Homework 3

40947007S 資工113 張欀齡

1. 針對所需的相關資訊，以下為詳細說明：
    - 機器軟硬體規格
        - 機器：MacBook Pro (13-inch, 2019, Two Thunderbolt 3 ports)
        - 處理器：1.4 GHz 四核心Intel Core i5
        - 記憶體：8 GB 2133 MHz LPDDR3
        - 顯示卡：Intel Iris Plus Graphics 645 1536 MB
    - 作業系統：MAC OS
    - 開發軟體版本：MAC OS Montery 12.6.3
    - 如何執行程式
        - `python3 abMinimax.py`
    - 聯絡電話：0905632902
2. 如何執行 `abMinimax.py`

    詳細註解已寫在程式碼中。
    - alpha-beta pruning
        - 請見程式碼 `abMinimax.py`
        - 如何執行：`python3 abMinimax.py`
3. 測試用輸入檔說明

    使用python `random` 函式來製作輸入檔，盤面大小為 $1 \leq n, m \leq 8$。以下為程式碼：

```python
import random

n = random.randint(1, 8)
m = random.randint(1, 8)

# 產生隨機的01數列
arr = [[random.randint(0, 1) for j in range(m)] for i in range(n)]

# 將數列寫入檔案
with open("input.txt", "w") as f:
    f.write(str(n) + " " + str(m) + "\n")
    for row in arr:
        f.write(" ".join(map(str, row)) + "\n")
```

自己製作的測資：(前三筆為作業提供測資)

下面用來舉例以及列出表現的部分都會利用這裡的測資，可對照測資編號。

測資編號	測資大小	測資
(1)	3*4	1 0 0 1 0 0 0 1 1 1 1 1
(2)	3*4	1 1 1 1 0 0 0 0 1 1 1 1
(3)	4*8	1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0
(4)	2*2	1 0 1 0
(5)	3*5	1 1 0 0 1 0 1 0 1 0 1 0 0 1 0
(6)	5*4	0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 1 1
(7)	2*3	0 1 0 0 1 0
(8)	5*2	1 1 0 1 1 1 1 0 1 1
(9)	7*5	0 1 1 0 0 0 1 1 1 1 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 1 1

測資編號	測資大小	測資
(10)	6*7	0 1 1 1 1 0 1 1 1 0 0 1 0 1 0 0 1 1 0 1 1 0 0 1 0 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 1 1
(11)	3*6	1 0 1 1 1 1 0 1 1 1 1 0 0 0 1 1 0 0
(12)	8*8	0 1 1 0 1 1 0 1 1 1 0 0 1 0 0 0 1 0 0 1 1 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1 1 0 0 0 0 0 1 0 0 1 0 1 1 1 0 0 1 1 0 1 0 0 0 1 0 1 0

4. 程式 `abMinimax.py` 詳細說明

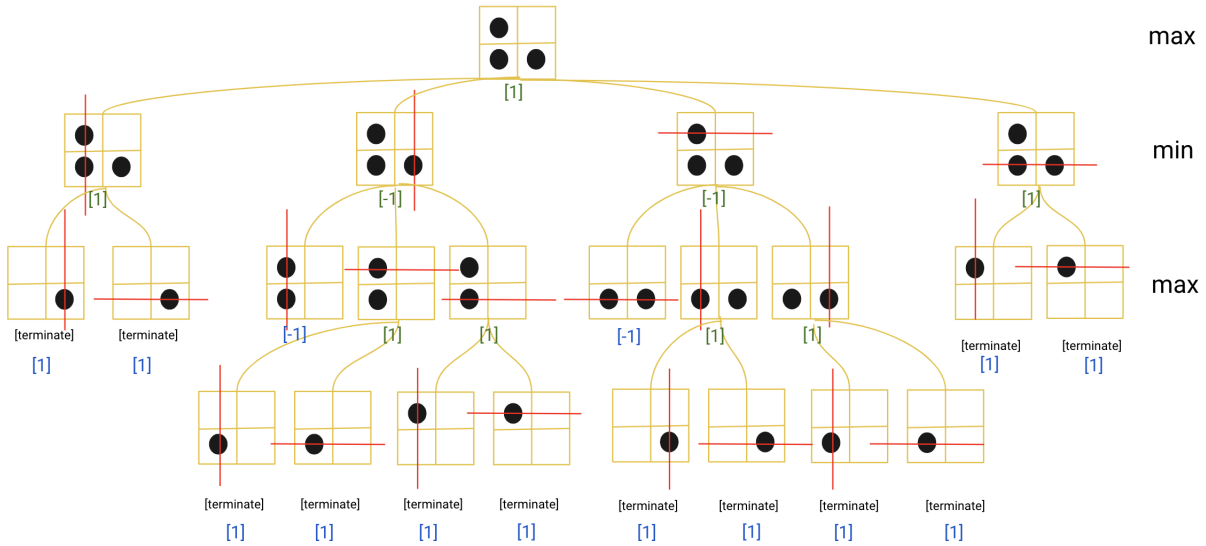
   以下詳細介紹。

   - 使用什麼方法

     使用 alpha-beta pruning 演算法來開發此下棋程式，實作 minimax 演算法，並使用 pruning 進行剪枝。

   - 資料結構

     資料結構如下圖，得到盤面後，列出所有可能移動，將走完的新盤面遞迴傳入下一層，藍色標號為該條路走完的分數，走完一條路可以得到該種走法最後的分數，接著利用 minimax 一層一層回去選擇分數，綠色標號代表 min / max 選擇的分數結果，可以看到下圖最後選完最高層的 max 在 [1, -1, -1, 1] 四種選擇中選擇最大的分數 [1]，因此最後結果為 1。



- 技術

  盤面表示：盤面使用 python 的`list`來存，例如上述圖的例子，我的盤面就是`[[1, 0], [1, 1]]`，因為 python 輸入時會以字串的形式輸入，因此有先將資料 map 成整數型態再存入 list。

  盤面資訊儲存：需要儲存的資訊有新的盤面、此層玩家是 max / min、alpha、beta、當前分數、走步資訊。我將走步資訊分開紀錄，其餘資訊在遞迴時會一併更新傳入。

  走步產生：設置一個全域陣列`best_step`，minimax 的回傳值有一欄為當前移動資訊，因此在 back 回來時會去紀錄並比較，若新的分數比當前分數好，則更新走步資訊。

  如何取答案：最終 max 選擇的分數即為最佳解，走步的第一步則取走步資訊`best_step`的第一筆。

  是否為最佳解：最終 max 選擇的分數即為最佳分數解，因為這個算法的目標是最大化自己的收益或最小化對手的收益，並且假設對手也是採取最佳的策略，Minimax演算法通過遞迴從葉節點向上更新每個節點的值，直到回到根節點。在回溯的過程中，玩家選擇的節點值是子節點值的最大值，而對手選擇的節點值是子節點值的最小值。這樣做的目的是選擇能夠使自己收益最大化的策略，同時假設對手也是採取最佳策略，因此對手會選擇使玩家收益最小化的策略，所以玩家可以根據當前的遊戲狀態找到最佳的行動策略，以達到最好的結果，找到最佳解。

  是否會跑不停：在盤面 8×8 以內的盤面不會跑不停。

  記憶體是否會爆掉：在盤面 8×8 以內的盤面記憶體不會爆掉。

  耗用的時間及空間：

  - 時間複雜度

因為原理是對遊戲的 tree 做 DFS，因此時間複雜度為 $O(bd)$, b 為 tree 的 branching factor，d 為 tree 的 最大深度。

- 空間複雜度

  $O(d)$ ，d 為 tree 的 最大深度，每次只需要紀錄一種遊戲的走法，從 root 到 leaf。

此支程式的技術主要分別為五個函式，以下詳細介紹

- `count_pieces(board)`

  傳入一個盤面，利用 `for` 迴圈計算盤面上的棋子數量，並回傳結果。

- `terminal_state(board)`

  判斷目前的盤面上的棋子數量是否等於 0，回傳遊戲是否結束。

- `get_legal_moves(board)`

  利用 `for` 迴圈從 Row 到 Column 將合法的走法加入 `legal_moves` ， 取得所有根據目前傳入盤面有哪些合法的走法。

- `make_move(board, move)`

  會傳入目前盤面以及走法，根據傳入的走法來消滅盤面上的棋子並計算消滅了幾顆棋子，回傳新盤面及消滅的棋子數。

- `minimax(board, is_max_player, alpha, beta, score)`

  主要實作 minimax 演算法部分，會遞迴去 call 這個函式，傳入盤面，目前是否為 max 玩家，做 pruning 的 alpha、beta 值，以及目前分數，進入函式後，會先判斷目前玩家是否為 max player，接著使用 `for` 迴圈跑在 `get_legal_moves(board)` 取得的所有合法走步，接著再遞迴傳入下一層，back 回來後會計算分數，更新最佳分數，更新 alpha / beta 值，在剪枝的部分則是判斷當 `alpha >= beta` 時就不跑。

以上為實作 alpha - beta pruning 及 minimax algorithm 時用到的技術。

- 表現

使用作業提供的三個測資的表現如下：

盤面大小 (括號內為測資編號)	執行時間
3*4 (1)	0.000906 (s)
3*4 (2)	0.000808 (s)
4*8 (3)	0.072508 (s)

以下使用測資編號（見上面第三題題目自己製作測資之表格）來進行舉例

舉例：

輸入測資編號(4)的測資，執行`python3 abMinimax.py`，我們可以得到輸出結果為

```
Column # : 1
2 points
Total run time = 5.602836608886719e-05 seconds.
```

輸入測資編號(5)的測資，執行`python3 abMinimax.py`，我們可以得到輸出結果為

```
Row # : 1
1 points
Total run time = 0.001088857650756836 seconds.
```

輸入測資編號(6)的測資，執行`python3 abMinimax.py`，我們可以得到輸出結果為

```
Row # : 5
3 points
Total run time = 0.006742238998413086 seconds.
```

輸入測資編號(7)的測資，執行`python3 abMinimax.py`，我們可以得到輸出結果為

```
Column # : 2
2 points
Total run time = 6.604194641113281e-05 seconds.
```

輸入測資編號(8)的測資，執行`python3 abMinimax.py`，我們可以得到輸出結果為

```
Row # : 1
2 points
Total run time = 0.002604961395263672 seconds.
```

輸入測資編號(9)的測資，執行`python3 abMinimax.py`，我們可以得到輸出結果為

```
Column # : 4
3 points
Total run time = 0.13986802101135254 seconds
```

輸入測資編號(10)的測資，執行`python3 abMinimax.py`，我們可以得到輸出結果為

```
Row # : 6
1 points
Total run time = 1.5099132061004639 seconds.
```

輸入測資編號(11)的測資，執行`python3 abMinimax.py`，我們可以得到輸出結果為

```
Row # : 1
3 points
Total run time = 0.001291036605834961 seconds.
```

輸入測資編號(12)的測資，執行`python3 abMinimax.py`，我們可以得到輸出結果為

```
Column # : 1
4 points
Total run time = 83.85330724716187 seconds.
```

可以發現當盤面到 8×8 時，執行時間需要約 80 幾秒，實際時間可能會因為盤面不同而不同，不過和一般僅使用 minimax 演算法相比，效率已經快了非常多。

5. 參考文獻、網站來源及參考哪些部分

   根據`GeeksforGeek`的`Minimax Algorithm`了解程式邏輯，參考以下幾個網站了解alpha-beta pruning 演算法，以 Minimax algorithm 為基礎改寫程式。

   Minimax Algorithm in Game Theory | Set 1 (Introduction) - GeeksforGeeks

   Alpha–beta pruning - Wikipedia

   Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning) - GeeksforGeeks

   Finding optimal move in Tic-Tac-Toe using Minimax Algorithm in Game Theory - GeeksforGeeks

6. 說明此次作業碰到的狀況及困難

在此次作業中，主要有三個困難與狀況，分別為：

1. 一開始不知道從哪裡下手，起初我以為要從根節點走到最底的葉節點，再一一走回來才能算最終分數，這樣的做法我在最底的葉節點回傳的分數是0，因為是回頭才算分數，這樣會導致在做 alpha - beta pruning 剪枝時答案錯誤，因為回傳 0 的關係會導致數據不正確，在深入思考研究後，發現從根節點開始跑時就可以一路算分數算下去，跑到最底的葉節點代表的分數為按照那條路玩的分數，最後再利用minimax 找到最佳解，在這個部分花了比較多時間。

2. 第二個問題是 alpha - beta pruning 剪枝的部分，花了一點時間在理解該如何剪枝，雖然在程式碼中只有短短幾行判斷當 `alpha >= beta` 時就不跑，但一開始有點難理解，後來理解到意思為：因為當我今天是 max 的那方，我會選最大的，而alpha 為 max 目前可能選的最大的值，因此在 max 下一層展開，若`alpha >= beta`，代表此值一定不會被 max 選到，因為 max 有更大的選擇可以選，因此不用跑那個部分。

3. 走步的紀錄以及盤面處理，此部分因為處理盤面及紀錄走步比較多面向需要處理，因此也花了一點時間在處理這個部分，例如：該如何決定此盤面有哪些操作的可能性，以及選擇了一種操作後該如何處理消掉那排棋子，並產生新的盤面。另外一個是走步的部分，這部分也因為遞迴一直傳下去，造成走步的紀錄需要多想一下，詳細的紀錄方式在上述第四題有說明。

在開發此下棋程式並使用了 alpha - beta 剪枝和 minimax 演算法之後，我學到了以下幾點心得：

1. 人工智能算法的優化

   minimax 演算法是一種常見的人工智能算法，用於遊戲策略的決策。然而，在處理較大的遊戲樹時，傳統的 minimax 算法效率較低。通過引入 alpha - beta 剪枝，可以大幅減少搜索的節點數量，從而提高運算效率。這個技巧使得算法可以在更深的層次上進行搜索，提高了遊戲策略的質量和執行速度。

2. 重要性的局部搜索

   minimax 算法在遊戲策略決策中具有廣泛應用，但在處理大型遊戲樹時，遞迴搜索的成本可能變得非常高。alpha - beta 剪枝技術可以適應這種情況，通過僅擴展對當前遊戲策略有意義的節點，避免了不必要的搜索。這樣的局部搜索技巧可以大幅提高算法的效率，同時保持較高的策略品質。

3. 演算法的優化和效能評估

   在開發程式時，我們不僅要實現 minimax 演算法和 alpha - beta 剪枝，還需要考慮算法的效能。評估和比較不同算法的效能是一個重要的過程，可以通過計算算法的搜索深度、節點擴展數量和遊戲策略的品質來衡量。

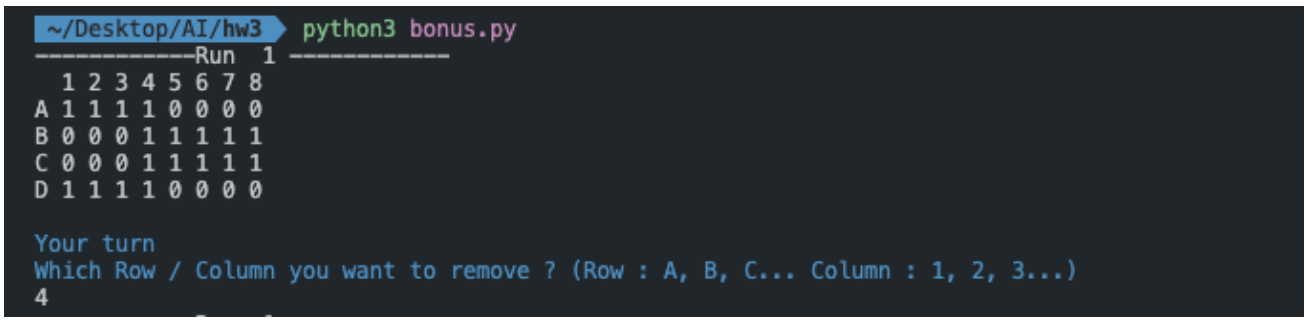
在開發此下棋程式時，我學到了人工智能算法的優化技巧、局部搜索的重要性和演算法的效能評估。此外，還實作了如何應用 alpha - beta 剪枝和 minimax 演算法來開發一個智能的下棋遊戲，這些技巧和經驗對於開發其他類型的遊戲和應用中的人工智能算法也是非常有價值的。

7. 額外加分

   見`bonus.py`，實作command line 的人機遊戲，玩家可和 AI 玩此款遊戲，會去讀input.txt 檔的盤面。

   如何執行：`python3 bonus.py`

   執行後會看見類似以下畫面，可依照指示去選擇想消除的Row / Column。



   遊戲結束後，會計算分數並印出玩家是哪一方。