Homework 2: Connect Four

1. Introduction

該報告在實作基於 Minimax Search 的 Connect Four 遊戲 AI。分為三個部分:第一部分是只使用 Minimax Search 的基礎 agent。第二部分是額外使用 Alpha-Beta Pruning 來縮短第一部分執行時間的 agent。並且探討使用該演算法相對於原本的 agent 的優劣點。第三部分是基於前述的演算法，實作出一個對抗使用 Alpha-Beta Pruning 的 agent 能有 50%以上勝率的 agent。依據修改的內容探討第二部分潛在可繼續改善的地方。

2. Implementation

Minimax Search

Algorithm:

從深度 0 的點開始，使用 DFS 的方式去探索深度為 4 的 tree。一旦探索到深度 4 或是到達的深度已經停止遊戲，就返回目前深度的棋局評估成績 get_heuristic(board)。當返回成績後，使用 boolen 去判斷目前是 maximizing(True)抑或是 minimizing player(False)。如果 maximizing，初始 value 設置為系統最小值。只要大於返回的成績大於 value，value 設置為該成績，並將 set 重設為該列數。如果等於則將該列數加入 set。反之亦然。

Results & Evaluation:

這是 Minimax 執行 100 次的執行結果及勝率。近 23 分鐘的執行時間，及 100%勝率。

```
Game 100/100 finished.
execute time 1320130.38 ms
Summary of results:
P1 <function agent_minimax at 0x000001507ABA7490>
P2 <function agent_reflex at 0x000001507ABA75B0>
{'Player1': 100, 'Player2': 0, 'Draw': 0}
========================================
        DATE: 2025/03/28
        STUDENT NAME: 何嘉婉
        STUDENT ID: 109550072
        ========================================
```

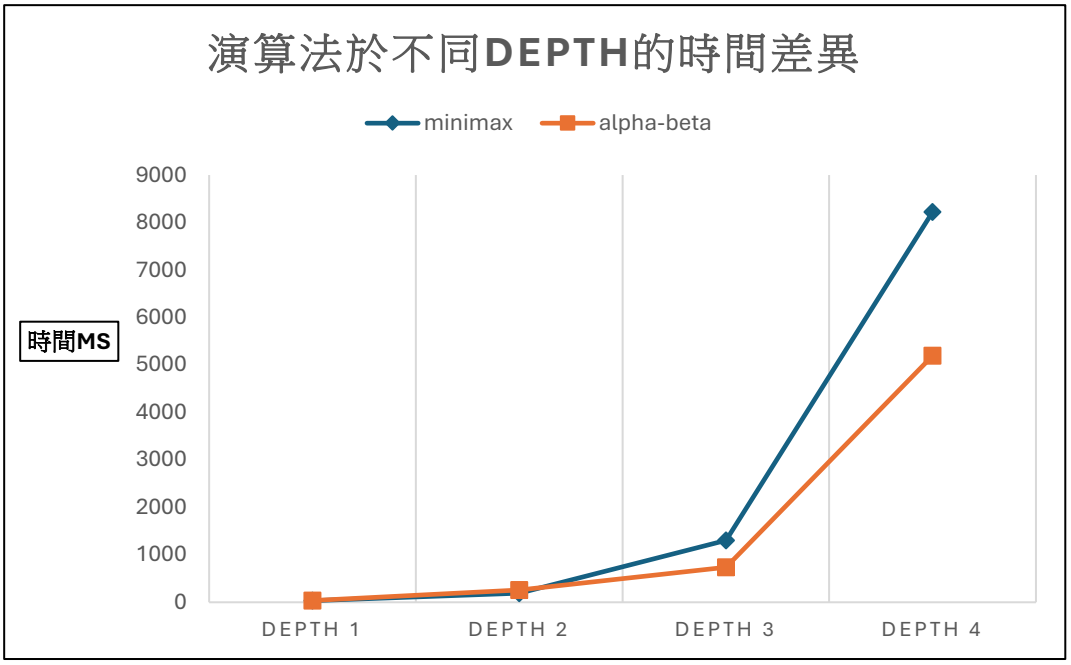
Alpha-Beta Pruning

Optimization:

該演算法會儲存 maximizing 能達到的最大值 alpha，minimizing 能達到的最小值 beta。在 maximizing 的節點，只要該節點的子節點出現大於等於 beta 就停止運算，因為大於 beta 的值不會被另一方 minimizing 選擇了。如果小於，那就更新 alpha 值。反之亦然。所以能減掉一部份的計算。在我的 code 中，基本上列數 set 只會存一開始遇到的極限值列數，以及如果有出現的相同極限值的列數。因此 set 的元素數量應該是介於 1 到 2 之間。

Results & Evaluation:

下面兩演算法各執行一次遊戲於不同深度的時間差異。能看出當深度越深產生的時間差異越大。也表示了，當該 tree 的規模越大，剪枝對於該 tree 的檢索時間有更大的效益。



這是 Alpha-Beta Pruning 執行 100 次的執行時間及勝率。近 6 分鐘的執行時間，96%勝率。執行時間相較 Minimax 減少 75%，但同時勝率也因為略過部分可能結果而下降。

```
Game 100/100 finished.
execute time 334872.60 ms
Summary of results:
P1 <function agent_alphabeta at 0x000002592FC3B520>
P2 <function agent_reflex at 0x000002592FC3B5B0>
{'Player1': 96, 'Player2': 4, 'Draw': 0}
========================================
        DATE: 2025/03/28
        STUDENT NAME: 何嘉婉
        STUDENT ID: 109550072
        ========================================
```

Stronger AI Agent

Techniques Used:

基本跟 Alpha-Beta Pruning 一致，但是由大小於跟等於判斷改成只有大小於判斷，增加決策的選擇。以及初始的列數執行順序改成由中心開始向外擴[3, 2, 4, 1, 5, 0, 6]，因為先佔據中心位置比較有靈活性。以及修改 Heuristic Function，詳細於下面說明。

Advanced Heuristic Function:

將算法原本只計算連 2、連 3 個棋，轉換成計算連續且旁邊有空的連棋。計算空一個位置及空兩個位置的各自數量。然後一樣對稱的算下來，4 棋、空兩頭的 3 棋、空一頭的 3 棋，以及空兩頭的 2 棋跟空一頭的 2 棋。4 棋、空兩頭的 3 棋出現就是定局了。然後只空一頭的 3 棋需要防堵或是攻下。而後兩個對棋局的影響較小，但不是沒有。這些變數去形成更複雜、精確的 value 算法。

How it counters Alpha-Beta:

由上述的 Heuristic Function 改版。可以確保該 agent 會去確保自己能夠達成如空兩頭的 3 棋，讓自己進入必贏狀態，然後也會去封鎖對方連棋。然後以及先掃視中心位置的順序，佔據更有贏面的地位。再來是增加決策的選擇，比 Alpha-Beta Pruning 的可選路徑更多。

Results & Evaluation:

這是 Alpha-Beta Pruning 與 Strong 執行 100 次的執行時間及勝率。近 19 分鐘的執行時間，82%勝率。因為都還是使用 Alpha-Beta Pruning 演算法，因此還是比純 minimax 快。

```
Game 100/100 finished.
execute time 1086932.06 ms
Summary of results:
P1 <function agent_alphabeta at 0x00000206D05F7520>
P2 <function agent_strong at 0x00000206D05F7640>
{'Player1': 14, 'Player2': 82, 'Draw': 4}
========================================
        DATE: 2025/03/28
        STUDENT NAME: 何嘉婉
        STUDENT ID: 109550072
        ========================================
```

3. Analysis & Discussion

於設計 strong heuristic 的難點，我認為是 score function 中各變數的係數調整。要讓每個參數都能發揮作用，但又不能影響到對其他參數的判斷。以我的情況我是每個數值隔開 100 倍，因為像是單空一頭連 2 棋可能可以超過 10 個，但不太可能到 100。以及我一開始將 win 及雙空連 3 棋，放置於同一個位置，因為我想說這兩個參數都是勝利。但是調適後發現，雙空連 3 棋容易勝利，但不一定導向勝利，因為有高度差異。而且放一起會導致原本可以贏，但是 agent 跑去做其他雙空連 3 棋的情況。為了讓此 agent 有更高的 win rate，我將 Alpha-Beta Pruning 條件修改，所以執行時間比較多。在執行速度上要弱於 Alpha-Beta Pruning，但比 Minimax 快。如果能調整深度，我認為可以更強。

4. Conclusion

Minmax 在對抗類遊戲有眾多應用，然而依據執行的 game tree 深度，會以指數型加劇其執行時間。Alpha-Beta Pruning 能夠有效的提高 Minimax 執行效率，雖然相對的勝率會些微下降。如果執行次數較少，我認為可以只用大小於來剪枝，保證勝率，且執行時間下降。但如果是大量執行，我認為等於剪枝還是要使用，進一步減少執行時間。最後在編寫自己的 agent 時。如何應用這些學習到的內容，去改善成更具競爭力的 agent。因為是想使用目前所學的，所以並沒有使用其他演算法，如 MCTS 等。所以只能從枝微末節地方去修改，可調整的內容量較少，頗具挑戰性。