assignment 3: five_in_a_line_excercise

1. Program performance:

因為我有做 double-free three-in-line, four-in-line-missing-one, free-three-in-line 的情況，所以下棋的情況會跟單純用 UCT( )去下來的不一樣。在單純用 UCT( ) 時，常常會發生一下狀況，明明差一顆或兩顆就能連成一線，但 UCT( )演算法卻會選擇下其他地方。因此我有另外寫 function 去尋找更好的下法。

可以看到在右圖中，player2 根據 UCT( )算法本來要走(0, 1)，但是因為該步驟在(2, 2)的地方有一個 three-in-line 的情況發生，所以反而選擇下(2, 2)。而在 player2 的下一步中，UCT( )算法本來要走(6, 5)，但是因為該步驟在(3, 2)的地方有一個 four-in-line 的情況發生，所以反而選擇下(3, 2)。最後就順利贏下比賽了。我會在接下來的報告講解我是怎麼實作的。

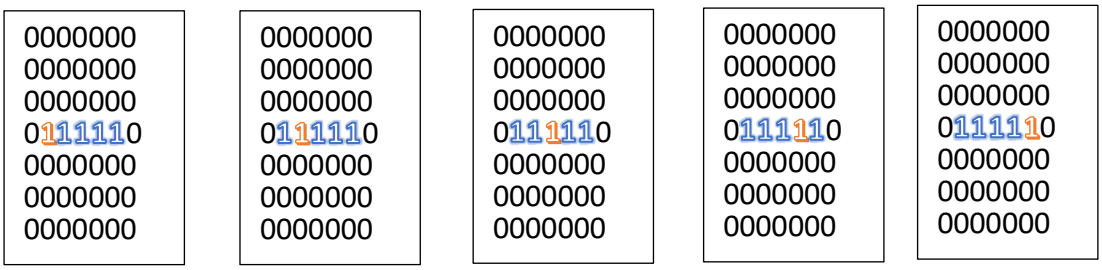
```
No one wins yet
*
better way 2 2 3
play# 18 , Player 2 Best Move: (0, 1)
0100000
0100000
1012201
0010002
2220200
0010001
0202100

No one wins yet
*
play# 19 , Player 1 Best Move: (5, 2)
0100000
0100000
1012201
0010002
2220210
0010001
0202100

No one wins yet
*
better way 3 2 4
play# 20 , Player 2 Best Move: (6, 5)
0100000
0100000
1012201
0010002
2222210
0010001
0202100

Player 2 wins!
```

2. IsFiveInLine( ) 實作:

傳入 IsFiveLine( )的參數是現在所走的那一步(x, y)，所以會以(x, y)為中心，去偵測是否有五點連線的情況。而五個點連成一條線主要有四種狀況，分別是**垂直、水平、上斜對角、下斜對角**。所以我在 IsFiveInLine() 內，分別呼叫四個 function，去檢查這四種狀況。

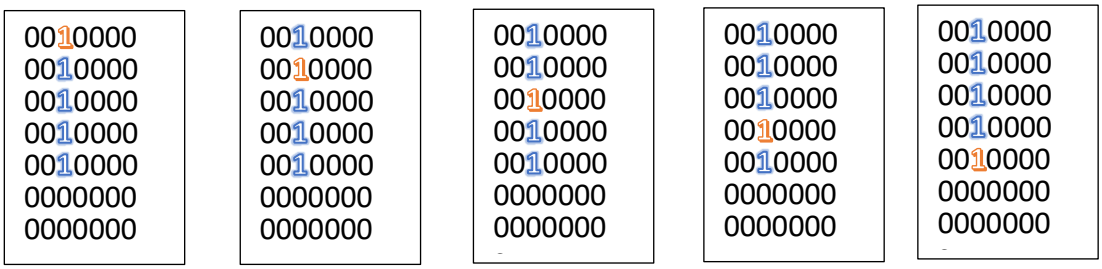
```
if self.Vertical(x, y, player)==True:
    return True
if self.Horizontal(x, y, player)==True:
    return True
if self.High_Diagonal(x, y, player)==True:
    return True
if self.Low_Diagonal(x, y, player)==True:
    return True
```

每種狀況又可以分為五個 case，以水平做舉例：因為傳入 IsFiveLine( )的參數是現在所走的那一步(x, y)，所以有可能是線中五個點當中的任何一點(如下圖)。除了判斷這 5 種狀況之外，也要判斷 player 是不是都是同一人，也要判斷下的位置是不是合法的(是否在棋盤內)。我這邊是把這五種 case 分別處理，利用 for 迴圈去偵測這五種 case:

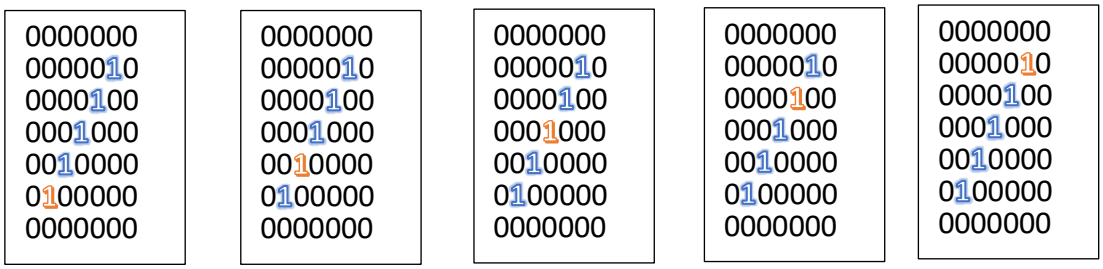


而垂直、上斜對角、下斜對角的情況，也是利用上述方法去判別 5 種 case。

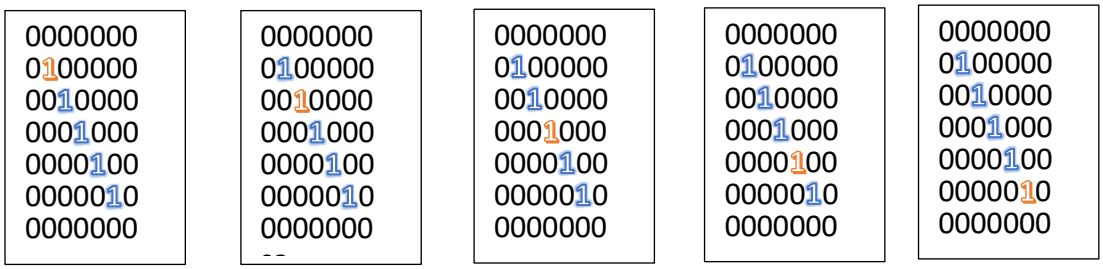
垂直：



上斜對角：



下斜對角：



```
def Horizontal(self, x, y, player):
    p=0
    for step in range(0,5):
        if self.IsOnBoard(x+step, y) and self.board[x+step][y]==player:
            p += 1
            if p==5:
                return True
    p=0
    for step in range(-1,4):
        if self.IsOnBoard(x+step, y) and self.board[x+step][y]==player:
            p += 1
            if p==5:
                return True
    p=0
    for step in range(-2,3):
        if self.IsOnBoard(x+step, y) and self.board[x+step][y]==player:
            p += 1
            if p==5:
                return True
    p=0
    for step in range(-3,2):
        if self.IsOnBoard(x+step, y) and self.board[x+step][y]==player :
            p += 1
            if p==5:
                return True
    p=0
    for step in range(-4,1):
        if self.IsOnBoard(x+step, y) and self.board[x+step][y]==player:
            p += 1
            if p==5:
                return True
    return False
```

Fig. 利用 for 迴圈去偵測 5 種 case

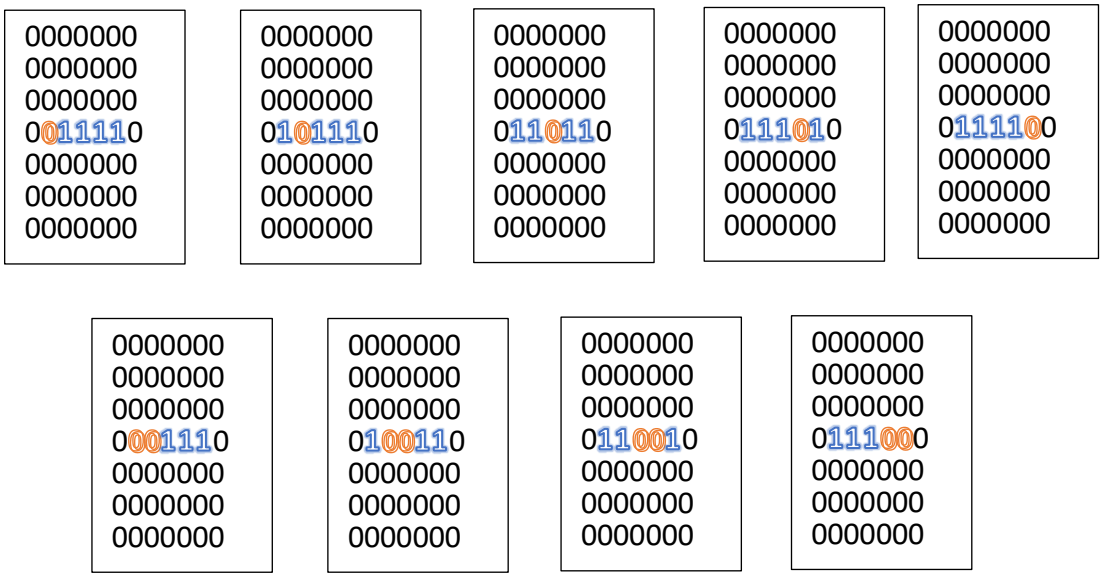
3. double-free three-in-line, four-in-line-missing-one, free-three-in-line 檢查

除了基本的 MCTS 算法之外，我還有實作一些演算法，去偵測 double-free three-in-line, four-in-line-missing-one, free-three-in-line 的情況。在 UCTPlayGame( )內，呼叫完 UCT( )後，我會互叫 detect_missing_one_in_line( )，去檢查是否有更好的下法。而在該 function 裡面，我會以 for 迴圈，遍歷整個 board 上的點，並以每個點為中心，去偵測他周遭是否有 three-in-line, four-in-line, free-three-in-line 的狀況。做法是把每個點輸入 Vertical_optimize( ), Horizontal_optimize( ), High_Diagonal_optimize( ),Low_Diagonal_optimize( )，看看在在垂直、水平、上斜對角、下斜對角是否有 three-in-line, four-in-line, free-three-in-

line 發生。

```
def detect_missing_one_in_line(state, m):
    player = 3 - state.playerJustMoved
    ori_x, ori_y = (m[0], m[1])
    for i in range(7):
        for j in range(7):
            p, x, y =  Vertical_optimize(state, i, j, player)
            if p==3 or p==4:
                return 1, x, y
            p, x, y =  Horizontal_optimize(state, i, j, player)
            if p==3 or p==4:
                return 1, x, y
            p, x, y =  High_Diagonal_optimize(state, i, j, player)
            if p==3 or p==4:
                return 1, x, y
            p, x, y =  Low_Diagonal_optimize(state, i, j, player)
            if p==3 or p==4:
                return 1, x, y
    return 0, ori_x, ori_y
```

接下來會以 Horizontal_optimize( )去具體解釋我怎麼偵測 three-in-line, four-in-line, free-three-in-line。實際作法跟 IsFiveInLine( )還蠻像的，我會去判別說，在之前講的那 5 種 case 中，是否有少一個就能連成一線的情況發生，或缺少兩個就能連成一線的情況發生，如下圖。只要去數每一個 case 當中，board 上面為 0 的位置有幾個，就能知道是 three-in-line 或 four-in-line。



```
def Horizontal_optimize(state, x, y, player):
    p=0
    for step in range(0,5):
        if state.IsOnBoard(x+step, y) and state.board[x+step][y]==player:
            p += 1
    if p==3 or p==4:
        for step in range(0,5):
            if state.IsOnBoard(x+step, y) and state.board[x+step][y]==0:
                return p, x+step, y
    p=0
    for step in range(-1,4):
        if state.IsOnBoard(x+step, y) and state.board[x+step][y]==player:
            p += 1
    if p==3 or p==4:
        for step in range(-1,4):
            if state.IsOnBoard(x+step, y) and state.board[x+step][y]==0:
                return p, x+step, y
    p=0
    for step in range(-2,3):
        if state.IsOnBoard(x+step, y) and state.board[x+step][y]==player:
            p += 1
    if p==3 or p==4:
        for step in range(-2,3):
            if state.IsOnBoard(x+step, y) and state.board[x+step][y]==0:
                return p, x+step, y
    p=0
    for step in range(-3,2):
        if state.IsOnBoard(x+step, y) and state.board[x+step][y]==player :
            p += 1
    if p==3 or p==4:
        for step in range(-3,2):
            if state.IsOnBoard(x+step, y) and state.board[x+step][y]==0:
                return p, x+step, y
    p=0
    for step in range(-4,1):
        if state.IsOnBoard(x+step, y) and state.board[x+step][y]==player:
            p += 1
    if p==3 or p==4:
        for step in range(-4,1):
            if state.IsOnBoard(x+step, y) and state.board[x+step][y]==0:
                return p, x+step, y
    return 0, x+step, y
```

Fig. 計算 board 上面為 0 的位置有幾個