

AI final project report

Group 34

紀政良109550004、林揚益109550133、林律穎109550042

github repo link: <https://github.com/tonychi2002/connect-four.git>

Content

Content	1
1. Introduction	2
1.1 屏風四子棋遊戲規則	2
2. Related work	4
3. Methodology	4
3.1 Game Tree	4
3.2 Minimax	5
3.3 Alpha-beta pruning	6
4. Experiments	6
4.1 探討先手優勢	6
4.2 本次專題設計之 agent	7
4.3 Minimax 和 Alpha-Beta 的反應速度比較	7
4.4 不同搜尋深度的戰力比較	8
5. Conclusion	8
6. References	9

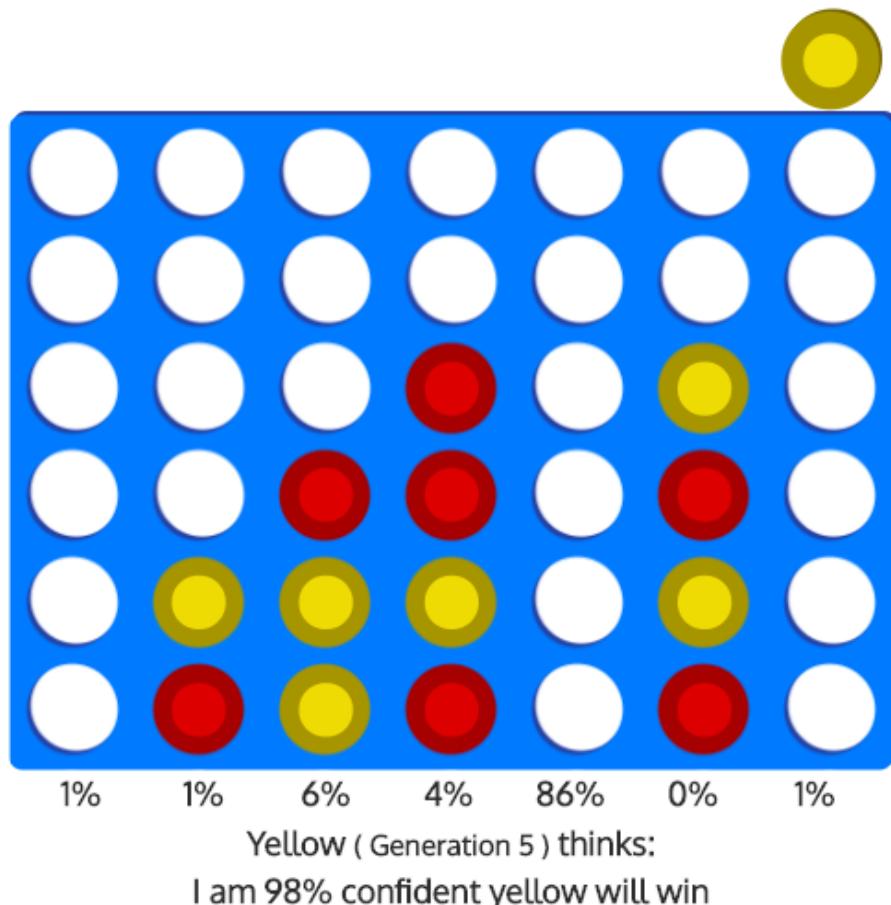
1. Introduction

本次期末專案，我們將以訓練 AI 玩屏風四子棋為主題做討論。屏風四子棋是一款風靡全球，家喻戶曉的遊戲，其規則簡單，又能讓玩家腦力激盪，深受世界各地人們的喜愛。但這項看似簡單的遊戲，玩起來卻不是那麼的容易，就連資深玩家來玩都很難有必勝法，因此也很難做出完每的模型，但我們願意為之嘗試。

由於我們三人皆對此遊戲有相當程度的熱愛，僅僅面對面腦力激盪並不能滿足我們征服屏風四子棋的欲望，若能設計出成功的 AI 四子棋模型，不但能應用在人工智慧概論這門課所學的知識，對其有更高的掌握度，還能從中獲得更多的成就感。

1.1 屏風四子棋遊戲規則

1. 雙方必須輪流把一枚己棋投入開口，讓棋子因地心引力落下在底部或其他棋子上。
2. 當己方4枚棋子以縱、橫、斜方向連成一線時獲勝。
3. 棋盤滿棋時，無任何連成4子，則平手。



因為旗子會落下到底，在玩的過程中其實很仰賴對方的旗子落點，因為這關乎自己能否快速達到需要的高度，或者搶下某個關鍵點，因此如何考慮對方的行動是訓練此遊戲 AI 相對重要的一環。

我們實做的方法是運用課堂中所學的 search 和 reinforce learning 去設計出好的 AI model，目標讓他擊敗資深玩家，以及打敗其他現有 AI。

實做部份，整個遊戲邏輯與架構，從零到有都是我們自己寫的，用python展現出OOP的精神，寫出一個可以運作，有正確邏輯的四子棋。並且為了方便後續演算法實做，寫出許多方便查詢的 function，算是對後續開發很友善的自產 interface。

2. Related work

已經有很多人針對 Connect 提出不同的策略，有些以觀察盤面的關鍵特徵（例如哪些盤面必勝、哪些盤面可能會讓對手勝出）並使用條件判斷的方式來擬定策略，有些則是利用搜尋樹來找尋當前盤面的最佳解。根據上網查 connect four 的相關研究結果，大部分皆是探討 minimax, alpha-beta pruning 為主，並討論如何加快運算速度，試著在有限的時間與記憶體內做出最大強度的 AI model。

網路上有人在探討如何壓縮狀態表示法的儲存空間，剪枝的強度能節省多少時間，或是在遊戲不同階段所對應討論的深度不同、剪枝程度上的不同，有的能顯著的壓縮時間，有的則能有效的提高勝率，以遊戲棋子個數多寡，去探討該階段所能找到的最佳解法。

3. Methodology

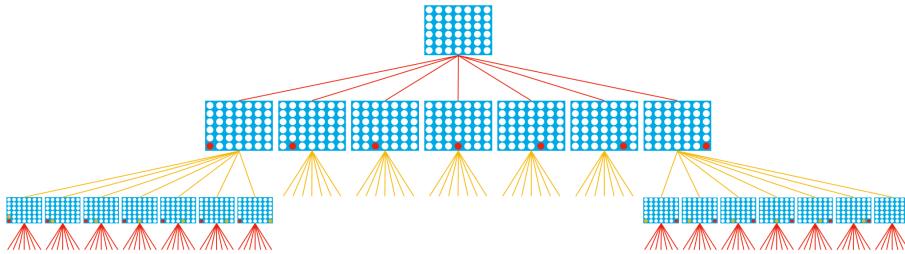
3.1 Game Tree

Game tree 是用來表達一個遊戲進行下去可能發生的所有情況，以樹狀圖的方式呈現。一個完整的 game tree 會包含每個中止狀況，也是所謂的葉節點，將遊戲中所有情況都推演到無法進行為止，換言之，就是很暴力又詳細的將所有未來可能發生狀況列出。

Game tree 的成長性跟遊戲每一步的合法行為個數高度相關，connect four 最多有七個合法行為，因此，以空棋盤為根之 game tree 最多會有 $(7!)^6$ 個數節點，也就是所有可能發生的遊戲狀況。

為了限制 game tree 的成長，通常人們或機器在評估時會訂出一個層數，往下推演到 n 層就好，在節省時間的條件下，查看遊戲接下來可能發生的狀況。

四子棋 game tree 狀態總數：4,531,985,219,092，數據來自網路，可見把整顆 game tree 列出來是不切實際的作法。



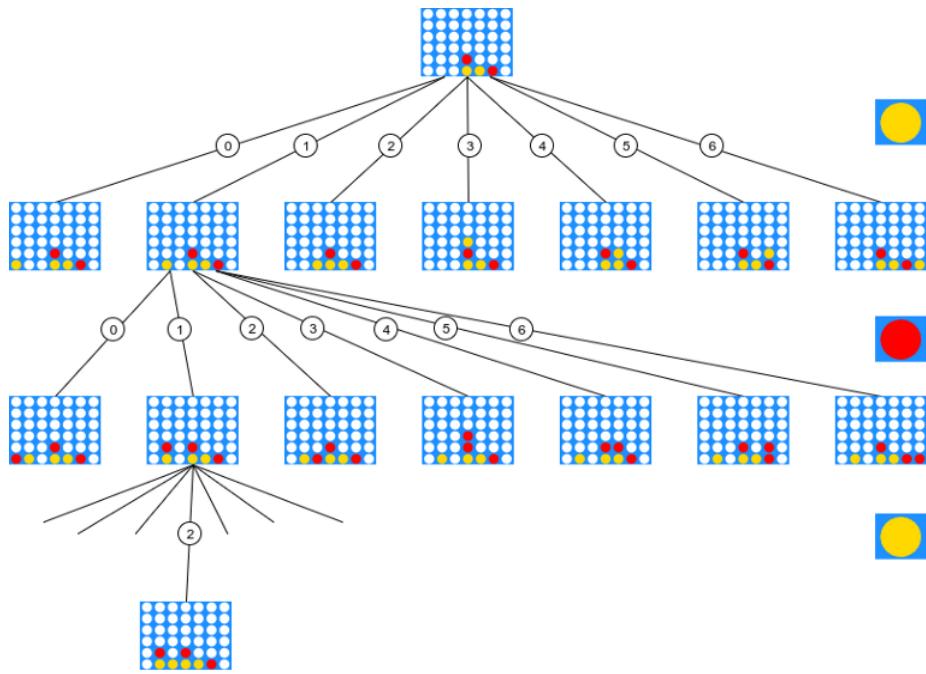
3.2 Minimax

Minimax 常用在正反兩方對抗時，假設對方都會採最佳走法，站在己方立場去預估接下來應該走哪一步。

中文稱作「極小化極大演算法」，在一盤遊戲中，假設雙方都會選擇對自己最有利的策略進行遊戲，因此可以去預先判斷對手的行動，在做出這步決定前，事先判斷對手會如何做出應對，再藉由此資訊決定策略。如此反覆進行。如果一路推到最底層（遊戲結束），四子棋的AI 將可以直接判斷出目前局勢會勝利、失敗或平手。然而，這需要相當大的運算資源。此外，遊戲最多會進行 42 步，因為無法在有限的記憶體內儲存計算過的結果，因此每次都要重算。這樣會需要非常久的運算時間。

我們將Minimax 往後預估的步數稱為 N ，若能預估整個棋局，也就是走到必勝或無法移動的情況，將可直接得到最佳解法，但此舉動為 N 的指數性成長，通常會造成龐大的運算資源消耗，故依據我們遊戲棋盤的大小去調整 N 之值，避免造成運算過久，影響遊戲體驗。

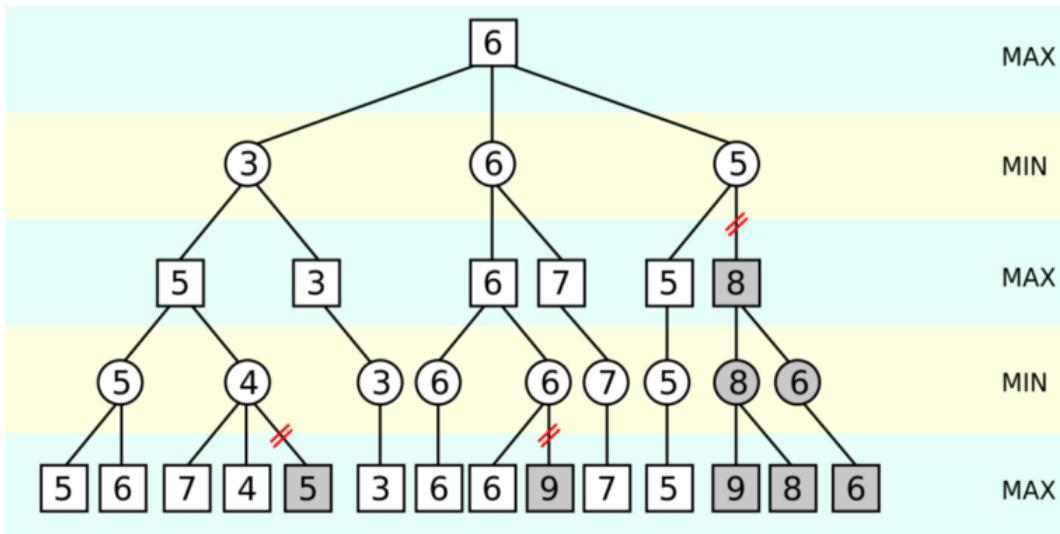
因此在實行 Minimax 演算法時，會先訂出一個向下推演的最大層數，也就是能往下預估幾步內的結果，再依據此結果去做出對自己最有利的選擇。可想而知， N 定的越大，結果會越接近真實結果，也就是對自己越有利。在執行時間考量下，我們只對 $N=2, 3, 4$ 進行Minimax，再更大的話，時間將會難以接受。



3.3 Alpha-beta pruning

在上述Minimax中，我們會把 game tree 在 N 層內完全的展開出來，是有效但相當暴力的作法，但其實可以更聰明。於是，就需要alpha-beta pruning的幫助。在推到

當演算法評估出某策略的後續走法比之前策略的還差時，就會停止計算該策略的後續發展。該演算法和極小化極大演算法所得結論相同，但剪去了不影響最終決定的分枝



4. Experiments

注：以下測試結果皆以三個元素為一組的 tuple (player 1 win, player 2 win, draw) 來表示先手勝利、後手勝利和平手的場數。

4.1 探討先手優勢

參考網路上得資料，若兩個 $n = 42$ 的 Minimax 對抗，第一手只要下中間，且後面保持最佳下法，則先手必勝，但礙於時間成本，本次 minimax 和 alpha-beta pruning 的 n 只有設計到4，故無法知道先手下中間以及後續下法如何必勝，然而我們仍能希望能從實驗結果觀察到先手方是否在搜尋深度限制情況下還具有優勢。

4.2 本次專題設計之 agent

agent_random():隨機挑一個合法column

agent_smart():看到己方有三個連線，會去下第四個

agent_smarter():看到己方有三個連線，會去下第四個，若無則會去阻擋對手的第四步

agent_score():引入heuristic的概念，看下一步誰分數高就選誰

agent_minimax():將agent_score()權重值得概念更深入的使用，看的不只是下一個，而是下n個，也就是極小化極大演算法(有深度限制)

agent_alpha_beta():使用 alpha-beta pruning 來加速 agent_minimax() 的運算速度

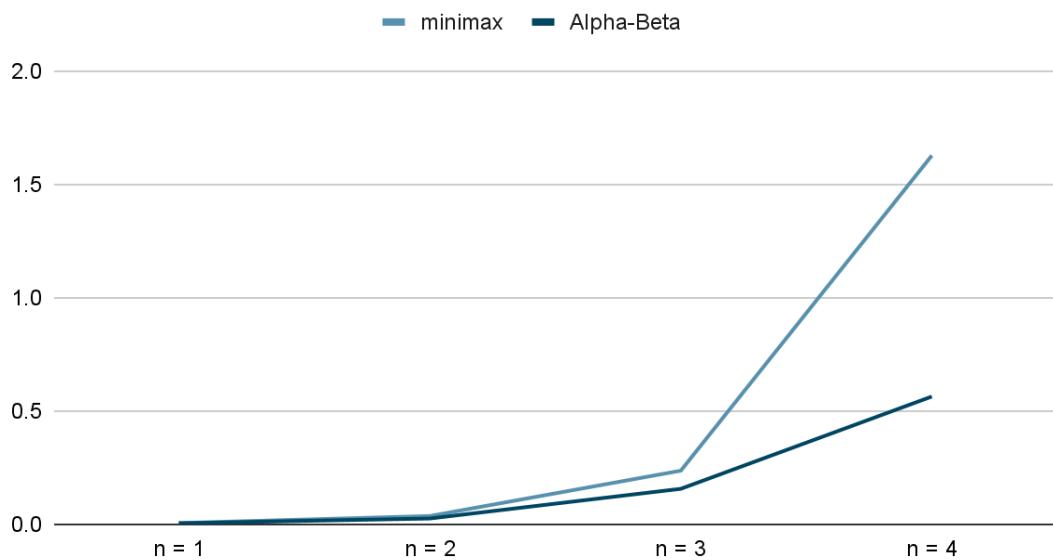
我們想關注其中的 agent_minimax() 和 agent_alpha_beta()，並和其餘沒有採用遊戲樹的 agent 比較。經過和 agent_random() 比賽來評估之後，觀察到使用遊戲樹的 Agent 會有比較好的表現。

random	smart	smarter	score	minimax n=2	alpha_beta n=2
545/451/4	849/151/0	819/181/0	993/7/0	999/1/0	1000/0/0

4.3 Minimax 和 Alpha-Beta 的反應速度比較

下表顯示 Minimax 和 Alpha-Beta 在不同搜尋深度時，每回合的平均反應時間。可以看出隨著搜尋深度越來越大，Minimax 的膨脹速度也明顯超過 Alpha-Beta，可見 Alpha-Beta 的加速效果十分顯著。

Points scored



4.4 不同搜尋深度的戰力比較

下表列出同樣使用agent_alphaBeta() 但帶入不同 n 去比較30次，將n = 1, 2, 3, 4互相比較的勝敗狀況。資料以每百場的〔先手勝場數 / 後手勝場數 / 和局數〕之格式表示。

p1\p2	n=1	n=2	n=3	n=4
n=1	21/9/0	4/24/30	7/23/30	3/27/0
n=2	25/4/1	12/13/5	17/11/2	4/23/3
n=3	27/3/0	19/8/3	18/12/0	8/14/8
n=4	30/0/0	8/12/10	25/2/3	10/14/6

從此表可看出幾點特點：

1. 先手優勢平均看來是存在的。以此表格之斜對角線分為左下和右上來觀察，比較相同 agent 的組合在不同先後手的表現，可以發現普遍先手時的勝場數多於後手時的勝場數。
2. n=4 打贏 n=3, 却打輸 n=2, 以及 n=2 打贏 n=3, 推測造成此原因為，當遇到相同權值時的選擇是具有 random 性質的，且因為時間關係，測試場數較少，所以無法大量的顯示出 agent 之間的穩定性。

5. Conclusion

本次人工智慧概論期末專案運用課堂所學設計 Connect Four 的遊戲 AI，並比較使用特定策略和 Game Tree 實作 Agent 的優劣，經實驗發現使用 Game Tree 的 Minimax agent 表現較佳，且搜尋深度限制越深勝率越高。然而由於 Minimax agent 的反應時間隨著搜尋深度限制變大而膨脹的狀況十分嚴重，因此我們以 Alpha-Beta Pruning 來嘗試改善狀況，於實驗後發現確實能夠大幅改善反應時間膨脹的問題。

此外，我們也發現就算無法得知目前盤面是否為必勝局面，採用同樣條件的 Agents 之比賽中，先手方也還是佔有優勢，然而使用更聰明的 Agent 還是能夠在後手條件中取勝。未來如果有機會，希望能夠嘗試盤面更大或有更多限制的遊戲版本，希望可以有新的發現。

雖然做出來的成品不如網路上的好，但在測試、修改與挑戰的過程中，對於沒有專案經驗的我們是十分難能可貴的。

6. References

<https://towardsdatascience.com/creating-the-perfect-connect-four-ai-bot-c165115557b0>

<https://www.kaggle.com/learn/intro-to-game-ai-and-reinforcement-learning>

<https://connect4.gamesolver.org/>

<https://www.youtube.com/watch?v=yDWPi1pZ0Po>