# 2022 年臺灣國際科學展覽會
# 優勝作品專輯

作品編號　**190032**

參展科別　電腦科學與資訊工程

作品名稱　**The GoClub-梅花棋演算法效率及適用性分析**

得獎獎項

就讀學校　臺北市立永春高級中學

指導教師　陳慶隆、蔡春風

作者姓名　張庭瑋、吳冠諺

關鍵詞　梅花棋、演算法、人工智慧

# 作者簡介

我是張庭瑋（右），目前就讀於臺北市立永春高中二年級數理資優班。我在高一時第一次接觸到程式碼，就深深被它吸引，因而決定一方面提升自己的數學能力、一方面投入資訊專題研究。「梅花棋」這個研究從最初的實體遊戲，到現在站上國際科展的舞台，在過程中我透過不斷學習及嘗試，發現自己對於人工智慧開發有著更高的堅持及專注。我希望日後能往這方面的主題鑽研，甚至能投入相關的領域工作。

我是吳冠諺（左），目前就讀於臺北市立永春高中二年級數理資優班。我直到上了高中才接觸到程式語言，當初的我什麼都不會，在寫程式的時候非常吃力，但是經過幾次的練習後漸漸地被它所吸引。在研究過程不斷的嘗試中也發現，我對演算法優化特別感興趣，也就是我會執著於追求一個問題最快、最簡單或最有效率的處理手法，因此跟我的組員合作完成這份研究。我希望以後能一直持續下去，並投入資訊相關的領域工作。

# Abstract

This study aims to design, develop, and intellectualize an originated chess game, the “GoClub”. The idea of GoClub originated from a math problem determining the distance between two points on the Cartesian coordinate system. The rule of this game is simple. Two players take turns locating their black and white chessmen somewhere on a 19-lined square chessboard. A player wins when a “Club”, a set of five chessmen containing a center and four vertices perpendicularly equidistant with it, is successfully established among all his chessmen on board.

We decided to design a C++ program for improving this game since man-made winning judgements became extremely difficult along with the increasing number of chessmen. Several versions of algorithm were proposed in order to improve the judging correctness and efficiency of the program. We named them the Average algorithm, the Pythagorean algorithm, the Vector algorithm, and the Reticular algorithm according to deriving order in sequence. The last one is widely used in current program design since we found it executed very well in test runs.

Finally, we attempted to upgrade our GoClub program with artificial intelligence. The decision-making rule “Minimax” was introduced when we analyzed the applicability via the concept of Victory notion. At this time, the gaming program “GoClub” allows men players to proceed complex competition, and also allows a layman player to compete with basic AI. Producing random chess composition and upgrading the level of AI will be the major works from now on.

# 摘要

本研究旨在研究一款自創棋類遊戲「梅花棋」，找出效率最佳的演算法及分析AI的適用性。遊戲規則如下：雙方玩家輪流在19階的棋盤上下棋，先手執黑子，後手執白子，任一方形成梅花即獲勝。隨著棋子的增加，肉眼判斷勝負愈發困難，因此希望借助電腦的力量完成它。我們透過C++編寫程式，持續改良優化演算法，提升電腦的精確度與流暢度。過程中依序提出了平均演算法、畢氏定理演算法、向量演算法、以及網狀編碼演算法。目前最新版本中，我們使用含有螺旋編碼表的網狀編碼演算法，這可使電腦快速正確地判斷勝負。得到最佳的演算法後，我們嘗試運用撰寫Minimax演算法編寫AI，並且不斷增加演算法的深度，從而提升電腦的實力。透過Victory notion的概念分析兩者間的相似度，判斷其對於梅花棋的適用性。透過不斷與Minimax演算法測試遊戲，將梅花棋規則中，先後手的優勢差距逐漸縮小。目前本研究已可順利進行單純的雙人對戰與複雜的人機對戰模式。

# 壹、研究動機

下課期間經常看見許多朋友在下五子棋，然而我們早已玩膩了，於是我們不禁思考是否有機會更改五子棋的規則（作者不詳，2016）進而研發出一款更有趣且運用到數學知識的遊戲。此時一道數學題目要求平面坐標2點間的距離，引發我們進行延伸思考的動機。我們想到定義棋盤上以任意一點為中心，找尋其他4點皆與其之距離相等（這 5 點合稱為梅花），即達成獲勝的條件。

但是棋類遊戲在現實中遊玩，實在有太多需要考量的繁雜問題了，像是攜帶不便或者是人眼不易判斷在複雜的棋局當中是否獲勝，如下圖所示，因此我們想利用 C++（Stephen Prata，2012；蔡志敏，2020）製作出能便利攜帶、快速且正確判斷勝負的程式。梅花棋遊戲結束及先手獲勝示意圖如下：

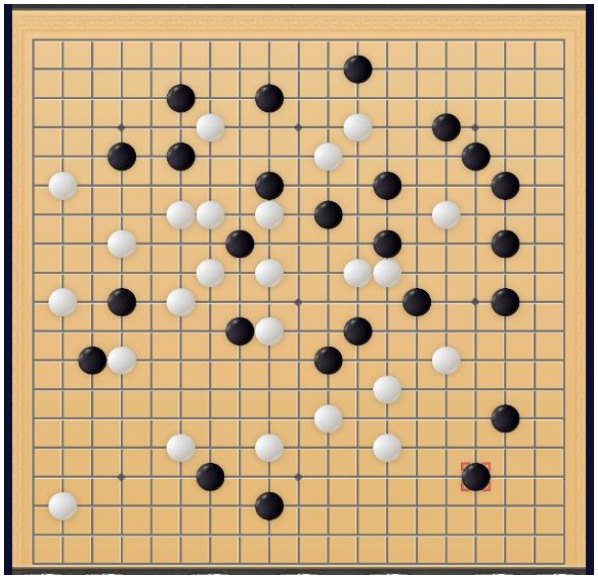

圖 1、梅花棋遊戲結束示意圖

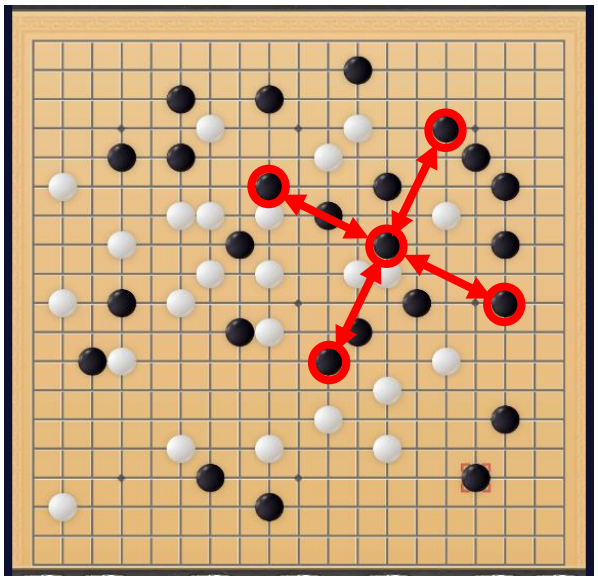

圖 2、梅花棋遊戲先手獲勝示意圖

然而現今社會中，人工智慧愈漸強大且應用於日常生活中，考慮到並非所有玩家都能找到對手，因此我們也希望能製作一套最完善的 AI 演算法，使得玩家能與電腦對弈競賽，綜上述原因我們希望蒐集分析研究相關資訊後，能自行開發 AI 演算法，並分析各種 AI 對於梅花棋的適用性。

# 貳、研究目的

本研究主要在研究並開發一款棋類遊戲「梅花棋」。以下是本研究規劃達成目標：

一、設計「梅花棋」遊戲，定義規則與獲勝條件。

二、推導並提出能順利進行遊戲、判斷勝負的演算法概念。

三、以 C++實作梅花棋演算法並比較其優劣，綜合開發出最佳化的模式。

四、以 C++及 Python 實作 AI 演算法並比較適用性的差異，進而找出最適用的演算法。

# 參、研究設計

## 一、研究設備及器材

（一）硬體

1. 個人電腦× 2

   CPU：AMD Ryzen 9 4900H with Radeon Graphics 3.30GHz。

   RAM：16.0 GB。

   硬碟空間：512 GB。

   作業系統：Windows10 家用版 x64。

   CPU：Intel®Core™ i7-10750H CPU @ 2.60GHz 2.59GHz。

   RAM：8.0 GB。

   硬碟空間：512 GB。

   作業系統：Windows10 家用版 x64

   用途：撰寫程式、架構開發、協助推導演算法。

2. 黑色棋子×191。

3. 白色棋子×190。

4. 19×19 方格棋盤。

（二）軟體及其他工具

1. Code::Blocks：

程式撰寫工具，使用版本為 17.12，是一個免費、開源且可以跨平臺的整合式開發環境，其使用了外掛程式架構，使之可以使用外掛程式自由地擴充。目前 Code::Blocks 主要針對開發 C/C++程式而設計。

2. Visual studio code：

程式撰寫工具，使用版本為 1.61.1，是一款由微軟開發且跨平台的免費原始碼編輯器。該軟體支援語法突顯、代碼自動補全、代碼重構、檢視定義功能，並且內建了命令列工具和 Git 版本控制系統。使用者可以更改主題和鍵盤捷徑實現個性化設定，也可以通過內建的擴充程式商店安裝擴充以拓展軟體功能。

3. C++：

程式語言，使用版本為 C++14，是一種常被廣泛運用的電腦程式設計語言。它是一種通用語言，支援多重程式設計模式，例如程序化程式設計、資料抽象化、物件導向程式設計、泛型程式設計和設計模式等。

## 二、研究流程架構

本研究依照下圖之流程進行。

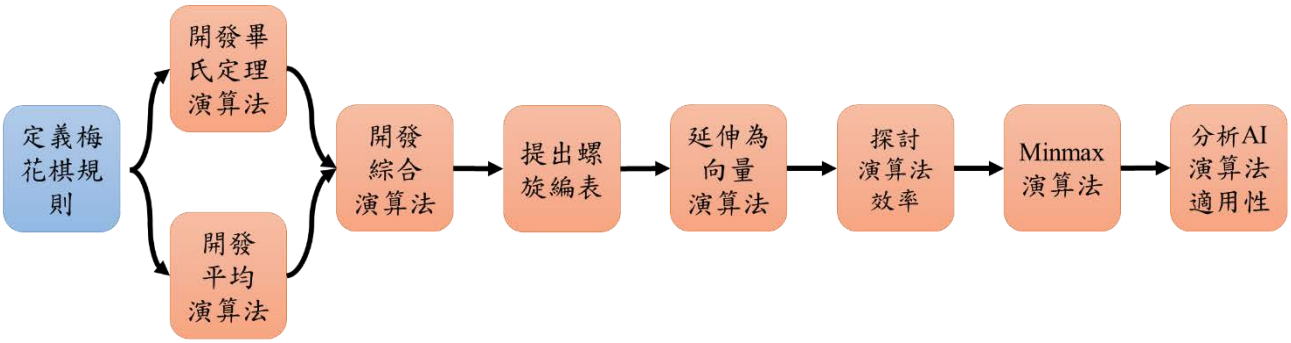


圖 3：研究流程圖

## 三、名詞定義

本研究內容中將大量提及以下幾項關鍵詞彙，為使閱讀者更加容易了解研究內容，在此解釋如下：

（一） 梅花：以棋盤上任意一點為中心，並具有 4 顆棋子與其距離相等，並且 4 顆棋子形成一正方形，此時這 5 顆棋子則稱為一朵梅花，如圖 4 所示：

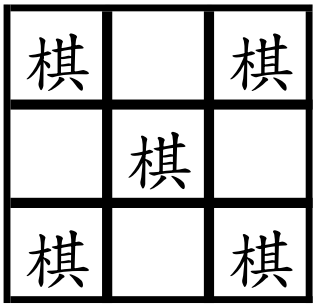


圖 4：標準梅花示意圖

（二） 花瓣：一朵梅花中，中點以外的 4 顆棋子皆稱為此梅花之花瓣，如圖 5 所示：

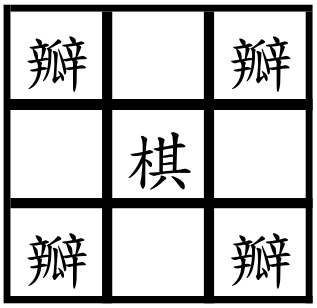


圖 5：花瓣示意圖

（三） 花蕊：一朵梅花中，中點稱為此梅花之花蕊，如圖 6 所示：

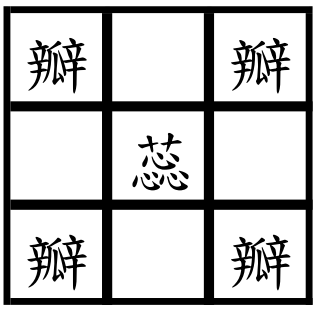


圖 6：花蕊示意圖

（四） 臂長：花蕊與各花瓣之距離稱為臂長，如圖 7 所示：

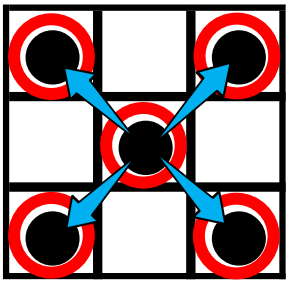

圖 7：臂長示意圖

（五） 弦長：各花瓣之間的距離稱為弦長，如圖 8 所示：

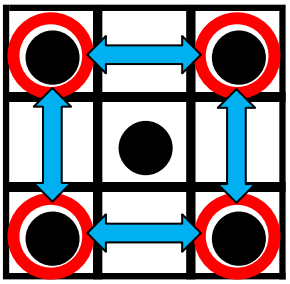

圖 8：弦長示意圖

（六） 活二：該梅花已經有兩顆棋子存在且沒有任一敵方棋子，則我們稱此圖為活二。如圖 9 所示：

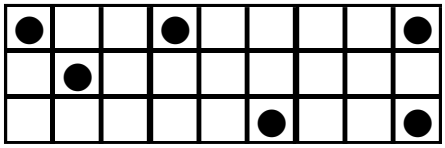

圖 9：所有活二示意圖

（七） 活三：該梅花已經有三顆棋子存在且沒有任一敵方棋子，則我們稱此圖為活三。如圖 10 所示：

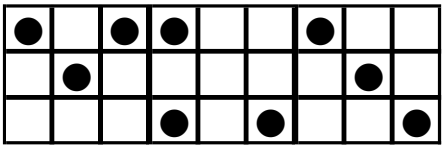

圖 10：所有活三示意圖

（八） 活四：該梅花已經有四顆棋子存在且沒有任一敵方棋子，則我們稱此圖為活四。如圖 11 所示：

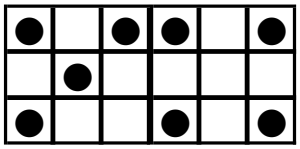

圖 11：所有活四示意圖

（九） Minimax algorithm-k search：指 Minimax algorithm 的某一特定版本。電腦每回合共模擬了自身及對手未來 k 步的所有情況，並從而選擇最有效益的位置作為本回合下棋的位置。

（十） Minimax algorithm-m by n system：指先手玩家使用 Minimax algorithm-m search 版本，而後手玩家使用 Minimax algorithm-n search 版本，模擬遊戲的進行。

（十一） 類人類思考程度：該模擬版本平均獲勝所需步數、人類理應獲勝步數及勝率的比較，也就是：

$$\frac{\text{人類理應獲勝步數}}{\text{模擬版本平均獲勝所需步數}} \times 100\% \times \text{勝率} = \frac{1700 \times \text{勝率}}{\text{棋子數量}}\%$$

# 肆、文獻探討

以下將探討本研究中人工智慧所使用的 Minimax 演算法及 Alpha-Beta 剪枝法。Minimax 演算法常用於棋類等由兩方較量的遊戲和程式。該演算法是一個零總和演算法，即一方要在可選的選項中選擇將其優勢最大化的選擇，另一方則選擇令對手優勢最小化的方法。而開始的時候總和為 0。很多棋類遊戲可以採取此演算法，例如井字棋（tic-tac-toe）。

## 一、Minimax 演算法

在下圖 12 的展開樹中，由於第 0 層代表我方下棋，所以我們取在第一層失分最少的步驟，而第 1 層代表對手下棋，所以假設對手也會採取對他們最有利的下法（也就是對我方最不利的、失分最多的），而這整顆展開樹的推論邏輯就在這種 Mini-Max 的過程中完成。當中必須補充說明的是，圖中的−∞與+∞通常代表該節點為樹葉節點，也就是整盤棋已經結束。換句話說即為有玩家獲勝。

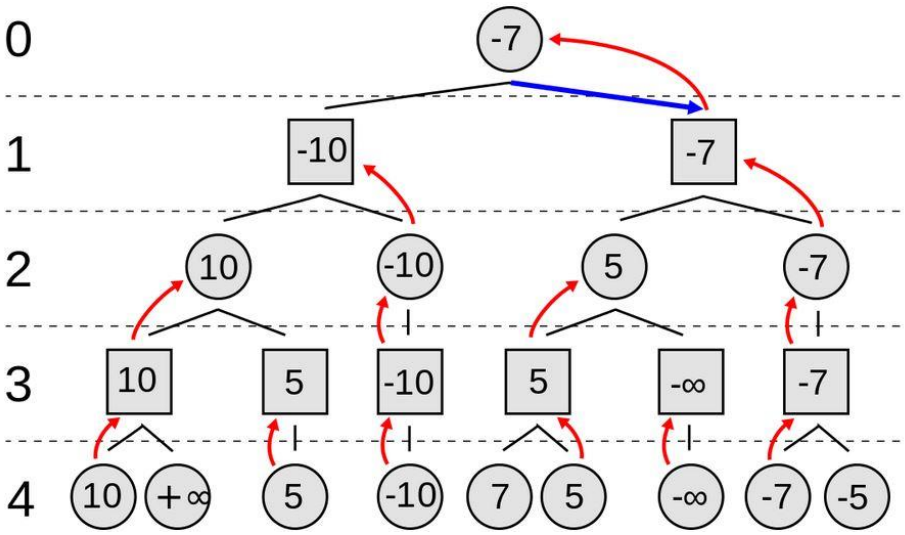


圖 12：Minimax 演算法示意圖

## 二、Alpha-Beta 剪枝法

在下圖中可以看到 Mini-Max 對每個節點都進行遞迴展開，這種展開的數量是很龐大的，因此即使電腦運算效能極高也無法展開太多層，所以我們必須透過「Alpha-Beta 剪枝法」減少展開的數量，下圖 13 是一個範例。

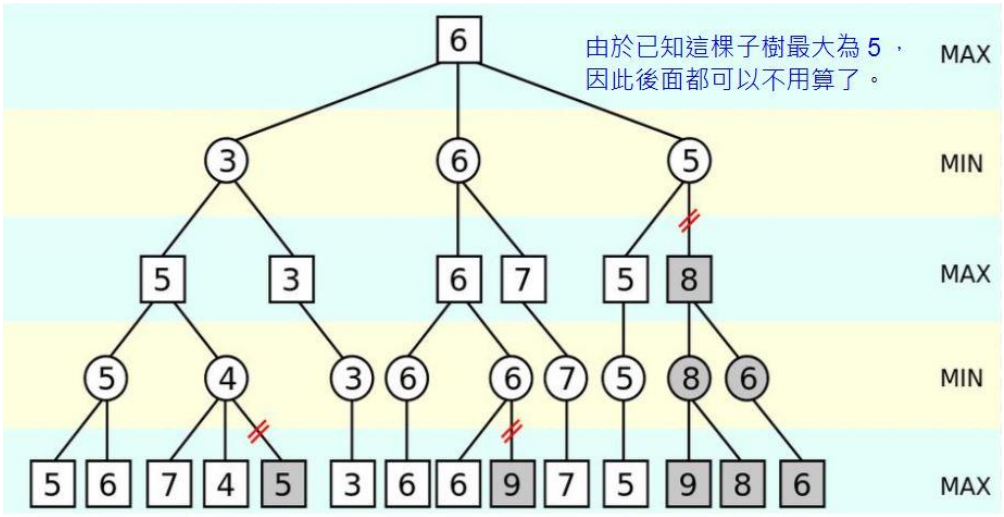


圖 13：Alpha-Beta 剪枝法示意圖

在上圖 13 中，請注意上面 Min 層的 5 節點，您可以看到當該節點最左邊子樹的分數 5 已經計算出來後，因為 5 比 8 還小，因此不管後面的節點分數為多少，都不可能讓其父節點變得比 5 還要大，所以右邊的子樹都可以不用再計算了，這就是 Alpha-Beta 剪枝法的原理。

「Alpha-Beta 剪枝法」其實是「Min-Max 對局搜尋法」的一個修改版，主要是在 Mini-Max 當中加入了$\alpha$與$\beta$兩個紀錄值，用來做為是否要修剪的參考標準。

# 伍、研究結果

## 一、梅花棋規則制定與實體遊戲設計

我們在設計梅花棋遊戲時，希望梅花棋遊戲可以建立在保留五子棋最大的特性「5 顆棋子即可滿足獲勝條件」的前提下，變換出更有意義的獲勝方式。但是並沒有過多更改五子棋的執行規則，依然採取雙方玩家各執一子，輪流下在棋盤交點上，且先手執黑子，後手執白子，直至其中一位玩家獲勝，其中執行規則唯一有所不同的是梅花棋在 19 階的棋盤進行，反觀標準的五子棋則是在 15 階的棋盤進行。

而在設計與五子棋差異最大也最具有遊戲本身意義的獲勝條件時，一道數學題目引發我們的靈感，題目是這樣的：有 A、B 兩點位於一個平面坐標系上，求 X 軸上一點與 A、B 兩點的最小距離和。這讓我們想到「對稱」的概念：將梅花棋遊戲的獲勝條件訂定為以任一顆棋盤上的棋子為中心（也就是花蕊），其他 4 顆棋子皆與花蕊距離相同（即為花瓣），並且此 4 顆花瓣形成正方形，則稱該玩家形成獲勝條件。至此我們定義梅花棋遊戲規則如下：「雙方玩家輪流在大小為 19 階方格棋盤上將棋下在交叉點上，先手執黑子，後手執白子，直至其中任一方形成一朵標準梅花即為該玩家獲勝」。梅花棋遊戲結束及先手獲勝示意圖如下：

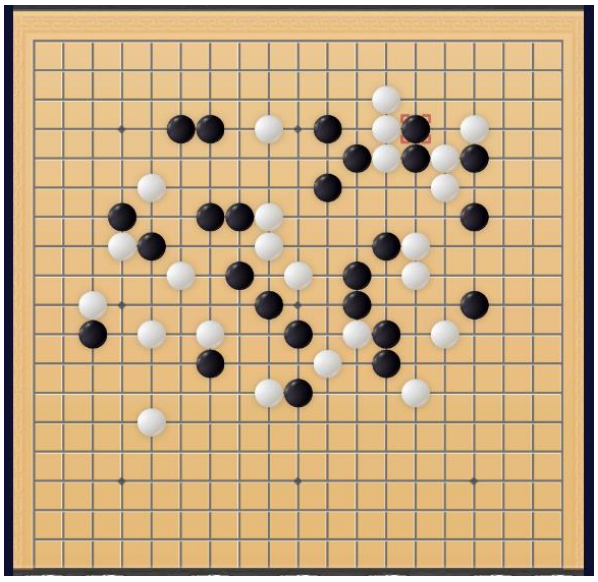

圖 14：梅花棋遊戲結束示意圖

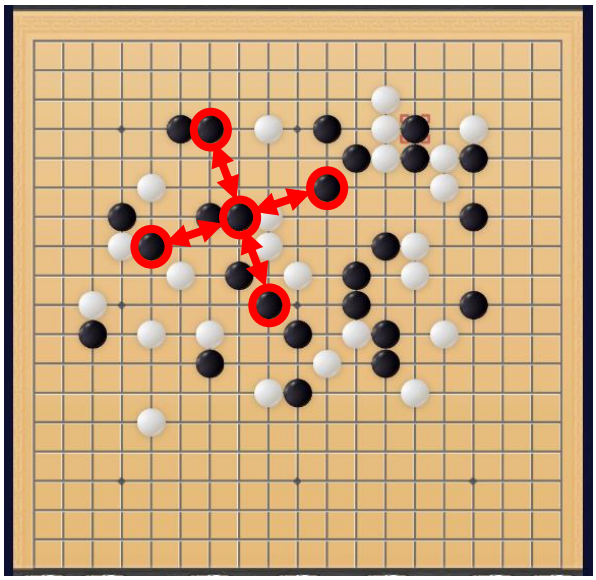

圖 15：梅花棋遊戲先手獲勝示意圖

觀察圖 14、圖 15 可以發現，此遊戲看似簡單，但隨著遊戲的進行，棋盤上的黑白子數量隨之增加，想要直接以肉眼迅速判斷是否形成梅花或哪一位玩家獲勝將愈發困難，因此我們希望能借助電腦強大的運算能力，進而達到精準且有效率的判斷梅花是否形成從而快速判斷遊戲是否結束。

## 二、梅花棋程式設計與開發

### （一） 平均演算法的開發

起初我們把棋盤定義為類似一個平面坐標，且將每個棋子都想像成一個坐標為(x,y)的點，而既然要計算兩點之間的距離，因此我們直覺想利用畢氏定理來作計算。但是經過紙上的思考與討論後發現，一朵梅花其具有非常重要的性質：由於梅花定義為「以棋盤上任意一點為中心，並具有4顆棋子與其之間最小的距離相等，並且4顆棋子形成正方形」，所以花蕊一定位於花瓣形成的正方形中央，也就是對於這朵梅花而言通過花蕊且斜率與正方形任一邊之延伸線相同的線段必為此梅花的對稱軸，因此可知若梅花4個花瓣的$x$坐標及$y$坐標分別為$(x_1,y_1)$、$(x_2,y_2)$、$(x_3,y_3)$、$(x_4,y_4)$，則平均必為花蕊的坐標$(x,y)$，綜上所述可推得平均演

算法計算關係式為

$$\frac{x_1 + x_2 + x_3 + x_4}{4} = x \wedge \frac{y_1 + y_2 + y_3 + y_4}{4} = y$$

至此我們將演算法流程設計如下圖 16 所示：

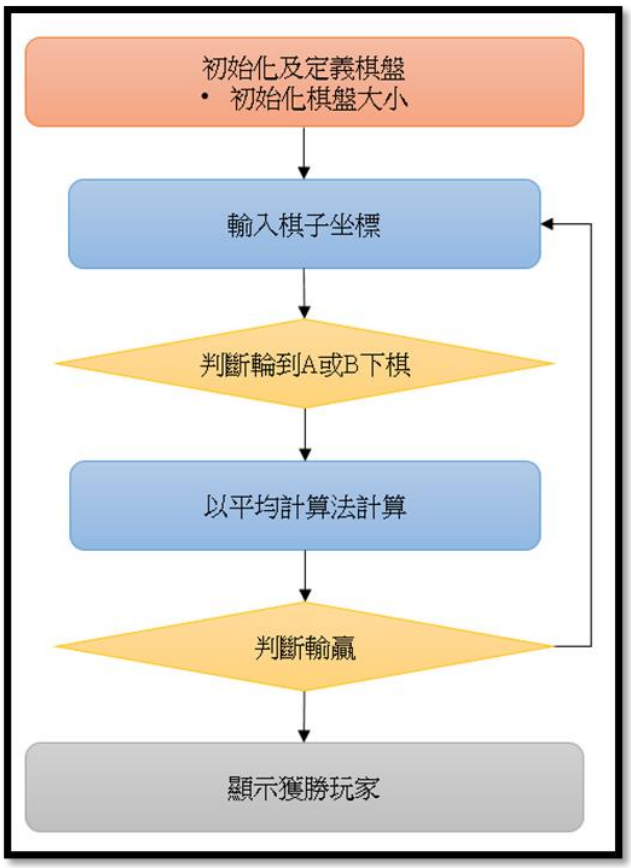


圖 16：平均演算法流程圖

（二） 平均演算法的概念

在平均演算法中，我們利用平面坐標的概念將圖 17 各點的坐標以表 1 的形式存入陣列中討論。取出所有坐標後我們發現四個花瓣的坐標平均必為中點之坐標，因此我們利用此特性判斷勝負。利用已經取出的花瓣取坐標的平均數接著和花蕊的坐標比較進而判斷是否獲勝，如圖 17 及圖 18 所示：

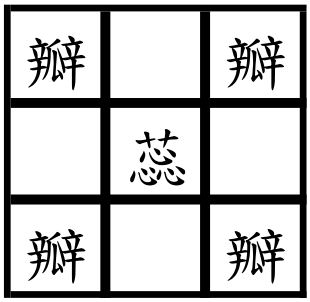


圖 17：實際棋盤中其中一方獲勝之情形

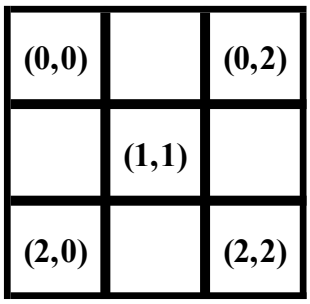


圖 18：實際棋盤中梅花之各點坐標

但在計算過程中可能會發生下表 1 的情況，梅花內部散落著零星的棋子，因此我們利用 4 層的巢狀迴圈（每層控制 1 個點，即一次計算 4 個花瓣加 1 個花蕊）將所有可能為花瓣的情況列出，分別計算直到擁有任一組符合梅花定義才算達成獲勝條件。

表 1：實際棋盤中旗子$x$、$y$坐標存入陣列之情形

| X坐標 | 2 | 0 | 0 | 0 | 4 | 4 |
|---|---|---|---|---|---|---|
| Y坐標 | 2 | 0 | 2 | 4 | 0 | 4 |

（三） 平均演算法測試版本（3×3）實作

在製作 3×3 的測試版本時，我們希望能夠做到將棋盤上棋子分布的樣態輸入後能透過演算法推算出正確的遊戲結果。因此我們將棋盤即時樣態儲存後，再分別將雙方玩家的棋子取出進行平均數的運算。

經過實際遊玩後發現，3×3 棋盤中的梅花不論縱向、橫向甚至對角線的切割都可以觀察到其圖形仍然是對稱圖形，因此得知 3×3 的棋盤上以梅花的觀點來說，不可能存在概念上的缺陷。測試情形如下圖 19：

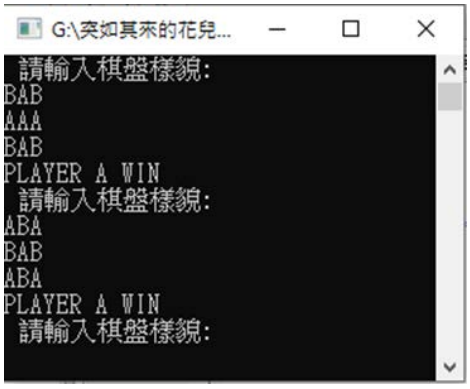


圖 19：3 × 3測試版本中玩家A獲勝之所有情形

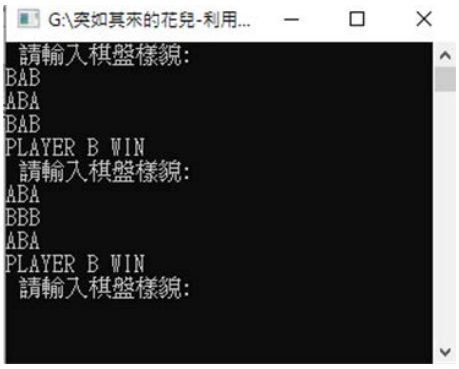


圖 20：3 × 3測試版本中玩家B獲勝之所有情形

當我們認為測試版本中不存在缺陷後，將其改寫成能夠讓雙方玩家即時遊玩的遊戲版，但礙於此遊戲版是使用 C++撰寫，因此我們讓玩家利用輸入點坐標的方式進行下棋，且為了避免有玩家肆意惡搞，因此設定如果輸入棋盤外坐標或已有棋子之坐標就會重新執行讓玩家重新輸入棋子之坐標。即便目前下棋的方式較不直觀，但現階段用 C++來測試演算法之正確性才是我們的最主要目的，因而此時暫不深究人性化界面的設計。

而在編寫過程中，我們增加了判斷該哪位玩家下棋的功能，利用變數 Turn 記錄輸入的坐標是本局棋盤中的第幾顆棋子，而當變數 Turn 為奇數時代表輪到先手玩家輸入坐標，反之，當 Turn 為偶數時則輪到後手玩家輸入坐標，因此我們得知 Turn 為奇數時只需判斷先手玩家

是否達成獲勝條件，因為後手玩家並沒有輸入新的坐標，自然就不可能達成獲勝條件，反之，Turn 為偶數時，亦僅需判斷後手玩家是否達成獲勝條件，利用此特性將可以大大減少電腦近一半的判斷時間。經過多次更改，我們僅顯示必要的資訊，過程如圖 21 及圖 22 所示：


圖 21：3 × 3遊玩版本之遊玩過程


圖 22：3 × 3遊玩版本之遊玩過程

（四） 平均演算法的缺陷

正當我們要將平均演算法推廣到更大的棋盤上時，我們發現一個致命的錯誤。該演算法的根本原理為判斷 5 顆棋子形成之圖形是否為對稱圖形，而即使是3 × 3這種範圍極小的棋盤依然存在梅花以外的對稱圖形，這也間接導致我們將棋盤所有可能出現的情形作測試時，電腦會誤判玩家獲勝，錯誤圖形及測試結果如下圖 23、24 所示：

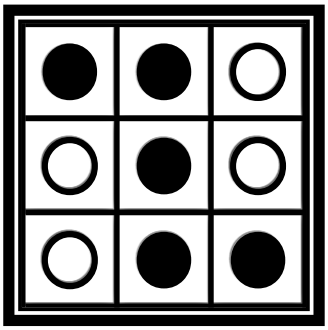

圖 23：錯誤圖形示意圖

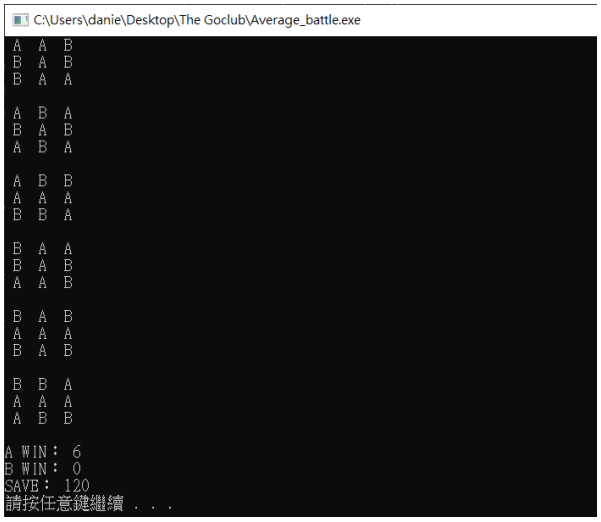


圖 24：測試結果示意圖

（五） 平均演算法軟體開發嘗試

即便在3 × 3的遊玩版本發現其缺陷，我們依然在發現之前就將控制棋盤大小的常數 digit 重新定義為標準棋盤的 19，如此一來棋盤大小即為19 × 19。畫面如圖 25 所示：

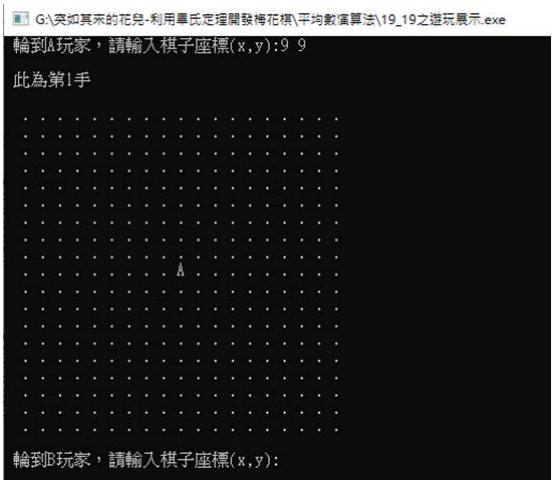


圖 25：19 × 19之空棋盤樣貌

但在經過測試後發現，一旦把棋盤放大至大於3×3時，就會形成平均演算法無法解決的缺陷，也就是如果僅單純使用平均演算法計算，而任意五顆棋子形成一條直線，則此時電腦也會判定獲勝，因為此直線圖形也是一對稱圖形，因此中點的坐標必為其餘 4 點的坐標平均。直線圖形如下圖 26 所示：

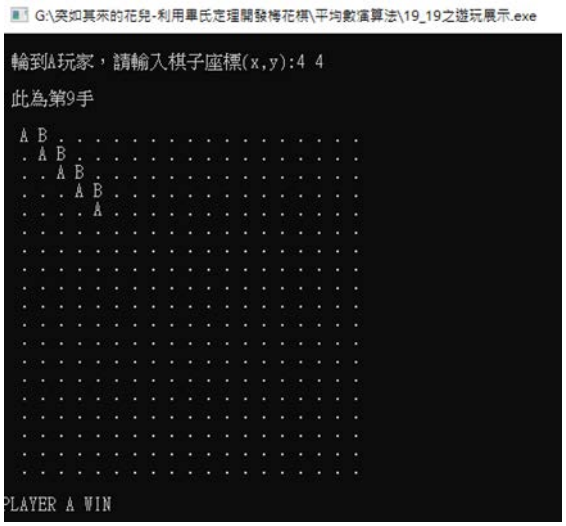


圖 26：任意5顆棋子形成一直線之示意圖

發現這個缺陷後，我們試圖以更改演算法內容或增加計算規則解決，但是始終找不到合適的方法，唯一的可能就是將所有非梅花的特定情況依序排除後再進行平均演算法的運算，但這麼做不僅極度耗費運算時間，在程式碼的編寫上也將提升一定的難度，因此我們決定更換演算法。在討論過後，我們依然決定以最直觀的畢氏定理來判斷花瓣與花蕊之間的距離，因此我們接著開發畢氏定理演算法。

（六） 畢氏定理演算法的概念

在推導過程中我們發現，如果只計算花蕊與花瓣之間的距離，有可能發生缺陷，因為棋盤上可能存在 4 段以上的臂長相同（皆以相同花蕊為中心），因此需要再判斷是否存在 4 個相同的弦長。4 段的臂長相同之示意圖如下圖 27。在畢氏定理演算法當中，既然要利用畢氏定理求各點之間的距離，我們認為將棋盤坐標化的概念依然是首選，因此如下圖將各點的坐標存入容器中討論。取出各點坐標後我們利用畢氏定理公式計算表中所有可能的臂長及弦長，

若某 4 段之值相等則可以判斷此 5 點形成一朵梅花。

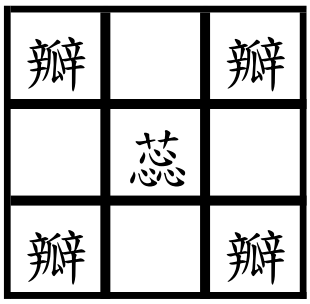

圖 27：實際棋盤中可能形成之梅花樣貌

至此，我們將畢氏定理演算法流程圖設計如下：

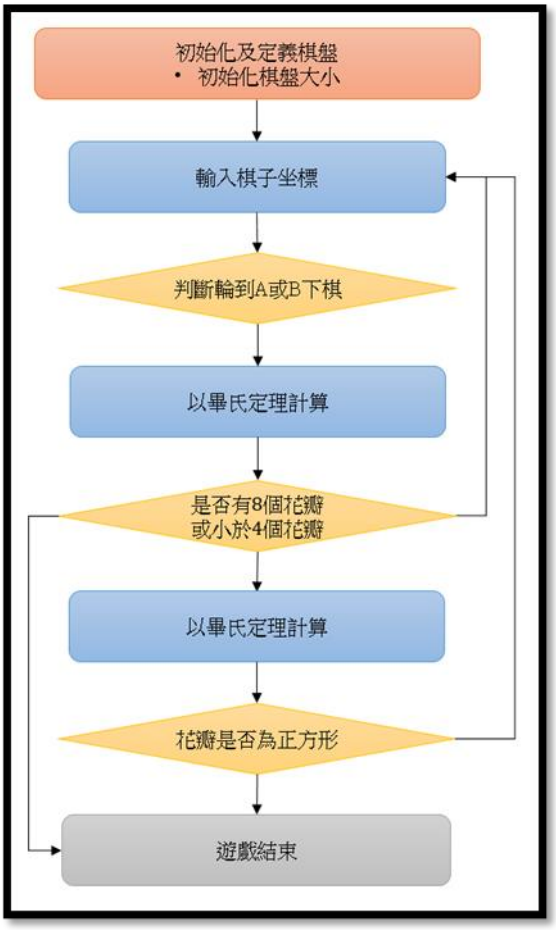


圖 28：畢氏定理演算法

（七） 畢氏定理演算法測試版本（3×3）實作

為了驗證新演算法的正確性，依然先製作3×3的版本且依然將棋盤坐標化，再分別將雙方玩家的棋子坐標取出，並且假設所有棋子都可能是花蕊，再判斷每一段臂長，假如相等且弦長也都相等，則此 5 顆棋子即達成梅花的條件，判定該玩家獲勝。遊玩過程如下圖 29：

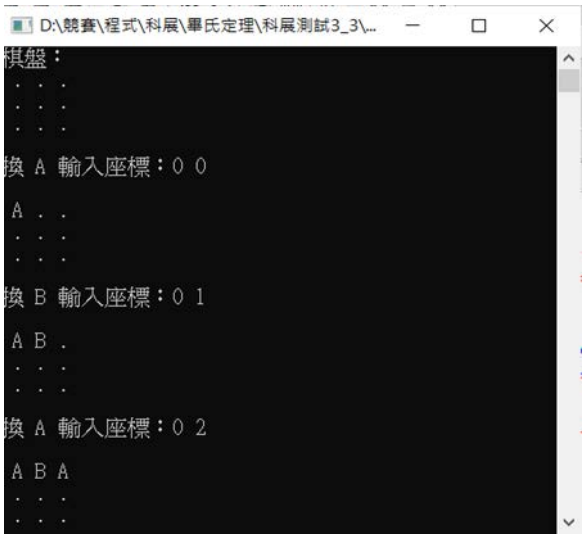


圖 29：畢氏定理演算法之3×3版本遊玩情形

較值得注意的是，在3×3中只要 4 個臂長相等則其 4 顆花瓣必形成正方形，因此為求效率在該測試版本中可以直接忽略此計算條件，至此，必形成正方形證明如下：

設棋盤中$c_5$為原點，使得$\overrightarrow{c_5c_1}(-1,-1)$、$\overrightarrow{c_5c_2}(0,-1)$、$\overrightarrow{c_5c_3}(1,-1)$、$\overrightarrow{c_5c_4}(-1,0)$、$\overrightarrow{c_5c_6}(0,-1)$、$\overrightarrow{c_5c_7}(-1,1)$、$\overrightarrow{c_5c_8}(0,1)$、$\overrightarrow{c_5c_9}(1,1)$

則兩種梅花之花瓣分別以向量內積計算可得$\begin{cases}\overrightarrow{c_5c_1}\cdot\overrightarrow{c_5c_3}=(-1,-1)\cdot(1,-1)=-1+1=0\\ \overrightarrow{c_5c_2}\cdot\overrightarrow{c_5c_4}=(0,-1)\cdot(-1,0)=0+0=0\end{cases}$

⇒ 兩四邊形對角線皆夾 90 度且對角線長度皆相等（臂長相等）

故兩者皆為正方形得證

為了再次驗證演算法的正確性，我們將3×3棋盤中的 126 種棋局利用 Greedy 算法全數檢測其是否擁有錯誤。而在執行完後我們確定了其可行性，並開始準備擴大棋盤範圍。演算

法檢驗結果如下圖 30 所示：

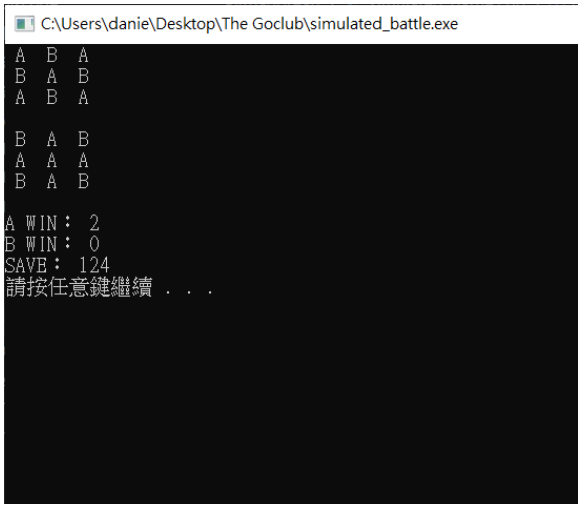


圖 30：畢氏定理演算法3×3棋盤下的檢驗結果

（八） 畢氏定理演算法軟體開發

在確定此版本中沒有缺陷後，我們希望將其改寫成能讓玩家自行決定棋盤大小的版本，因此我們將控制棋盤大小的常數 digit 改為變數並讓玩家能自行輸入，但在測試數次後發現，在3×3的棋盤中不會有像下圖 31 一樣，臂長相同的個數超過 4 個的情況，導致在n×n的棋盤中會有過早判斷玩家獲勝的可能性發生。

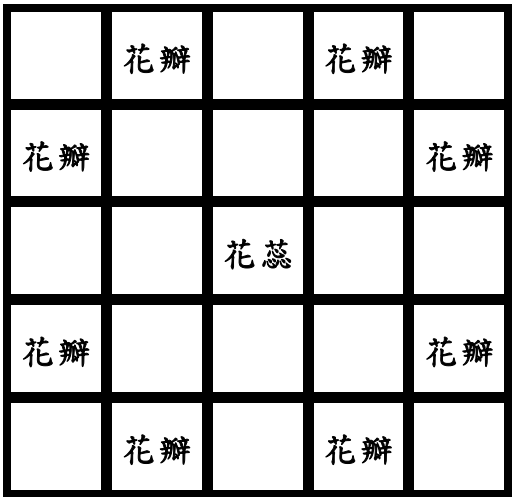


圖 31：n×n棋盤中之特例情形

因此我們新增了條件，由上圖 29 可知花瓣的數量可能大於 4 個，所以假如判斷臂長相等的花瓣僅有 3 組或以下，就不可能獲勝（棋子數已大於 5 顆），因此所有形成梅花當下臂長相同的情況是大於或等於 4，而不是恰等於 4，如果達成上述條件，則將臂長相等的花瓣取出來，計算各花瓣之間的弦長並存到新的容器中排序，假如有一樣的距離出現 4 次即 4 個花瓣形成正方形，此時才可以判斷該玩家獲勝。而但其中臂長相等的個數為 7 個時，我們就可以提前判斷此玩家獲勝。

先前演算法系統的判斷中，依然可能使系統出現誤判的情形，如下圖 30 所示，即系統依然會將這 5 顆棋子判斷為一梅花導致後手玩家無故輸掉遊戲。而經過導正，我們發現只要在儲存端加上排序功能即可有效的防止錯誤發生。至此，我們的畢氏定理演算法時間複雜度約莫$O(n^6)$。

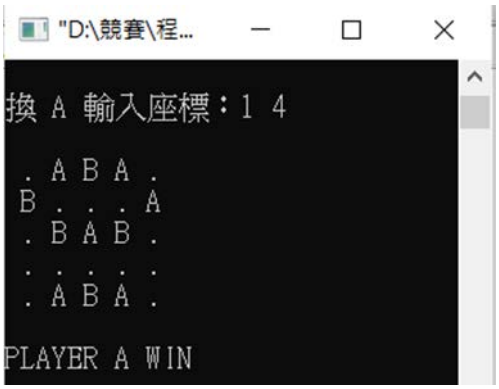


圖 32：畢氏定理演算法可能存在的缺陷情形

（九） 綜合演算法的概念

在我們整理畢氏定理演算法導正前所有缺陷的圖形時發現，缺陷只可能是上圖的同構，因此我們認為只要將此圖形的缺陷清除，程式即可正常運作。而在仔細觀察後發現，先前兩個演算法是互補的，因此只要增加平均演算法的計算條件，再判斷花瓣是否形成正方形即可。演算法流程圖 33 如下：

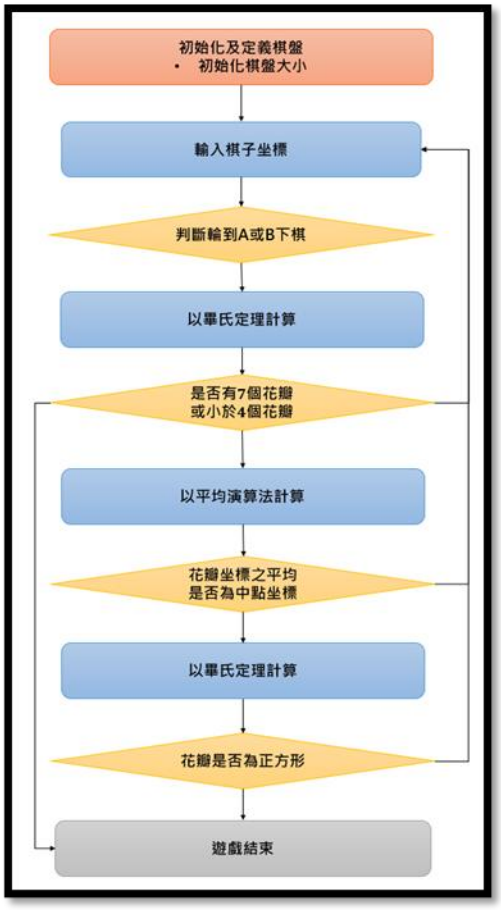


圖 33：綜合演算法流程圖

（十） 綜合演算法測試版本（5×5）開發

在訂定綜合演算法的流程後，我們利用 C++實作測試版供確認是否存在缺陷。程式中我們首先用畢氏定理演算法計算，並假設每顆棋子皆有機會成為花蕊，對其計算是否擁有 4 個或以上的臂長相等，並找出所有可能形成梅花的棋子排序後，把花蕊存在容器的第 0 項，並把所有可能的花瓣依序存在容器中。接著利用平均演算法計算，將可能的棋子數縮小為 5 顆後傳回原本的容器中並把不可能形成梅花的棋子覆蓋，最後再利用畢氏定理演算法中新增的條件計算弦長，判斷花瓣是否形成正方形，從而判斷玩家是否獲勝。程式執行過程如下圖 34 所示：

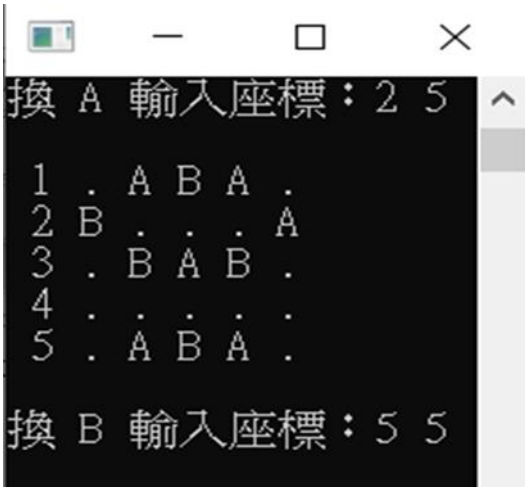


圖 34：綜合演算法執行過程

我們雖然找出了不存在缺陷的演算法，但以我們的成品來說時間複雜度仍然有$O(n^7)$，因此我們希望簡化此演算法，但在思考簡化方式時，我們發現了另外一種判斷方法。

## （十一）　螺旋編碼表

我們觀察到若以棋盤上任意一顆棋子為中心（即假設其為花蕊），則其餘所有格子（即為花瓣）皆只能形成相對應唯一的一種梅花形式，我們稱之為梅花的唯一性。至此，我們基於上述原因建立螺旋編碼表如下表，建表的概念是當每個花瓣皆只可能形成一種梅花時（花蕊固定），每個格子都不會有重複填寫的可能，也就是每個格子僅會擁有一種編碼，所以電腦僅需判斷除了花蕊之外是否存在4顆編碼相同之花瓣。

表2：螺旋編碼表

| 91 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 90 | 73 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 |
| 89 | 72 | 57 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 75 |
| 88 | 71 | 56 | 43 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 59 | 76 |
| 87 | 70 | 55 | 42 | 31 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 45 | 60 | 77 |
| 86 | 69 | 54 | 41 | 30 | 21 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 33 | 46 | 61 | 78 |
| 85 | 68 | 53 | 40 | 29 | 20 | 13 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 23 | 34 | 47 | 62 | 79 |
| 84 | 67 | 52 | 39 | 28 | 19 | 12 | 7 | 4 | 5 | 6 | 7 | 8 | 15 | 24 | 35 | 48 | 63 | 80 |
| 83 | 66 | 51 | 38 | 27 | 18 | 11 | 6 | 3 | 2 | 3 | 4 | 9 | 16 | 25 | 36 | 49 | 64 | 81 |
| 82 | 65 | 50 | 37 | 26 | 17 | 10 | 5 | 2 | 棋 | 2 | 5 | 10 | 17 | 26 | 37 | 50 | 65 | 82 |
| 81 | 64 | 49 | 36 | 25 | 16 | 9 | 4 | 3 | 2 | 3 | 6 | 11 | 18 | 27 | 38 | 51 | 66 | 83 |
| 80 | 63 | 48 | 35 | 24 | 15 | 8 | 7 | 6 | 5 | 4 | 7 | 12 | 19 | 28 | 39 | 52 | 67 | 84 |
| 79 | 62 | 47 | 34 | 23 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 13 | 20 | 29 | 40 | 53 | 68 | 85 |
| 78 | 61 | 46 | 33 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 21 | 30 | 41 | 54 | 69 | 86 |
| 77 | 60 | 45 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 31 | 42 | 55 | 70 | 87 |
| 76 | 59 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 43 | 56 | 71 | 88 |
| 75 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 57 | 72 | 89 |
| 74 | 73 | 72 | 71 | 70 | 69 | 68 | 67 | 66 | 65 | 64 | 63 | 62 | 61 | 60 | 59 | 58 | 73 | 90 |
| 91 | 90 | 89 | 88 | 87 | 86 | 85 | 84 | 83 | 82 | 81 | 80 | 79 | 78 | 77 | 76 | 75 | 74 | 91 |

在上表2中，我們將19 × 19之標準棋盤上可能形成的所有梅花皆以數字表示，但如果要以此表格直接應用在新的演算法當中，則此表必須符合兩個條件：

1. 梅花的坐標必須符合臂長相等且兩兩相互垂直。
2. 所有梅花之形狀均必須為獨一無二。

## （十二） 梅花唯一性的證明

若要滿足螺旋編碼表中的第二個條件，則第一個條件必須同時成立，且必須僅有一組坐標（4個花瓣）滿足向量內積為0，我們參考了數學課本中的向量表示方式並證明（游森棚，2021），證明的過程如下。

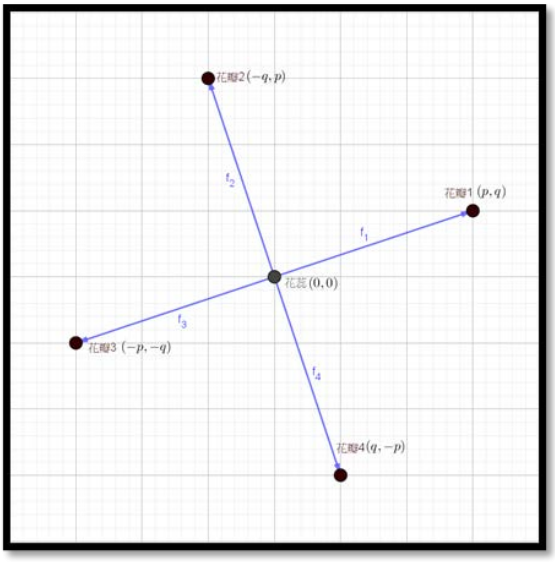


圖 35：螺旋編碼表概念圖（一）

上圖為以一顆棋子為原點計算其周圍是否存在4顆臂長相同的花瓣概念圖。在梅花唯一性的證明當中，我們假設花蕊為原點，並設其位於第一象限的花瓣坐標為$(p,q)$，令此兩點的向量為$v_1$，再分別推算出其餘三個象限中的花瓣坐標，並利用向量內積和為0之性質來證明任意向量$v_i(1 \le i \le 4)$皆成立。證明過程如下及圖34：

$$\begin{cases} v_1 \cdot v_2 = (p,q) \cdot (q,-p) = pq - pq = 0 \\ v_2 \cdot v_3 = (q,-p) \cdot (-p,-q) = -pq + pq = 0 \\ v_3 \cdot v_4 = (-p,-q) \cdot (-q,p) = pq - pq = 0 \\ v_4 \cdot v_1 = (-q,p) \cdot (p,q) = -pq + pq = 0 \end{cases}$$

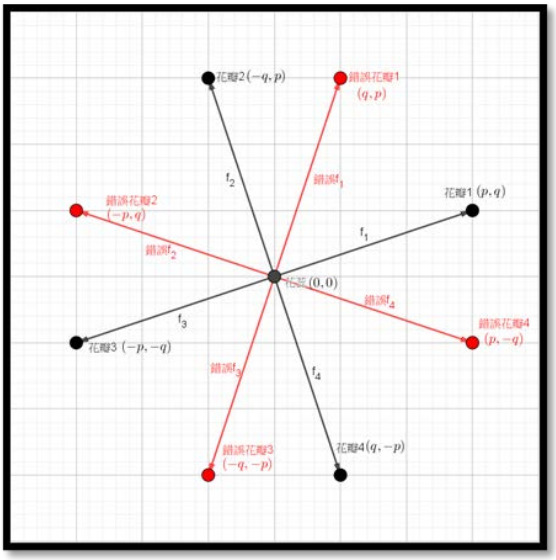


圖 36：螺旋編碼表概念圖（二）

而在上圖36中，由於先前開發畢氏定理演算法時，我們發現一顆花蕊會有超過4個臂長相同，因此我們再次利用向量內積為0之性質得證，當花蕊及其中一顆花瓣固定時，僅會有另外3顆花瓣符合形成梅花之定義，證明過程如下：

$$\begin{cases} \begin{cases} 錯誤 v_1 \cdot v_1 = (q,p)\cdot(p,q) = pq + pq = 2pq \\ 錯誤 v_1 \cdot v_2 = (q,p)\cdot(-q,p) = -q^2 + p^2 = (p+q)(p-q) \end{cases} \\ \begin{cases} 錯誤 v_2 \cdot v_2 = (-p,q)\cdot(-q,p) = pq + pq = 2pq \\ 錯誤 v_2 \cdot v_3 = (-p,q)\cdot(-p,-q) = p^2 - q^2 = (p+q)(p-q) \end{cases} \\ \begin{cases} 錯誤 v_3 \cdot v_3 = (-q,-p)\cdot(-p,-q) = pq + pq = 2pq \\ 錯誤 v_3 \cdot v_4 = (-q,-p)\cdot(q,-p) = -q^2 + p^2 = (p+q)(p-q) \end{cases} \\ \begin{cases} 錯誤 v_4 \cdot v_4 = (p,-q)\cdot(q,-p) = pq + pq = 2pq \\ 錯誤 v_4 \cdot v_1 = (p,-q)\cdot(p,q) = p^2 - q^2 = (p+q)(p-q) \end{cases} \end{cases}$$

因此可知每顆花瓣所形成的梅花都具有獨立性，其中任何一顆花蕊甚至花瓣都不能被其他棋子替代，因此在判斷輸贏時可利用預先建好的螺旋編碼表依序檢查所有棋子是否達成勝利條件，又依表格可知我們僅需判斷除了花瓣之外是否存在4顆編碼相同之花瓣即可。

我們在撰寫此演算法前推斷其時間複雜度約僅有$O(n^3)$，比起其他演算法將大幅降低編寫難度，且同時極大提升程式的執行效率，但此研究旨在推導最佳的演算法，而非最容易編寫，且此演算法效率過多浪費於建表過程，因此我們認為若更改建表方式也許能繼續使電腦之運算效率提升。

## （十三）　向量演算法概念

當我們再次探討梅花形成的原理時發現，如果我們固定任意一顆花瓣及一顆花蕊，則必定可以確定一朵梅花的樣貌，至此我們開始思索如何以效率最佳的方式找到其他花瓣。而我們發現在先前螺旋編碼表的螺旋編碼表中，可以證明所有花瓣皆與花蕊距離相同且具有獨立性僅方向不同，所以我們認為以向量的計算透過直接找點的方式可以最快找到其餘的花瓣位置，並確認該玩家是否獲勝。至此我們推導出了向量演算法：

在向量演算法中，我們首先將固定的花瓣稱為花瓣 1，之後利用花瓣 1 及花蕊間的向量平移，找出以花蕊為旋轉點旋轉 180 度的花瓣 2，並搜尋玩家是否在此點下棋，至此，我們將此方法稱為直線花瓣搜尋法。直線花瓣尋找法示意圖如下圖 35。

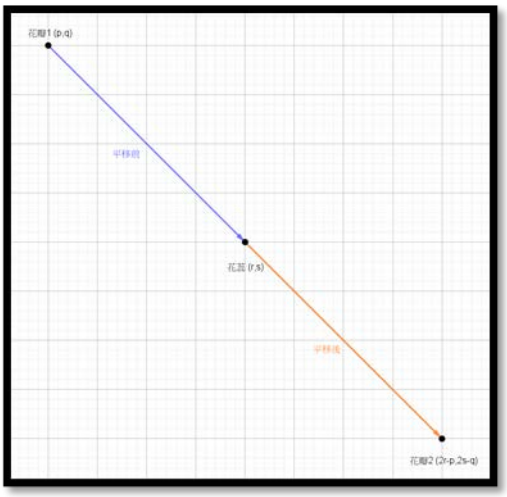


圖 37：直線花瓣尋找法示意圖

接著，要尋找左下角的花瓣，依然使用花瓣 1 及花蕊形成的向量，但是通過觀察我們可知此向量需要經過處理，透過下圖 38 可以發現，若假設原向量$v = (p, q)$，則只要將花蕊加上$v' = (-q, p)$即可找到位於左下角的花瓣 3，至此，我們將此方法稱為垂直花瓣尋找法。垂直花瓣尋找法示意圖及花瓣 3 尋找示意圖如下圖 38。

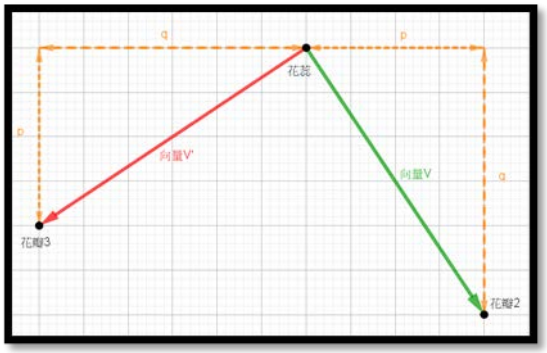


圖 38：垂直花瓣尋找法示意圖

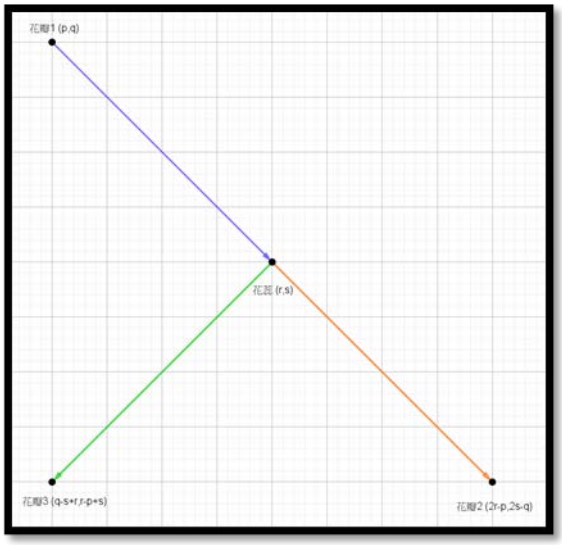


圖 39：花瓣3尋找示意圖

最後，要尋找位於右上角的花瓣 4，其中共有兩種方法可以找到其花瓣坐標：第一，直線花瓣尋找法，利用花瓣 3 及花蕊形成之向量平移；第二，再次利用垂直花瓣尋找法，利用花瓣 1 或花瓣 2 與花蕊形成的向量推導。經過計算，我們認為直接使用直線花瓣尋找法效率較高。直線花瓣尋找法示意圖如下圖 40。

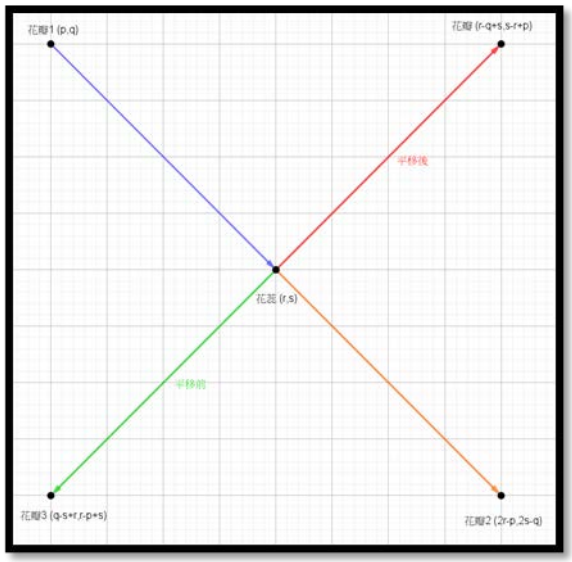


圖 40：直線花瓣尋找法示意圖

若要證明此演算法的正確性，僅須證明花瓣 2 及花瓣 3 與花蕊形成之夾角為直角即可，因為先前花瓣 2 及花瓣 4 皆為利用直線花瓣尋找法尋找得到，因此若證明其中一個角為直角

則其餘 3 個角也都必為直角。至此，證明過程如下：

$$v \cdot v' = (p, q) \cdot (-q, p) = -pq + pq = 0$$

原理呈現如上算式。實際數字代入計算過程如下。

$$\begin{aligned}\vec{v} \cdot \overrightarrow{v'} &= (2r - p - r, 2s - q - s) \cdot (q - s + r - r, r - p + s - s) \\ &= (q - s, r - p) \cdot (r - p, s - q) \\ &= qr - qp - sr + sp + sr - qr - sp + qp \\ &= 0\end{aligned}$$

（十四）　向量演算法軟體開發

本研究目前推算出最具效率的演算法即為向量演算法，其時間複雜度僅$O(n^2)$，此演算法已經可以正常執行。且在研究此演算法的過程中，我們認為目前棋盤的表示方法將嚴重影響到玩家對於下棋的判斷，因此我們重新定義了棋盤在 C++執行畫面中的表示方法。向量演算法執行畫面如下圖 41。


圖 41：向量演算法執行畫面

## （十五）　網狀編碼演算法軟體開發

經過推導，我們認為演算法的效率應該還可以更進一步的提升。在先前的向量演算法當中，對於每顆棋子都搜索一次，因此造成其複雜度達$O(n^2)$，而我們希望透過完善的建表方式，使得電腦不再需要每回合都進行複雜的計算，至此，新的網狀編碼法如下：

1. 將整個棋盤對於中心編碼一次。
2. 將中心點向外擴張後對於整圈皆編碼一次。
3. 以此類推直到中心點繞完所有可能的格子（最外圍一圈不可能）。

至此，網狀編碼演算法執行畫面如下：

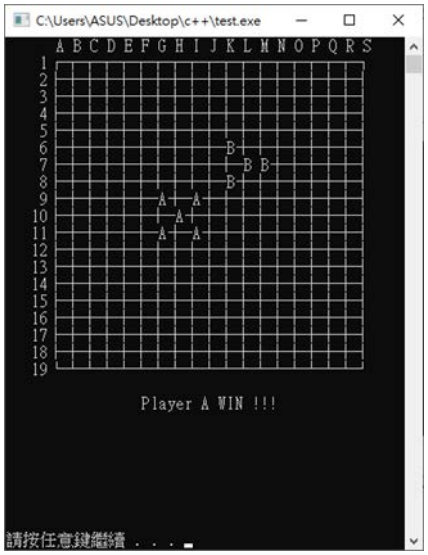


圖 42：網狀編碼演算法執行畫面

## （十六）　遊戲規則更動：三人版本

在實作出目前效率最佳的演算法後，經過多次的遊玩及測試發現：若玩家僅有兩位，則先手將會有必勝的可能性，也就是此時只要先手的玩家照著一定的規律下棋，則後手玩家將

必定無法防守成功，因此我們重新將規則更改為三人遊玩，程式內將有玩家 A、玩家 B 及玩家 C。向量演算法三人遊玩版示意圖如下圖 43。

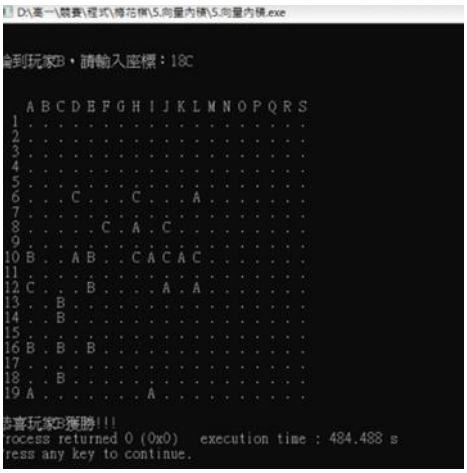


圖 43：向量演算法三人遊玩版示意圖

## 三、人工智慧實用性分析及應用

（一）梅花棋雙人版之先手必勝法：Victory notion

經過測試，我們發現在雙人版的梅花棋遊戲當中，是有機會形成先手必勝的，而這顯然必須建立在先手玩家非常聰明且不會下任何無意義的棋子。我們依照假設後手玩家也非常聰明的情況模擬遊戲的進行，透過下圖 44 可知，即使白棋每一步都完美預測黑棋下一步可能的進攻方式進行防守，黑棋依然能在至多 5 步內形成活四。

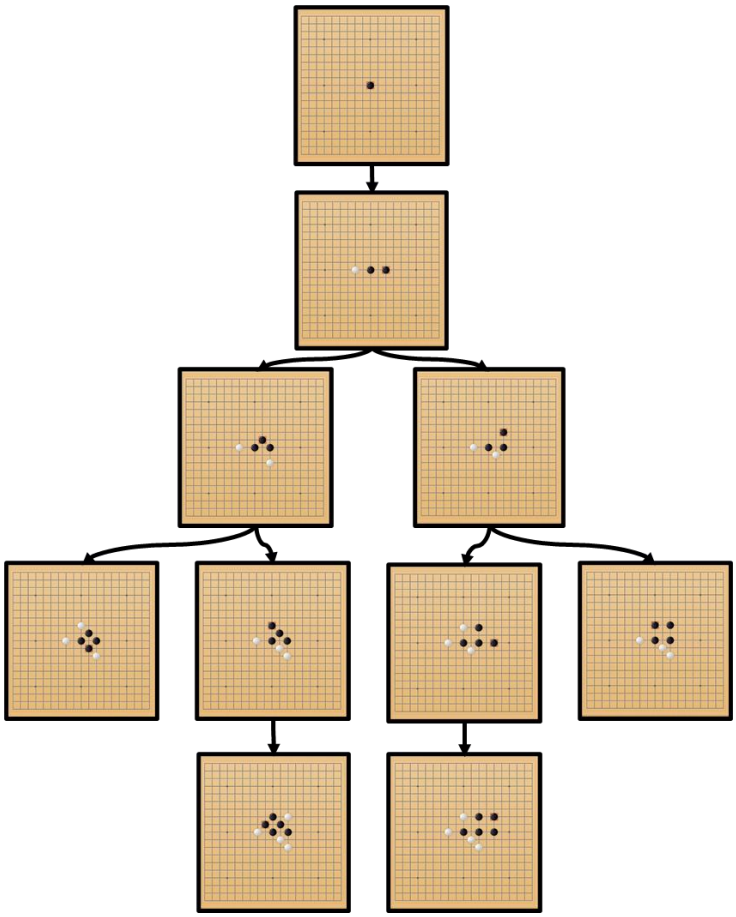

圖 44：先手活四示意圖

透過下圖 45，又在先手形成活四之後，即使後手玩家先前完美防守此圖形，此時先手玩家進攻的方式依然有 4 種，而這 4 種當中又有各 2 種方法一樣只是方向不同，且其中兩種在往後的效益較高，因此我們以此方法繼續模擬後可發現，後手玩家無論如何防守都無法阻止先手玩家形成雙活四。故推得「雙人版梅花棋先手 17 步必勝」。

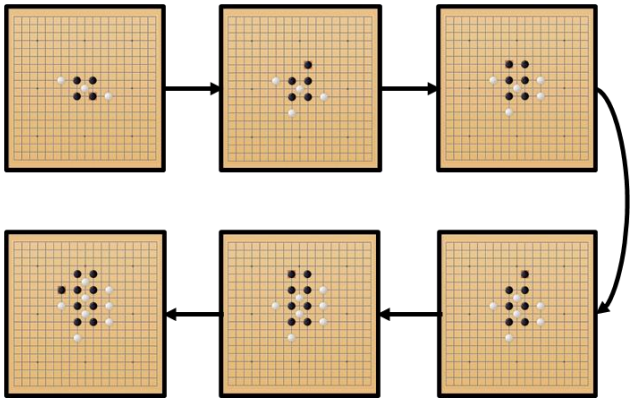

圖 45：先手必勝示意圖

（二）Minimax algorithm-one search 1.0 版

在發現 Victory notion 後，我們進一步思考如何解決此情形。經過討論我們希望透過人工智慧來嘗試是否有更佳的防守策略是我們沒有發現的，或透過人工智慧的運算來逐步制定先手玩家的限制，使得雙方玩家在實力近乎相等時勝率能趨近 50%。

在製作 Minimax algorithm-one search 1.0 版之前，我們需要先定義各種形態下梅花半成品的分數，也就是類似五子棋的活二、活三、活四等圖形（作者不詳，時間不詳），至此，我們將分數定義如下表 3：

表 3：梅花半成品分數定義

| 圖形 | 分數 |
|---|---|
| 活二 | 2 |
| 活三 | 3 |
| 活四 | 4 |
| 梅花 | ∞ |

由於最終的分數我們採取累加制，也就是將所有預判定義的分數相加，而根據編碼表可知若某號編碼中同時擁有雙方玩家的棋子，即該號梅花不可能成形。至此，較值得注意的是在進行分數計算時，必須要忽略已有其他玩家棋子的編碼，否則可能會造成誤加分數的情形。

在 Minimax algorithm-one search 1.0 版中，我們利用亂數取一空棋盤的位置計算分數，再與整面棋盤中沒有棋子的所有格子比較分數。每回合都重複以上步驟，模擬對於電腦進攻的效益以及對於對手的防守效益。最後判斷後找出當回合最具效益的棋路。Minimax algorithm-one search 1.0 版執行畫面如下圖 46 所示：

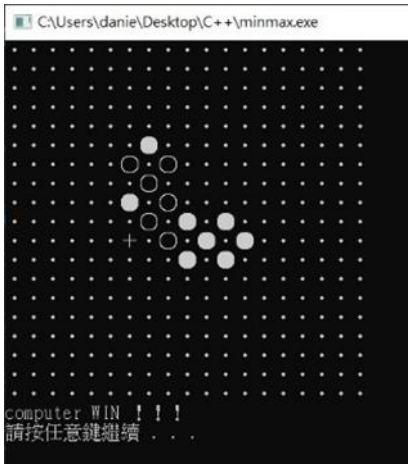


圖 46：Minimax algorithm 執行畫面

較特別的是，每場遊戲的第一顆棋子皆會下在亂數取的位置，因為 One search 層數過低第一回合無法模擬出效益。此外我們單就 Minimax algorithm-one search 1.0 版建立了 Minimax algorithm-one by one 系統，經過約莫 4 天的時間，我們成功模擬了 1 百萬場棋局。在這一百萬筆資料中，先手玩家獲勝了 511122 場，即勝率約 51.11%，且類人腦思考程度平均僅 $\frac{17}{28.08338} \times 51.11\% \approx 30.94\%$，因此可以很明顯的發現 One search 並無法滿足 Victory notion 的規則。Minimax algorithm-one by one 中先手 Minimax algorithm-one search 1.0 版獲勝步數統計圖如下：

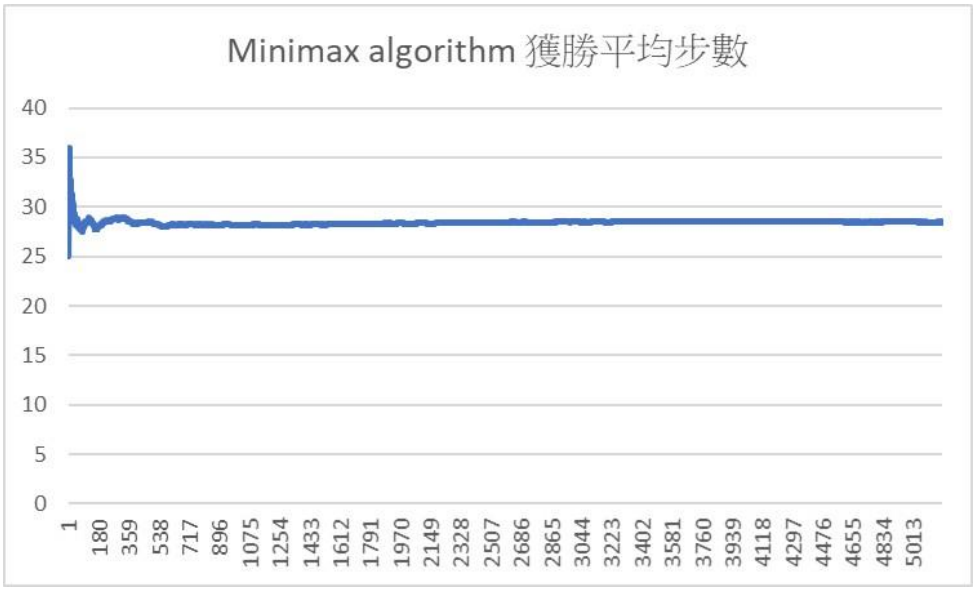


圖 47：先手 Minimax algorithm-one search 1.0 版獲勝步數折線圖

（三）Minimax algorithm-two search 1.0 版

在 Two search 中有一點需要特別注意，在第二階段的模擬是仿造對手下棋，而分數定義採取進攻效益及防守效益分開判別，也就是當進攻效益或防守效益不相同時會直接採用較高的一方進行比較，因此必須把 Minimax 中的 Mini 值也更改為 Max 值的運算。Minimax algorithm-two search 1.0 版執行畫面如下。而我們初步判斷若預測到未來 4 步，也就是必定能形成活四後，理應可以依照 Victory notion 執行，因此我們希望將分數的判斷及棋盤的模擬一層一層增加，製作出 Three search 及 Four search。

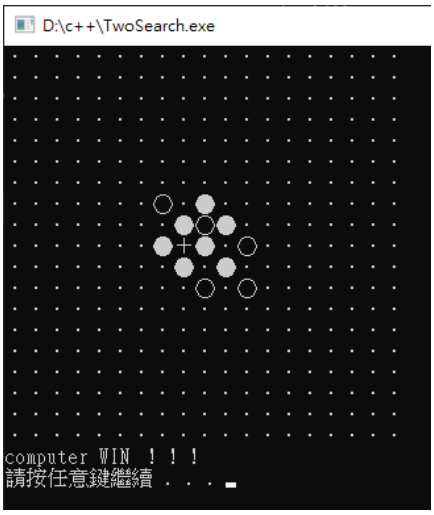


圖 48：Minimax algorithm-two search 執行畫面

（四）Minimax algorithm-two search 2.0

因為目前層數較低，所以經常會有分數相同的情況，而透過螺旋編碼表可知，整張棋盤的中心點可以形成最多種梅花樣態。這種情形在原先的版本中，大概率會被其他以亂數取得且分數相同的格子取代或直接下在第一顆分數相同的格子，可想而知這並不是最理想的判斷方法，因此我們新增了一個比較條件（包含 Minimax algorithm-one search 1.0 版）:「若任兩格分數相等時，則依其可形成之梅花總數作為判斷標準」。至此，我們開發出了較完善的 Minimax algorithm-one search 2.0 版及 Minimax algorithm-two search 2.0 版演算法。

（五）Minimax algorithm-m by n system

運用人工智慧演算法開發至今，我們已經成功研發了需多不同版本的 Minimax algorithm-one search 及 Minimax algorithm-two search，其中兩者較為完整的必屬 Minimax algorithm-one search 2.0 版及 Minimax algorithm-two search 2.0 版所有。而我們使用此兩者分別建立了 Minimax algorithm-one by one system、Minimax algorithm-one by two system、Minimax algorithm-two by one system 及 Minimax algorithm-two by two system。在這些系統中，我們將各自使用的棋子數統計分析以評估後續的適用性及類人類思考程度。其各自運行所使用之棋子數結果如下表 4 所示：

表 4：運行結果（每列表先手玩家所使用的版本，每行則表後手玩家）

| 版本 | one | two |
|---|---|---|
| one | 25 | 21 |
| two | 13 | 17 |

上表中較為特別的是即便預測到第二層，電腦的實力仍無法得到大幅度的提升，以至於後手玩家使用 Minimax algorithm-two search 2.0 時依然無法戰勝 Minimax algorithm-one search 2.0 版，不過根據表格第 2 行的數據可知，Minimax algorithm-two search 2.0 版的確實力提升了不少。此時我們觀察到，隨著對手實力的增強 Minimax algorithm-one by two system 所使用的棋子數與 Minimax algorithm-one by one system 相比理應增加，但他卻不增反減，這使我們感到困惑，於是將所有系統的棋盤樣貌甚至下棋順序輸出以觀察原因。所有系統的棋盤樣貌及下棋順序示意圖如下：

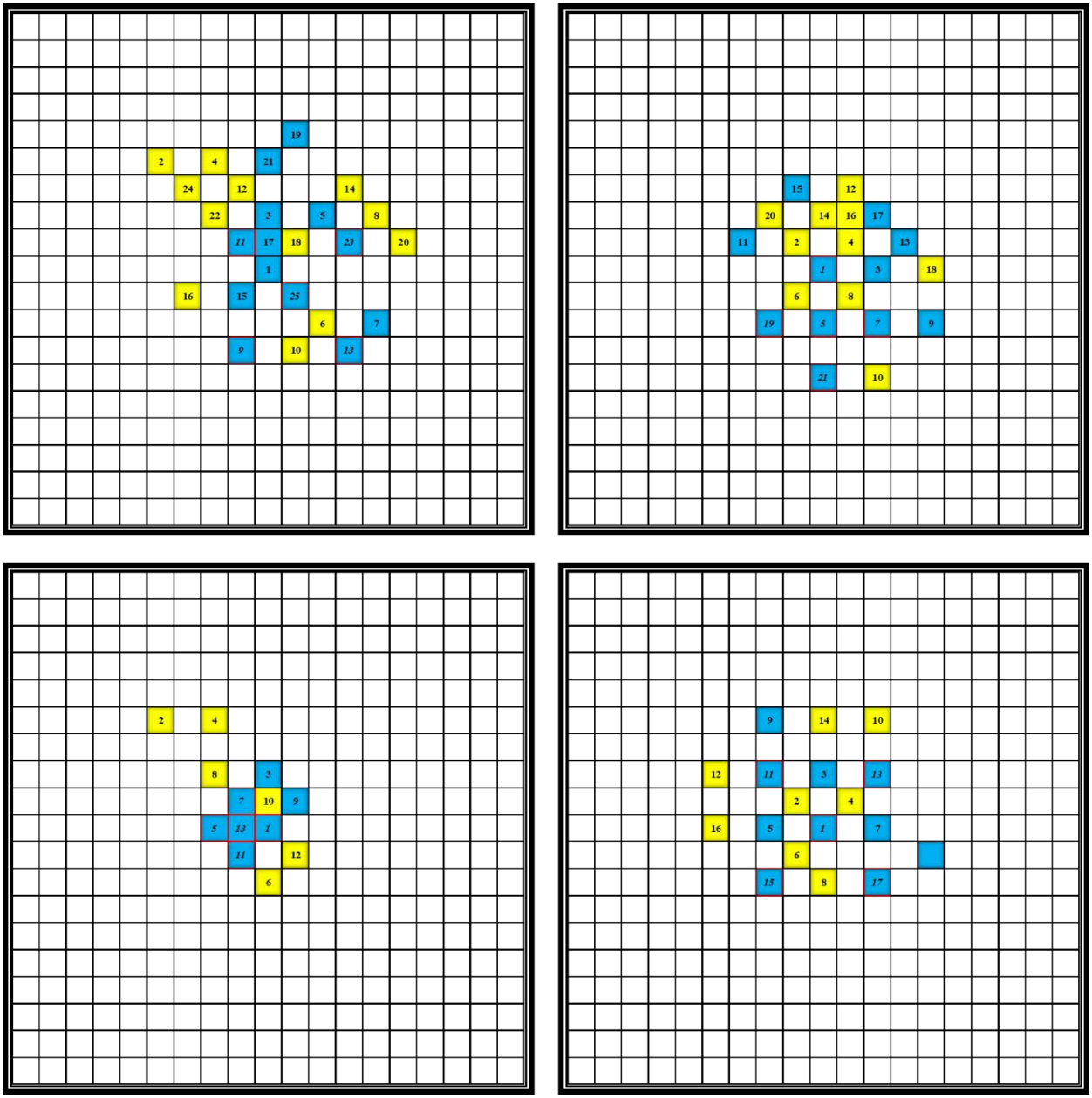

圖 49：所有系統的棋盤樣貌及下棋順序示意圖

（六）Minimax algorithm-two search 2.0 版本缺陷及優化概念

經過觀察我們發現，在 Minimax algorithm-one by two system 的第 16 步所選擇的棋子並不理想。根據 Victory notion 可知，任一方玩家形成口字型活 4 則必定獲勝，因此該步棋若下在原位的右方一格，則會直接形成 Victory notion，但是就理論上來說形成 T 字形活 4 後依然會步入 Victory notion 的流程中。示意圖如下：

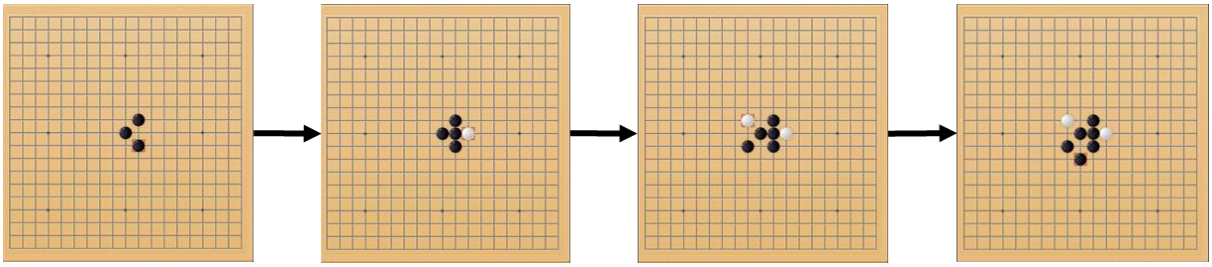

圖 50：T 字形活 4 Victory notion 的流程示意圖

不過我們發現一個特例：「在正常情況下，即使 T 字形的棋子彼此相連，則該玩家依然可以利用反向的活 3 來形成 Victory notion，但當反向活 3 已被阻擋時這樣做顯然不通，而 Minimax algorithm-one by two system 及遇到這樣的問題，而解決的方法很簡單，只要在形成活 4 時讓電腦選擇口字形即可增加後續的發展，以此達成 Victory notion。」解決方法示意圖如下：

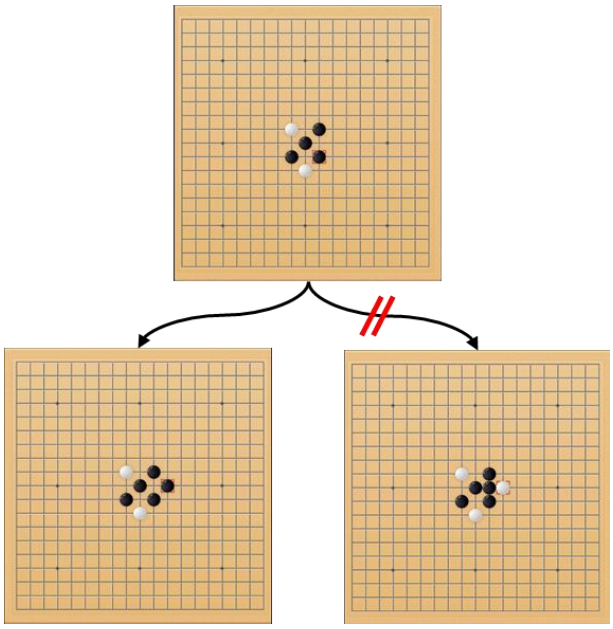

圖 51：Minimax algorithm-one by two system 第 16 步分析

（七）Minimax algorithm-m by n system 類人類思考程度及適用性分析

本研究礙於巨量時間運算因素，尚未將龐大的適用性定義完成，故目前僅能以類人類思考程度及自身的部分主觀意識判斷個演算法間的優劣，Minimax algorithm system 人類思考程度分析結果如下圖 52 所示：

| 版本 | one | two |
|---|---|---|
| one | 68% | 81% |
| two | 131% | 100% |

圖 52：人類思考程度分析結果

其中較值得深入探討的是，當先手選用 Minimax algorithm-three search 時，類人類思考程度高的離譜，我們目前研判大概因為兩個原因造成：「目前直接使用棋子數量作為主要判斷依據，這很可能是圖形截然不同而數量剛好。」，因此往後我們在定義適用性時，會優先以棋面樣貌的相似度搭配類人類思考程度進行加權。

# 陸、討論

在研究初期，我們發現選用平均演算法會造成程式具有許多此單一演算法無法解決的缺陷，例如像是原先的五子棋圖形，電腦會直接判斷該玩家獲勝，因此我們改用效率略低的畢氏定理演算法，但在最後測試版本中又發現，即使此演算法解決了平均演算法無法解決的缺陷，卻衍生出另外一些平均演算法能解決之項目，例如可能會有判斷順序的問題造成電腦誤判。

在觀察後我們發現，將兩種基礎演算法合併能完全解決所有缺陷，但這麼做將會拖慢電腦執行效率，因此我們改以利用向量找點的方式開發向量演算法，但我們依然認為演算法效率可以更快。改用建表的方式將其他演算法中的提出坐標及計算距離的步驟省略，以此大幅提升了電腦執行效率。各演算法比較如下表 5 所示：

表 5：各演算法優缺點比較

| 演算法名稱 | 編寫難易度 | 是否存在 BUG | 時間複雜度 | 演算法可用性 |
|---|---|---|---|---|
| 平均 | 低 | 存在 | $O(n^5)$ | 低 |
| 畢氏定理 | 中 | 不存在 | $O(n^6)$ | 中 |
| 綜合 | 高 | 不存在 | $O(n^7)$ | 中 |
| 向量 | 最低 | 不存在 | $O(n^2)$ | 高 |
| 網狀編碼 | 中 | 不存在 | $O(n)$ | 最高 |

在人工智慧的開發方面我們使用 Minimax algorithm 的概念首次寫出了梅花棋的 One search 版本，並且發現了雙人版有先手必勝的情況。而在模擬了 1000000 場對局並將其延伸為 Two search 版後，我們發現電腦過於依賴亂數所選的棋子，因此我們加上了對於目前 Minimax algorithm 最為重要的判斷準則：「若任兩格分數相等時，則依其可形成之梅花總數作為判斷標準」。

研究至此，我們建立了各個版本間總計 4 個 Minimax algorithm system，並各別運行後排除特例的情況，也就是儘可能降低被亂數影響的因素。最後，我們綜合分析了各演算法間的類人類思考程度，並提出了定義適用性的初步方向。

# 柒、結論與未來規劃

## 一、目前成果

本研究截至目前為止，已經推導出沒有 BUG 且效率來到O(n)的網狀編碼演算法，並利用C++實作出棋盤大小為$19 \times 19$的標準梅花棋軟體，但礙於 C++所撰寫的軟體僅適合用以檢驗演算法的正確性，並沒有人性化的介面設計（非本研究主要目標），因此我們尚未將軟體的使用者介面製作出來，暫時成為此研究目前的遺憾。不過，這是未來稍加設計就可解決的問題。

另外我們也實作 Minimax algorithm-One search1.0 版本，並且對其進行類人類思考程度的分析。而我們透過一百萬筆資料的模擬，分析出 Minimax algorithm-One search1.0 版目前的類人類思考程度如下表 6：

表 6：Minimax algorithm-One search 棋力及適用性

| 勝率 | 使用棋子數 | 類人類思考程度 |
|---|---|---|
| 51.11% | 28.08338 | 30.94% |

最後我們推廣 Minimax algorithm-One search1.0 版，實作了 Minimax algorithm-One search 及 Minimax algorithm-Two search 的 2.0 版本，並建立了 4 套 Minimax algorithm system，並且透過兩個 Minimax algorithm 間目前的最高版本，也就是 Minimax algorithm-one search 2.0 版及 Minimax algorithm-two search 2.0 版分析了類人類思考程度。類人類思考程度計算結果如下圖 53 所示：

| 版本 | one | two |
|---|---|---|
| one | 25 | 21 |
| two | 13 | 17 |

| 版本 | one | two |
|---|---|---|
| one | 68% | 81% |
| two | 131% | 100% |

圖 53：類人類思考程度計算結果

## 二、未來展望及應用

由於目前的網狀編碼演算法仍擁有其缺點，例如目前演算法中，我們是把每顆棋子分別假設其可能為花蕊再依序對棋盤編碼，如此一來，棋盤上每一格都需要存取非常多編碼，因此我們希望在未來能夠建立一套完善的編碼方式，使得電腦可以更加節省記憶體空間。其中的差異便是螺旋編碼表中的編碼表必須對於每顆棋子周圍的進行編碼，例如某玩家下了 30 顆棋後，則電腦必須編碼至少 30 次才能確定該回合是否形成梅花，而在未來我們希望能成功建

立一個新的編碼表，使得電腦在每回合判斷勝負時 ，都只需要對於整個棋盤編碼一次即不須理會盤面有幾顆棋子。另外我們也希望能夠利用 processing 或其他適合的編譯器製作出 C++不易設計的使用者介面，使得玩家能夠以更直觀的方式進行本遊戲，甚至利用一些建模軟體做出 3D 的棋盤。

目前我們已經做完的 Minimax algorithm 最多預測到未來兩步的所有可能，但這樣電腦的實力顯然不足，我們認為這是因為電腦預測的步數不足而導致。因此在未來我們希望首先將適用性的計算完整定義，並且將 Minimax algorithm 推廣到 Three search 甚至 Four search 來依序分析其適用性。我們也希望能實作深度神經網路，使電腦自行學習如何遊玩梅花棋，再分析電腦是否能自行找出 Victory notion。

我們發現如果玩家只在平面上遊玩可能會造成玩家的選擇過少，因此有機會造成先手必勝的可能性，且我們認為在 3D 立體空間中更加需要透過電腦的運算來判斷勝負，因為 3D 的棋盤在現實中幾乎不可能實現，即使實現也會很不切實際，另外也因為先前提到的 C++無法良好顯示 3D 介面，所以我們希望能夠尋找到適合的編譯器來將此遊戲推廣至 3D 立體空間，讓玩家在下棋時能有更多的選擇。

在此之前，我們已經先將立體中的標準梅花型態定義完成，我們將梅花定義為兩種狀態；第一種為 1 顆花蕊加上 8 顆花瓣，其中 8 顆花瓣可以形成正方體；第二種為 1 顆花蕊加上 6 顆花瓣，其中 6 顆花瓣可以與花蕊(假設為原點)形成一個類似三維座標系的 XYZ 軸，且 6 顆花瓣皆與花蕊等距。此兩種梅花示意圖如下圖。

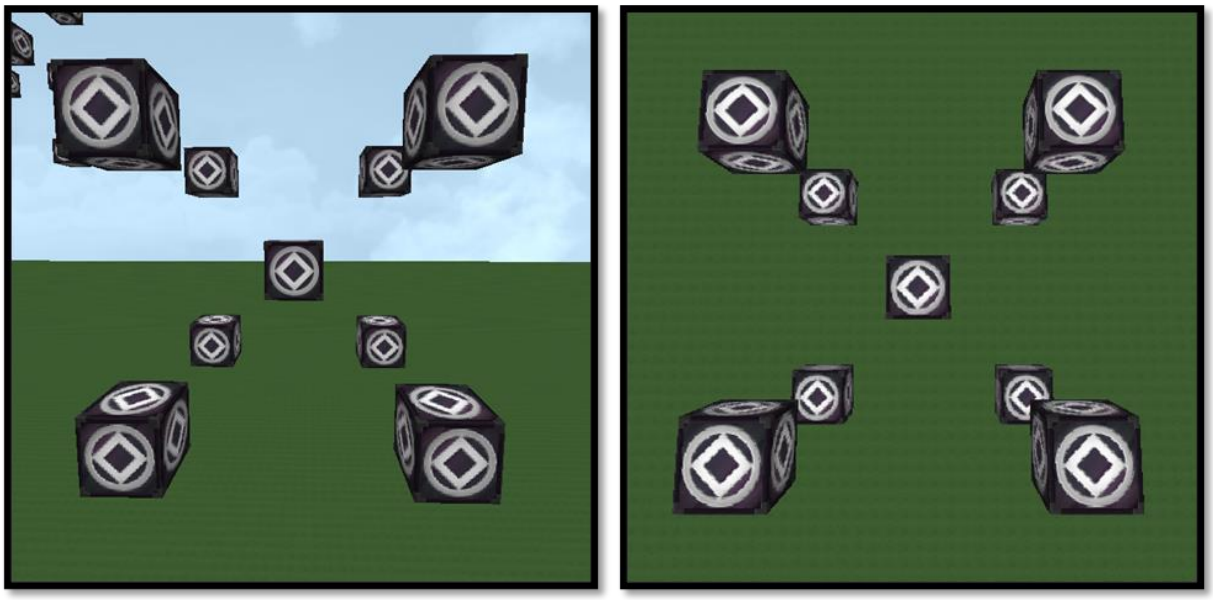

圖 54、55：正方體梅花示意圖

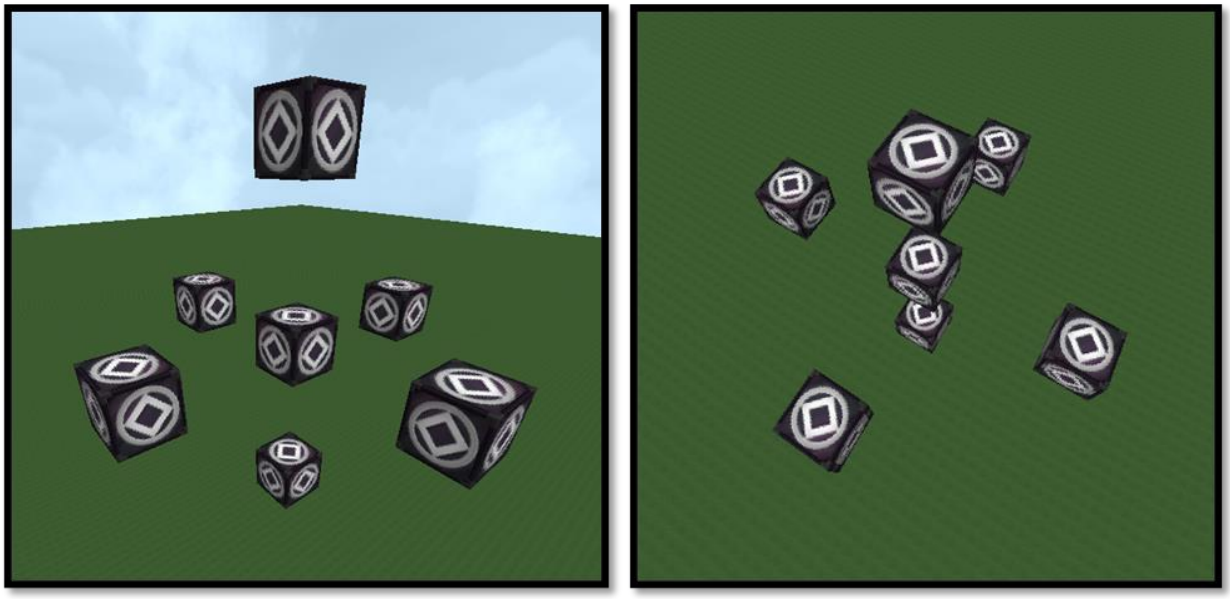

圖 56、57：三維坐標系梅花示意圖

# 捌、參考資料


1. Stephen Prata（2012）。**C++ Primer Plus 中文版**（第六版，蔡明志譯）。臺北市：碁峰。
2. 作者不詳(2016)。五子棋規則與開局。2020 年 12 月 4 日取自 http://chiuinan.github.io/game。
3. 作者不詳（2022）。臺北市 110 學年度國民小學一般智能資賦優異學生鑑定安置說明。2021 年 10 月 24 日取自 http://www.thps.tp.edu.tw。
4. 作者不詳（時間不詳）。五子棋術語。2021 年 10 月 24 日取自 http://587.renju.org.tw/teach/teach017.htm。
5. 陳鍾誠(2014)程式人雜誌。2021 年 10 月 12 日取自 http://programmermagazine.github.io。
6. 萌萌的皮卡丘（2019）。C++之簡單的五子棋 AI。2021 年 10 月 14 日取自 https://www.twblogs.net。
7. 游森棚（2021）。**普通高中數學 A 第三冊**。臺南市：翰林。
8. 蔡志敏（2020）。**Hello！C++程式設計**（第二版）。臺北市：碁峰。

## 【評語】190032

參賽者根據五子棋之規則發想有趣的梅花棋遊戲。唯此遊戲之公平性、可玩性皆還需要更多研究證明。目前遊戲規則可能導致玩家在不知情的情況下獲勝，失去遊戲原應有策略與娛樂性。此遊戲具有發展潛力，但仍需要更多研究驗證與改進玩法，期盼參賽者持續增進此遊戲之遊戲規則。