Project 2

109550087 單宇晟

在這次的 project 中，我選擇用 minimax 來當作我主要的演算法，主要有以下幾點原因：

1. minimax 就是根據 two-player 的遊戲來設計，加上本次遊戲的 outcome 又十分明確，讓 minimax 可以正確地做出 optimal choice。
2. minimax 相對於其他演算法來較為簡單，比較可以根據自己的想法做出調整(像是 evaluation function)
3. 雖然本次遊戲的可能性很多，但只要加上 alpha-beta pruning，minimax 就能更有效率的搜尋整個 game tree，做出更好的決策

程式結構：

```
if np.sum(mapStat == 0) > 8:
    max_depth = 3
else:
    max_depth = 5

def isGameOver(mapStat):
    return np.sum(mapStat == 0) == 0
```

首先，我的 max_depth 會根據剩下的格子數改變，因為越接近結尾的決策就越重要，因此我會在剩下 8 格的時候將 max_depth 提升到 5；後面則是簡單判斷遊戲結束與否的 function。

```python
def getLegalMoves(mapStat):
    moves = []
    free_boxes = np.where(mapStat == 0)

    for i in range(len(free_boxes[0])):
        x = free_boxes[0][i]
        y = free_boxes[1][i]
        # for d in range(1, 7):
        #     moves.append([(x, y), 1, d])
        moves.append([(x, y), 1, 1])
        for l in range(1, 3):
            for direction in range(1, 7):
                next_x, next_y = Next_Node(x, y, direction)
                if next_x >= 0 and next_x < 12 and next_y >= 0 and next_y < 12 and mapStat[next_x][next_y] == 0:
                    flag = True
                    for step in range(l - 1):
                        cross_x, cross_y = Next_Node(next_x, next_y, direction)
                        if cross_x >= 0 and cross_x < 12 and cross_y >= 0 and cross_y < 12 and mapStat[cross_x][cross_y] == 0:
                            next_x, next_y = cross_x, cross_y
                        else:
                            flag = False
                            break
                    if flag:
                        moves.append([(x, y), l + 1, direction])
    return moves
```

```python
def applyMove(state, move):
    new_state = np.array(state)
    start_pos, length, direction = move
    x, y = start_pos
    new_state[x, y] = player
    for i in range(length - 1):
        next_x, next_y = Next_Node(x, y, direction)
        new_state[next_x, next_y] = player
        x, y = next_x, next_y
    return new_state
```

接著是我的 getLegalMove() 和 applyMove()，顧名思義，getLegalMove() 先找到所有合法的路徑，再由 applyMove() 去更新 map state。

```python
def minimax(state, depth, alpha, beta, maximizingPlayer):
    if depth == 0 or isGameOver(state):
        return None, evaluate(state, maximizingPlayer)

    # player = 1 if maximizingPlayer else 2

    possibleMoves = getLegalMoves(state)

    if len(possibleMoves) == 0:
        return None, evaluate(state)

    bestMove = None
    if maximizingPlayer:
        bestScore = float("-inf")
        for move in possibleMoves:
            nextState = applyMove(state, move)
            _, score = minimax(nextState, depth-1, alpha, beta, False)
            if score > bestScore:
                bestScore = score
                bestMove = move
            alpha = max(alpha, score)
            if beta <= alpha:
                break
    else:
        bestScore = float("inf")
        for move in possibleMoves:
            nextState = applyMove(state, move)
            _, score = minimax(nextState, depth-1, alpha, beta, True)
            if score < bestScore:
                bestScore = score
                bestMove = move
            beta = min(beta, score)
            if beta <= alpha:
                break

    return bestMove, bestScore

if player == 1:
    maximizingPlayer = True
else:
    maximizingPlayer = False

bestMove, _ = minimax(mapStat, max_depth, float("-inf"), float("inf"), maximizingPlayer)
```

再來就是我的 minimax 本身了，不過我並沒有特別更改某個步驟，主要就是從所有可能的路徑中，盡可能地選出最好的路徑。

```python
def evaluateMax(mapStat):
    num_empty_boxes = np.sum(mapStat == 0)
    # print(num_empty_boxes)
    num_player_boxes = np.sum(mapStat == 1)

    if num_empty_boxes == 1:
        return 1000
    if num_player_boxes == 0:
        return -1000

    if num_empty_boxes < 5:
        if num_empty_boxes % 2 == 1:
            return 1000
        else:
            return -1000

    if num_empty_boxes == 5:
        return 1000

    return 0
```

最後，是我認為除了演算法本身的選擇之外，這次 project 影響 performance 最大的部分 — evaluation function。由於 minimax 會需要兩個 evaluation function，一個負責找 min-value、另一個則負責 max-value，而我的寫法中，兩者只差了一個負號，所以這裡就只以 max-value 舉例。最基本的，當然就是判斷勝負，如果只剩 1 個格子是空的，就代表我贏；如果沒有剩，代表我走到最後一步，也就是我輸了，所以根據這兩個情況分別給 reward 的值(1000、-1000)。接著是格子剩 5 個以下的狀況，如果我走完之後的格子剛好剩五格，那不管格子的分布、連接狀況如何，有很高的機率會是我贏(這部分是我自己窮舉的)，雖然有時候還是可能會敗北，但對我來說剩五格幾乎等同於獲勝，所以一樣 return 1000；而如果剩不到五格，我自己是假設這個情況下，每個人每次都只走一格，那剩下奇數個格子的話，就也會是我贏(對手-我-對手...)，反之則會是我輸，所以分別 return 1000、-1000。

總結

這次 project 我花的時間主要都是在想 evaluation function 上，在一開始主要架構的部分寫好後，只要我有新想法想要嘗試，我就讓我寫的程式自己跟自己打(舊版本 vs 新版本)，確保我在 evaluation function 中新加上去的 code 有助於程式做出正確的判斷，如果錯了就再想新的評估方式，如此往復，才變成我現在的 evaluation function。