2023 年臺灣國際科學展覽會
優勝作品專輯

作品編號　190004

參展科別　電腦科學與資訊工程

作品名稱　針對梅花棋遊戲之人工智慧實作與分析

得獎獎項

就讀學校　臺北市立永春高級中學

指導教師　吳毅成、陳慶隆

作者姓名　張庭瑋、吳冠諺

關鍵詞　梅花棋軟體、演算法開發及優化、
人工智慧適用性分析

作者簡介

我是張庭瑋（右），目前就讀於臺北市立永春高中三年級數理資優班。「梅花棋」這個研究從最初的實體遊戲，到現在二度站上國際科展的舞台，在過程中我不斷創新及嘗試，發覺自己對於資訊領域的堅持，也發覺自己在人工智慧上的不足。此次挑戰現役最強演算法的架構，令我更加確信自己的愛好。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，也期望有朝一日能學有所成。

我是吳冠諺（左），就讀臺北市立永春高中數理資優班。高中開始接觸研究，發現了做研究的樂趣，不斷嘗試不斷失敗，但卻能從失敗中獲得更多的靈感、發現更多問題，我想這也是程式吸引我的地方。對同一個題目花兩年半的時間投入其中，我從這個題目中學習到不少東西，努力和堅持是我這兩年中最大的收穫，想做好一篇研究是辛苦的，但從中獲得的快樂只有自己知道，很開心能夠再度入選國際科展，希望未來能繼續專研資訊領域並進入職場回饋社會。

Abstract

This study aims to address the issues that previous studies hoped to address. In this study, two artificial intelligence algorithms are proposed for symmetric and asymmetric chess games. They are Minimax and Monte Carlo Tree Search, respectively. Among them, Minimax is divided into one, two and three layers of exploration depth, and MCTS is divided into 100, 300, 500, ..., more than 1900 versions according to the number of simulations. With the current results, we don't think its win rate is ideal. The main reason is that the results of all the current algorithms are too random, even if we optimize the UCB formula, although the win rate has improved, it still does not meet our expectations. In order to solve the above problems, we hope to fundamentally solve the problem of low operation efficiency, and the most obvious way is to complete the artificial intelligence training before the game runs, that is, to directly give a set of strategy running time. Game runs. The game starts, so the computer doesn't need to do additional game simulations. To sum up, we started to implement the framework of Tuple-Network, TD Learning and AlphaZero, but due to time constraints, the models have not been trained yet.

摘要

本研究旨在解決先前研究未解決的問題。而在本研究中對於對稱規則及非對稱規則的梅花棋遊戲，各提出兩大人工智慧演算法。分別是 Minimax 及 Monte Carlo Tree Search。而在這之中，Minimax 又被分為探索深度一層、兩層及三層、MCTS 則是以模擬次數分為 100、300、500、…、1900 多個版本。而以目前的成果來說，我們認為其勝率並不理想。而主要的原因還是要歸咎於目前所有演算法的結果過於隨機化，而即使我們對於 UCB 公式進行優化，雖然勝率有所提升但仍然不符合我們的期待。為了解決上述問題，我們希望從根本解決運行效率過低的問題，而最顯而易見的方法就是在遊戲運作前先將人工智慧訓練完畢，也就是在遊戲開始時直接給予一套策略，令電腦無須再做額外的遊戲模擬。綜上所述，我們開始實作 Tuple-Network、TD Learning 及 AlphaZero 的相關架構，但礙於時間關係，模型尚未被訓練。

壹、研究動機

在先前的研究中，我們對於梅花棋遊戲已經有了一些初步的成果，但是當時還是有需多問題諸如勝負判別函式的效率過低、提出 Victory Notion 的猜想並且沒有得到解決、人工智慧演算法的實力過於薄弱、Minimax 演算法無法增加探索層數...等。而我們希望在此基礎上，逐步解決以上問題，並開發完整的梅花棋遊戲軟體。

貳、研究目的

一、 改良勝負判別函式的效率問題。

二、 實作梅花棋遊戲的 Minimax 演算法。

三、 將 Alpha-Beta 剪枝法套用至 Minimax 演算法。

四、 實作梅花棋遊戲的 Monte Carlo Tree Search 演算法。

五、 實作梅花棋遊戲的 Tuple-Network 演算法。

六、 實作通用演算法「AlphaZero」並完成模型訓練。

七、 分析並比較各人工智慧演算法對於梅花棋遊戲的棋力差距。

參、研究回顧（張庭瑋、吳冠諺，2022 年）

在先前的研究中，推導出沒有 BUG 且效率來到$O(n^4)$的向量演算法（網狀編碼演算法被發現其複雜度為$O(n^4)$），並利用 C++實作出棋盤大小為19×19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，因此當時尚未將軟體的使用者介面製作出來，這成為該研究的遺憾。

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 5：

表 5：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，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所示：

版本	one	two
one	25	21
two	13	17

版本	one	two
one	68%	81%
two	131%	100%

圖：類人類思考程度計算結果

肆、研究設備及器材

一、硬體

（一） 桌機及筆電×42（同時運作的最大數量）

（二） 黑色棋子×191。

（三） 白色棋子×190。

（四） 19×19 方格棋盤。

二、軟體及其他工具

（一） 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。

（二） 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。

（三） 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。

（四） Python：

程式語言，使用版本為 Python 3，是一種廣泛使用的直譯式、進階和通用的程式語言。其支援多種程式設計範式，包括函數式、指令式、結構化、物件導向和反射式程式。它擁有動態型別系統和垃圾回收功能，能夠自動管理記憶體使用，並且其本身擁有一個巨大而廣泛的標準庫。

（五） Pygame：

Pygame 是跨平台的 Python 模組，專為電子遊戲設計。包含圖像、聲音。建立在 SDL 基礎上，允許即時電子遊戲研發而無需被低階語言，如 C 語言或是更低階的組合語言束縛。基於這樣一個設想，所有需要的遊戲功能和理念都完全簡化於遊戲邏輯本身，所有的資源結構都可以由高階語言提供，如 Python。

伍、研究過程

一、研究流程架構

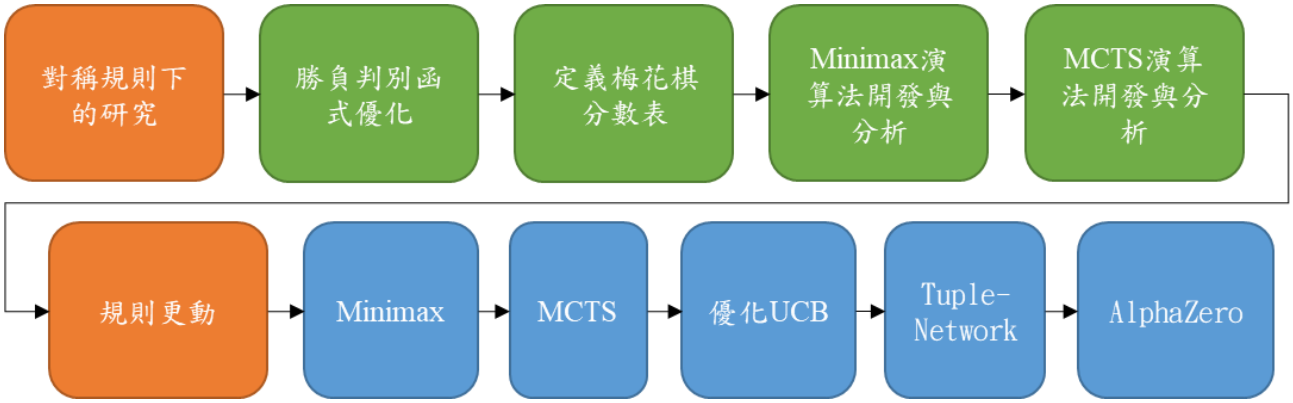

圖 1：研究流程圖

二、研究過程

本研究基於「The Goclub-梅花棋演算法效率及適用性分析」之成果，進行延伸研究。後續將依照研究時間軸分為兩大階段進行呈現，其分別為梅花棋遊戲的原始規則及後續改良版本。而原先的梅花棋遊戲規則符合對稱賽局的定義，因此我們稱之為對稱規則，而我們後續將此規則詳細定義如下：

1. 使用大小為 9 路的方格棋盤。
2. 先手執黑子後手執白子，雙方依序下子於棋盤交點。
3. 任意一方玩家形成梅花即為獲勝。

（一）梅花棋遊戲對稱規則下的研究

1. 勝負判別函式的改良

在先前的研究中，我們分別提出了平均演算法、畢氏定理演算法、綜合演算法、向量演算法及網狀編碼演算法等 5 個不同的判別方式，並證明各自的可行性。而經過驗證，當時選用的網狀編碼演算法效率並不高。當初的作法可以理解為：「在一個n階棋盤上，枚舉出所有梅花的型態，並對其編碼。在運行時就個別判斷所有不同編碼的梅花是否形成，即可得知遊戲是否結束。」因此複雜度即編碼總數，而經過計算，若以棋盤的邊長作為單位計算，則可得複雜度約為$O(n^4)$，這顯然不是最理想的演算法。

我們回頭改良向量演算法。在新的向量演算法中仍然將棋盤座標化，但為求函式的運行效率，利用一維陣列存取棋盤狀態。並在下棋時，分別將雙方玩家棋子的座標儲存於不同的兩個串列，而每次我們都只需要辨別該回合某方玩家的落子是否達成勝利條件。而判別方法如下圖 2 所示：

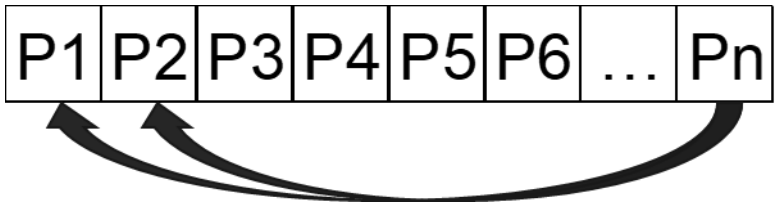

圖 2：梅花棋勝負判別流程示意圖

在上圖中，我們假設P_n為該回合某玩家的落子位置，並與P_1進行判斷。首先以P_1作為花蕊P_n作為其中一個花瓣，利用向量平移及向量內積的性質找出其餘三處花瓣的位置，判斷此五處是否形成梅花，而這樣的過程稱為「判斷$\overrightarrow{P_nP_1}$是否形成梅花」。接著以同樣的方式判別$\overrightarrow{P_1P_n}$是否形成梅花。以此類推判斷$\overrightarrow{P_nP_2}$、$\overrightarrow{P_2P_n}$…$\overrightarrow{P_nP_{n-1}}$、$\overrightarrow{P_{n-1}P_n}$是否形成梅花，以上即完成某一回合的完整判斷，複雜度約為$O(n)$。

2. Minimax 演算法

Minimax 演算法又稱為極小化極大演算法（維基百科編者，2022 年 5 月 22 日），是一種零總和演算法。常被應用於棋類等雙方資訊皆公開的零和遊戲（維基百科編者，2022 年 7 月 8 日），而零和遊戲則是指非合作賽局，也就是遊戲開始時所有玩家的分數皆為零或一個常數，而若任一玩家獲得一定的利益，則其餘玩家必有損失，使得其分數總合仍為零或某個常數，例如井字棋、五子棋等遊戲。而梅花棋也符合零和遊戲的定義，因此 Minimax 演算法在理論上適用，但實際上卻可能受某些因素影響，譬如 Minimax 演算法的複雜度是指數級的，所以在某些複雜度高的遊戲中不被廣泛應用。以下為 Minimax 演算法的實際流程：

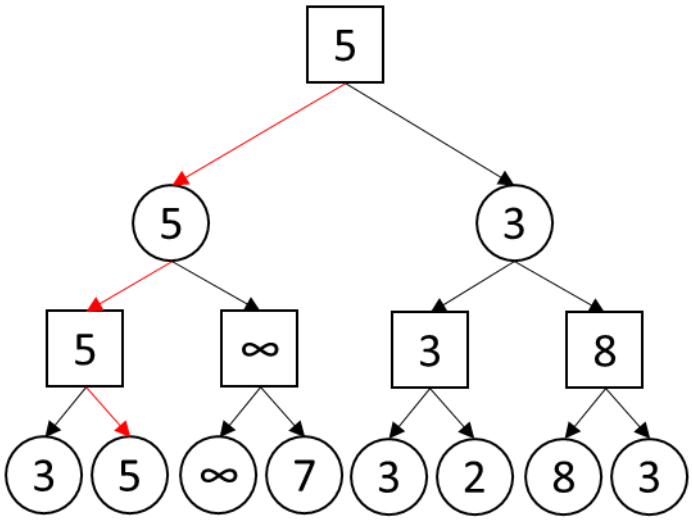

圖 3：Minimax 演算法示意圖

以上圖為例，假設方形是我方的回合，則我方會選取利益最大的方法作為該回合的選擇，也就是其子節點中數字最大的，而圓形則是代表輪到對手的回合，在此我們需要假設對手極度聰明，也就是會選擇對他自己最有利的情況，那麼他會盡可能使我方的分數有最小值。以此類推可以得到該回合的最佳解應是紅色這條路線。

3. Alpha-Beta 剪枝法

Alpha-Beta 剪枝法是一種用於減少搜索樹分支的方法，其被廣泛應用於 Minimax 演算法上。主要原理在於將搜索資源用於有意義的分支，也就是判斷節點是否存在計算的意義，以下為 Alpha-Beta 剪枝法的實際流程：

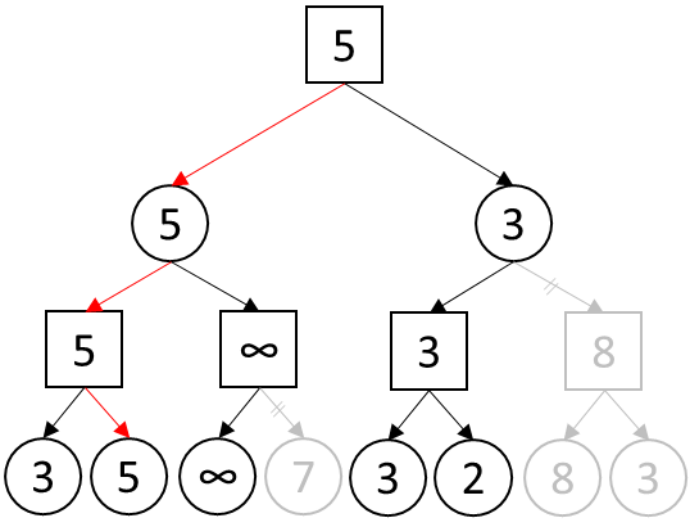

圖 4：Alpha-Beta 剪枝法示意圖

以上圖為例，在這棵二叉樹中以 Alpha-Beta 剪枝法優化過後，其灰色區域為不需要計算的節點。首先看到分數為 7 的葉節點，祖父節點要取其子節點的最小值，而因為葉節點的兄弟節點分數為無限大，回傳後大於葉節點其父節點的兄弟節點，又葉節點的父節點要取最大值，因此即便葉節點的分數比兄弟節點大，也不會被祖父節點選取，所以此節點後續的所有情形都可以被忽略。而後面的灰色區域也可以用相同的概念來省略計算。綜上所述，再加入 Alpha-Beta 剪枝法後的 Minimax 演算法可以節省不少時間。

4. 以 Alpha-Beta 剪枝法實現 Minimax 演算法實作與分析

在本次研究中，我們重新定義梅花棋各狀態下的分數表，並將新評估函式改為以新分數表為基準做總和。以下為新的分數表：

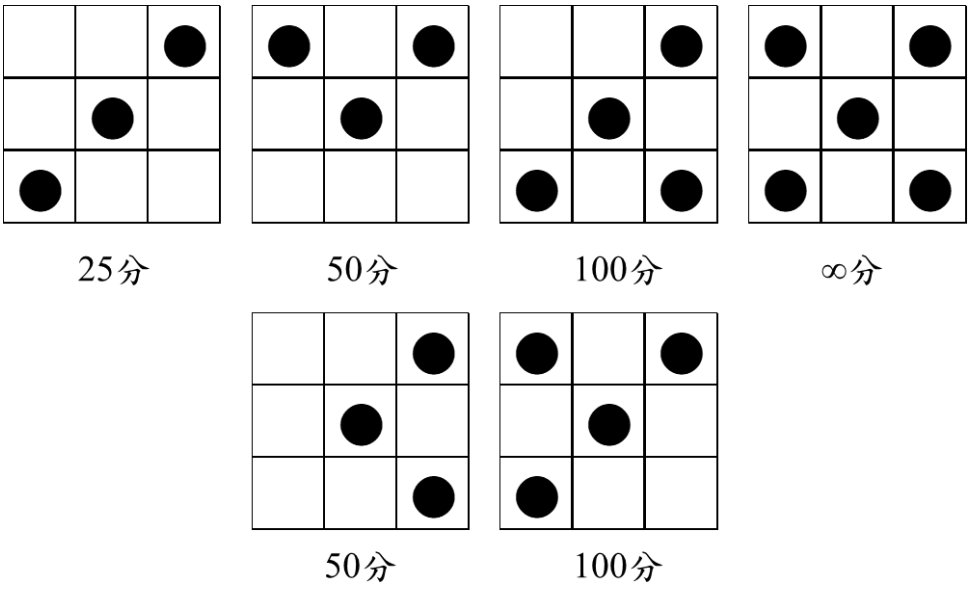

圖 5：新分數表示意圖

完成定義後，我們希望透過勝率對不同探索深度的演算法進行比較。為此我們首先讓先後手都分別使用隨機的方式進行遊戲模擬。我們總共模擬了 1000 場遊戲，透過下圖可以觀察到其勝率大約會收斂至 55%左右。

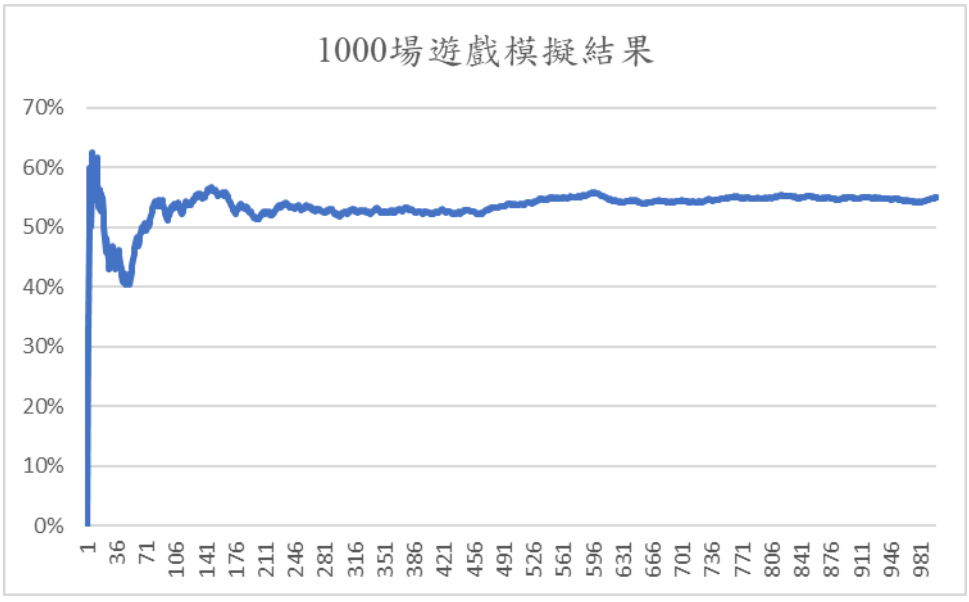

圖 6：1000 場隨機遊戲模擬結果

之後我們以同樣的作法對不同的探索層數進行比較。也就先手方分別持有樹高一層、兩層、三層...等不同的探索層數，而後手玩家則是統一以隨機的方式進行遊戲。同樣都以 1000 場為標準可以得到勝率分別會趨近於 96%、63%及 87%（這裡的結果僅顯示三層以內的數據，詳細原因稍後會說明）。這樣的結果無疑是意外的，普遍來說探索層數越高代表考慮的可能性越多，勝率也就應該越高，觀察後會發現探索層數為兩層的勝率異常的低，這顯然不符合直覺。以下是先手分別持有探索層數為一層、兩層、三層時勝率的收斂圖：

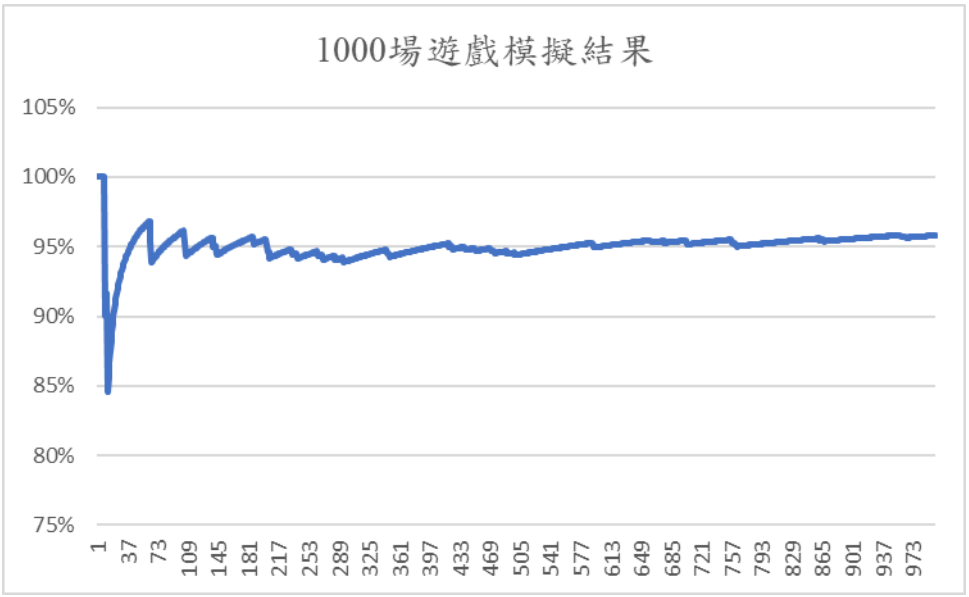

圖 7：1000 場探索層數為一層的模擬結果

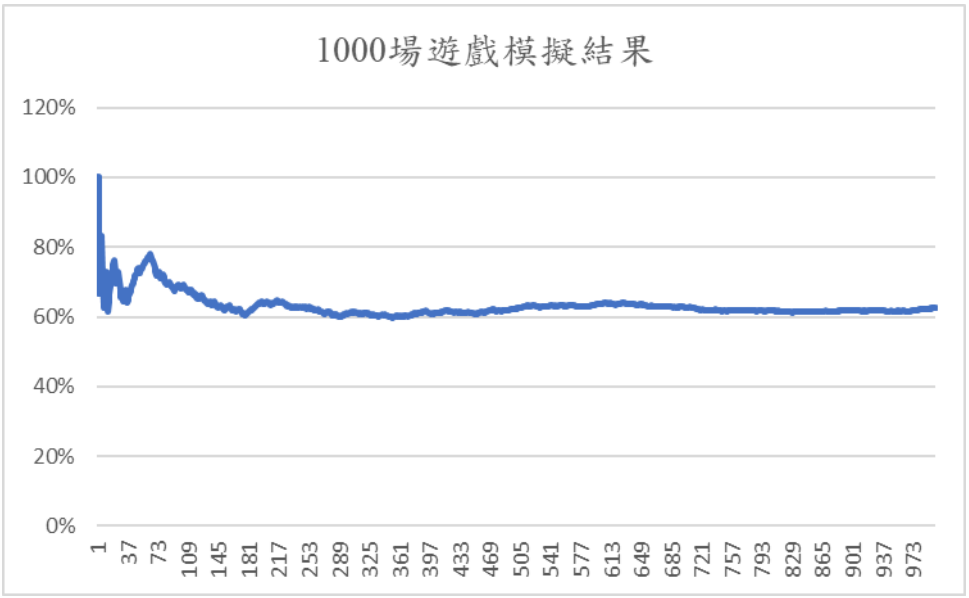

圖 8：1000 場探索層數為兩層的模擬結果

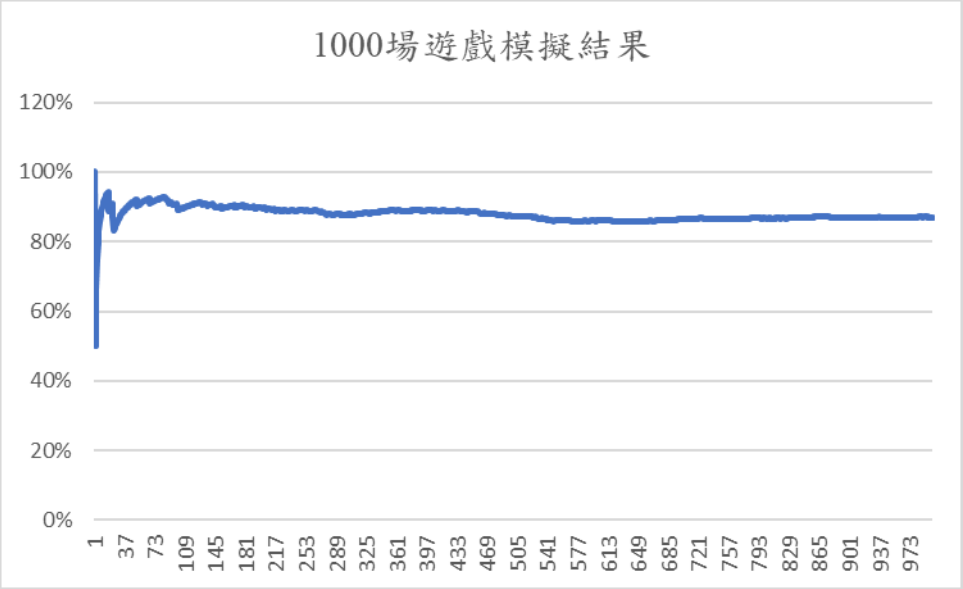

圖 9：1000 場探索層數為三層的模擬結果

經過討論我們認為造成這樣的原因可能有以下幾點：目前探索層數對梅花棋來說過低，導致資料無參考價值。偶數層的葉節點為敵方玩家的回合，導致發生一些無法預計的原因，從而影響電腦的判斷。目前的分數表定義方式仍有改善空間。因此日後我們會再對以上幾點進行分析。

而先前提及的層數問題，也就是資料僅有探索層數前三層的問題，則是因為 Minimax 演算法的複雜度是指數級成長的，因此到了四層時，運算的時間過於冗長，加上 1000 場的基準量導致我們尚無法給出模擬的結果。

5. Monte Carlo Tree Search（蒙特卡羅樹搜尋）

Monte Carlo Tree Search 又稱為蒙特卡羅樹搜尋（以下簡稱 MCTS），是一種用於進行決策的啟發式搜尋演算法（克里斯托弗·Z.穆尼，2018 年）。。他是一種基於隨機模擬並抽樣的決策演算法，其被廣泛應用在遊戲中，例如圍棋、即時電子遊戲、不確定性遊戲等。

MCTS 會不斷的模擬棋局並將結果回傳以更新節點上的數據，直到完成指定的模擬次數為止。而每一次的模擬都包含四個階段，分別是 Selection（選擇）、Expansion（擴充或展開）、Simulation（模擬或仿真）、Backpropagaion（反向傳播）。以下是四個階段的示意圖：

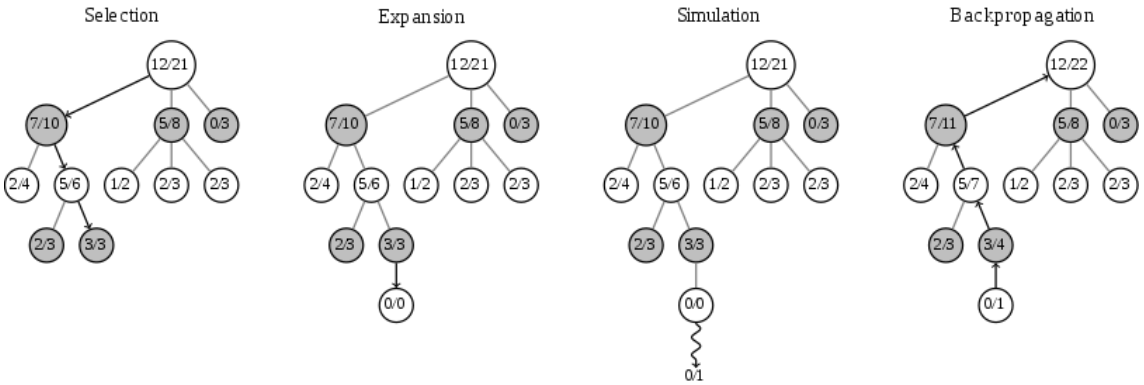

圖 10：MCTS 四個階段示意圖

以上圖為例，第一個階段是 Selection：從根節點向下選擇子節點，直至葉節點為止。選擇子節點的方式有很多種，本研究以最常見的 UCB 公式作為依據。接著是 Expansion：若任意一方玩家在葉節點獲勝使得遊戲結束，則不進行此步驟，否則將葉節點後續的所有可能作為子節點，並選取其中一個節點。再來是 Simulation：從 Expansion 選擇的節點開始，以隨機策略進行遊戲直到結束（即某玩家獲勝或平手），此步驟又被稱為 Playout 或者是 Rollout。最後是 Backpropagation：將 Simulation 的遊戲結果，一路向上更新到根節點。

而所謂的 UCB 公式，其全名為 Upper Confidence Bound（上限信賴區間演算法），它是由兩個構造組合而成。分別是遊戲勝率及探索率，而要形成這兩個構造需要四個參數，w_i：第i次模擬時，該節點當下的總獲勝次數、n_i：第i次模擬時，該節點的總模擬次數、N：總模擬次數、C：探索參數，是一個常數用以平衡勝率與探索率，因此若此參數值越大，則 Selection 時選取的點會越以探索率低的節點優先，反之則

以勝率高為優先。其公式為：

$$UCB = \frac{w_i}{n_i} + C\sqrt{\frac{\log(N)}{n_i}}$$

6. MCTS 實作與分析

在 MCTS 中，我們仍然透過勝率的分析，對不同總模擬次數的演算法進行比較。底下藍色區域表示先手玩家持有的演算法版本，而橘色部分則是後手玩家所持有的。首先看到紅色區域，可以發現各模擬次數間的勝率關係有著如同 Minimax 演算法的問題，而不太一樣的是所有勝率都非常低，甚至比隨機模式還要誇張。

	Random	100	300	500	700	900	1100	1300	1500	1700	1900
100	61%		100%	67%	99%	100%	100%	100%	100%	100%	98%
300	50%	94%		98%	95%	90%	98%	98%	96%	94%	98%
500	41%	86%	98%		98%	100%	98%	98%	94%	100%	92%
700	31%	92%	96%	94%		94%	96%	96%	98%	94%	96%
900	43%	94%	98%	96%	100%		98%	96%	92%	93%	95%
1100	48%	92%	55%	96%	98%	94%		100%	96%	96%	98%
1300	53%	94%	92%	43%	96%	98%	90%		86%	98%	98%
1500	51%	88%	96%	96%	100%	96%	88%	96%		94%	98%
1700	41%	86%	98%	98%	98%	98%	86%	98%	96%		98%
1900	42%	90%	92%	94%	100%	92%	96%	96%	98%	96%	

圖 11：MCTS 各版本間的遊戲模擬結果

而後面的其他資料，除去綠色區域莫名的低以外，剩下都高到離譜。我們並不瞭解出現這種結果的具體原因。而在先前的研究中曾提出關於 Victory Notion 的猜想，也就是梅花棋這款遊戲似乎在對稱規則下存在必勝法。在本次研究中，我們發現梅花棋遊戲符合 Strategy-stealing argument 的相關定義(Wikipedia contributors. April 5, 2022)，進一步證明了先手方必不敗(此處是基於雙方玩家都極度聰明的情況下)，也就是先手具有絕對的優勢。因此我們懷疑是模擬次數不同的演算法間棋力相差無幾，加上先手優勢過大，才會導致勝率如此驚人。

（二）梅花棋遊戲非對稱規則下的研究

在非對稱規則中，我們新增一條限制。玩家欲形成梅花時，不得將花蕊作為最後落子，也就是不能將下圖中紅色的棋子作為最後的落子，而這樣一來也就不符合 Strategy-stealing argument，因此也就解決了公平性的問題。

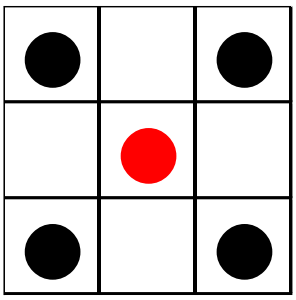

圖 12：非對稱規則特徵示意圖

1. Minimax 演算法實作與分析

我們同樣希望透過勝率對不同探索深度的 Minimax 演算法進行比較。為此首先讓先後手都分別使用隨機的方式進行遊戲模擬。在模擬了 1000 場遊戲後，可以發現到勝率大約會收斂至 54%左右。之後我們也各別以同樣的作法對不同探索層樹的 Minimax 演算法進行遊戲的模擬，最後得到其勝率的收斂值分別為 97%、61%、89%，以下圖所示。而這樣的結果和對稱規則極為相似，因此我們初步排除受先手優勢影響的可能。

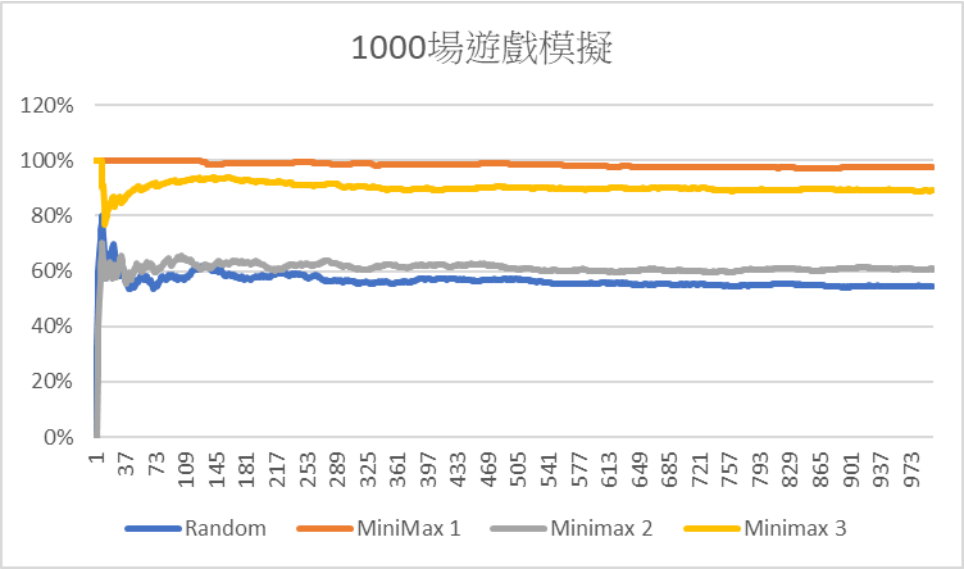

圖 13：1000 場遊戲模擬的勝率收斂圖

2. MCTS 演算法實作與分析

在非對稱規則下的 MCTS 中，仍然透過勝率的分析，對不同總模擬次數的演算法進行比較。下圖藍色區域表示先手玩家持有的演算法版本，而橘色部分則是後手玩家所持有的。看到綠色區域，可以發現各模擬次數間的勝率關係有著類似於前面所有版本的問題，但其中的勝率仍然非常低，甚至依然比隨機模式還要誇張。

	Random
100	66%
300	41%
500	48%
700	42%
900	49%
1100	53%
1300	39%
1500	45%
1700	36%
1900	47%

圖 14：MCTS 各版本間的遊戲模擬結果

3. UCB 公式的改良與分析

儘管結果和想像中並不相同，並且尚無解決方法，但是我們仍然希望精進 MCTS 演算法。而我們首先想到的即是 UCB 公式，可以想像一下目前的 MCTS 中，每一次的模擬中，Selection 都是對 UCB 公式取最大值，也就是選取勝率最大的子節點。這樣對先手方來說非常合理，但對於後手方則會有些邏輯上的瑕疵，也就是演算法並沒有假設後手玩家是最聰明的，因為對先手方勝率高的子節點對於後手玩家來說並不高。

綜上所述，我們將 UCB 公式重新定義為三種不同的情況。首先我們假設M為該節點的勝率，C為探索率，則有原始$UCB = M + C$，此為第一種情況，也就是原始的狀況。第二種情況是針對後手玩家的 UCB 公式作改良，計算後可以發現若某節點先手玩家的勝率為P則後手玩家的勝率$P' = 1 - P - 平手的機率$，而平手的機率趨近於零，因此定義$UCB = (1 - M) + C$。最後一種情況則是直接將 UCB 值以亂數表示。依照這三種不同的公式，可以得到新的演算法勝率趨近於下圖顯示：

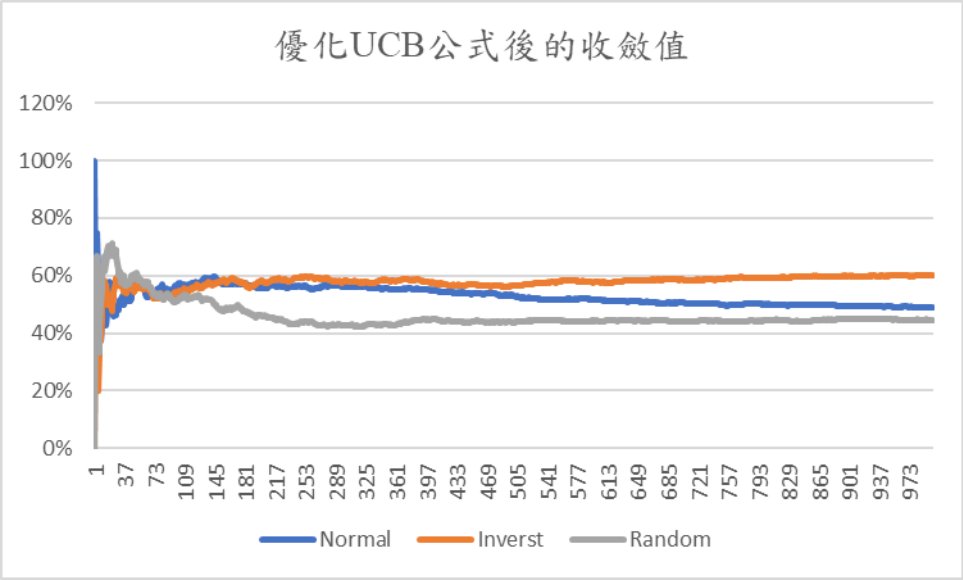

圖 14：優化 UCB 公式後遊戲模擬的勝率收斂圖

而從圖中可以很清楚的發現，不同的 UCB 公式勝率確實有差異。因此得到結果

為$Inverst > Normal > Random$，詳細觀察實際數據後可以得到下列的勝率表格：

Inverst	Normal	Random
60	49	44

圖 16：各 UCB 公式下的實際收斂勝率

4. GoClub 遊戲封裝與 UI 設計

目前我們基於現有的所有人工智慧，設計出一套含有人性化界面的應用程式。利用 Pygame 繪製簡單的圓形於棋盤上，並將滑鼠點擊時的座標誤差進行高斯運算，進而取得準確的落子位置。從而使的遊戲順利運行，以下是遊戲進行畫面：

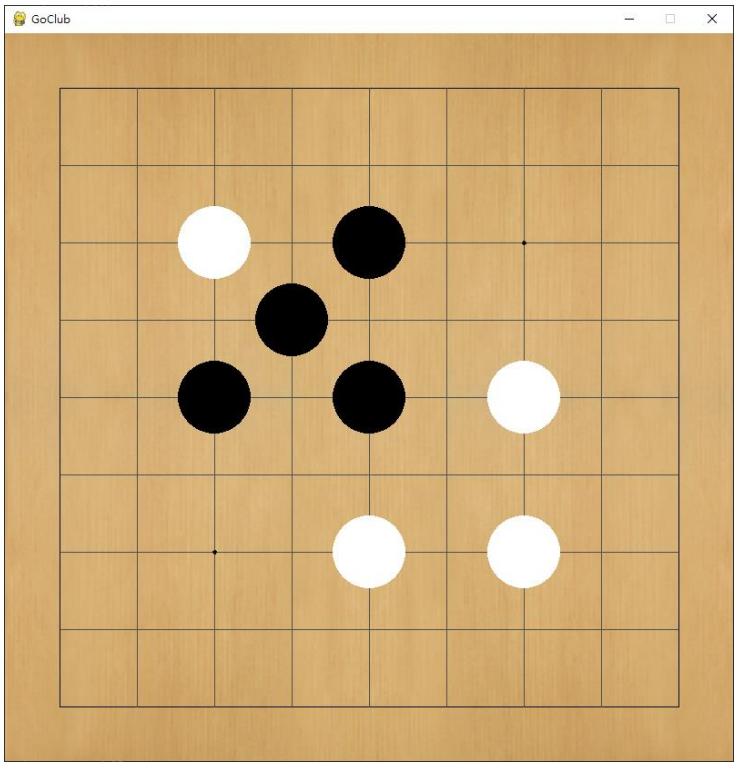

圖 16：遊戲實際運行畫面

以目前的成果來說，其功能還非常簡陋。日後我們希望在棋盤右方新增資訊頁，屆時將可以在面板上輸出各種關於當下棋局的資訊，甚至可以提供新手一些落子的建議。另外也希望設計主畫面。在主畫面中我們希望可以提供玩家在程式內設定各

種參數及遊玩人數等，例如人工智慧的版本。

5. Tuple-Network

這裡我們使用 2048 這款常見的遊戲舉例（Chu-Ling Ko，2021 年 1 月 1 日），遊戲開始時會隨機生成兩個數值為2^1或2^2的方塊，每一回合會有兩個階段，首先玩家可以選擇往上、下、左、右其中一個方向移動；接著電腦會隨機在某個空位置生成數值為2^1或2^2的方塊，當兩個數字相同數值的方塊2^k合在一起時，則會被更新成一個2^{k+1}的方塊，同時獲得2^{k+1}的分數，如此重複上述動作，直到所有位置被填滿且無法繼續得分。

而這款遊戲的目的顯然是盡可能拿得高分，但是若只考慮未來 1 步的情況是不夠的，舉例來說，下圖中往左或往右移動都能透過合併 2+2 和 8+8 四塊方塊來獲得 20 分而往上或往下僅 2+2 總計 4 分，比較之下會認為當下選擇往左或往右是最佳的，但是對於整個盤面的未來而言，往下滑是前途最好的，在未來能獲得更多的分數、存活更久，而盤面前途的定義正是我們需要的，也就是能幫助估算盤面潛力值的函數：「評估函數」。

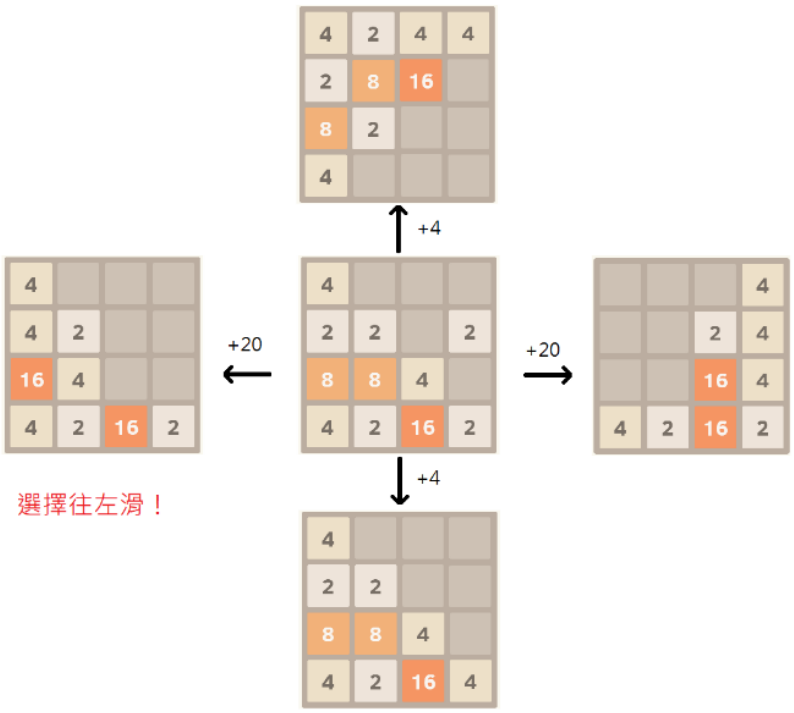

圖 17：未來一步內分數分析示意圖

所謂的評估函數則是指「以某個盤面計算其未來能獲得的分數」。一但取得這個函數，我們就能讓 Agent 更聰明地選擇方向了，那就是永遠選擇「該次滑動賺的分數」加上「滑完的盤面預計還能賺的分數」最高的滑動方向。

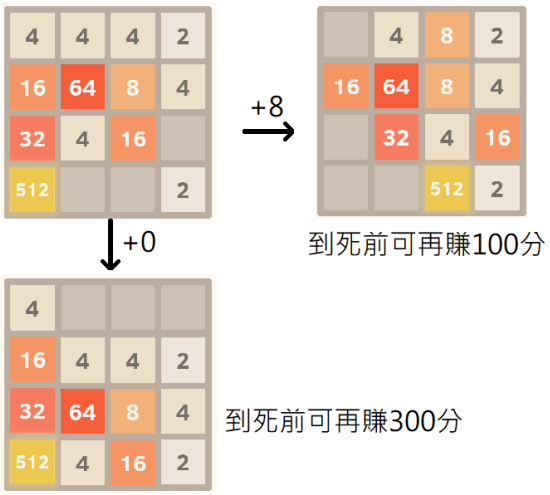

圖 18：評估函數示意圖

以上圖為例，往右滑則當回合可以獲得 8 分並且評估未來還能再賺 100 分才結

束遊戲，因此往右滑能獲得到的分數總共是 108 分；而往下滑當回合可以獲得 0 分，並且評估能再賺 300 分才結束遊戲，因此往下滑能獲得到的分數總共是 300 分，所以 Agent 應該選擇向下滑。較特別的是，這樣的策略完全依賴評估「前途」的函數，函數回傳的數值愈準確，Agent 就會愈強大。而若想要精準的評估盤面的「前途」，最合理的方法就是直接看盤面在實際的遊戲中的未來究竟賺了幾分。

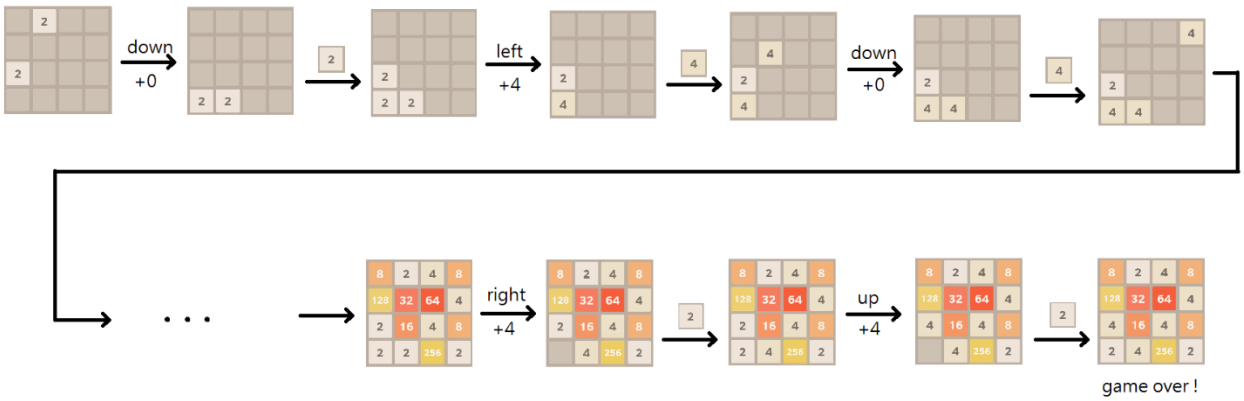

圖 19：評估函數的計算

以上圖為例，假設此圖為某局盤面從開始到結束的過程，而遊戲結束後總共賺了 987 分。有了這筆數據，我們就能回推整局遊戲中，不同的盤面「前途」分別是多少。較特殊的是函數只在乎「滑動後」的盤面有多少潛力，因為該回合往什麼方向滑是我們可以決定的，而系統會在哪裡跳出新的磚塊則是隨機的。

因此我們只要重複遊玩遊戲，並蒐集所有盤面的前途數值，即可幫助預估每個盤面的潛力值。而數據愈來愈多 Agent 預估的也會愈來愈精準，所以評估函數的結果也會愈來愈有參考價值。不過若有相同的盤面在不同場遊戲中重複出現，則可能得到不一樣的最終分數，以下圖為例：

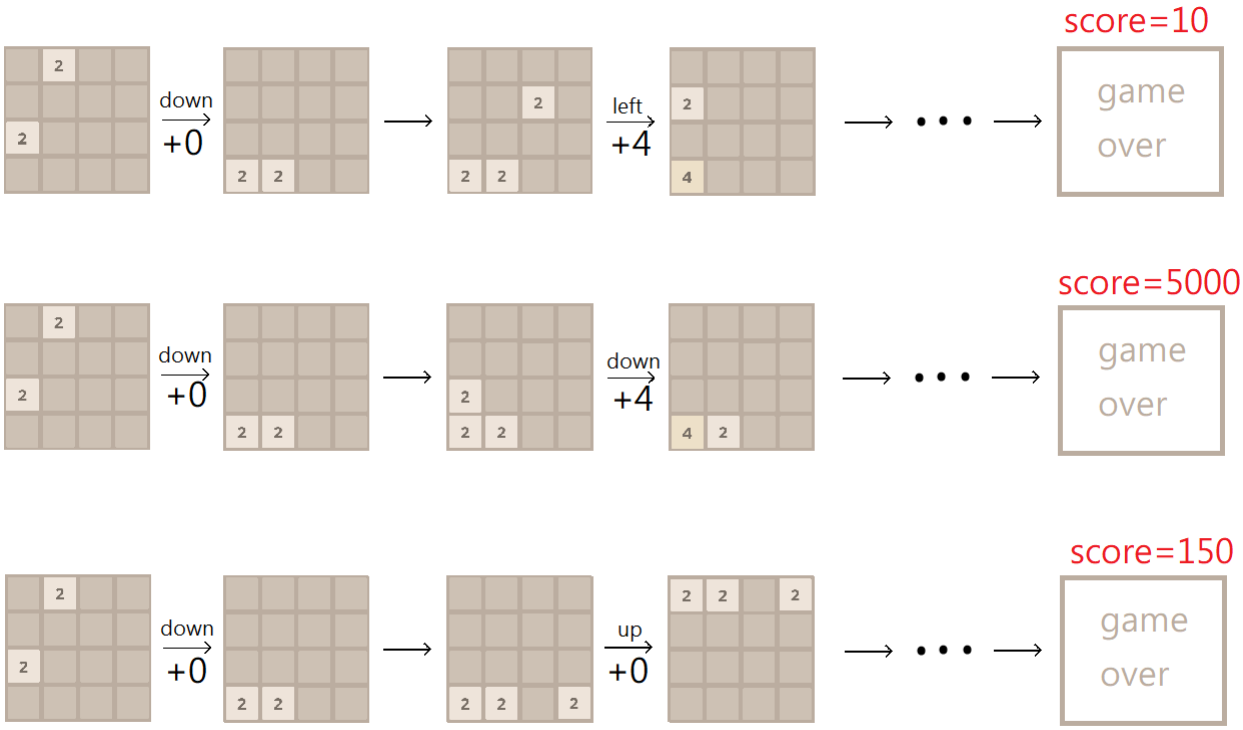

圖 20：同盤面不同結果示意圖

此時我們首先想到的解決方法就是：將三個數據取平均值作為該盤面的評估數值。這樣的方法看似合理，實則有個極嚴重的缺陷，例如上圖 10 分或是 5000 分這種極端值會主導平均值；再者，因為 Agent 會不斷更新評估函數的資料，因此愈舊的遊戲數據愈不具有參考價值，不應該等比例平均。

而解決辦法如下：首先設定一個 Learning Rate，然後獲得新的數值時，就把新舊的數值差乘上 Learning Rate 並加上舊的數值。然而 Learning Rate 的值一般定義在 0 和 1 之間，但實際應該使用多少，則會根據遊戲種類的不同而改變。

而現在假設某盤面在眾多棋局中出現三次，那我們有以下幾種方法更新盤面估計值：Monte Corlo Tree Search（簡稱 MCTS）、Temporal difference learning（簡稱 TD learning）、N-tuple network 等。那麼根據前面的假設，我們制定以下的訓練步驟：

1. 把所有盤面的潛力值設為 0。

2. 玩一場遊戲。

3. 用該場遊戲的結果跟新潛力值。

4. 用新潛力值再玩一次。

不斷重複上述第三及第四步驟直到潛力值收斂。觀察並比較後可以發現，蒙特卡羅法每次都是使用「真實」的數據，但礙於隨機選擇，所以結果非常浮動；而 TD learning 則相對穩定，但是其全然相信「傳來」的數據。

TD learning，是 reinforcement learning 的種類之一，在訓練 model 時，不會等到事件發生時才改變 model，而會在事件發生前調整預測，使結果不會過度隨機化。舉例來說，以往的考試 A 學生都考了$x = \{x|80 \le x \le 85, x \in N\}$ 分，結果 model 預測他將考 30 分，但是事實上應該猜測 80 分左右的分數較為合理。也就是 model 預測錯了，那此時就可以調整 model，而不用等到確切結果揭曉。為了更好理解我們簡化遊戲來舉例，下圖這場遊戲只有 4 次滑動就結束，總共賺了$2 + 0 + 4 + 2 = 8$分，而這裡我們只在意移動後的分數。

圖 21：2048 遊戲示意圖

以下的更新中，我們使用一樣使用 learning rate$= 0.1$。首先，對於 D 盤面來說，它是這場遊戲的最後一個盤面，沒有辦法繼續賺取分數，因此可能獲得的分數就是 0 分。更新$V(D) \leftarrow 0 + 0.1 \times (0 - 0) = 0$

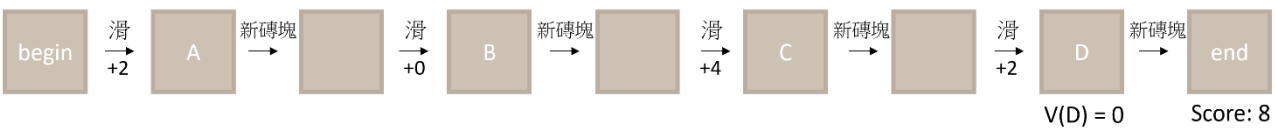

圖 22：2048 遊戲示意圖

對於 C 盤面來說，它在這場遊戲中賺了 2 分後成為 D 盤面，因此它的可能得到的分數是$2 + V(D)$。這裡不管最後真正拿了多少分，model 都全然相信$V(D)$會是對的，又以目前的$V(D)$值來說，$2 + V(D) = 2$。更新$V(C) \leftarrow 0 + 0.1 \times (2 - 0) = 0.2$。依此類推可得到：

$$V(B) \leftarrow 0 + 0.1 \times (4.2 - 0) = 0.42$$

$$V(A) \leftarrow 0 + 0.1 \times (0.42 - 0) = 0.042$$

這裡雖然拿到八分，但是$V(A)$卻僅 0.042 分，而這是因為模擬的場次不夠。只要模擬的場次夠多分數自然就會收斂到穩定又可信的值。我們認為 Monte Corlo Tree Search 和 TD learning 都可以嘗試在梅花棋遊戲上實做，如果想結合兩種方法的特色，則可以使用 TD$-\lambda$ learning。其中有一個參數 λ 來決定「考慮多少比例的實際賺分結果」，而蒙地卡羅法其實就是 TD-1，完全只考慮「實際賺分結果」；而剛才介紹的 TD learning 其實就是 TD-0，完全只考慮「下一個狀態」。

以下圖為例，這場遊戲總共賺了$a + b + c + d$分，在 TD-1（蒙地卡羅法）中，A 盤面的潛力估計值即為$b + c + d$；在 TD-0中，A 盤面的潛力估計值即為$b + V(B)$；而在λ介於 0 到 1 之間的 TD$-\lambda$中，所有組合都會是目標，而所有值都會跟據λ為比例組合起來，成為 A 盤面的潛力估計。以下表規則進行操作

表 1：$target(A)$各項紀錄表

項	係數
$b + V(B)$	$1 - \lambda$
$b + c + V(C)$	$\lambda \times (1 - \lambda)$
$b + c + d + V(D)$	$\lambda \times \lambda \times (1 - \lambda)$
$b + c + d$	$\lambda \times \lambda \times \lambda$

$$target(A) \leftarrow (1-\lambda) \times \big(b + V(B)\big)$$
$$+\big(\lambda \times (1-\lambda)\big) \times \big(b + c + V(C)\big)$$
$$+\big(\lambda \times \lambda \times (1-\lambda)\big) \times \big(b + c + d + V(D)\big)$$
$$+(\lambda \times \lambda \times \lambda) \times (b + c + d)$$

雖然上述演算法在梅花棋中看起來都是可行的，但是梅花棋的棋盤實在太大了，可能性太多，要訓練好一個評估函數需要花費過多的時間。不過訓練 AI 的精隨就是見微知著，並非看過一模一樣的圖才知道那是什麼物品。因此對於 2048 遊戲和梅花棋的盤面來說，我們希望能藉由學習特徵來讓相似的盤面能夠被預測出類似的潛力值，這時就需要用到 N-tuple-network，我們可以選取一塊區域作為盤面的「feature」，而計算估計值時也只針對盤面上的那一小塊 feature 做操作。

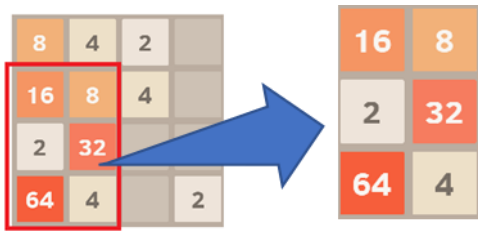

圖 23：2048 遊戲中特徵示意圖

而同樣的位置應該要在 4 個不同旋轉方向×2 種鏡像總共 8 種 Isomorphism 中各取一次。以 A1 圖來舉例，「左下角六塊」的 feature 應該要取以下 8 次，並把 8 個估計值加總起來，成為 A1 這個盤面真正的估計值。

圖 24：特徵的旋轉示意圖

由於盤面前途的估計值是所有 feature 的對應值的加總，所以更新盤面估計值時也要把更新的量平均分給每個 feature。舉例來說，假設我們設計的演算法是 4 個 feature，再乘上 8 個 isomorphism 就有 32 個 feature。若目前某個盤面的 32 個 feature 對應值加起來是 100，代表這個盤面目前的潛力估計值是 100，並且這個盤面的更新目標為 200，這個差距是+100，再乘以 Learning Rate= 0.1，得知這個盤面的潛力估計值估計值要再加10。因此，這個盤面上的 32 個 feature 所對應的值通通要一視同仁地加上$\frac{10}{32} = 0.3125$。如此一來，經過不斷的訓練後，相似有前途盤面所共有的 featur 就會擁有很大的值；相似的遊戲要結束盤面所共有的 feature 就會有很小的值，甚至是負數。

6. AlphaZero 模型

我們根據現役最強棋類演算法 AlphaZero(又稱為通用演算法)為架構，以 Python 構築出其通用架構，並將梅花棋遊戲的相關規則套用其中，期望能獲得前所未有的勝率。AlphaZero（以下稱為本算法）主要由三大演算法構築而成，分別是深度學習、強化式學習及賽局樹演算法（布留川英一，2021 年 1 月）。本算法巧妙利用此三種演算法各自的特點，諸如深度學習預測最佳棋步的直覺、強化式學習藉由自我對弈獲得的經驗以及賽局樹演算法的預知能力，將之結合形成通用演算法。

但礙於原始 AlphaZero 的訓練環境包含 5000 個 TPU 的運算資源，這是一般家用電腦無法企及的，因此我們將神經網路逕行適度的簡化，以縮短訓練時間。我們首先設計強化式學習循環，本算法是由對偶網路、自我對弈模組、訓練模組及評估模組等 4 個部分組成。其流程圖如下所示：

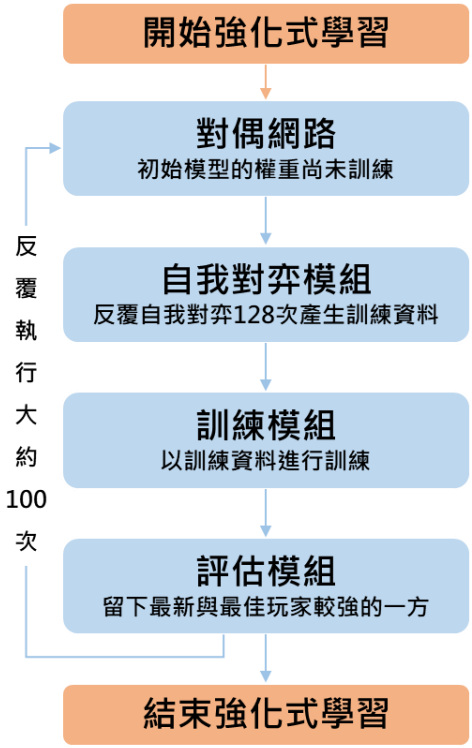

圖 25：強化式學習循環流程圖

接著我們實作以上 4 個模塊，分別封裝為各自的類別並測試無誤後，再組合完成整體演算法。首先是對偶網路（Dual Network）：是二合一的網路模型，也就是可以同時預測策略和局勢價值的神經網路（Neural Network），其利用當前盤面作為輸入，然後同時輸出策略及局勢價值。然而實際上對偶網路是以殘差網路（ResNet）為基礎構成的，也就是其中包含大量的殘差塊及卷積塊，作為盤面特徵的提取。其網路架構如下圖所示：

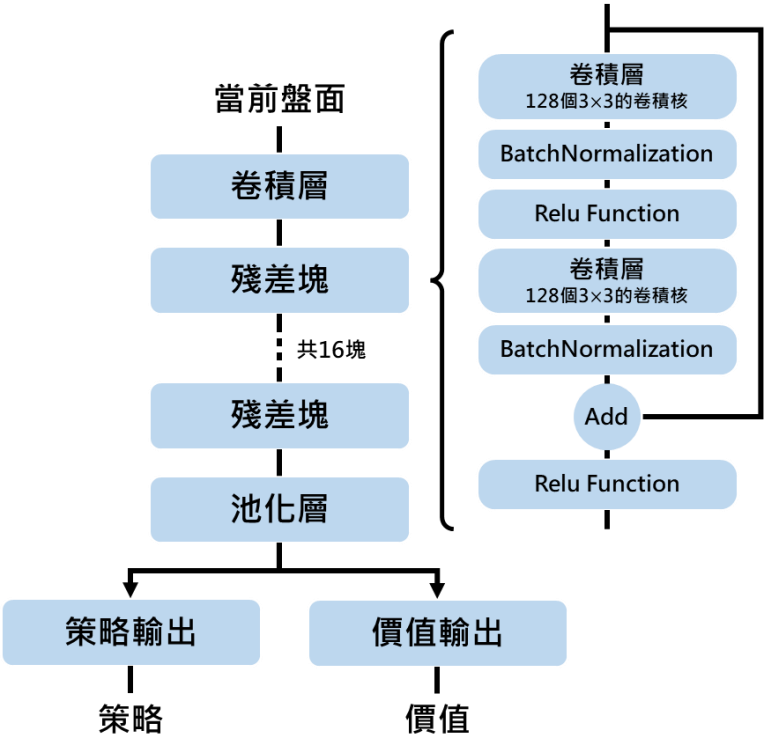

圖 26：對偶網路架構圖

而針對對偶網路的兩個輸出，其分別各為一個 List 型態。首先策略是一個長度為棋盤大小（9×9＝81個）的串列，其經過 Softmax 函數運算後，形成各項機率總合為 1，如此一來最大的項目即為該回合最佳的落子位置。接著價值的部分就單純許多，其為長度為 1、值介於 0～1 之間的串列。

接著是本算法包含的第二個演算法：價值策略蒙特卡羅樹搜尋（Policy Value Monte Carlo Tree Search，以下簡稱 PV MCTS）。而這裡的 PV MCTS 和前面 MCTS 的差距只在由 UCB 公式轉為 PUCT 公式的其中一種改良，其公式如下：

$$PUCT = \frac{w}{N} + C \times P \times \frac{\sqrt{T}}{1+N}$$

上面公式的符號分別代表；W：此節點的累積價值（即對偶函數輸出的價值）。N：此節點的試驗次數。C：用於調整「勝率」與「棋步預測機率×探索率」之間的平衡常數。P：棋步的機率分布（即對偶函數輸出的策略）。T：累積的總試驗次數。

也就是說 PV MCTS 不再利用 Random Play 取得公式中的數據，而是採用對偶網路的輸出。因此對偶網路的強度會直接影響 PV MCTS 的結果。

根據上面的解釋，可以想像提升對偶網路的實力成為最重要的關鍵。而要訓練對偶網路，最重要的是需要大量的訓練資料。至此我們建立了自我對弈模組，希望本算法藉由自己學習到不同變化的棋路。在自我對弈模組中，我們除了以對偶網路搭配 PV MCTS 進行對弈模擬外，為了達成不同變化的棋步，我們將策略以波茲曼分布（Boltzmann distribution）重新計算，公式如下所示：

$$波茲曼分佈 = \frac{N^{\frac{1}{r}}}{\sum_{i=1}^{n} N_i^{\frac{1}{r}}}$$

完成自我對弈模組後，我們開始建構訓練模組用的類別。根據五子棋的經驗，我們認為至少需要 4480 場模擬才有機會和人類的實力相提並論。但如果一次把 4480 場訓練資料模擬出來，那麼可想而知，這些資料並不符合人類的實力，因此我們將其拆成每次模擬 128 場，模擬完即訓練對偶網路以更新其中的超參數，並重複以上循環 35 次即可達成 4480 場的需求量。至此，我們已經完成所有 AlphaZero 需要的模組，其強化式訓練循環如下所示：

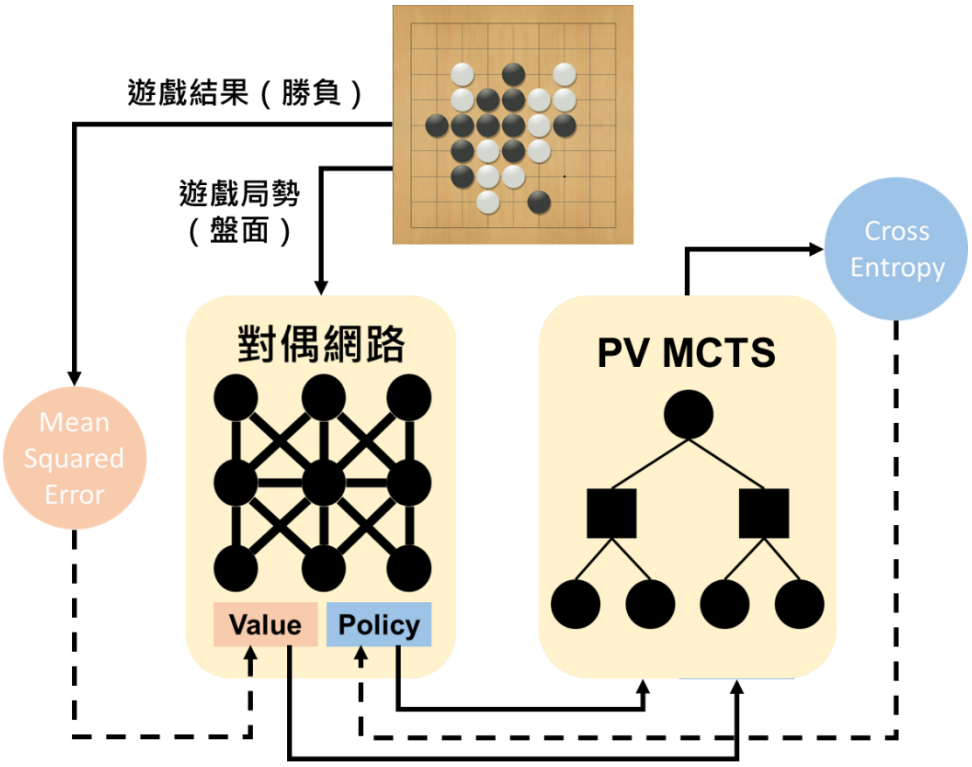

圖 27：強化式訓練循環

經過簡單的遊戲模擬及計算，我們確定無法在短時間內完成訓練。主要原因在於自我對奕的時間太長，光是完成一筆資料就要花費約莫 4 分鐘的時間，而拉長到 128 場來看則需要 8.5 個小時，而這並不包含其他訓練時間，且以上的計算都是基於使用 GPU 運行的結果。所以目前為止尚未完成模型的訓練，僅完成架構的實作，並確定其可運行。

陸、初步結論

在本次研究中，我們針對對稱規則及非對稱規則的梅花棋遊戲，進行了各人工智慧演算法間的比較。其中包含主要的三個演算法，分別是隨機行為、極小化極大值演算法及蒙特卡羅樹搜尋。而在這之中，極小化極大值演算法又被詳細區分為探索深度一層、兩層及三層，蒙特卡羅樹搜尋則是以模擬次數分為 100、300、500、700、900、1100、1300、1500、1700、1900 等多個版本。上述的所有勝率如下圖所示：

對稱規則下的梅花棋勝率分析													
Random	Minimax			Monte Corle Tree Search									
	一層	兩層	三層	100	300	500	700	900	1100	1300	1500	1700	1900
55%	96%	63%	87%	61%	50%	41%	31%	43%	48%	53%	51%	41%	42%
非對稱規則下的梅花棋勝率分析													
Random	Minimax			Monte Corle Tree Search									
	一層	兩層	三層	100	300	500	700	900	1100	1300	1500	1700	1900
54%	97%	61%	89%	66%	41%	48%	42%	49%	53%	39%	45%	36%	47%

圖 25：不同版本間的人工智慧勝率分析表

在觀察後可以發現，不論是對稱規則下的演算法勝率，抑或是非對稱規則下都非常不盡人意。而我們認為主要的原因還是要歸咎於目前所有演算法的結過過於隨機化，以 Minimax 來說，探索層數為三層時，其判斷的依據樣本過少，因此並無法給出準確的落子位置；反觀 MCTS 演算法，在該演算法的模擬次數過低時也會有同樣的問題，由於模擬次數低導致不同節點上的 UCB 值尚無法收斂，因而造成選擇落子位置時形同隨機行為。

此時我們對 MCTS 中的 UCB 公式進行改良。將其分為三種不同的形式，分別為 Normal、Inverst 及 Random 狀態。而我們同樣對其進行遊戲模擬，得到的結果為$Inverst(60\%) > Normal(49\%) > Random(44\%)$。而這樣的結果表示改良 UCB 公式對於 MCTS 的勝率確實有顯著得提升，但即便如此 60%的勝率仍然不高。

為了解決上述問題，我們開始朝兩種方向思考：首先，改良現有的 Minimax 演算法及 MCTS 演算法，希望提升他們運行效率，使得在同一時間下的探索層數及模擬次數提高。第二種方法則是為了從根本解決運行效率過低的問題，而最顯而易見的方法就是在遊戲運作前先將人工智慧訓練完畢，也就是在遊戲開始時直接給予一套策略，令電腦無須再做額外的遊戲模擬。綜上所述，我們開始實作 Tuple-Network、TD Learning 及 AlphaZero 的相關架構，但礙於時間關係，模型尚未被訓練。

柒、未來展望

一、進一步完善梅花棋軟體之 UI 介面及其他功能。

二、對梅花棋遊戲提出棋力的相關定義，使得人工智慧演算法間的比較更數據化。

三、持續精進 Minimax 演算法的分數定義（如訓練殘差網路作為評估函式）。

四、將 Minimax 演算法以 DP 形式編寫，進一步提升執行效率。

五、持續精進 MCTS 演算法的運行效率，增加模擬次數。

六、尋找合適的計算資源以利模型訓練（Tuple-Network、AlphaZero）。

捌、參考資料

1. Chu-Ling Ko。(2021 年 1 月 1 日)。從 2048 學習遊戲對局 AI 設計。CLKO 技術筆記。https://ko19951231.github.io/2021/01/01/2048/
2. Wikipedia contributors. (April 5, 2022). Strategy-stealing argument. Wikipedia. https://en.wikipedia.org/wiki/Strategy-stealing_argument
3. 克里斯托弗·Z.穆尼（2018 年）。蒙特卡羅模擬。漢語大詞典出版社。
4. 布留川英一（2021 年 1 月）。強化式學習：打造最強 AlphaZero 通用演算法。旗標出版社。
5. 張庭瑋、吳冠諺（2022 年）。2022 臺灣國際科學展覽會參展作品專輯：「The GoClub-梅花棋演算法效率及適用性分析」。國立臺灣科學教育館。https://twsf.ntsec.gov.tw/activity/race-2/2022/infornation.html
6. 維基百科編者。(2022 年 5 月 22 日)。極小化極大演算法。維基百科。https://zh.wikipedia.org/w/index.php?title=%E6%9E%81%E5%B0%8F%E5%8C%96%E6%9E%81%E5%A4%A7%E7%AE%97%E6%B3%95&oldid=71759496
7. 維基百科編者。(2022 年 7 月 8 日)。零和賽局。維基百科。https://zh.m.wikipedia.org/zh-tw/%E9%9B%B6%E5%92%8C%E5%8D%9A%E5%BC%88
8. 維基百科編者。(2022 年 5 月 22 日)。Alpha-Beta 剪枝法。維基百科。https://zh.wikipedia.org/zh-tw/Alpha-beta%E5%89%AA%E6%9E%9D

【評語】190004

此作品對自己定義玩法的梅花棋使出兩種人工智慧演算法（分別是 Minimax 及 Monte Carlo Tree Search）來讓人與電腦對戰並進行效能研究和比較，但這兩種方法都是使用 AI 在下棋時行之有年的方法，因此本作品的貢獻主要在實作這兩種方法且應用在梅花棋上。針對梅花棋的一些複雜變化和情況，本作品有進行探討，此外作者也開始研究 Tuple Network 及 TD Learning 的相關知識，但這些構想都尚未被實作出來。