人工智慧概論 HW2 黃庭筠 112550105

1. Introduction (5%)

在本次作業中利用 Python 實作 Minimax、Alpha-Beta，擬定遊玩 Connect-four 的遊戲策略，並且透過增加不同策略以及改變 heuristic 的判斷方式，嘗試實作出勝率更高的 AI。

```
==========================================
        DATE: 2025/04/02
        STUDENT NAME: 黃庭筠
        STUDENT ID: 112550105
        ==========================================
```

2.1 Minimax Search (5%)

● Explain the algorithm:

```
if depth == 0 or grid.terminate():
  return get_heuristic(grid), best_moves

if maximizingPlayer == True:
  best = float('-inf')
  for move in valid_moves:
    new_grid = game.drop_piece(grid, move)
    score = minimax(new_grid, depth - 1, False)[0]
    if score == best:
        best_moves.add(move)
    elif score > best:
        best_moves = {move}
        best = score
  return best, best_moves
```

○ How it recursively explores the game tree.

呼叫下一層的函式形成遞迴直到最深的深度，即符合 if depth == 0 的條件調用 get_heuristic 函數預估 score，再一層一層依據不同玩家選擇對應的值及 move，回傳最佳值以及能夠抵達最佳值的 move。

○ How it selects moves for the maximizing and minimizing players.

Maximizing 選擇深一層的節點中具有最大 score 的 move，minimizing 則選擇深一層節點中具有最小 score 的 move。

○ The role of get_heuristic(board) in evaluation.

評估最深處節點的 score，並回傳至其他層，奠定所有 score 的基礎。

● Results & Evaluation:

```
Game 100/100 finished.
execute time 1368959.43 ms
Summary of results:
P1 <function agent_minimax at 0x000002A21CA896C0>
P2 <function agent_reflex at 0x000002A21CA89800>
{'Player1': 100, 'Player2': 0, 'Draw': 0}
==========================================
        DATE: 2025/03/30
        STUDENT NAME: 黃庭筠
        STUDENT ID: 112550105
        ==========================================
```

2.2 Alpha-Beta Pruning (5%)

● Explain the optimization:

○ How α-β pruning reduces unnecessary node expansions.

α 為當前 max 層中最大值，若分支沒有比 α 大的值，則可以剪枝，因為就算遍歷也不會改變最大值，此時的 β 是淺一層的最小值，也就是若在 Maximizing 層，某個分支的最大值 α ≥ β（即 α 已經比 Minimize Player 可接受的最小值還大），則可以剪枝，因為 Minimize Player 不會選擇這個分支，即便繼續遍歷也不會影響最終決策，這樣可以節省不必要的計算時間。

○ When and how pruning occurs in your implementation.

在 maximizing player 的情況下更新 α 值為最大值，在 minimizing player 的情況下更新 β 值為最小值，而加入 if 判斷式，當 α >= β 時剪去不必要的分支，利用 if-break 的方式跳出迴圈。

```
if maximizingPlayer:
    best_value = float('-inf')
    for move in valid_moves:
        new_grid = game.drop_piece(grid, move)
        score, _ = alphabeta(new_grid, depth - 1, False, alpha, beta)

        if score > best_value:
            best_value = score
            best_moves = {move}
        elif score == best_value:
            best_moves.add(move)

        alpha = max(alpha, score)
        if beta <= alpha:
            break

    return best_value, best_moves
```

● Results & Evaluation:

○ Compare **execution time** of **Minimax vs. Alpha-Beta**.

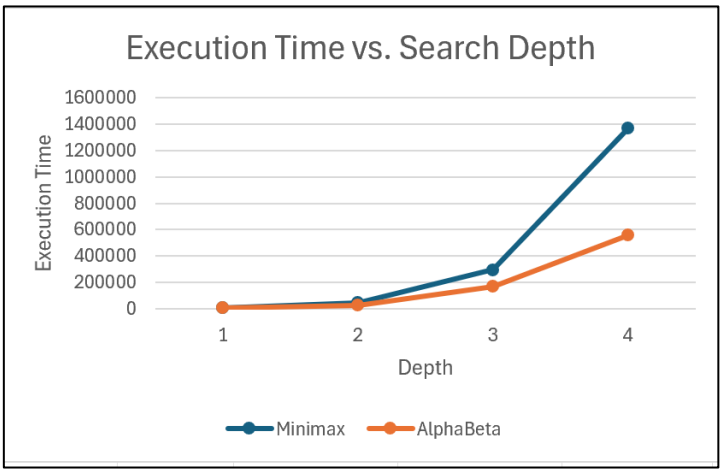
Minimax 的 execution time 幾乎是 Alpha-Beta 的兩倍以上。

```
Game 100/100 finished.
execute time 559017.07 ms
Summary of results:
P1 <function agent_alphabeta at 0x0000019BBA089800>
{'Player1': 95, 'Player2': 5, 'Draw': 0}
========================================
        DATE: 2025/03/30
        STUDENT NAME: 黃庭筠
        STUDENT ID: 112550105
        ========================================
```

○ **Plot: Execution Time vs. Search Depth** to show efficiency improvement.

我嘗試用深度 1~4 做比較，發現當深度增加，透過 Alpha-Beta 改善的時間變化就越顯著。

	1	2	3	4
Minimax	7197.76	44119.11	293927.36	1368959.43
AlphaBeta	5815.9	22188.86	165756.44	559017.07



2.3 Stronger AI Agent (agent_strong()) (5%)

Techniques Used:

○ Explain how you developed agent_strong(), detailing the key strategies and optimizations implemented. Discuss the design choices made to improve decision-making, and heuristic evaluation.

我的 agent_strong()以 alpha-beta 為基礎，做了一些改良，第一個是把 α >= β 改成 α > β，如果把相等的剪掉雖然可以剪掉更多分支節省時間，但有可能剪掉自己贏的機會；另外在進入遞迴判斷前先確認是否有下一步就能夠獲勝的方法，如果有就回傳 inf 為 score 強烈建議這一步，反之如果是敵人有下一步就能獲勝的方法就回傳-inf 為 score 避免動到這一步。

● **Advanced Heuristic Function (get_heuristic_strong()):**

○ How it improves move evaluation over get_heuristic(board).

幾乎跟原本的判斷相同，加入了深度的判斷，如果深度越深則長遠的評估則越為重要，因此將 score 乘上(1 + 0.1*depth)。

○ How it **counters Alpha-Beta (Depth 4)** by prioritizing:

在 agent_strong()中透過右圖中的函式，在更新 move 的 score 前先判斷是否能夠獲勝，若能夠直接獲勝則將 score 設定為正無限大；反之，若敵人即將獲勝，則將 score 設定為負無限大，達成預估 forced wins& blocking imminent opponent wins 的效果。

```
if new_grid.win(maximizingPlayer):
  return float('inf'), {move}
if new_grid.win(3 - maximizingPlayer):
  return float('-inf'), {move}
```

● Results & Evaluation:

○ **Show Win Rate of agent_strong() vs. Alpha-Beta (Depth 4).**

○ Demonstrate that agent_strong() wins more than **50% of games over 100 games**.

```
Game 100/100 finished.
execute time 1760532.13 ms
Summary of results:
P1 <function agent_alphabeta at 0x000001E0FF489940>
P2 <function agent_strong at 0x000001E0FF489A80>
{'Player1': 8, 'Player2': 87, 'Draw': 5}
========================================
        DATE: 2025/04/02
        STUDENT NAME: 黃庭筠
        STUDENT ID: 112550105
        ========================================
```

3. Analysis & Discussion (5%)

● What were the difficulties in designing a strong heuristic?

一開是在實作的時候主要想用往中間區域下的這個策略來做，但就算從中間排序 valid_moves，遍歷時還是會優先選擇 score 高的不會直接下中間，也嘗試過第一步都先下中間但勝率並不會超過 50%，後來決定改為朝著阻擋敵方獲勝以及當自己離或勝差一步時優先檢查，不進入一般的 score 判斷，成功讓勝率超過 50%。

● Did agent_strong() have any weaknesses (e.g., high computation time, failure in some cases)?

- 計算時間過長：加入一些特殊的判斷並且保留了 α = β 的情況，導致被保留的分支較多，計算時間更長。
- 無法應對陷阱或複雜戰術：目前的 agent_strong()無法完全識別陷阱設置、未來幾步的威脅或形成兩邊都可以連成四子情況等複雜戰術的應對。

● How could agent_strong() be further enhanced?

- 可以考慮使用 MCTS 搜索方式會比 Minimax 更有效率，這種方法不像 Minimax 一樣需要遍歷所有情況，而是透過隨機選擇評估最佳策略。
- 加入移動排序技術，例如優先選擇更有潛力的步驟（例如選擇中心列的 move），可以讓 Alpha-Beta 剪枝過程加速，提高搜索效率，減少需要評估的節點數量。

4. Conclusion (5%)

在本次作業中，利用不同方法與 AI 進行 connect-four 的遊戲並嘗試創造更高的勝率，首先利用 Minimax 方法可以 100%打敗預設的 AI，但會花比較久的時間；後利用 AlphaBeta 方法讓特定情況剪枝節省時間，但因為設計問題會剪掉可能獲勝的分支，讓勝率很難到 100%，但幾乎都能維持在 85%以上；最後設計更強的 AI 嘗試打敗 AlphaBeta，我使用的是優先檢查是否能有一步獲勝的方法以及防守對方一步獲勝，回傳對應的 score 增加此 AI 的勝率，並且在 heuristic 函數加入 depth 的影響，讓 AI 在深度深的時候能有更長遠的判斷，最後成功在對戰時達成勝率超過 50%的成果。