# 國立交通大學

## 資訊科學與工程研究所

## 碩士論文

# 一個蒙地卡羅之電腦圍棋程式之設計

# A Design of Monte-Carlo Computer Go Program

研 究 生：陳俊嶧

指導教授：吳毅成　教授

中華民國 九十九 年九月

# 一個蒙地卡羅之電腦圍棋程式之設計
# A Design of Monte-Carlo Computer Go Program


研 究 生：陳俊嶧　　Student：Chun-Yi Chen

指導教授：吳毅成　　Advisor：I-Chen Wu


國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

September 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年九月

# 一個蒙地卡羅之電腦圍棋程式之設計

研究生：陳俊嶧　　指導教授：吳毅成

國立交通大學 資訊科學與工程研究所

## 摘要

蒙地卡羅的方法應用在電腦圍棋上，使得電腦圍棋棋力有了很大的突破。在本論文中，會以蒙地卡羅方法設計出九路電腦圍棋程式 AI。並說明在好步篩選及壞步過濾的 AI 策略中，對於電腦圍棋程式棋力成長的幫助。

# A Design of Monte-Carlo Computer Go Program

Student: Chun-Yi, Chen Advisor: I-Chen, Wu

Institute of Computer Science and Engineering
National Chiao Tung University

# Abstract

Implementing computer Go program based on Monte-Carlo method has great improvement on the strength of Go Program. In this paper, we implement our 9x9 computer Go program based on Monte-Carlo method. To help improve the strength of Go Program, we describe how to choose good moves and to delete bad moves.

# 誌謝

這篇論文可以順利完成，首先要感謝吳毅成教授的悉心的指導，使我得以一探人工智慧電腦圍棋領域的深奧，並感謝教授時常犧牲睡眠時間與我討論到半夜，指點我正確的方向、實作的方式與做事的態度，讓我獲益良多。而老師對學問的嚴謹更是我學習的典範。

在此研究期間，也非常感謝台灣師範大學博士班的黃士傑，提供了許多電腦圍棋相關的資訊，使得我在論文的方向與想法上得到許多建議與幫助。

再來我要感謝實驗室的林秉宏、林宏軒學長，對於程式部分給了我一些建議，讓我在實作上可以更加順利；接著也感謝 97 級實驗室的同學，陳柏廷、林彥成、鄒忻芸、廖家茵與黃郁雯，在這兩年裡，有輕鬆的閒聊、有歡笑、有趕作業的革命情感，讓我的生活很輕鬆愉悅；還有感謝實驗室的學弟楊景元、左存道，在比賽期間一起努力，使得程式最後得以順利完成，並且獲得佳績；也感謝學弟韓尚餘在研究期間，維護實驗用的機器，使得我的實驗數據可以如期完成。

最後感謝我的家人，在研究的期間，給了我許多的生活上的鼓勵與建議，使得我可以抱持的熱情與穩定的心情在研究上，謝謝你們。

最後，謹以此篇論文，獻給所有在我這兩年研究路上，協助及陪伴我的老師、朋友及家人，謝謝你們。

# 目錄

# 圖表目錄

# 表格目錄

# 第一章、介紹

在這一章節，會有圍棋的介紹，以及電腦圍棋發展狀況，並提出研究動機。在 1.1. 章節之中會有圍棋規則的介紹及名詞的定義，在 1.2. 章節中會介紹電腦圍棋發展概況，在 1.3. 章節會提出研究動機與目的。

## 1.1 圍棋介紹

圍棋是一種雙方圍地的遊戲，分為黑白兩方，持黑者先下，輪流各下一子，直到最後，圍地者多的人勝利。為了平衡先下者有利，會有貼目的制度，因此，黑方得勝必須圍地數量必須多於白棋圍地數目加上貼目的數目。圍棋是現在公認最複雜度最高的對局遊戲，複雜度為 $10^{360}$[13]，以下介紹圍棋相關名詞與定義。

### 1.1.1 棋盤

圍棋主要使用的棋盤大小分別有 19*19、13*13、9*9，本篇論文主要是實作在 9*9 的棋盤上，如圖 1。因此，在本篇論文提及有關圍棋的項目，都會以 9*9 棋盤為對象。

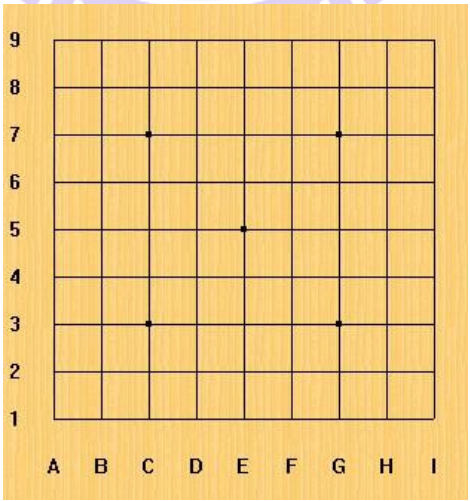



圖 1 圍棋 9*9 棋盤

### 1.1.2 棋點

由棋盤上，水平線與垂直線相交的點，也是黑白兩方可以下棋子的點，如圖2。

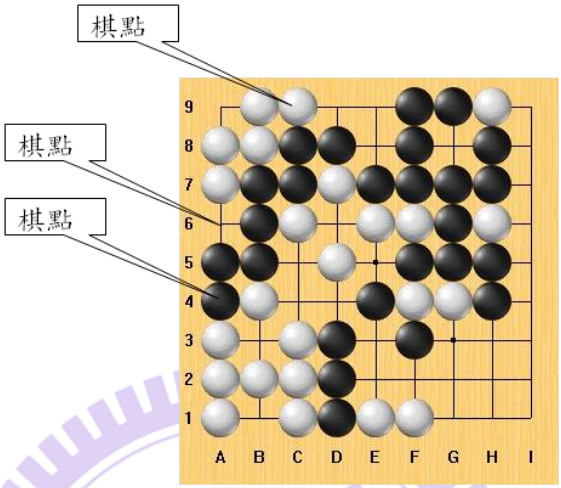
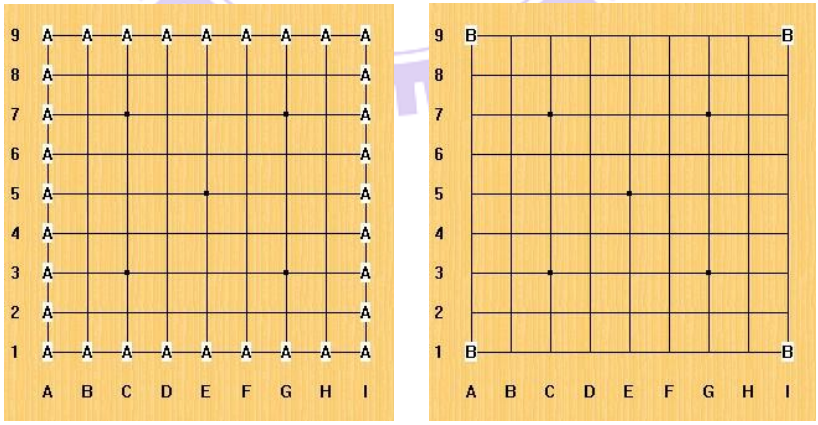



圖 2 棋點

### 1.1.3 邊、角

邊是棋盤最外圍的棋點，共有四個邊，如圖 3 (a)。角是由兩個邊相交的棋點，共有四個角，如圖 3 (b)。




（a） （b）

圖 3 （a）A 棋點皆為邊，（b）B 棋點皆為角

### 1.1.4 線

由棋點到邊的最短距離，由 1 線開始。如圖 4 紅線為 1 線，綠色為 2 線，藍色為 3 線，紫色為 4 線。

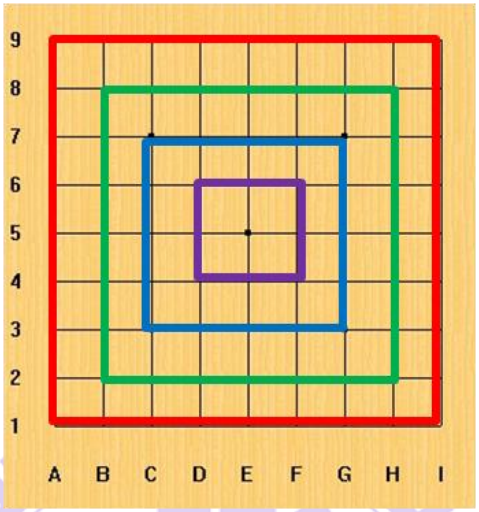



圖 4 線

### 1.1.5 棋子

黑白雙方下在棋盤上的子，分為黑子與白子，如圖 5。

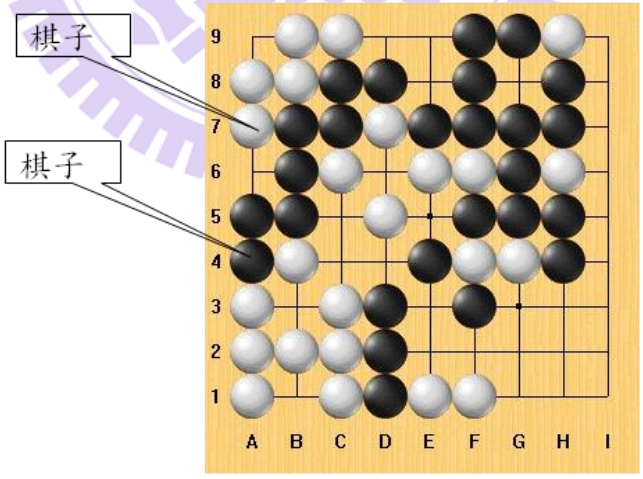



圖 5 棋子

### 1.1.6 空點

棋點上，沒有棋子存在。棋局的初始盤面，每一個棋點的初始狀態，都是空點，如圖 6。

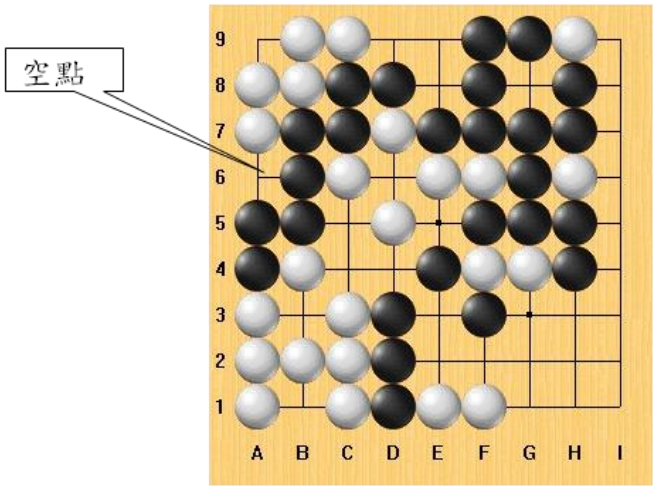



圖 6 空點

### 1.1.7 棋串

由相同顏色相連起來的棋子，一子到多子，如圖 7。

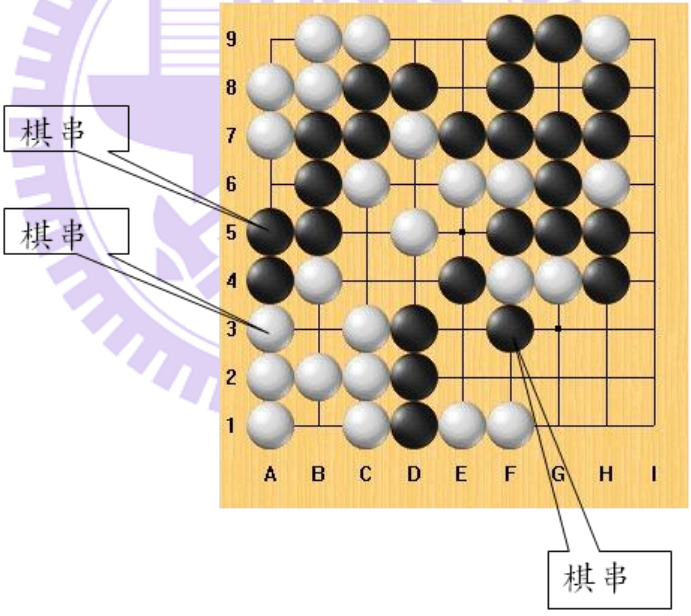



圖 7 棋串

### 1.1.8 氣數

與棋串相連的空點數，如圖 8。

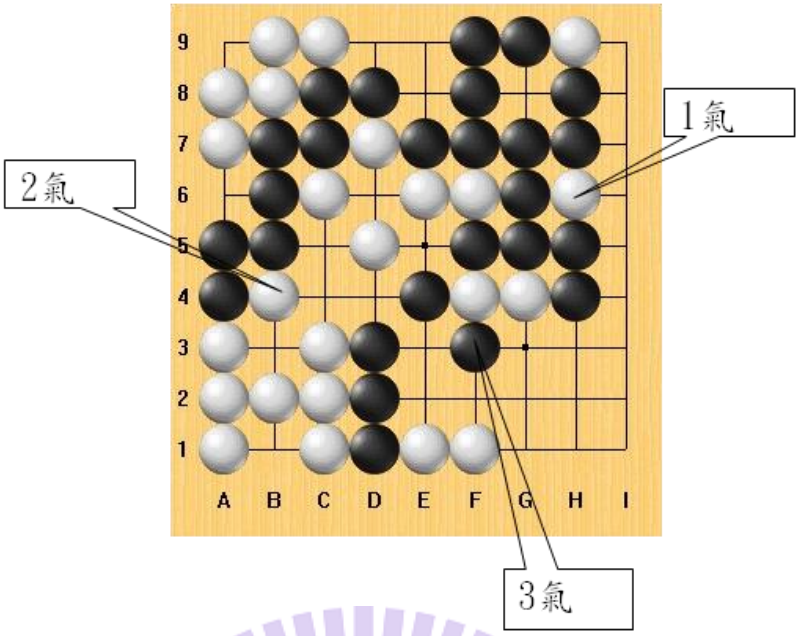



圖 8 氣數

### 1.1.9 真眼

棋點為空點，四個方向都由相同顏色棋子所包圍。棋點在二線以上時，四個斜對角對多只能有一個空點或不同顏色的棋子。棋點在一線時，斜對角不能有空點或不同顏色的棋子，如圖 9。

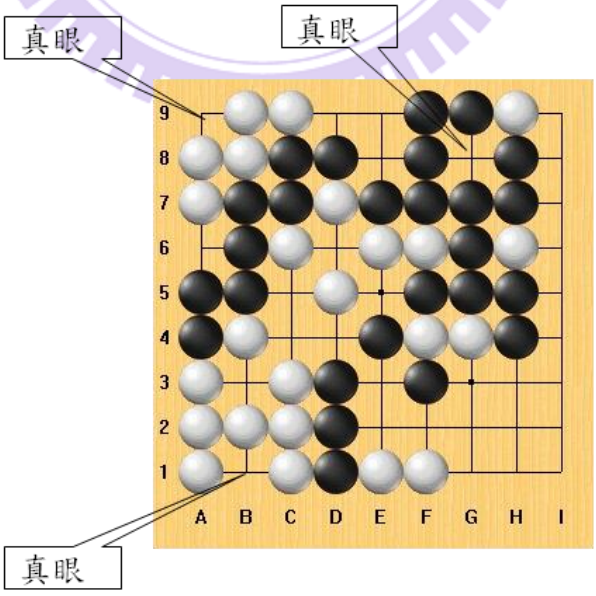



圖 9 真眼

### 1.1.10 假眼

棋點為空點，四個方向都由相同顏色棋子所包圍，且不為真眼，就稱為假眼，如圖 10。

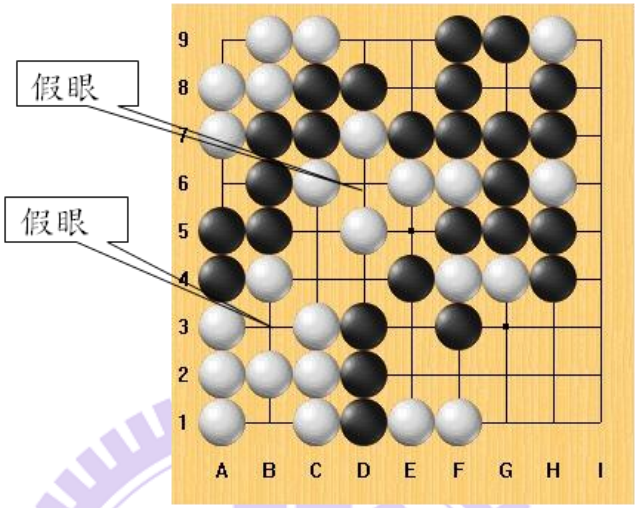



圖 10 假眼

### 1.1.11 活棋

棋串包含了兩個真眼以上，對方無法提掉此棋串，此棋串稱為活棋，如圖 11。

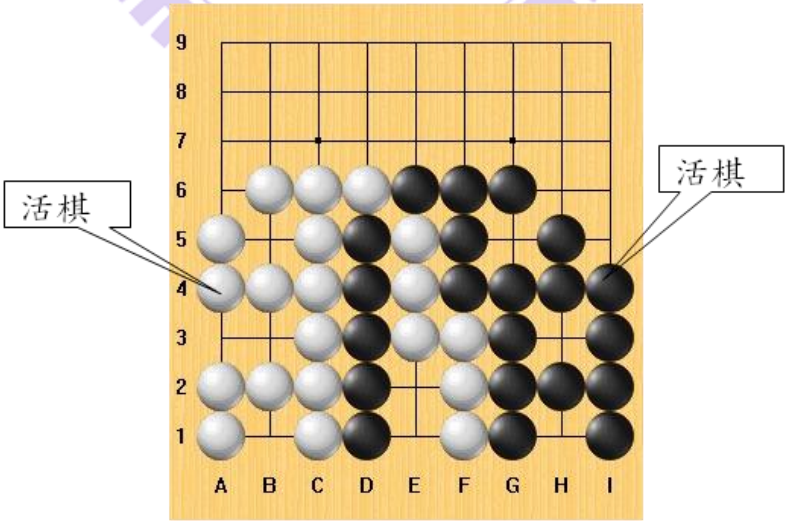



圖 11 活棋

### 1.1.12 雙活

雙方棋串擁有共同的氣點，無論雙方下在此氣點上，都會造成己方的棋串剩下一氣，而造成此棋串可能會馬上被對方提掉，因此在此情況時，可能造成雙方都不下氣點的位置，而造成兩方棋串都可以存活，稱為雙活，如圖 12。

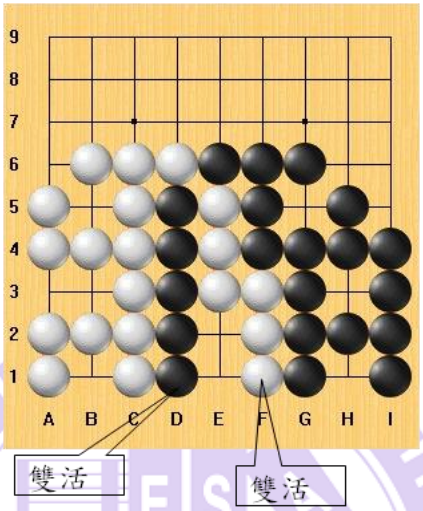



圖 12 雙活

### 1.1.13 提子

棋串的氣數為 0。在圍棋規則裡，氣數為 0 的棋串，會將棋串的棋子從棋盤上移除，如圖 13。

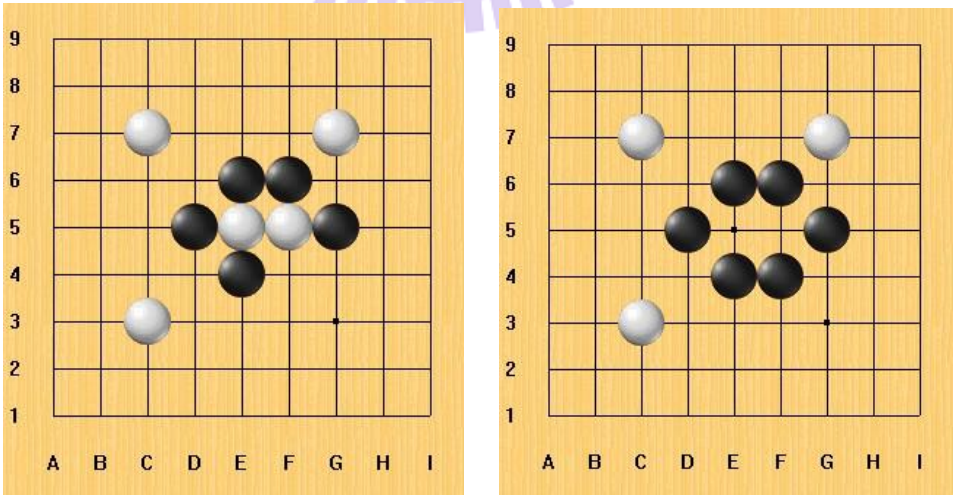



圖 13 提子

### 1.1.14 叫吃

棋串的氣數為 1，如圖 14。

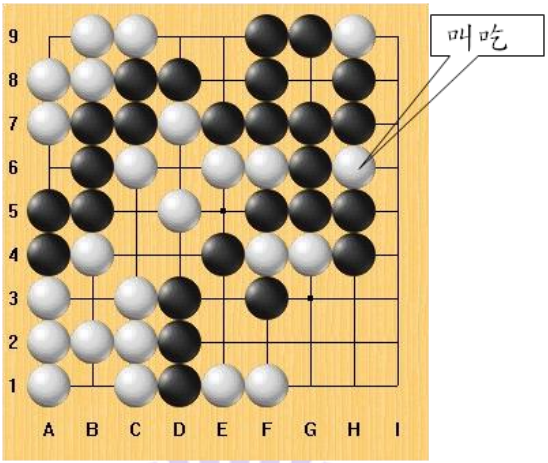



圖 14 叫吃

### 1.1.15 劫

我方下一個棋子之後，將對方一個棋子的棋串提掉，且這棋子也只剩下一氣，則稱為劫。在圍棋規則裡，對方無法馬上提掉我方這棋子，必須等到雙方都進行一回合下棋後，才可以再提掉這方子，如圖 15。

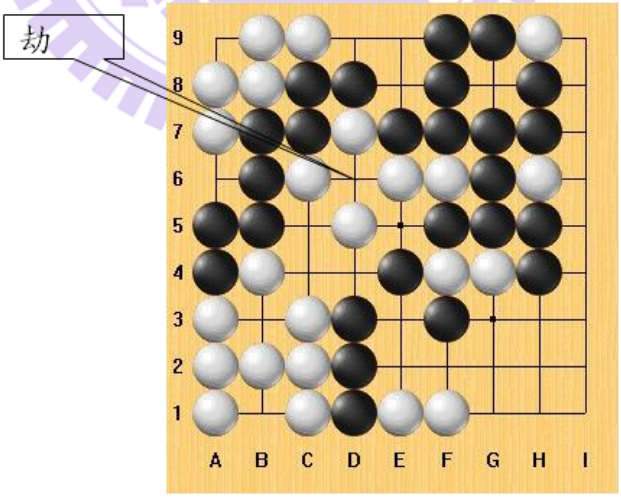



圖 15 劫

### 1.1.16 征子

對方棋串處於被叫吃的狀況，對手試著下在剩下一氣的棋點後，對方棋串會

只有兩氣，我方可以再次叫吃對方棋串，持續叫吃到最後，可以提掉對方的棋串，如圖 16。

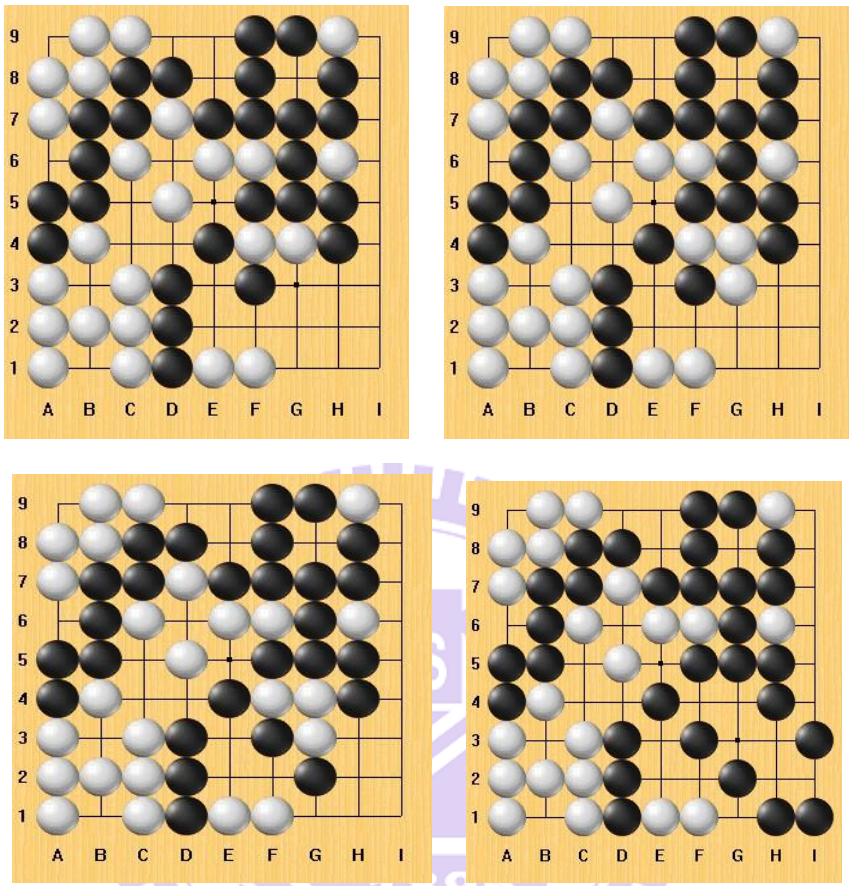



圖 16 征子

## 1.1.17 虎型

棋點為空點，周圍四點有三個為相同顏色的棋子，如圖 17 A 點為虎型。

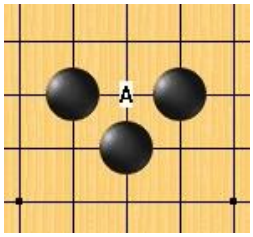



圖 17 虎型

## 1.1.18 撲

我方下一個棋子之後，只剩下一氣，且沒有提掉對方子，如圖 18 A 點為撲。

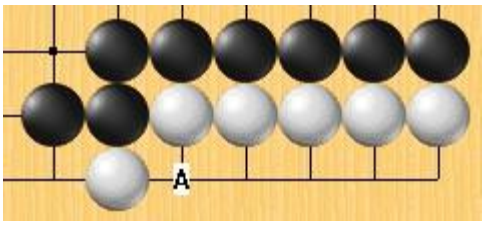



圖 18 撲

# 1.2 電腦圍棋概況

發展於西洋棋的電腦 AI 程式深藍，在 1997 年時，擊敗了西洋棋世界冠軍。表格 1 顯示各遊戲的複雜度[13]，圍棋的複雜度在現有棋類中，是複雜度最高的棋類。因此對於圍棋的電腦 AI 發展，讓許多研究人員更加期待與投入。

| 棋類遊戲 | 複雜度 |
|---|---|
| 圍棋 | $10^{360}$ |
| 日本將棋 | $10^{226}$ |
| 象棋 | $10^{150}$ |
| 西洋棋 | $10^{123}$ |

表格 1 棋類遊戲複雜度

西洋棋與象棋的電腦 AI 的搜尋樹，可以依照一些資訊來進行局面的優劣分析，例如：棋子的位置、棋子的數量等等。因此在樹的展開，可以依照每個局面給予的審局函數，進行挖深深度的搜尋。然而，電腦圍棋要得到一個很好的審局判斷方式，是比較困難的。例如：有些棋型在不同盤面會有不同的優劣，有時邊上的點也會因為攻殺而提伸為好點。

電腦圍棋程式 AI 在 80 年代時，每年大約以兩級的棋力在成長。到 90 年代，每年大約以一級的棋力在成長[1]。到 2000 年之後，棋力的成長，並不像之前年代順利成長了，原因可能是，要一個很好的審局函數是很困難的[5]。

直到 2006 年，11 屆奧林匹克電腦圍棋競賽，在 9*9 的棋盤中，Crazy Stone 獲得了冠軍[9]。Crazy Stone 是採用 Monte-Carlo 加上傳統的圍棋知識所研發出的

電腦圍棋程式，Monte Carlo 也受到矚目。

之後，隨著 UCT 演算法應用在電腦圍棋上[14]，電腦圍棋的棋力有了不錯的成長。在 2007 年，12 屆奧林匹克電腦圍棋競賽，以 UCT 演算法及 Monte-Carlo 製作而成的 Mogo[11]，在 9*9 棋盤中得到亞軍，在 19*19 棋盤中，得到了冠軍[9]。以 UCT 演算法為架構的電腦圍棋程式 Fuego[8]，也在 2009 年 14 屆的奧林匹克電腦圍棋競賽獲得了不錯的成績，分別在 9*9 棋盤得到冠軍，19*19 棋盤得到亞軍[9]。在 2009 年台灣公開賽，Mogo 在 19*19 的棋盤上，與周俊勳(9 段)對奕兩場，周俊勳讓 7 子，結果是 1:1 平手[8]，這消息對於電腦圍棋的發展是一個有利的助力。

## 1.3 研究目的

上一章節介紹了電腦圍棋近期主要是以UCT以及Monte-Carlo為主要架構，在競賽中也獲得了很好的成績。

因此本篇論文的目的是在王永樂、吳毅成實作的電腦圍棋程式 HappyGo 上進行研發。本篇論文採用電腦圍棋程式 HappyGo 的介面，程式 AI 部分，是另外研發。本篇論文主要是發展出一個電腦圍棋程式基於 UCT 以及 Monte-Carlo 為主要架構，研究加入盤面判斷的策略，以及在選點時，可以依照好壞點的判斷，經由剪裁及初始值的設定，達到更好的選點能力，進而提升電腦圍棋程式 HappyGo 的棋力。也在電腦圍棋領域提供了一支以 UCT 為主的電腦程式作為可以測試的平台。

## 1.4 論文架構

在第二章會介紹研究背景，對於蒙地卡羅、UCT、RAVE 及先前 HappyGo

程式做介紹。在第三章會介紹本論文的研究方法，針對好步篩選及壞步過濾的實作做說明。在第四章為實驗結果，對於實做的電腦圍棋程式與 GNUGo、先前 HappyGo、Fuego 對弈。第五章會對本篇論文做結論及未來展望。

# 第二章、研究背景

在這章節中，在 2.1 章節會對於蒙地卡羅做介紹。在 2.2 章節會針對 Bandit Problem 做解釋。在 2.3 章節介紹為了解決 Bandit Problem 而發展出的公式 UCB。在 2.4 章節會介紹 UCT 演算法。在 2.5 章節時，會介紹 Monte-Carlo Search Tree 的流程，在 2.6 章節時，會介紹 RAVE，一個在選點時可以參考的策略。在 2.7 章節時，會針對王永樂、吳毅成研發的 HappyGo 做介紹。在 2.8 章節對於電腦圍棋程式 GNU Go 的概況做介紹。在 2.8 章節對於電腦圍棋程式 Mogo 的概況做介紹。

## 2.1 蒙地卡羅(Monte-Carlo)

蒙地卡羅是一種大數法則的方法，使用隨機取樣，取樣數越多時，其誤差相對會較低。

1993 年時，B. Bruegmann 發表了將蒙地卡羅方法應用在圍棋上[2]，但是當時成績還並沒有很顯著。直到 2006 年，以蒙地卡羅方式發展出的 Crazy Stone 在奧林匹克 9*9 獲得冠軍了冠軍[9]，蒙地卡羅方式應用在圍棋上，也備受期待。

蒙地卡羅應用在圍棋上時，在進行模擬時，會隨機下子，直到盤末，在隨機過程中不填真眼，以確保棋局結束。因此可以依據各點模擬的結果，對於盤面進行優劣判斷，進而從中選出好點。蒙地卡羅方式所研發出的圍棋程式具有幾個優點：1. 使用者可以隨時中斷，立即回傳當時分析出的最好著手。2. 只需要少許的圍棋知識，就可以依據大數法則而選出好的著手。

## 2.2 吃角子老虎問題(Bandit Problem)

吃角子老虎問題是對於眾多的機器中，要如何挑選出最有利益的機器的問題。在這些機器中，可以依據“探勘”(exploitation )和“探索”(exploration)給予每一

台機器不同的權重，最後可以依據此權重，選出最有利益的機器。

“探勘”是依據之前的經驗，給予機器權重。主要是在於深度的搜尋。

“探索”是依據測試的次數，給予機器權重。主要是在於廣度的搜尋。

## 2.3 UCB

UCB(Upper Confidence Bound)是為了解決吃角子老虎問題所發展出來的公式。公式如下：

$$UCB_i = \overline{X}_i + c * \sqrt{\frac{\log_{10} n}{T_i(n)}} \quad \text{--- (1)}$$

代表第 i 個測試機器的 UCB 值，$\overline{X}_i$是平均獲得利益(平均勝率)，c 是常數，n 表示總共測試次數，$T_i(n)$此機器在 n 次測試中，被測試的次數。

表格2針對三台機器做給予測試次數和勝利次數，來算出各機器的UCB值：(c = 0.1)。

| 機器碼 | 1 | 2 | 3 |
|---|---|---|---|
| 測試數 | 10 | 12 | 4 |
| 勝利數 | 5 | 9 | 1 |
| UCB 值 | $\frac{5}{10} + 0.1 * \sqrt{\frac{\log_{10} 26}{10}}$ | $\frac{9}{12} + 0.1 * \sqrt{\frac{\log_{10} 26}{12}}$ | $\frac{1}{4} + 0.1 * \sqrt{\frac{\log_{10} 26}{4}}$ |

表格 2UCB 值計算

## 2.4 UCT

UCT(UCB for Tree Search)是一種不平衡的 MiniMax Search tree，UCT 在選點時，使用 UCB 值作為選點的依據，也就是在選點展開時，會選出 UCB 值最高的點進行深度搜尋。在平均報酬率有某種程度準確且常數 c 值給予恰當的情況下，UCT 會具有一個特性，對於好點的深度相較於壞點會挖的比較深，如圖 19。

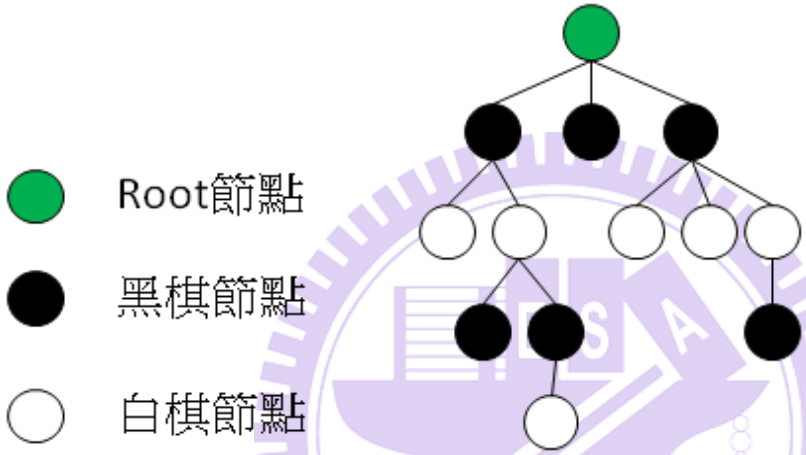



圖 19 UCT

## 2.5 蒙地卡羅樹搜尋(Monte-Carlo Tree Search)

蒙地卡羅樹搜尋，簡稱 MCTS，是在平均獲得利益的地方，採用模擬的結果，來給予平均利益。以圍棋進行 Monte-Carlo tree search 為例，選點時，會採用 UCB 回傳的值，做為選點的依據。當選點葉節點時，會展開一個新的節點。接著進行模擬，直到盤面結束，會給予勝負值。最後進行平均報酬率的更新。至此算是一個輪迴。Monte-Carlo tree search 進行多次的輪迴後，對於好點的節點深度會挖比較深，進而有利最後結束時，可以回傳一個好點。圖 20 是 Monte-Carlo Tree Search 的流程：

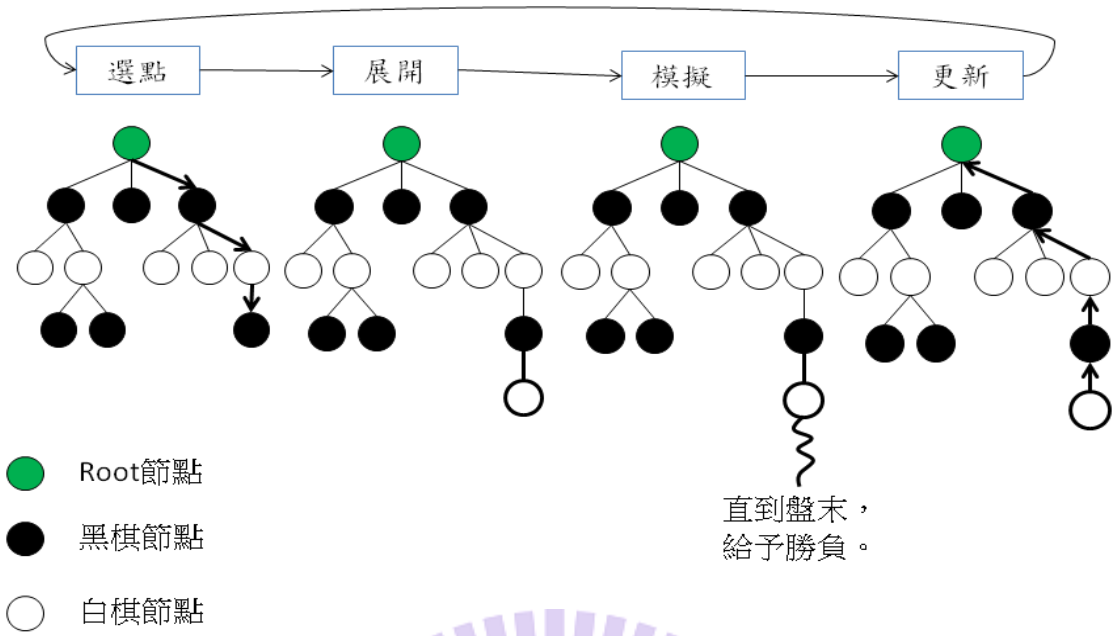
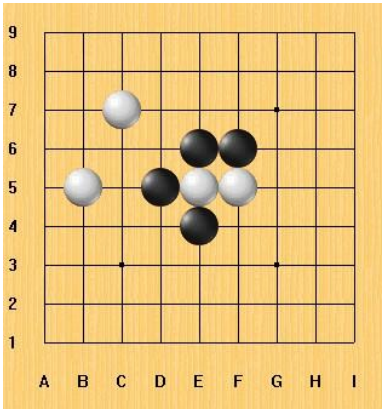



圖 20 Monte-Carlo Tree Search

## 2.6 RAVE

RAVE(Rapid Action Value Estimation)是一種將曾經下過的好步，都當成全局的好步，也就是在下此好步之前，這個好步在之前的盤面也會是好步。是一種 all-move-as-first 的概念[10]。因此在選點的時候，可以依據之前下過的盤面，每一個好步在此盤面會有額外的權重，藉此可以擁有更多的資訊在選點時可以參考。圖 21 舉一個圍棋例子，在第一輪時，黑方曾經下過 G5、C6，且這個盤面最後結果是黑方勝利。那在第一手的時候，對於 C6 這個點，會給予額外的權重，因為 C6 這點可能也會是一個好步選擇。



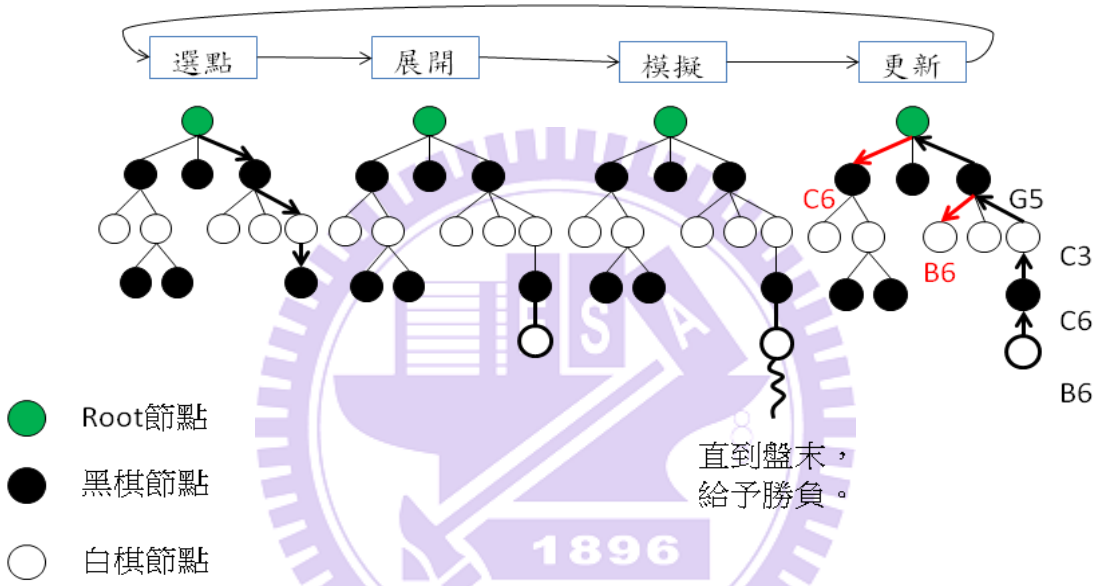




圖 21 上圖為盤面，下圖為 RAVE 更新

# 2.7 原 HappyGo 版本

HappyGo 是由王永樂、吳毅成所研發的電腦圍棋程式，程式語言是由 MFC 加 C++所寫成，在原先 HappyGo 的 AI 部分，是採用蒙地卡羅與 UCT 的架構，加入了贏著優先、二氣三氣棋串處理、愚型過濾、征子判斷及封閉區域的概念[19]。圖 22 為 HappyGo 介面。


圖 22 HappyGo 介面

在模擬次數 10K 時，對抗電腦圍棋程式 GNU Go3.7.10 版本，Level 10 平均勝率有 63.9%，模擬次數到達 70K 時，平均勝率有 87.8%。

# 2.8 GNU Go

是一支開放電腦圍棋程式[12]，並非使用蒙地卡羅方法，因為是公開的程式，且可以調整棋力。因此也成為了許多電腦圍棋程式可以對弈的對象。

# 2.9 Mogo

是一支以 UCT 和 Monte-Carlo 為架構的電腦圍棋程式[11]，在 2007 年奧林匹克電腦圍棋比賽獲得了冠軍[9]，並且在 2009 年與周俊勳在 19*19 的棋盤上，周俊勳讓七子，結果是 1:1 平手[8]。

Mogo 在 UCT 及蒙地卡羅的架構中，加入了 RAVE 概念，並且在模擬時，加入了一些圍棋 AI，例如：上一步被叫吃的棋串，會試著救此棋串。對於好型

的棋型比對。盤面可以提子時，直接提掉對方子。否則就隨機產生一個合法步。

# 第三章、研究方法

在這章節，會依照 Monte-Carlo Tree Search 的各流程中，本篇論文實作的方式。在 3.1 章節中，介紹選點的機制與公式。在 3.2 章節中，依照 AI 策略，在節點展開時，分析好步的選擇與壞步的過濾。在 3.3 章節中，介紹模擬時的 AI 策略。在 3.4 章節中，介紹在更新時的處理。

## 3.1 選點公式

在這章節會對於選點公式做一些分析，3.1.1 說明在不同拜訪次數時的選點構想。3.1.2 介紹選點公式。3.1.3 對於初始值設定。

### 3.1.1 選點分數構想

本篇論文對於選點分數的構想，希望先由初始值來決定權重[6]，接著著重於 RAVE 值，到達一定的拜訪量，著重於 UCB 值。

在一開始，此節點還未拜訪時，本論文會依照一些 AI 的策略，給予每個合法步初始值，因此被設定較高初始值的節點，會被優先展開。

當此節點經歷一些拜訪測試之後，雖然 UCB 值可信度較高，但是在拜訪次數不夠多時，也就是以往經驗不足時，UCB 值還是不可靠的。且由於 RAVE 值是依據每一盤面曾經下過的好點，都會進行 RAVE 值得更新，因此相對於 UCB 值，RAVE 值參考的量會比較多。因此在這階段時，本篇論文在 RAVE 值與 UCB 值的權重調配上，會比較信賴 RAVE 值。

隨著此節點拜訪次數的提高，UCB 值可以參考的量也提高了，本篇論文認為當 UCB 可以參考的量夠多時，UCB 值相對 RAVE 值來說，比較足以信任，因此在權重分配上 UCB 的權重會逐漸提高，並且會超過 RAVE 值的權重。

### 3.1.2 選點公式

本篇論文在選點公式部分，參考了 Computer Go 論壇提出的公式，並且將他實做出來，公式如下：

$$\text{Score} = W_r * R_v + (1 - W_r) * (\overline{X}_j + c * \sqrt{\frac{\log_{10} n}{T_j(n)}}) \quad (1)$$

$W_r$代表權重，越高代表越注重 RAVE 值，越低代表越注重 UCB 值。$R_v$代表 RAVE 值。$\overline{X}_j + c * \sqrt{\frac{\log_{10} n}{T_j(n)}}$為 UCB 公式。

### 3.1.3 初始值設定

初始值設定相當於 AI 的智慧(Knowledge)。在展開時，會根據此論文實作的程式 AI 給予初始值。例如：可以救活我方棋串的棋點、叫吃對方等，都會將初始值設定相較其他棋點還高。相反地，多子自殺著或者將有機會成真眼的假眼填掉，這些初始值相對來講會比較低。

本篇論文在設定初始值的部分，主要是對 RAVE 值做設定，RAVE 值最高為 1，最低為 0。因為本論文所實作的程式，在一開始 RAVE 值權重會比 UCB 來的高，因此好點將 RAVE 值提高，分數就會高於其他的點。相對地，較差的點，RAVE 值設定較低，則分數就會低於其他的點。圖 23 中輪到黑方下子，黑方會設定 A 棋點具有提子及救活的能力，因此具有較高的初始值，B 棋點也可長氣就兩子，也具有較高的初始值。然而，C 棋點填了自己有可能成為真眼的假眼，所以初始值相對較低。D 棋點是無用的自殺步，所以初始值也較低。表格 3 是對於圖 23 上的棋點做初始值設定。

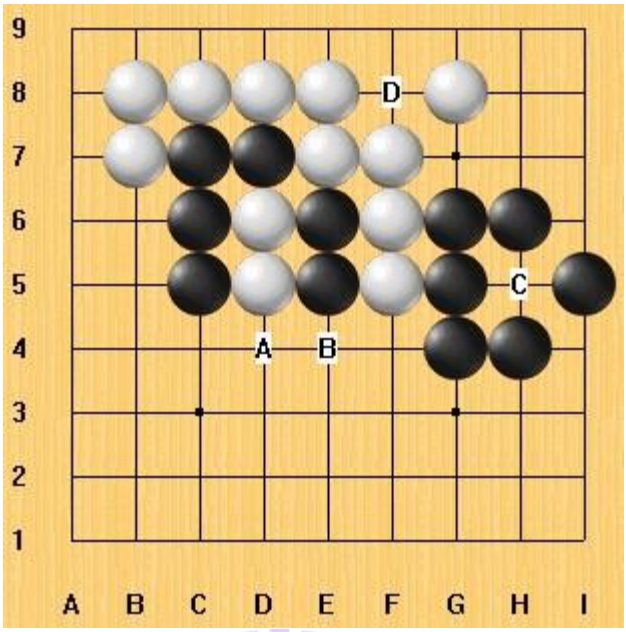



圖 23 初始值設定

| 標記 | RAVE 值 | 描述 |
|---|---|---|
| A | 0.9 | 提子且救兩子 |
| B | 0.7 | 救兩子 |
| C | 0.2 | 自己填假眼 |
| D | 0.2 | 無用自殺著 |

表格 3 初始值設定

# 3.2 展開節點

在本章節對於展開節點會做分析，可以對於好步的篩選而優先展開，並且對於壞步會進行過濾。因為在好步篩選時，會依照好型來判斷好步，所以在 3.2.1 會先介紹本論文採用的鑽石型模組。在 3.2.2 章節中，會介紹封閉區域的概念，封閉區域可以讓本論文實作的電腦程式可以更早得知已活棋串。在 3.2.3 章節中，會介紹本論文實作的電腦程式 AI 好步篩選的機制。在 3.2.4 會介紹壞步的過濾。

## 3.2.1 鑽石型模組(Diamond Pattern)

3.2.1.1 會介紹鑽石型模組。3.2.1.2 介紹鑽石型模組目錄(Diamond Pattern Table)大小和資料結構。3.2.1.3 對於鑽石型模組與 3*3 模組做比較。

### 3.2.1.1 鑽石型模組

鑽石型是由距離中心點相距兩個點的所有點所組成。總共有 12 個點[3]。圖 24 “o”即為此 12 個點。

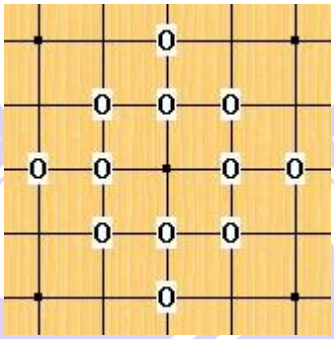
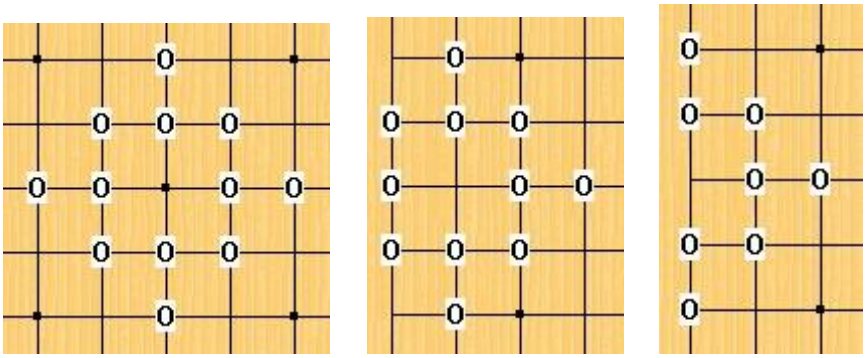


圖 24 鑽石型模組

相較於 3*3 模組，會額外增加四個點可以判斷。

### 3.2.1.2 鑽石型模組目錄

每一個點可能有黑子、白子、空點三種狀況，邊界部分我們拆成各種模組來分析，總共會有六種模組狀況，如圖 25。



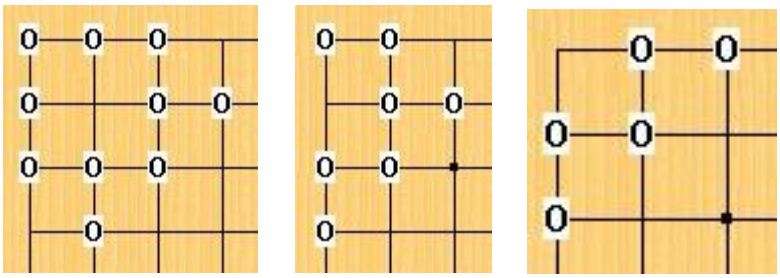


圖 25 鑽石型所有型態模組

大小分別為：(a)$3^{12}$ (b)有四個方向，$3^{11}$＊4(c)有四個方向，$3^{8}$＊4 (d)有四個方向，$3^{10}$＊4 (e)有八個方向，$3^{7}$＊8 (f)有四個方向，$3^{5}$＊4。因此加起來總共是需要 1,520,937 個項目，才能包含所有模組。

鑽石型模組目錄的資料結構：因此資料結構每增加一個 Byte，就會佔用記憶體 1,520,937Byte。模組輸入的方式有人工輸入與程式自動產生。人工輸入只需要輸入一個方向的模組後，就會自動產生八個方向。

人工輸入：

1. 真眼：可以直接比對真眼模組來判斷此棋點是否為真眼，如圖 26。

```
真眼
 ※
ＢＢＢ
※Ｂ・Ｂ※
ＢＢ＊
 ※
```

・：空點
Ｂ：黑子
Ｗ：白子
ｂ：黑子或空點
ｗ：白子或空點
＋：黑子或白子
＊：黑子或白子或空點
※：黑子或白子或空點或邊界
—：邊界

圖 26 真眼模組

2. 假眼：可以直接比對假眼模組來判斷此棋點是否為真眼，如圖 27。

```
假眼
 ※
＊Ｂ＊
※Ｂ・Ｂ※
＊Ｂ＊
 ※
```

・：空點
Ｂ：黑子
Ｗ：白子
ｂ：黑子或空點
ｗ：白子或空點
＋：黑子或白子
＊：黑子或白子或空點
※：黑子或白子或空點或邊界
—：邊界

圖 27 假眼模組

3. 好型：可以判斷此棋點是否為好型，如圖 28。

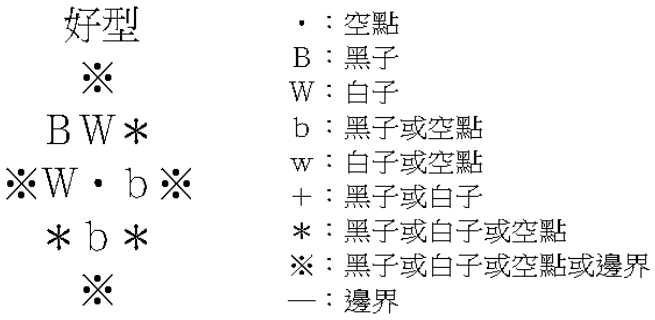



圖 28 好型模組

4. 假眼提升成真眼：判斷此棋點為假眼時，可以找出下哪個棋點可以使得他成為真眼，如圖 29。


圖 29 假眼提升成真眼模組

5. 產生虎型：判斷此棋點下了之後，有沒有機會成為虎型，如圖 30。

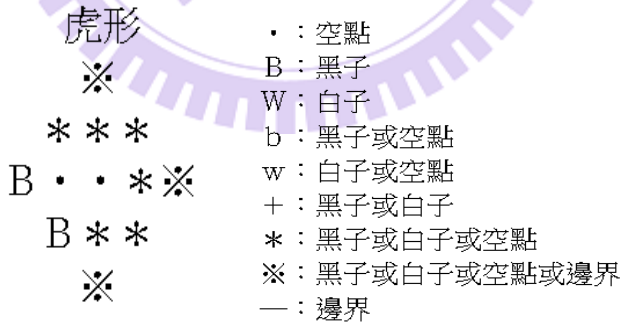
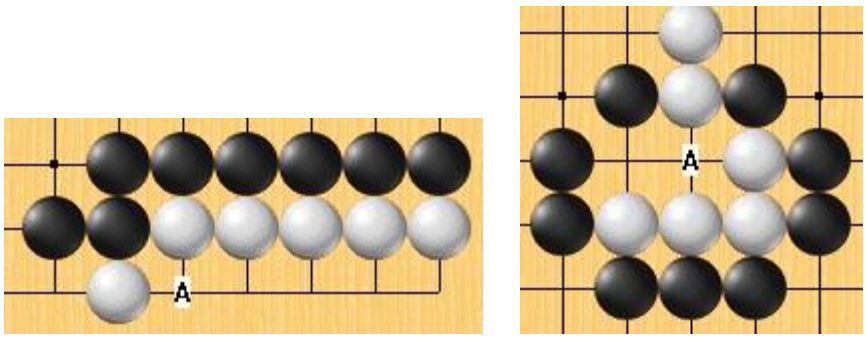



圖 30 產生虎型模組

6. 撲：在一子撲的判斷中，為了避免無用的自殺著，本論文認為有用的一子撲分為四類。消眼、反提、打劫、殺子，如圖 31。除此之外，本論文都認為是無用的自殺著。根據這四種狀況，我們可以依照這四種狀況建立模組，以利本論文實作的電腦圍棋對於撲的判斷更有效率，如圖 32。




(a) (b)

(c) (d)

圖 31 撲（a）消眼（b）反提（c）打劫（d）殺子

一子撲

※

BW＊

B・・W※

bW＊

※

・：空點
B：黑子
W：白子
b：黑子或空點
w：白子或空點
＋：黑子或白子
＊：黑子或白子或空點
※：黑子或白子或空點或邊界
—：邊界

圖 32 一子撲

程式自動產生：周圍黑子、白子、空點的數量。圖 33，周圍 1 黑子、2 白子、1 空點。

W

BWB

WW・BB

B・B

B

・：空點
B：黑子
W：白子
b：黑子或空點
w：白子或空點
＋：黑子或白子
＊：黑子或白子或空點
※：黑子或白子或空點或邊界
—：邊界

圖 33 周圍黑子、白子、空點數量

### 3.2.1.3 鑽石型模組與 3＊3 模組的比較

在這章節會分別對模組目錄大小、效能、鑽石型模組額外棋型應用方面做比較。

1. 模組目錄大小：3＊3：每一點會有黑子、白子、空點的狀況，邊界部分我們拆成各種模組來分析，總共會有三種模組狀況。大小分別為(a)$3^8$ (b)有四個方向，$3^5$＊4(c)有四個方向，$3^3$＊4 。總共有 7,641 個項目，如圖 34。

   鑽石型：在 3.2.1.2 中計算總共是 1,520,937 項目。

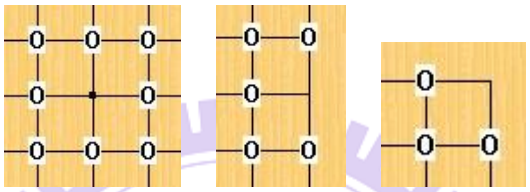


圖 34 3＊3 所有模組

2. 效能比較：表格 4 分為三種狀態來顯示。

   A. UCT+模擬：模擬部分，隨機下子，不填真眼。

   B. UCT+模擬加強：模擬增加逃跑、提子、好型判斷的 AI。

   C. UCT 選點加強+模擬加強：UCT 增加篩選機制和初始值設定。

| | 狀態 | Games/sec | 效能差 |
|---|---|---|---|
| 3X3 Pattern | A. UCT+模擬 | 14,000 | 降低 40% |
| Diamond Pattern | A. UCT+模擬 | 10,000 | |
| 3X3 Pattern | B. UCT+模擬加強 | 11,666 | 降低 33% |
| Diamond Pattern | B. UCT+模擬加強 | 8,750 | |
| 3X3 Pattern | C. UCT 選點加強+模擬加強 | 7,777 | 降低 22% |
| Diamond Pattern | C. UCT 選點加強+模擬加強 | 6,363 | |

表格 4 Diamond 與 3＊3 效能比較

3. 鑽石型模組額外的棋型應用：

A. 產生虎型。

B. 好型。

C. 一子撲。

## 3.2.2 封閉區域

在這章節會對於封閉區域做定義介紹，3.2.2.1 介紹封閉區域，3.2.2.2 會介紹搜尋封閉區域的機制。

### 3.2.2.1 封閉區域介紹

封閉區域是由相同棋串所圍成的區域且無相異顏色的棋串，目的是可以更快得知棋串的死活狀況，因為封閉區域在某些條件下，可以列為至少一眼，如圖 35。

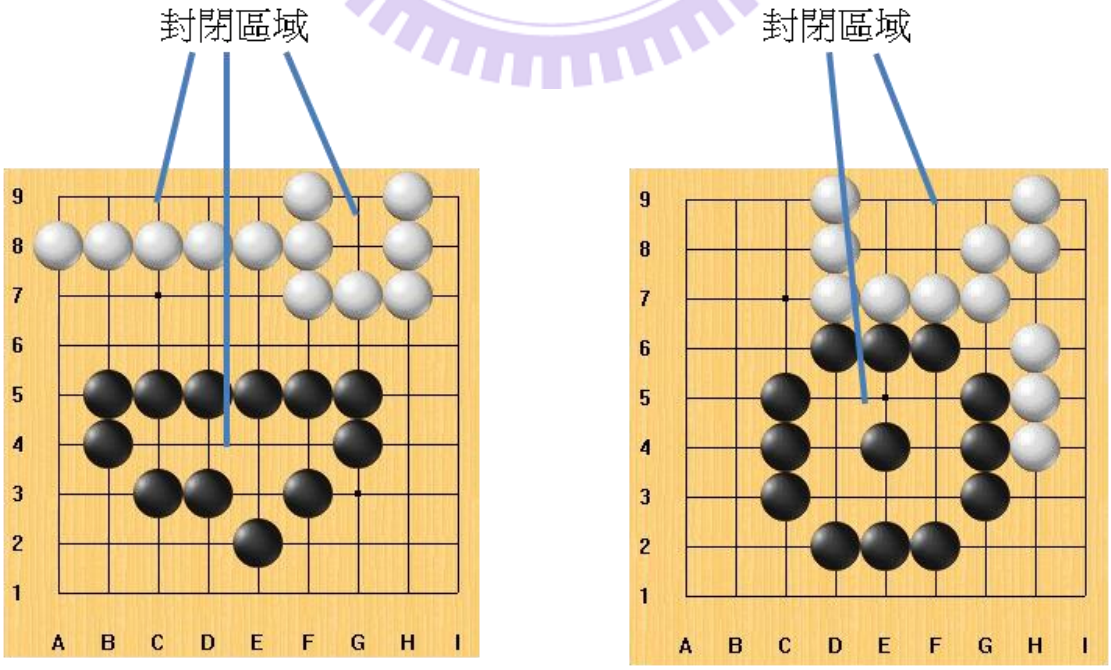


圖 35 封閉區域

### 3.2.2.2 封閉區域搜尋機制

使用模組的方式進行比對，因為使用模組比對時，可以很快得知下一次要比對的空點方向，以及有無對方子，因此可以得到更好的效率。封閉區域的大小本篇論文限制為八子以下，因為超過八子時，可能對方可以在封閉區域內做出活型，如圖 36。

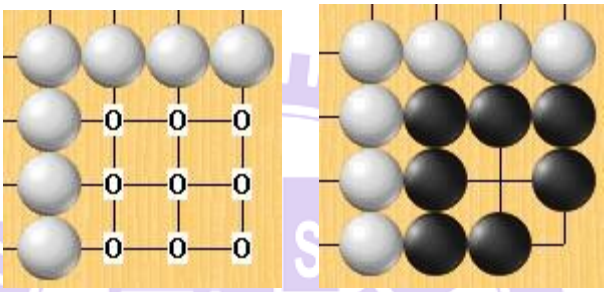


圖 36 九子封閉區域

搜尋封閉區域的模組主要是依據 Corner0(C0), Corner1(C1), Corner2(C2)，Corner 的意義代表周圍有棋子的角狀況，當一個角周圍有子時，且角的棋點為空點或與周圍棋子不同顏色，則稱為缺角。如果在一個棋點的 3＊3 大小內看到一個缺角，稱為缺一角。看到兩個缺角，稱為缺兩角。Corner0 代表無缺角，因此都由 Corner0 而成的封閉區域，是沒有缺角的，如圖 37。Corner1 代表此棋點顯示邊界缺一角，因此有一個或多個 Corner1 而成的封閉區域，會有一個或多個缺角，如圖 38。Corner2 代表缺兩角，因此有一個或多個 Corner2 而成的封閉區域，會有兩倍 Corner2 的缺角，如圖 39。

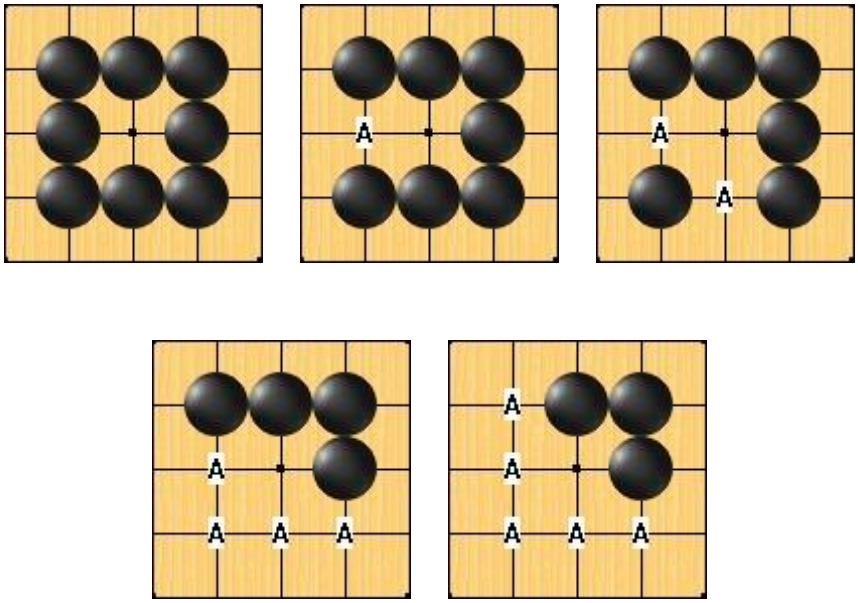



圖 37 C0

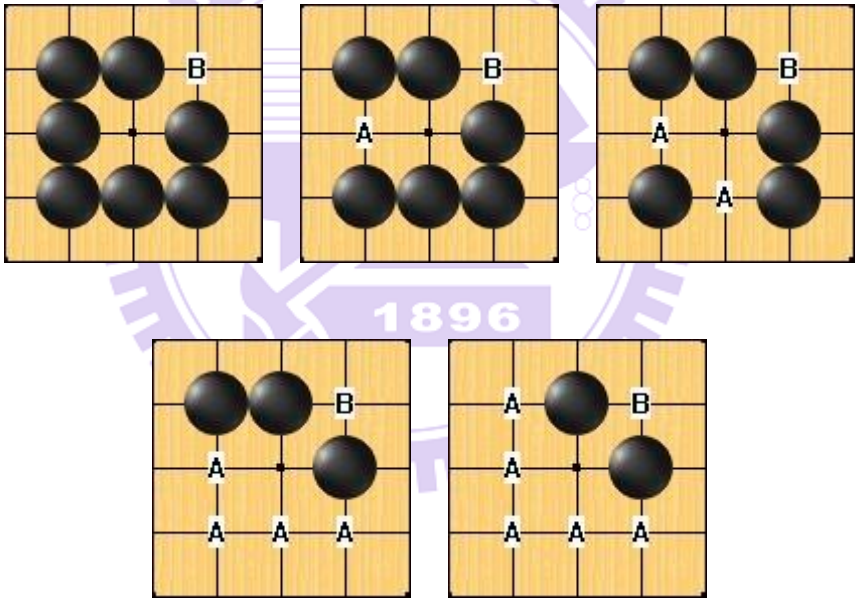
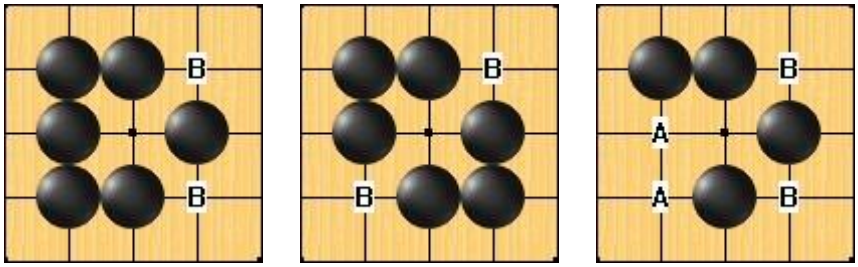



圖 38 C1



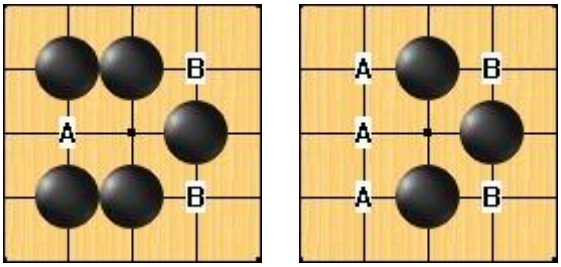



圖 39 C2

至於 Corner3(C3), Corner4(C4)，分別代表缺三角與缺四角(圖 40)。由 C3 與 C4 所組成的封閉區域必定只有一個空點，而且為假眼，本篇論文認為假眼與封閉區域在分析上會有不同狀況，因此在搜尋封閉區域時，不比對 C3 與 C4。

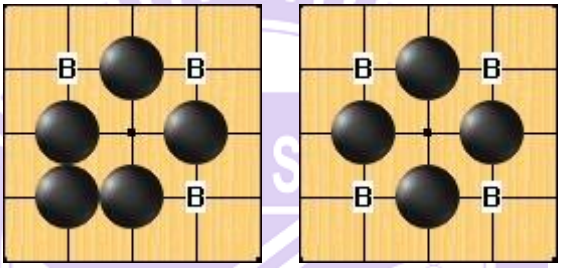



圖 40 C3, C4

圖 41 顯示封閉區域內，標明棋點屬於 C0, C1, C2 的種類。0 代表為 C0，1 代表為 C1，2 代表為 C2。

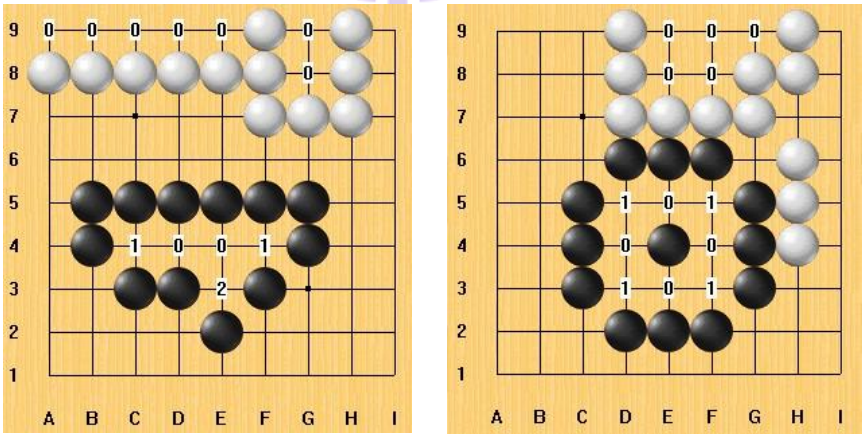


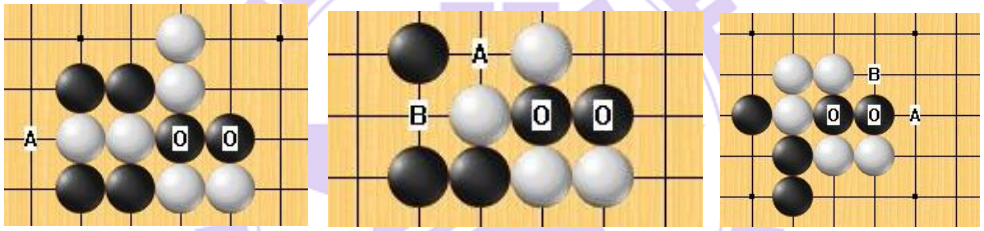

圖 41 封閉區域內部種類

## 3.2.3　好步篩選

在這章節會對於本論文實作在電腦圍棋程式的好步判斷做介紹。3.2.3.1 二氣棋串增加氣點篩選。3.2.3.2 一氣做活篩選。3.2.3.3 好型篩選。3.2.3.4 攻擊二氣棋串篩選。3.2.3.5 提子篩選。3.2.3.6 點眼篩選。3.2.3.7 假眼提升成真眼篩選。

### 3.2.3.1　二氣增加氣點

對於我方剩兩氣的棋串試著增加氣數。增加氣數的方式有提子、叫吃、延長氣點，如圖 42。

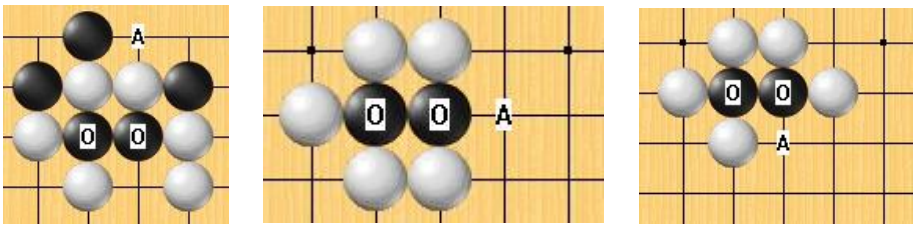



（ａ）　　（ｂ）　　（ｃ）

圖 42 （ａ）A 為提子，（ｂ）A 為叫吃，（ｃ）A、B 為延長氣

### 3.2.3.2　試著一氣做活

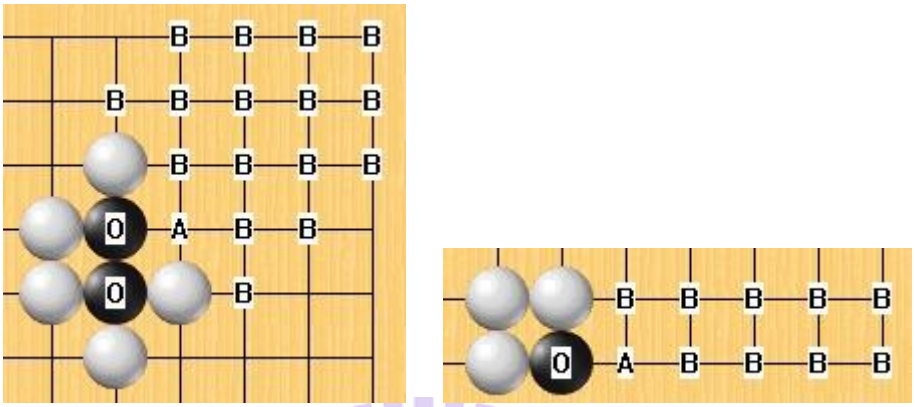
對於我方剩一氣的氣串試著做活。做活方式有提子、延長氣，在延長氣的部分必須判斷是否會造成征子，如果征子成立，則不進行長氣，如圖 43。




（ａ）　　（ｂ）　　（ｃ）

圖 43 （ａ）A 為提子，（ｂ）A 為延長氣，（ｃ）A 需征子判斷

在征子的判斷[19]，假設我方代表逃跑方，則在逃跑的路線上，只要有我方子，就認定征子不利。相反地，都無我方子，代表征子成立。征子的路線部分，在二線逃跑時，路線是梯型路線。在一線逃跑時，路線是直線，如圖 44。

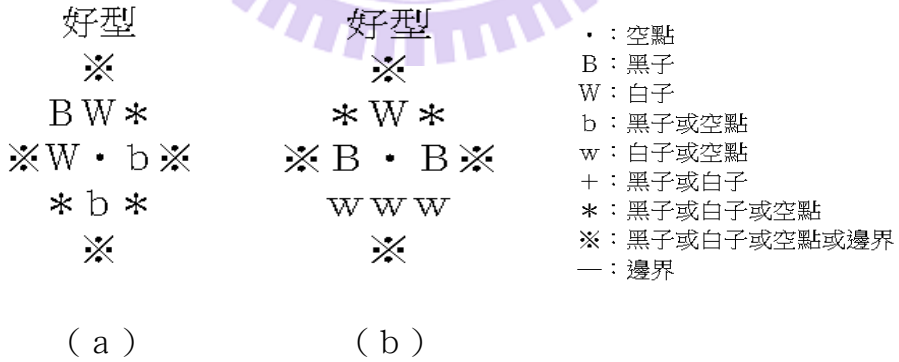



（ａ）　　（ｂ）

圖 44 （ａ）二線以上征子，（ｂ）一線逃跑

## 3.2.3.3 好型

由對方上一部周圍的12個點進行好型的比對。在此篇論文中的好型的模組，是由業餘五段棋力的人進行分析。如果期點模組吻合好型，則就認定這會是一個好點，如圖 45。



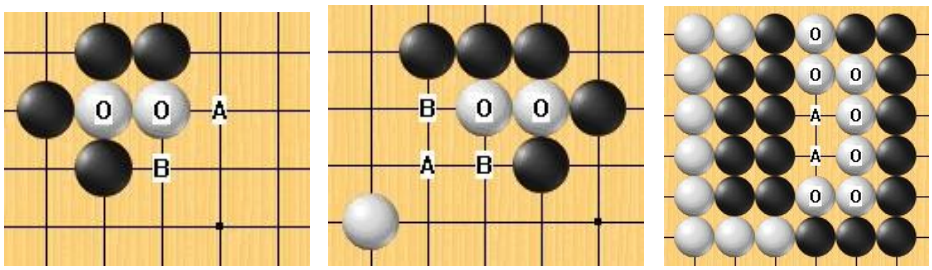

（ａ）　　（ｂ）

圖 45 （ａ）二線以上征子，（ｂ）一線逃跑

## 3.2.3.4 攻擊二氣棋串

對方剩餘兩氣的氣串，且判斷對方無法逃跑[17][18][20]，則會試著叫吃或者

門吃[7][4]。但是在此要注意雙活狀況[16]，因為當叫吃對方時，如果我方反而剩一氣，則必須針對我方棋串的棋型來分析是否為雙活，如圖 46。




（ａ）　　（ｂ）　　（ｃ）

圖 46 （ａ）A 為征子攻擊點，（ｂ）A 為門吃，（ｃ）A 為雙活的氣點

### 3.2.3.5 提子

對方剩餘一氣的棋串，則提掉對方，如圖 47。

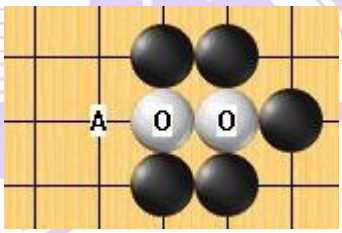



圖 47 提子

### 3.2.3.6 點眼

當上一步對方提子之後，則判斷對方周圍的棋串是否有額外的眼位，如果沒有，則可下中心點，以利對方無法或較困難的做出兩眼活，如圖 48。

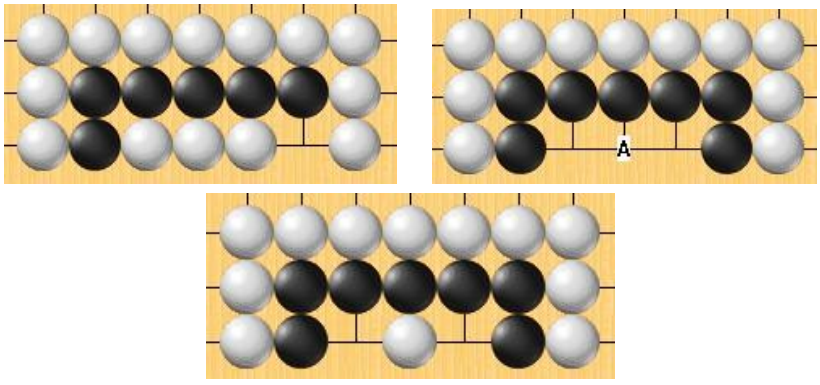



圖 48 點眼

### 3.2.3.7　假眼提升成真眼

當假眼存在時，判斷四個斜對角，如果已經佔據兩個斜對角，則另外兩個斜對角若有至少一個為空點，則此空點可以使假眼變成真眼，如圖 49。

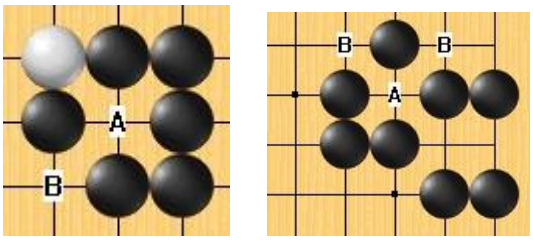



圖 49 假眼提升成真眼

## 3.2.4　壞步過濾

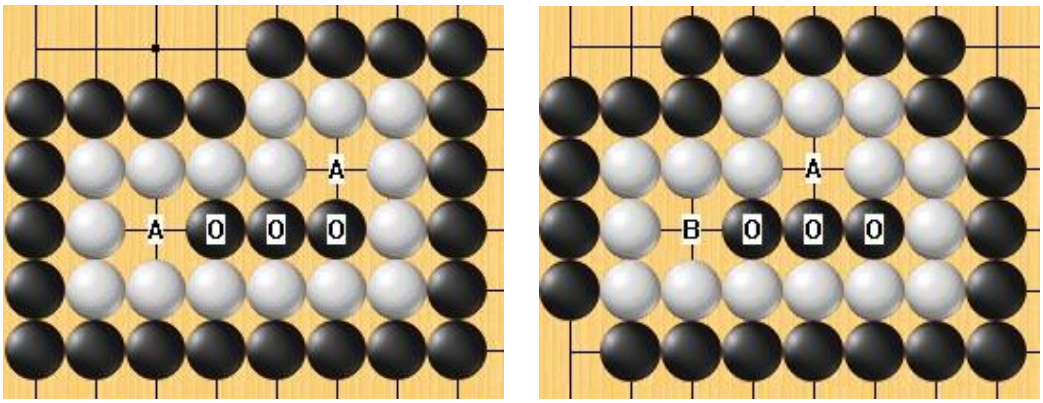
在這章節會對於本論文實作在電腦圍棋程式的壞步過濾做介紹。3.2.4.1 征子過濾。3.2.4.2 自殺步過濾。3.2.4.3 安全封閉區域過濾。

### 3.2.4.1　征子

在 3.2.3.2 介紹了征子的判斷，如果征子成立時，則將逃跑的氣點列為壞著。

### 3.2.4.2　自殺步

對於我方下一子之後，會造成我方棋串變為 1 氣，則必須進行判斷。判斷在下了自殺著之後，我方可否依據點眼來使得對方棋串只剩一眼，如果沒法點眼，則自殺步列為壞著。3.2.1.2 時介紹了撲，撲也是下了一子之後，我方棋串剩 1 氣。撲如果沒有任何效果，也就是沒有消眼、反提、打劫、殺子的情況下，都列為壞著，如圖 50。




（a）　　（b）

圖 50 （a）A 為無用自殺著，（b）A 為有用自殺著，B 為無用自殺著

### 3.2.4.2 安全封閉區域

當封閉區域都是由 C0 組成時，可以定為至少一眼。如果封閉區域內部的空點沒有在一線的棋點，則可以容許 1 個 C1 存在時，依然是至少一眼。因此當封閉區域周圍的棋串都至少兩眼時，則封閉區域算是安全的，此內部的空點歸列為壞著。如 51 圖上邊白子棋串，具有兩個封閉區域，且兩個都由 C0 所組成，因此都各自至少一眼，加起來就是至少兩眼，所以上邊白棋為已活棋串。兩個封閉區域因為都由 C0 組成且周圍的白棋棋串為活棋，所以兩個封閉區域都判斷為安全封閉區域[15]。

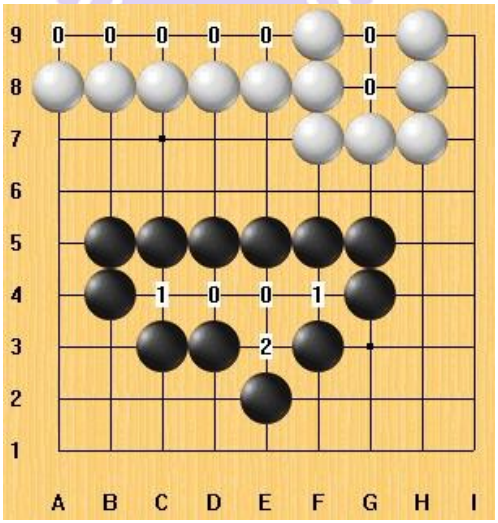



圖 51 安全封閉區域

## 3.3 模擬棋局

在這章節會分析在模擬棋局時，各點的優先權排序，以及有效率的提升模擬棋局的效能。在 3.3.1 章節對於各點優先權做排序。3.3.2 對於提升模擬棋局的效能提出方法。

### 3.3.1 模擬棋局下子的優先權

模擬棋局的優先權，將會決定在模擬棋局時，只要高優先權的狀況符合，就會直接下子。因為在模擬棋局時，希望可以達到一定的準確度外，也希望可以有效率的進行模擬棋局，才能在有限時間內進行更多的模擬棋局。

優先權的排序如下：

1. 對手提子時，判斷是否點眼。
2. 上一步被叫吃的棋串，試著對一氣做活。
3. 上一步周圍鑽石型的 12 個點的好型比對。
4. 攻擊二氣棋串。
5. 提子。
6. 隨機下子。包含假眼提升成真眼、自殺著判斷、安全封閉區域不下。

上述的 AI 在 3.2.3 及 3.2.4 有做詳細介紹。表格 5 會對於 UCT 篩選機制與模擬優先權做一些比較。

| | UCT 篩選機制 | 模擬優先權 |
|---|---|---|
| 對手提子，判斷是否點眼 | 有 | 有 |
| 試著一氣做活 | 有<br>(全部一氣棋串偵測) | 有<br>(上一步被叫吃的棋串) |

| | | |
|---|---|---|
| 好型比對 | 有 | 有 |
| 攻擊二氣棋串 | 有 | 有 |
| 提子 | 有 | 有 |
| 假眼成真眼 | 有 | 有 |
| 自殺著判斷(撲) | 有 | 有 |
| 征子判斷 | 有 | 有 |
| 安全封閉區域不下 | 有 | 有 |
| 二氣增加氣數 | 有 | 沒有 |

表格 5 UCT 篩選機制與模擬優先權比較

### 3.3.2 使模擬棋局提前結束

為了增加模擬棋局的效能，在這提供了兩種方法，可以提前結束回傳勝負狀況。

1. 提子數量：當有一方提子數目大於對方 27 子時，得勝。
2. 我方佔有的面積：棋盤大小為 9＊9，貼目 7.5 目。則黑方佔有 45 目時，則黑方勝利。若白方佔有 37 目時，則白方勝利。

## 3.4 更新節點

在模擬結束後，會依照勝負來做節點的更新。此時可以順便對於各節點的勝負值做更新。下述的勝敗，為黑方的勝敗。在節點為黑子時，也就是兒子為白方應手時，若底下兒子的節點“全”為勝時，則該節點為勝。若底下兒子“有一”為敗時，則該節點為敗。當節點為白子時，也就是兒子為黑方

應手時，則底下的兒子“有一”為勝，則該節點為勝。若底下的兒子“全”為敗時，則該節點為敗(圖 52)。

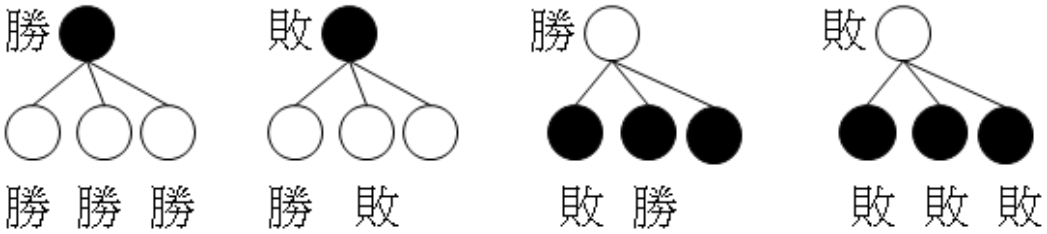



圖 52 節點狀況

# 第四章、實驗

在 4.1 節中，會以 GNU Go3.8 的 Level 10 為對弈對象，模擬次數為 3000，並且分析逐一加入新的策略是否可以提升棋力。在 4.2 節中，會以之前 HappyGo 的 AI 為對弈對象，採取每一步一秒的機制。在 4.3 節中，會以 FueGo 為對弈對象，採取每一步一秒的機制。電腦配備為：AMD 64 X2 Dual 4600+ 2.41GHz, 1.87GB 的 RAM。

## 4.1 對抗 GNU Go

對弈場數為黑白雙方各五百場，GNU Go 版本為 3.8 的 Level10。模擬次數為 3,000 盤。基本設定為 UCT 篩選機制和模擬加強的 AI 只有試著一氣逃跑、好型比對、提子，表格 6 顯示 GNU Go 對戰勝率，由上而下，會一一加入新的策略。在加入封閉區域時，平均每秒可下的盤面減少至 5000 的盤面，但是相對地，在勝率方面也有了 12%的成長，對於活型判斷能力的提升，有助於提升電腦圍棋程式的棋力。之後加入自殺步判斷與點眼、假眼提升成真眼、撲的判斷，這些判斷花費時間並不多，但是在棋力的提升上也有很好的提升，顯然壞著的過濾，也是可以提升電腦圍棋的棋力。在 UCT 篩選機制中，因為展開的節點並不像模擬棋局時執行這麼多次，因此效能並不會降低太多，但是在勝率方面也提升了 2%的勝率，在 UCT 篩選做精確的策略判斷，或許會比在模擬棋局加強還要容易得到比較好的棋力。表格 7 針對兩個較不花時間的策略，個別在基本設定和加入封閉區域下，與 GNU GO 的勝率。顯示加入自殺步判斷與點眼、加入假眼提升成真眼都能較好。至於一子撲因為屬於自殺步判斷的部分，因此沒有各別獨立測試。

| 狀態 | HappyGo 持黑勝率 | HappyGo 持白勝率 | 平均勝率 | 平均每盤/秒 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| 基本設定 | 51% | 63.8% | 57.4% | 7659 |
| 加入封閉區域 | 70% | 69% | 69.5% | 4985 |
| 加入自殺步判斷與點眼 | 71.4% | 69.4% | 70.4% | 5000 |
| 加入假眼提升成真眼 | 69.8% | 74.6% | 72.2% | 4850 |
| 加入一子撲判斷 | 71% | 76.8% | 73.9% | 4985 |
| 攻擊二氣棋串 | 73.2% | 76.4% | 74.8% | 4576 |
| 增加二氣棋串氣數 | 74.8% | 80% | 77.4% | 4552 |

表格 6 與 GNU Go 勝率

| | | | |
|---|---|---|---|
| 加入自殺步判斷與點眼 | 71.4% | 69.4% | 70.4% |
| 加入假眼提升成真眼，未加入自殺步判斷與點眼。 | 65.8% | 73.6% | 69.7% |

表格 7 個別加入與 GNU Go 勝率

# 4.2 對抗先前 HappyGo

對弈場數為黑白雙方各一百場，以每步一秒對弈。表格 8 顯示 HappyGo 對戰勝率。

| 實作論文後 HappyGo 持黑勝率 | 實作論文後 HappyGo 持白勝率 | 平均勝率 |
|---|---|---|
| 89% | 94% | 91.5% |

表格 8 與先前 HappyGo 勝率

## 4.3 對抗 Fuego

對弈場數為黑白雙方各五百場，Fuego 版本為 0.4 版，以每步一秒對弈。表格 9 顯示 Fuego 對戰勝率。

| HappyGo 持黑勝率 | HappyGo 持白勝率 | 平均勝率 |
|---|---|---|
| 42% | 56% | 49% |

表格 9 與 Fuego 勝率

# 第五章、結論與未來展望

本篇論文在 UCT 篩選機制及模擬加強部分，做了改善，在模擬次數為 3,000 盤的狀況下，與 GNU Go 對弈的結果，最後可以達到接近 8 成的勝率。且在 Inter® Core 2 Duo 3.34GHz, 3.99GB Ram 的電腦測試，第一手平均一秒可產生 7,777 的盤面。

加入封閉區域概念之後，可以越早判斷活型及安全區域的狀況下，由實驗數據得知提升了 12%左右的勝率，顯示越早得知死活狀況，及過濾下在安全區域的棋點，就會得到越好的效率。在 UCT 篩選機制中，加入增加二氣棋串的氣數，也提升了勝率，因此本論文認為在 UCT 篩選機制時，可以多花時間來做判斷與初始值設定，這樣可以更有效率的挖深好點的深度。在模擬棋局時，AI 可以比較少策略，除此可以增加每秒模擬的數量，也為了避免過多的 AI 造成模擬盤面不夠隨機，而喪失大數法則的原則。

最後在未來方向，可以加強封閉區域的判斷，來加強死活的判斷。封閉區域可以在對方下子於封閉區域內時，依然可以判斷我方此封閉區域還是有機會成眼數量，這樣可以避免對方下子於封閉區域內造成封閉區域被瓦解，而降低判斷死活的能力。在 UCT 篩選機制中，可以加入更多的 AI 判斷，加入跳、尖、雙等的判斷來加強篩選能力，或者過濾壞著，都是未來可以思考的方向。

# 參考文獻


[1] 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.

[2] B. Bruegmann, Monte Carlo Go, 1993.

[3] Chen, Keh-Hsun, Dawei Du, and Peigang Zhang, A Fast Indexing Method for Monte-Carlo Go, Computers and Games, 92-101, Volume 5131, 2008.

[4] Chen Keh-Hsun, Peigang Zhang, A New Heuristic Search Algorithm for Capturing Problems in Go, Lecture Notes In Computer Science, Proceedings of the 5th international conference on Computers and games, pp. 26-36, 2006.

[5] Jay Burmeister, and Janet Wiles, An introduction to the computer Go field and associated Internet Resources , Technical Report 339, Department of Computer Science, University of Queensland, 1995.

[6] Guillaume Chaslot, Christophe Fiter, Jean-Baptiste Hoock, Arpad Rimmel, and Olivier Teytand, Adding Expert Knowledge and Exploration in Monte-Carlo Tree Search, Lecture Notes in Computer Science, Volume 6048/2010, 1-13, 2010.

[7] Tristan Cazenave, Playing the Right Atari, International Computer Games Association Journal 30, pp. 35-42, 2007.

[8] M. Enzenberger, and M. Muller, Fuego http://fuego.sourceforge.net/.

[9] ICGA Tournaments http://www.grappa.univ-lille3.fr/icga/.

[10] Gelly, S., Silver, D., Combining online and offline learning in UCT, 17th International Conference on Machine Learning, pp. 273-280, 2007.

[11] Gelly, S., Wang, Y., Munos, R., Teytaud, O., Modification of UCT with Patterns in Monte-Carlo Go, Technical Report 6062, INRIA. 2006.

[12] GNU Go http://www.gnu.org/software/gnugo/.

[13] H. Jaap van den Herik, Jos W.H.M. Uiterwijk, and Jack van Rijswijck, Games Solved: Now and in the Future. Artificial Intelligence, 134(1-2):277-311, 2002.

[14] L. Kocsis, and C. Szepesvari, Bandit based monte-carlo planning, In 15th European Conference on Machine Learning, pages 282-293, 2006.

[15] Martin Muller, Playing it Safe: Recognizing Secure Territories in Computer Go by Using Static Rules and Search, InH. Matsubara, editor, Proceedings of the game Programming Workshop in Japan 97, pages 80-86, Computer Shogi Association, Tokyo, Japan, 1997.

[16] Xiaozhen Niu, Akihiro Kishimoto, and Martin Muller, Recognizing Seki in Computer Go, Lecture Notes in Computer Science, Vol. 4250/2006, pp. 88-103, 2006.

[17] Shunsuke Soeda, Tomoyuki Kaneko, and Tetsuro Tanaka, Dual Lambda Search and Shogi Endgames, Lecture Notes in Computer Science, Volume 4250/2006, 126-139, 2006.

[18] Thomas Thomsen, Lambda-Search in Game Trees – with Application to Go, Lecture Notes in Computer Science, 2001, Volume 2063/2001, 19-38.

[19] Wang , Yung-Le, Design and Implementation of 9x9 Computer Go Program HappyGO Base on UCT, Master Thesis, 2009.

[20] Zhang , Peigang, and Chen, Keh-Hsun, Monte Carlo Go Capturing Tactic Search, New Mathematic and Natural Computation, World Scientific Publishing Company, Volume. 4, Issue. 3, pp.359-367, 2008.