整體架構使用 alpha beta purning

Getstep(): 我把整個 code 包在 GetBestStep()裡面，而這整份 code 並沒有使用到 gameStat 的部分。

```
def Getstep(mapStat, gameStat):
    #Please write your code here
    #TODO
    step = GetBestStep(mapStat)
    return step
```

GetLegalMovements(): 先用範例 code 本身就有的 checkRemainMove 來拿到有哪些點是還沒被 occupied 的，而這些點都能夠單獨成為一個 legal move，所以先把它們加入 list。然後對這些點只找 1, 2, 3 方向有沒有合法長度超過 2 的 legal move，因為方向 4-6 的會有跟 1-3 一樣的 movement，這樣就可以找到所有不重複的合法步數。

```
def GetLegalMovements(mapStat):
    legal_points = checkRemainMove(mapStat)
    legal_movements = []
    for point in legal_points:
        legal_movements.append([point,1,1])
    for point in legal_points:
        for i in range(1,4):
            x, y = point[0], point[1]
            for l in range(2,4):
                x,y = Next_Node(x,y,i)
                if(x>=0 and x<12 and y>=0 and y<12 and mapStat[x][y]==0):
                    legal_movements.append([point,l,i])
                else:
                    break
    return legal_movements
```

UpdateMapStat(): 更新地圖

```
def UpdateMapStat(mapStat, step):
    tmp = mapStat.copy()
    x = step[0][0]
    y = step[0][1]
    l = step[1]
    dir = step[2]
    for i in range(l):
        tmp[x][y] = 1
        x,y = Next_Node(x,y,dir)
    return tmp
```

GetBestStep(): 包含兩個小 function，Min/MaxAgent，兩個 Agent 做的事情基本上相同，先看遊戲有沒有結束，結束了就直接回傳贏或輸，否則就去找所有合法步數然後初始化極大/極小值，然後對所有的 movement 做 MinMax 搜尋，直到層數足夠或是被 alpha 或 beta 剪枝掉，然後就會找到一個最佳的 movement。

```
def GetBestStep(mapStat):
    DEPTH = 4
    def MaxAgent(mapStat, depth, alpha=float('-inf'), beta=float('inf')):
        if isEnd(mapStat):
            return GetScore(mapStat) * 10, [0]
        moves = GetLegalMovements(mapStat)
        bestScore = float('-inf')
        score = float('-inf')
        bestMove = moves[0]
        for move in moves:
            score = MinAgent(UpdateMapStat(mapStat, move), depth, alpha, beta)
            if score > bestScore:
                bestScore = score
                bestMove = move
            alpha = max(alpha, bestScore)
            if alpha >= beta:
                return bestScore, bestMove

        return bestScore, bestMove

    def MinAgent(mapStat, depth, alpha=float('-inf'), beta=float('inf')):
        if isEnd(mapStat):
            return -10 * GetScore(mapStat)
        moves = GetLegalMovements(mapStat)
        bestScore = float('inf')
        score = float('inf')
        for move in moves:
            if depth == DEPTH - 1:
                score = -1 * GetScore(UpdateMapStat(mapStat, move))
            else:
                score, _ = MaxAgent(UpdateMapStat(mapStat, move), depth + 1, alpha, beta)
                if score < bestScore:
                    bestScore = score
            beta = min(beta, bestScore)
            if beta <= alpha:
                return bestScore
        return bestScore
```

這是 GetBestStep 的下半部分，處理掉一些極端值，如果合法步數只有一個的話就不特別計算，如果有太多合法步數的話會算不完，就隨機挑選一個，還有如果他在 minmax 的途中有出現錯誤也能夠透過 random 來稍微修正。

```
BestMove = []
moves = GetLegalMovements(mapStat)
if len(moves) == 1:
    return moves[0]
elif len(moves) > 19:
    print('random')
    return random.choice(moves)
BestScore, BestMove = MaxAgent(mapStat, 0)

if BestMove == 0 or BestScore == float('inf'):
    print('random')
    return random.choice(GetLegalMovements(mapStat))
return BestMove
```

GetScore(): 如果 legal 的起始點是 0 個就代表對方輸了，勝 1 代表對方贏了，因為給 GetScore 的地方都是放”給對手”的地圖，所有讓對方輸，自己可以拿到正分，而同時希望越少點數留在場上越好，越能夠推算出必勝的棋盤。

```
def GetScore(mapStat):
    legal_points = checkRemainMove(mapStat)
    if(len(legal_points)==0):
        return 100
    elif (len(legal_points) == 1):
        return -100
    else:
        return -1 * legal_points
```

isEnd(): 概念和 GetScore 的判斷依據類似，用剩下的點數來知道場上是不是已經沒有地方可以填了。

```
def isEnd(mapStat):
    legal_points = checkRemainMove(mapStat)
    return len(legal_points)==0
```

Experiments: 透過和 Sample_2 對打，不論先手或後手在 10 次的測試之下都能夠全勝，和朋友對打也有全勝和 55 開的紀錄，但我發現我的程式在先手的情況下勝率會略高一點，但也不排除是實驗誤差。

Experiences: 我在最一開始的時候只能夠算 3 層且 15 個 legal move 以下的深度，但在經由方向的限制和剪枝之後，可以算到 4 層且 19 個 legal move 的複雜程度，大大的增加了我的勝率。我也可以算 5 層 13 個 move 的難度，但因為很容易比其他人晚開始計算，所以會比我現在的參數還容易輸。