國立台北大學資訊工程學系專題報告

愛因斯坦棋 EinStein würfelt nicht! - 棋類遊戲人工智慧

專題組員：徐子揚、徐家榮、陳治霖、張峻瑋

專題編號：PRJ-NTPUCSIE-108-011
執行期間：108 年 09 月 至 109 年 06 月

一. 摘要

在資訊工程領域之中，人工智慧的研究一直是個熱門的主題，在諸多的人工智能應用之中，電腦對局的研究是我們頗有興趣並且也十分期待去挖掘的一塊；對於多數大學生而言，電腦對局這個名詞可能是在人工智慧領域中最親近的一塊，畢竟，電腦遊戲是每個人或多或少都接觸過的，故此，我們想要針對這個領域做更深一步的探討。

愛因斯坦棋，一個源自於德國的棋類遊戲，透過擲骰子和移動棋子來進行遊戲；規則看似簡單易解，實則因為遊戲本身的高隨機性質讓獲勝變得十分困難，贏下遊戲的方法亦變得複雜許多；也因此，愛因斯坦棋更成為國際奧林匹亞電腦遊戲程式競賽的指定棋類遊戲之一。

二. 簡介

1. 專題之研製背景

老實說，在製作專題之前，組員對於愛因斯坦棋並不十分了解，所以對這個主題有些害怕，擔心自己沒辦法做得很好。然而，在進行多場的遊戲並參考一些相關文獻之後，我們對於研究愛因斯坦棋的興趣便提高了許多，再加上指導教授不斷給予我們鼓勵和意見，所以才有了開始製作專題的信心與勇氣，一步一步突破各種難關，最後達成目標。

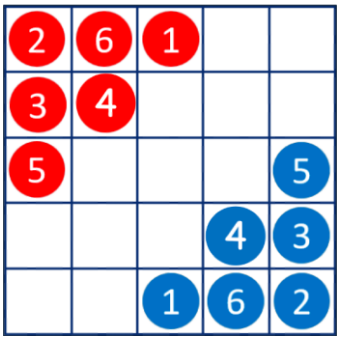
2. 專題研究目標

透過專題的製作，了解人工智慧程式的設計方式、製作過程並且能夠設計出擁有一定棋力水準的愛因斯坦棋人工智慧程式。

3. 愛因斯坦棋遊戲介紹

初始棋面：

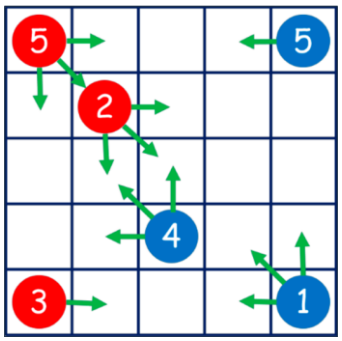
- 持棋方分為兩方。
- 5 x 5 棋盤，棋子置於方格中。
- 雙方各擁有編號 1~6 之棋子各一只。一方位於棋盤右下角，另一方則位於棋盤左上角。
- 開始前，雙方皆能對六只手棋做任意位置之擺放，唯需注意的是：位於左上角之玩家需將手棋置於棋盤左上角之三角形區域中；而位於右下角之玩家則需將手棋置於棋盤右下角之三角形區域中。



圖一、初始棋面之範例

移動方式：

- 雙方在進行每一手行動，皆需先擲一六面骰子，透過骰得點數決定此手應移動之棋子，若骰得點數為 5，則此回合僅能移動編號為 5 的棋子。
- 位於左上角之玩家僅能朝右方、下方、右下方移動(若位於盤面最右方，則僅能朝下移動；若位於盤面最下方，則僅能朝右移動。)
- 位於右下角之玩家僅能朝左方、上方、左上方移動(若位於盤面最左方，則僅能朝上移動；若位於盤面最上方，則僅能朝左移動。)

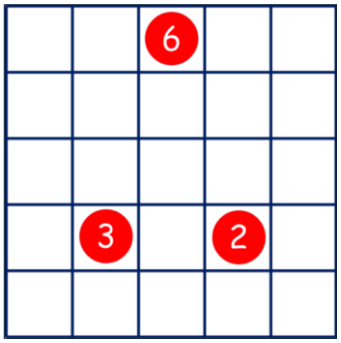


圖二、雙方移動方向之範例

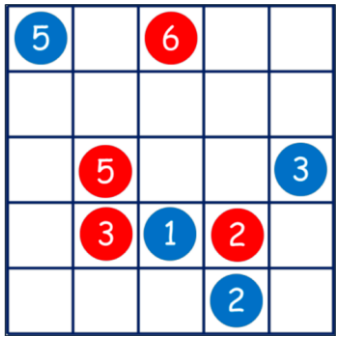
- 特殊情況：若移動前骰出數字為 3 且此時盤面上已無編號 3 之棋子，則玩家可自行選擇大於 3 最接近或小於 3 最接近之棋子移動之(如：盤面僅剩 2、5、6 三只棋子，則玩家能選擇移動 2 號棋或者 5 號棋。)
- 若欲移動之目的方格內有其他棋子，則不管敵方或者我方，棋子都會被吃掉；而棋上編號僅與骰得點數有關，棋子彼此之間並無大小關係。

勝利條件：

- 清除對手：其中一方失去全部棋子，則他方獲勝。
- 抵達對角終點：位於左上角之玩家抵達最右下角之方格；或者位於右下角之玩家抵達最左上角之方格，則該方獲勝。



圖三、左上玩家清除對手



圖四、右下玩家抵達對角終點

三. 專題進行方式

1. 蒙地卡羅樹搜尋法

由於愛因斯坦棋龐大的隨機性

質，所以程式必須仰賴大量的隨機模擬，盡可能地看透整個棋局的發展，藉以決定出最合適的走步，所以我們選擇使用在棋類遊戲人工智慧程式中十分著名的蒙地卡羅樹搜尋法(Monte Carlo tree search，簡稱 MCTS)。

蒙地卡羅樹搜尋法是一種啟發式的最佳優先搜尋演算法，在每一次要尋找最佳走步前，先將當前的盤面作為搜尋樹的根節點，之後再藉由選擇(Selection)、擴充(Expansion)、模擬(Simulation)以及反向傳播(Back propagation)這四個步驟的不斷循環，藉此選擇出最佳的走步，以下就各個步驟一一介紹其所代表的意涵以及執行方式：

- **選擇(Selection)：**

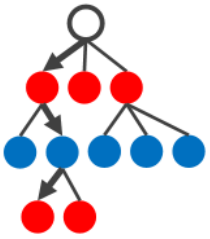
從根節點 R 開始，持續選擇連續的子節點向下至葉節點 L，而每一次選擇節點的依據則是目前節點 p 每一個子節點 p_i 的信賴上界值(upper confidence bound，簡稱 UCB)，其定義如下：

$$UCB_i = \frac{W_i}{N_i} + c\sqrt{\frac{\log N}{N_i}}$$

公式一、UCB 公式

其中 W_i 為 p_i 這個節點所代表的盤面經由模擬之後贏的次數，N_i 為 p_i 所代表之盤面進行模擬的局數，N 為 p 總共模擬的局數；第一項為開發項(exploitation)，即為 p_i 節點的勝率；第二項為勘探項(exploration)，是用來控制去嘗試其他非最佳棋步的頻率，其中 c 稱為勘探參數(exploration parameter)。

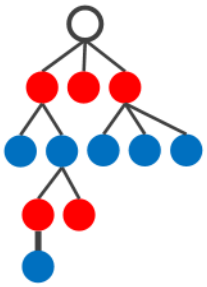
藉由這個選擇節點的方式，就能夠讓遊戲樹朝向最佳的方向擴充，這是蒙地卡羅樹搜尋法的精要所在。



圖五、Selection 示意圖

- **擴充(Expansion)：**

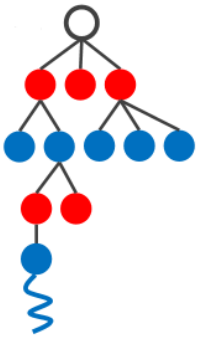
藉由前一步驟選擇到的葉節點 L，除非任一方的輸贏使得遊戲在此節點結束，否則就由此葉節點下面擴展出下一步會出現的所有可能棋盤 C。



圖六、Expansion 示意圖

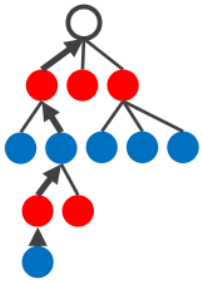
- **模擬(Simulation)：**

針對前一步驟所產生的所有新節點 C 進行數次的模擬遊戲，直到分出勝負，而其是採用隨機策略來進行，此步驟又稱為 playout 或者 rollout。



圖七、Simulation 示意圖

- **反向傳播(Back propagation)：**對於前一步驟所模擬出來的遊戲結果，更新從目前節點 *C* 到根節點 *R* 路徑上所有節點的資訊。



圖八、Back propagation 示意圖

2. MCTS 與愛因斯坦棋

在程式的實作上，每一次 AI 程式要選擇最佳走步時，程式會先依照目前擲出的骰子點數來判斷所有可能的走步(例如骰到點數 3，且 3 號棋子可以朝三種方向移動，則有三個可能走步)，並且將這些走步分別作為樹的根節點來執行蒙地卡羅樹搜尋法。

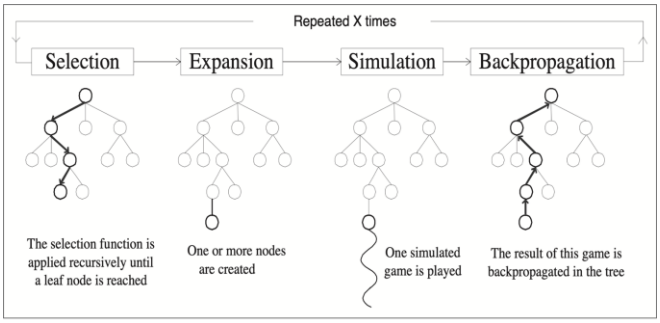
我們程式的遊戲樹在擴展時，會將目前節點所代表之棋局下一步對方的各種走步可能性都展開，但是在展開之前，程式會先檢查某棋子是否還存在棋盤上，若不存在，則不擴展與這顆棋子相對應走步的節點；另外程式也會檢查某存在棋盤上的棋子的某一走步是否有超出棋盤邊界，若此走步不合法，則亦不擴展此走步的節點。也就是說，在每一節點下方最多會擴展出十八個子節點(1~6 號棋子朝三種方向移動)。如果此節點之棋局已經使某一方分出勝負，則會略過擴展以及模擬的步驟，直接將此棋局之勝負結果進行回傳。

在擴展完畢後，在程式中可以設定對每一個擴展出的節點所代表之棋局採用隨機策略進行固定次數的模擬，而對於每一個節點，程式都會記錄其總模擬次數以及在這些模擬中 AI 方的勝場次數，勝場次數除以總模擬次數即為此節點之勝率，當設定的模擬次數越多，對於此節點勝率的估計也會越準確，也可以為選擇節點時提供更加精確的數據。

在模擬結束之後，這些擴展出來的節點就會得到一定數量的勝場次數及模擬次數，接下來就會將這些數據進行反向傳播，意即針對從目前節點到根節點路徑上所有的節點以該數據進行更新。舉例來說，目前擴展出三個節點，每個節點在經過各 10 次的模擬之後，其勝場次數/模擬次數分別為 3/10、5/10、6/10，在進行反向傳播之後，這個三個節點的父節點之數據即為 14/30，依此類推。

做完一輪蒙地卡羅樹搜尋法的四個步驟稱為一個循環(iteration)，而程式中亦可設定要執行幾次循環，循環得越多次，遊戲樹就會擴展得越完整，也能夠讓遊戲樹的主要路徑盡可能地擴展至終局盤面。

而 MCTS 的流程圖如下圖：



圖九、MCTS 流程圖

當 AI 方的每一個可能走步都執行完蒙地卡羅樹搜尋法以後，這些走步

的節點內就會儲存其勝場次數以及模擬次數，接下來程式就會依照該數據選擇勝率最高的走步作為此回合 AI 的走步。

3. 多執行緒(multi-thread)

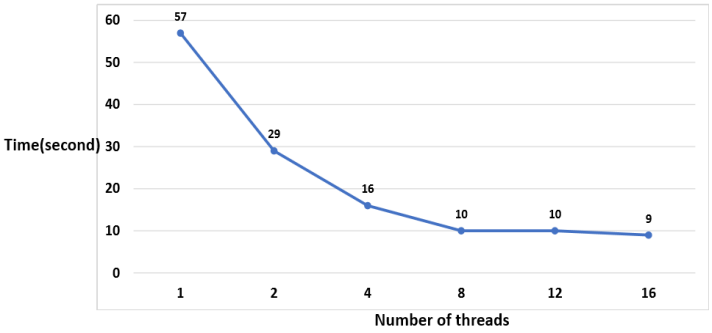
多執行緒是一種提高具有硬體多執行緒的 CPU 整體效率的技術，同時多執行緒允許多個獨立的執行執行緒更好地利用現代處理器架構提供的資源，並且達到節省時間的效果。

4. 執行緒池(threadpool)

在多個執行緒同時運作的情況下，因為各執行緒分別獨立運作的關係，會有結束時間不一的情況，造成先執行完的執行緒停止運作，而沒辦法使運作時間達到最大效益，因此需要用執行緒池，原理是可讓先執行完的執行緒放回佇列，等到要用時再呼叫出來使用，而不需再次開啟一個新的執行緒導致系統浪費時間及效能。

5. MCTS 結合 threadpool

然而 MCTS 運作，只會一次模擬單一的葉節點，如此一來所耗費的時間相當龐大，因此將 MCTS 的葉節點模擬結合上 threadpool，讓在有限的 CPU 核心下，可以同時執行多個葉節點的模擬，此設計可節省不少時間，由實驗後的圖表可看出在不同數量的執行緒下，執行的時間曲線圖如圖十所示，依照 CPU 核心的數量，若執行緒數量超過 CPU 可同時執行的數量，曲線將趨近平緩而時間不再縮短(測試使用之電腦有八個邏輯核心，所以當 thread 數量大於 8 之後，所花費之時間就無明顯下降)。

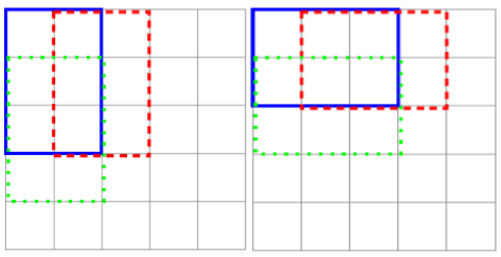


圖十、執行時間曲線圖

6. N-Tuple 設計

雖然使用蒙地卡羅樹搜尋可以為 AI 提供較佳走步之資訊以提高勝率，但如果遊戲樹中主要路徑之最底層的節點已經擴展至終局盤面，此時再做更多的模擬就比較無法為 AI 提供有效的資訊，所以若是能夠加入事先訓練的知識，就可以為 AI 提供更加多元的資訊，並進一步提高 AI 之勝率。

根據參考資料[2]文中的內容，愛因斯坦棋若是將整個棋盤視為一個單位的話，其變化約有 13^{25} 種，數量相當龐大，所以需要把棋盤切分為許多個小區塊來個別處理，在文章中的實驗結果，他們認為將棋盤分為每 6 格一個單位的效果最佳，也就是 6-tuple，而在這個設計中，使用 3x2 以及 2x3 兩種長方形在棋盤上推移並記錄此長方形區域內的特徵，示意圖如下：



圖十一、6-tuple 示意圖

使用這個方法的話，對於每一個棋盤都會產生 24 個 6-tuple，如此一來，

就能將棋盤的變化大幅縮減至 24×13^6 種，能夠節省許多空間。

而訓練方法則是使用兩個表格分別儲存每一個 6-tuple 的勝場次數及拜訪次數，並且讓兩個使用隨機策略的 AI 進行對打，在紅方進行移動之後，會將目前棋盤所代表的 24 個 6-tuple 暫時放在紅方的待更新區，藍方亦然。當棋局分出勝負後，會將贏家待更新區中的所有 6-tuple 之勝場次數及拜訪次數加一，輸家則只會增加待更新區中的所有 6-tuple 的拜訪次數。

7. 事先訓練結合 MCTS

在參考資料[2]中亦提到許多方法可以將事先訓練所得到的數據與 MCTS 結合，以提高 AI 之勝率，首先需要先使用一個向量儲存事先訓練中每一個 6-tuple 的勝率，即勝場次數/拜訪次數；再針對不同的棋盤計算出其對應的 24 個 6-tuple 的平均勝率，即 24 個 6-tuple 的勝率相加後再除以 24，而這個數值將其稱為 V(b)。以下介紹各種將事先訓練與 MCTS 結合的方法要如何運用 V(b)：

•Progressive Bias(PB)：

此方法是將原本的 UCB 公式(公式一)替換為以下公式：

$$UCB_i + C_p \times \frac{V(b)}{count_i}$$

公式二、PB 公式

使用此公式的話，就可以很直覺地將是先訓練的知識直接加入到 MCTS 之中，使其在選點時可以更加精確地選到勝率較高的節點。其中常數 C_p 以及 $count_i$ 在參考資料中的實驗結果分別使用 10 以及敗場次數的效果最佳(敗場次數為拜訪次數減去勝場次數)。

•Prior Knowledge(PK)

此方法則是針對每一個 MCTS 在擴充階段擴展出的新節點藉由事先訓練的經驗給予一些初始的勝場次數(w_i)及模擬次數(n_i)：

$$n_i = N_0$$
$$w_i = V(b) \times N_0$$

公式三、PK 公式

如此一來就能夠稍微減少 MCTS 在模擬階段中所需要的模擬次數，以提高程式執行的效率。其中 N_0 在參考資料的實驗結果使用 175 的效果最佳。

• ε -Greedy

此方法則是針對 MCTS 的模擬階段進行改良，其原理是在 MCTS 進行模擬時，會有(1-ε)的機率會選擇 V(b)數值最高的走步，而其他則會有 ε 的機率會採隨機策略走步。相較於完全隨機的模擬，這個方法會產生更加合理的遊戲模擬，也能讓對於某盤面勝率的測量更加準確，同時也保留些許嘗試的空間給予隨機走步。在參考資料的實驗中 ε 使用 0.2 的效果最佳。

四. 主要成果與評估

1. 連接至遊戲界面

為了展示上的便利，所以我們將程式連接在相關的遊戲介面，以下為遊戲操作介紹：(紅方為程式電腦，藍方為手動玩家)。

在遊戲開始之前，己方可以交換己方的棋子來擺放位置，準備完成後即可按右方的按鈕「擺盤完成」來開始遊戲(圖十二)。

圖十二

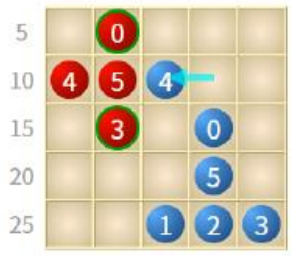
在圖十三中右方的方框中之數字即為當輪到己方時骰子擲到的點數，而左方有邊框的棋子為目前可移動的棋子以及可以移動的方格。

圖十三

網頁中左方會顯示棋子目前狀態(圖十四)。

圖十四

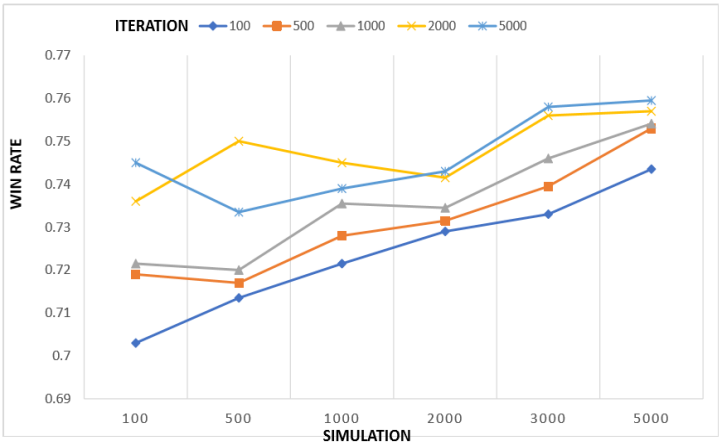
當出現骰子點數，棋子卻不存在的狀況時，便會出現兩種可能移動的棋子，玩家可自行點選欲移動之棋子(圖十五)。



圖十五

2. 實驗數據及結論

如前面所提到，執行一輪 MCTS 的四個步驟(Selection, Expansion, Simulation, Backpropagation)稱為一個循環(iteration)，而電腦 AI 的強度也和循環次數與模擬次數有著很大的關聯。在我們的實驗中，我們測試多組不同循環次數與模擬次數的搭配，並各與採用隨機策略的 AI 對戰 2000 場遊戲測試其勝率，實驗結果如下：

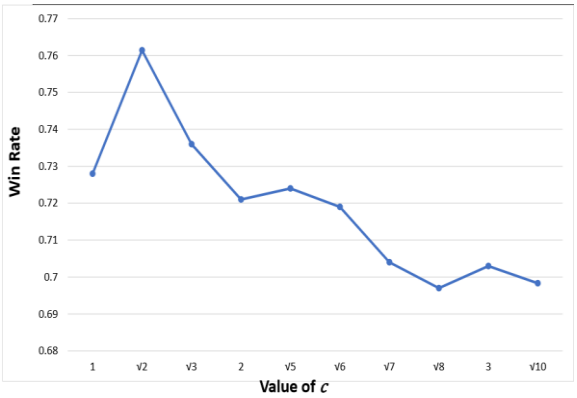


圖十六、MCTS 循環與模擬次數之勝率

由上方圖表可以看出，當模擬次數增加時，勝率也會有小幅度的上升，但也不會持續地上升，其上限約為 75%左右。

另外，我們也針對 UCB 公式(公式一)中的常數 c 嘗試許多不同的數值，

並搭配循環次數與模擬次數皆為 5000 的 MCTS 與使用隨機策略的 AI 進行 2000 場遊戲測試其勝率，結果如下圖：



圖十七、不同 c 值的勝率

可以明顯看出當 c 使用根號二時所得到的結果最佳，所以我們在程式中就選擇使用根號二作為公式中的 c 。

五. 結語與展望

這次的專題研究中，在所有組員努力之下，透過蒙地卡羅樹搜尋法，以及其他能夠用來增強 AI 的方式，來精確的預測棋步未來走向和獲勝的機率，成功地提升了人工智慧的勝率，與真人對下中可以感受到該難度，並不像一般的隨機程式，在未來也希望能找到更多的方法或是結合方式，來更加的提升此人工智慧的勝率，而速度方面，也需要提高更好的效率來節省模擬及運算的時間，雖然現今未能做出最好的人工智慧，但相信日後努力的研究，在未來的某一天，也能夠參賽，與各地高手一較高下，站上世界的舞台。

六. 銘謝

首先十分感謝指導教授這一年來的指導與鼓勵，教授在我們製作專題的過程中提出許多寶貴的意見以及學習資源，才能讓我們擁有現在的成果。

另外也要感謝實驗室的學長們不辭辛勞地協助我們解決在專題進行中遇到的困難與瓶頸，並且提供我們程式的改進方式，使我們程式執行起來更加有效率。

最後也必須感謝小組中每個成員的互相配合與付出，這個團隊中的成員缺一不可，在專題製作的過程中每個人都盡自己最大的心力讓專題的進行更為順利。

七. 參考文獻

[1] 曹少剛，”The Initial Research of EinStein würfelt nicht! with Deep Learning”，2017

[2] Yeong-Jia Roger Chu, Yuan-Hao Chen, Chu-Hsuan Hsueh, I-Chen Wu, “An Agent for EinStein Würfelt Nicht! Using N-Tuple Networks”, 2017

[3] Sylvain Gelly, David Silver, “Combining Online and Offline Knowledge in UCT”

[4] Guillaume M.J-B. Chaslot, Mark H.M. Winands, H. Jaap Van Den Herik, Jos W.H.M. Uiterwijk, “Progressive Strategies for Monte-Carlo Tree Search”, 2008

[5] Guillaume Chaslot, Christophe Fiter, Jean-Baptiste Hoock, Arpad Rimmel, Olivier Teytaud, “Adding expert knowledge and exploration in Monte-Carlo Tree Search”

[6] Emanuel Oster, “An MCTS Agent

For EinStein würfelt nicht!”，
2015
[7] 徐讚昇，“電腦對局導論”，
ISBN：978-986-350-237-1，2017