剩二氣，而且我方不處理或虛手(pass)的話，對方就能吃掉我方棋串。也就是對方叫吃時我方無法利用剩餘的氣點獲得更多的氣或吃掉相連的棋串而讓對方無法吃掉我方棋串。一旦符合條件，表示必須處理，否則將會被吃掉，而處理的目標當然是設法讓我方棋串不會被吃掉，因此，處理的方式為：第一，檢查相連的對方棋串是否為二氣，而且我方可以吃掉。第二，檢查相連的對方棋串是否為一氣。上述情況成立的話，則選擇棋子數目較多的棋串叫吃或吃掉， 若上述情況皆不成立的話，則檢查是否可以利用氣點獲得更多氣。



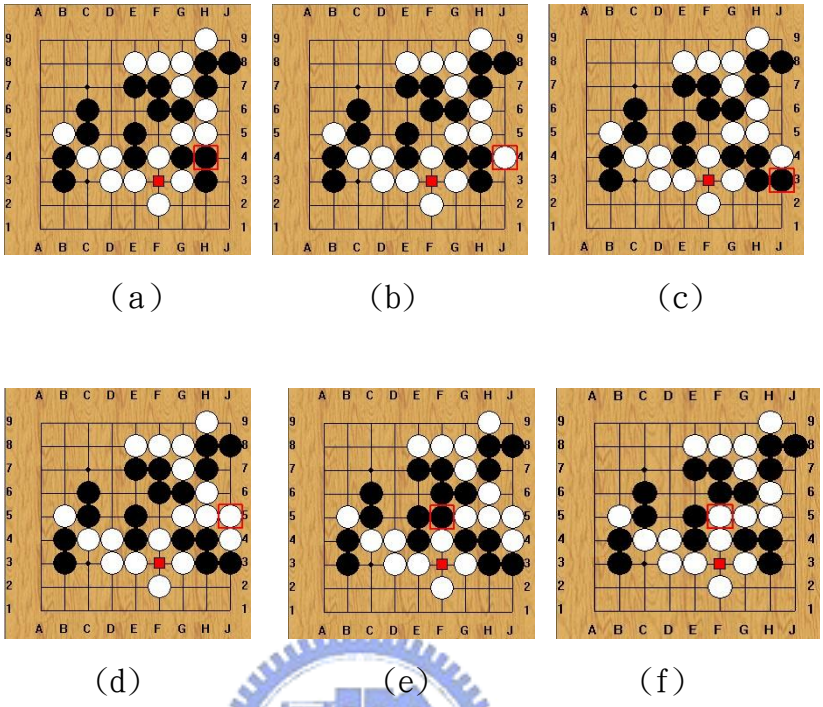
（a）　（b）　（c）　（d）

圖 3.11　二氣棋串處理

3.2.5.4　三氣棋串處理

如同二氣棋串的處理，當對方的著手造成我方棋串只剩三氣時，在某些情況下，如圖 3.12(a)，如果不予以處理而直接使用棋形比對或隨機方式，將會形成圖

3.12(b)至圖 3.12(e)。如果加以處理，則可以找到正確著手而形成圖 3.12(f)。因此，對於三氣棋串有必要予以檢查與處理。



（a）　（b）　（c）

（d）　（e）　（f）

圖 3.12 三氣棋串處理

針對這種情況，我們設定的檢查是否成立的條件為：對方著手造成我方棋串的氣數只剩三氣而且我方棋串的棋子數目大於二，並且沒有其它需要處理而且可以處理的我方二氣棋串。

這裡要特別說明一下，如果我方棋串的棋子數目為一時，剩下三氣如同沒有受到任何威脅，我方棋串的棋子數目為二時，剩下三氣，則因為尚有一半的氣，難以確認其有立即的危險，故而棋子數目大於二才需要處理。而當有其它我方棋串只剩二氣需要處理，並且可以處理時，依危險程度而言，當然先處理二氣棋串。

處理三氣的棋串，以讓我方棋串獲得更多氣或者讓相連的對方棋串之氣數小於我方棋串的氣數為目標，因此，處理方式為：檢查是否有相連的對方棋串之氣數小於三，如果有，則沒有立即的危險，所以不用現在處理。否則，檢查是否有相連的對方棋串之氣數為三，如果沒有，則不處理，因為無法達到目標（讓我方棋串獲得更多氣或者讓相連的對方棋串之氣數小於我方棋串的氣數）。如果有三氣的對方棋串，則檢查雙方共同的氣點是否能達到目標，如果有，則隨機選擇其一，如果沒有，則檢查是否我方氣點能讓我方棋串獲得更多氣。

在我們的程式中目前並不對四氣及更多氣的棋串作檢查與處理，而是由 UCT 的演算法來處理。

3.2.5.5 愚形過濾

在棋局的進行中，兩個或更多棋子會因為相連或距離而形成一些形狀，稱之為棋形，一些能讓未來的發展對自己有利，或能提供強大的防禦/攻擊效果，或能獲取最大利益的棋形，稱為「好形」。相反的，不利未來發展，不能提供防禦/攻擊效果，甚至有損利益的棋形，稱為「愚形」。因此，在棋局進行中，應避免我方愚形的產生。

判別「好形」需要豐富的圍棋知識，且同一個棋形在不同的情況，有時是好形，有時則否，而這也是傳統圍棋人工智慧程式的主要發展重點及挑戰之一。而對於使用 UCT 演算法為基礎的圍棋程式，只需要基本的圍棋知識是其特色之一，因此，定義及使用好的棋形並不是必要的。

但是，一些常見的「愚形」卻是容易定義及判別的，而且在大部份的情況下都是有效的。圖 3.13 為一些常見的愚形，其中左上方的棋形稱為空三角，而且空三角是其它愚形的一部份或形成的必經過程。因此，只需要檢查空三角並去除之就可同時處理多種愚形。在此我們定義要去除的空三角之棋形定義如圖 3.14。

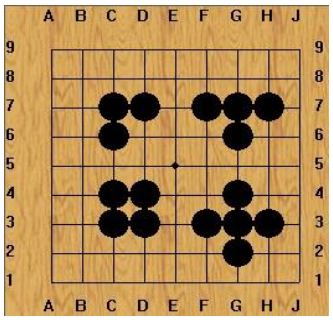


圖 3.13 愚形

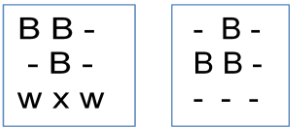


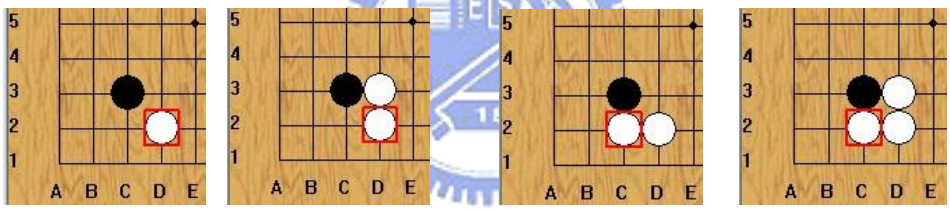
圖 3.14 愚形棋形定義

但是在收官階段，有時空三角也是無可避免的，因此，為了避免在收官階段造成錯誤，設定成只有在前 40 手才作此項檢查與去除。

3.2.5.6　二線特殊棋形比對

在收官階段,二線上的著手經常扮演重要的角色。如圖 3.15(a)及圖 3.15(b)，在這兩種情形下，黑棋如果不擋住白棋，則白棋可以繼續深入侵略而持續獲得利益如圖 3.15(c)及圖 3.15(d),除非在其它地方有更大的利益可以獲取，但是要判斷是否在其它地方是否有更大的利益，是複雜且需要大量時間，而且結果可能是否定的，在需要快速進行的模擬棋局中，這樣的成本是不符合效益。

因此，為了避免損失繼續擴大，利用棋形的快速比對找出此種情況，並加以處理。定義棋形如圖 3.16。使用此項策略，黑棋對於圖 3.15(a)及圖 3.15(b)的情況將會回應如圖 3.17(a)及圖 3.17(b)。



(a)　(b)　(c)　(d)

圖 3.15　二線特殊棋形

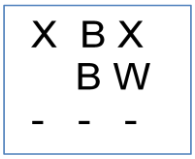
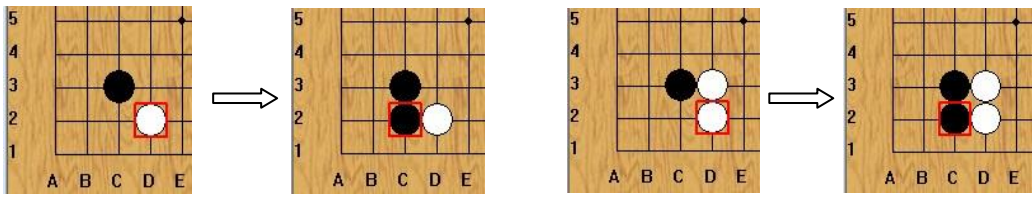


圖 3.16　二線特殊棋形定義



(a)　(b)

圖 3.17 符合二線棋形時的處理方式

3.2.5.7　叫吃對方棋串

如果在前面的檢查中沒有找到任何合適的棋步,則在檢查已叫吃棋串以及使用隨機方式選擇候選步之前，如果對方有只剩二氣的棋串，則檢查是否可以叫吃對方棋串，且對方無法逃脫，如果有找到符合的棋步，則叫吃之。

第四章 實驗分析

我們使用 GnuGo 為測試的對象，使用的版本是 3.7.10，棋力設定成 Level 10。測試的盤數為 HappyGO 的每組設定各 500 盤。

4.1 HappyGO 設定說明

HappyGO 的設定共分成八組，分別說明如下：

1. 基本設定
   此設定中使用候選步產生及管理機制，加上第一次展開時的候選步篩選，將不合法及不適合的棋步予以去除。其餘的七種設定中皆包含此設定。

2. 模擬加強
   此設定中除了基本設定之外，加入 3.2.5 節中所有提及的棋局模擬加強策略。

3. 贏著優先
   此設定中除了基本設定之外，加入 3.2.3 節中提及的贏著優先策略。

4. 贏著優先+模擬加強
   此設定為基本設定加上贏著優先以及模擬加強的策略。

5. RAVE
   此設定是除了基本設定外，將 MoGo 中使用的 RAVE 選擇節點策略加入。

6. RAVE+模擬加強
   此設定是除了基本設定外，加上 RAVE 及模擬加強策略。

7. RAVE+贏著優先
   此設定為基本設定加上 RAVE 及贏著優先策略。

8. RAVE+贏著優先+模擬加強
   此設定為基本設定加上 RAVE，贏著優先以及模擬加強策略。

測試所使用的機器配備為 Intel Core 2 Duo CPU，工作頻率為 3.3GHz，搭配 4GB 的記憶體。表 4.1 為各組設定之執行效率表。

表 4.1 執行效率表

設定	第一手		平均	
	盤/秒	棋步/盤	盤/秒	棋步/盤
基本設定	8630	104	11382	63
模擬加強	7437	101	9506	61
贏著優先	8623	104	10881	67
贏著優先+模擬加強	7449	101	9262	65
RAVE	8629	104	10473	66
RAVE+模擬加強	7446	101	9146	61
RAVE+贏著優先	8625	104	10087	67
RAVE+贏著優先+模擬加強	7425	101	8697	66

由表 4.1 的數據，可獲得幾項資訊：第一，就第一手時的結果而言，有包含「模擬加強」策略的設定，其執行效率降低約 14%±3%，每盤平均少三個棋步。雖然相差不多，但是相信可以持續加強而節省更多棋步。第二，就平均時的結果而言，有包含「模擬加強」策略的設定，其執行效率降低約 14%±3%，每盤少二至五個棋步。第三，RAVE 及贏著優先策略，對執行效率的影響比模擬加強策略小，究其原因乃是因為此二項策略僅是對模擬棋局的結果加以利用運算而已，所以相對地只需要較少的時間。第四，開局時每秒可執行的盤數比平均值要小，每盤所需棋步比平均值要多，顯示在佈局階段必須耗費較多的時間，這項資訊可以提供比賽時各個階段的時間分配的參考。

表 4.2 為測試結果數據。其中基本設定因為只具備圍棋必要的基本知識及少數棋形，模擬棋局中使用隨機選擇方式選擇棋步的機會最多，棋力也是各種設定中最不穩定，震盪幅度最大的一組設定。例如在持白棋時，模擬棋局次數 40K 時的勝率比模擬棋局次數 10K 時的勝率還低。而其它各項策略的加入，目的是要提升程式的棋力，也就是說，基本設定是其它各組設定的比較基準，我們將以基本設定的結果為基準，針對其它七組設定分析各項策略的成效。

表 4.2 測試結果數據

模擬次數	設定	持黑勝率	持白勝率	平均勝率
10K	基本設定	57.20%	48.00%	52.60%
	模擬加強	45.60%	62.40%	54.00%
	贏著優先	48.40%	38.00%	43.20%
	贏著優先+模擬加強	70.80%	47.60%	59.20%
	RAVE	52.20%	42.80%	47.50%
	RAVE+模擬加強	87.20%	57.00%	72.10%
	RAVE+贏著優先	58.40%	20.80%	39.60%
	RAVE+贏著優先+模擬加強	53.40%	74.40%	63.90%
20K	基本設定	85.80%	39.80%	62.80%
	模擬加強	65.80%	53.40%	59.60%
	贏著優先	45.60%	63.20%	54.40%
	贏著優先+模擬加強	64.00%	41.00%	52.50%
	RAVE	65.60%	68.80%	67.20%
	RAVE+模擬加強	50.20%	57.00%	53.60%
	RAVE+贏著優先	72.20%	90.40%	81.30%
	RAVE+贏著優先+模擬加強	78.20%	46.80%	62.50%
40K	基本設定	63.60%	32.80%	48.20%
	模擬加強	59.40%	92.20%	75.80%
	贏著優先	94.20%	78.00%	86.10%
	贏著優先+模擬加強	84.00%	60.00%	72.00%
	RAVE	94.80%	73.60%	84.20%
	RAVE+模擬加強	85.80%	76.60%	81.20%
	RAVE+贏著優先	76.80%	65.00%	70.90%
	RAVE+贏著優先+模擬加強	78.00%	80.80%	79.40%
70K	基本設定	95.80%	66.00%	80.90%
	模擬加強	91.80%	57.60%	74.70%
	贏著優先	87.40%	53.00%	70.20%
	贏著優先+模擬加強	66.20%	92.20%	79.20%
	RAVE	88.20%	84.20%	86.20%
	RAVE+模擬加強	75.60%	88.20%	81.90%
	RAVE+贏著優先	92.60%	81.80%	87.20%
	RAVE+贏著優先+模擬加強	82.40%	93.20%	87.80%

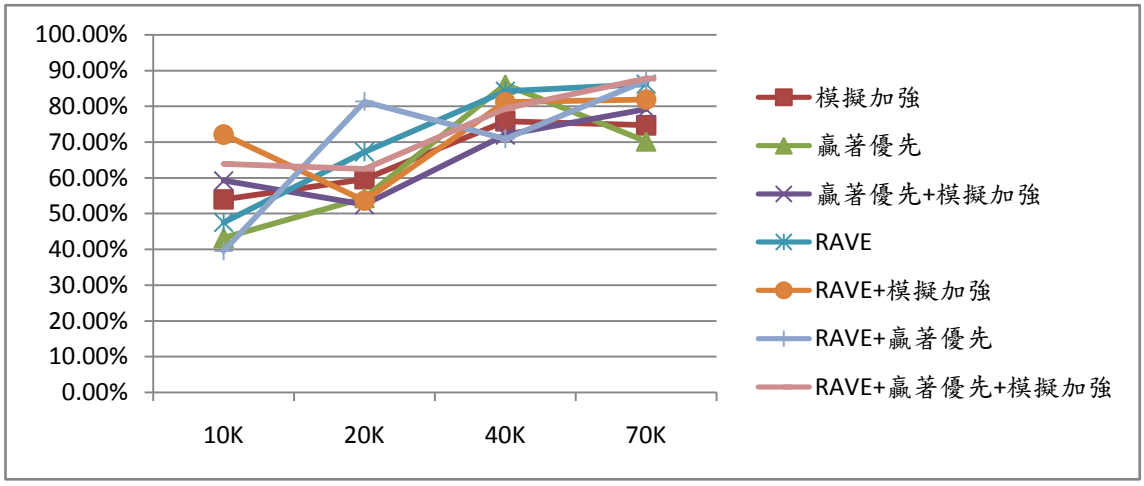


圖 4.1 七種設定的平均勝率

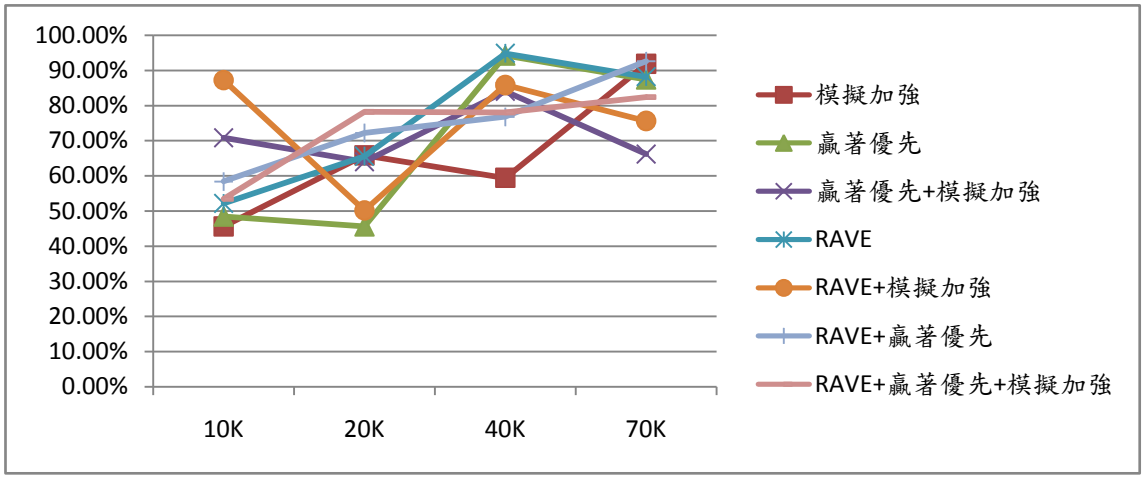


圖 4.2 七種設定的持黑勝率

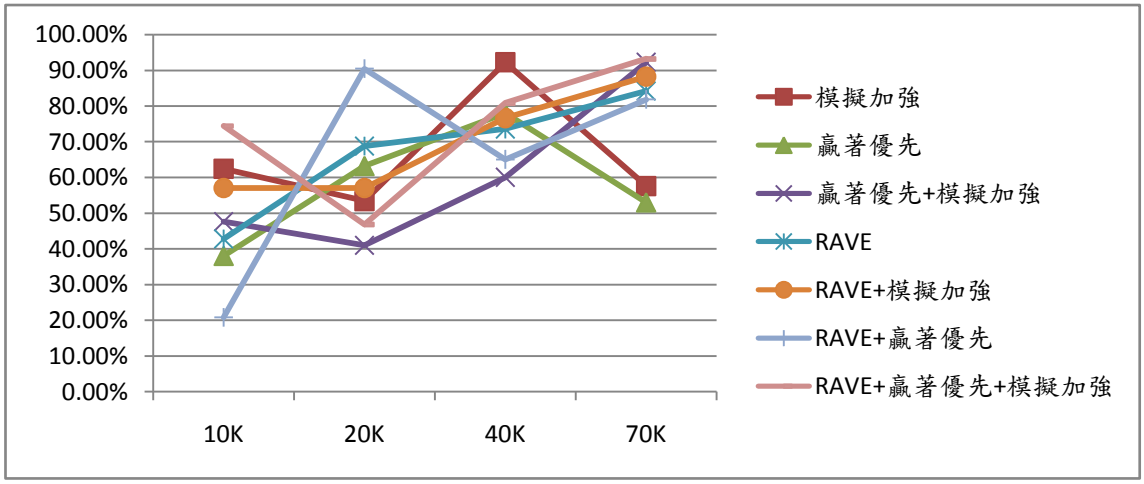


圖 4.3 七種設定的持白勝率

圖 4.1 為七種設定的平均勝率線圖，圖 4.2 為七種設定持黑棋時的勝率折線圖，圖 4.3 為七種設定持白棋時的勝率折線圖。由圖 4.1 的測試結果顯示，棋力呈現不穩定現象，表示尚有許多問題需要解決，是未來努力的方向。就勝率的穩定性而言，由圖 4.2 及圖 4.3 的結果可知，持黑比持白穩定，即持黑時的勝率震盪幅度明顯地比持白時的勝率震盪幅度要小。另外，結果顯示不同的策略會互相影響，但是並不完全是相互輔助的，有時會互相牽制，以圖 4.2 中「贏著優先＋模擬加強」設定為例，在模擬次數 20K 時，勝率小於只有模擬加強設定的勝率，模擬次數 40K 時，勝率小於只有贏著優先設定的勝率。此種現象並不是我們所預期的結果，而找出造成此現象的原因並加以解決，也是我們未來努力的方向。

由圖 4.1 的結果可知，模擬的次數愈多，不同設定間的勝率差距愈小。模擬次數 10K 時，「RAVE+模擬加強」的勝率為 72.1%，「RAVE+贏著優先」的勝率為 39.6%，差距為 32.5%。模擬次數 70K 時，「RAVE+贏著優先+模擬加強」的勝率為 87.8%，「贏著優先」的勝率為 70.2%，差距為 17.6%。差距由 10K 時的 32.5%縮小成 70K 時的 17.6%。圖 4.2 持黑勝率及圖 4.3 持白勝率也顯示相同的現象。持黑時，勝率差距由 10K 時的 41.6%(「RAVE+模擬加強」的 87.2%減「模擬加強」的 45.6%)，縮小至 70K 時的 26.4%(「RAVE+贏著優先」的 92.6%減「贏著優先+模擬加強」的 66.2%)。持白時，勝率差距由 10K 時的 53.6%(「RAVE+贏著優先+模擬加強」的 74.4%減「RAVE+贏著優先」的 20.8%)，縮小至 70K 時的 40.2%(「RAVE+贏著優先+模擬加強」的 93.2%減「RAVE+贏著優先」的 53%)。

由測試結果及上述的分析可知，策略的增加可以提升勝率，但是會有震盪的情形，而震盪的幅度可以經由模擬次數的增加而減小。

第五章 結論

5.1 現階段成果

由實驗結果可知，增加模擬棋局的合理性與正確性，以及增加不同的圍棋策略，確實可以提升程式的棋力。

在選擇節點方面，加入「贏著優先」策略，藉由檢查必勝節點並對必勝節點優先測試的方式加深搜尋。而在第一次展開節點之前，經由候選步的篩選，將不合理的候選步先予以去除，減少樹分支及不必要的棋局模擬。在棋局模擬的部份，加入了許多的圍棋策略，讓棋局模擬更加合理。

假眼的檢查與處理，增加形成真眼的機率，同時增加棋串活定的機會。征子處理、二氣棋串處理、三氣棋串處理以及愚形過濾，則是藉由圍棋知識的運用，讓棋局模擬的進行更加合理，也更接近人類棋士的下法。二線上的特殊棋形比對，則可以加強棋局模擬時收官階段的合理性。最後，嘗試吃掉對方棋串則可以提高我方的攻擊，增加吃子及獲勝的機率。

5.2 未來方向

棋力不穩定現象的原因，是否與隨機選擇有所關連，有待進一步確認。不同策略間的交互影響，能否有效加以控制，使其能相輔相成，發揮加乘的效果，也是未來努力的方向之一。而在候選步的篩選與棋局模擬部份，仍然有著相當大的改善空間，除了利用強大的電腦硬體運算能力增加模擬棋局次數之外，候選步的篩選有待持續加強以減少樹分支及不必要的模擬測試，而圍棋知識的實作與加強也是值得努力的方向。例如盤面的優劣分析、局部的死活判斷等都是人類圍棋棋士在下棋時會使用的方法，將之加入程式中以加強棋力，也是未來可以努力的目標。

參考文獻

1. D.J. H. Brown, Dowsey, S. “The challenge of Go.” New Scientist, Vol.81, pp.303-305,1979
2. Zobrist, A. L. “Feature Extraction and Representation for Pattern Recognition and the Game of Go”, University of Wisconsin, Ph.D. Dissertation, 1970
3. 顏士淨, 許舜欽,「電腦圍棋的發展概況」,Communications of the Institute of Information and Computing Machinery,Vol.1,No.2, pp.23~30, April, 1997
4. Jay Burmeister, Janet Wiles, “An Introduction to the Computer Go Field and Associated Internet Resources”, Technical Report 339, Department of Computer Science, University of Queensland, 1995
5. N.Metropolis, S.Ulam, “The Monte Carlo Method”,Journal of the American Statistical Association, Vol.44, No.247, pp.335-341, 1949
6. B.Bruegmann.” Monte Carlo Go”,1993
7. P.Auer,N.Cesa-Bianchi,P.Fischer,” Finite-time analysis of the multiarmed bandit problem”,Machine Learning, Vol.2, No.47, pp.235-256, 2002
8. L.Kocsis,C.Szepesvari,” Bandit based monte-carlo planning”,15th European Conference on Machine Learning(ECML), pp.282-293, 2006
9. Gelly,S.,Wang,Y.,Munos,R.,Teytaud,O.,” Modification of UCT with patterns in Monte Carlo Go”, Technical Report 6062, INRIA, 2006
10. http://www.lri.fr/~teytaud/mogo.html
11. http://go.nutn.edu.tw/2009/result.htm
12. Gelly,S.,Silver,D.,” Combining online and offline learning in UCT”,17th International Conference on Machine Learning, pp.273-280, 2007