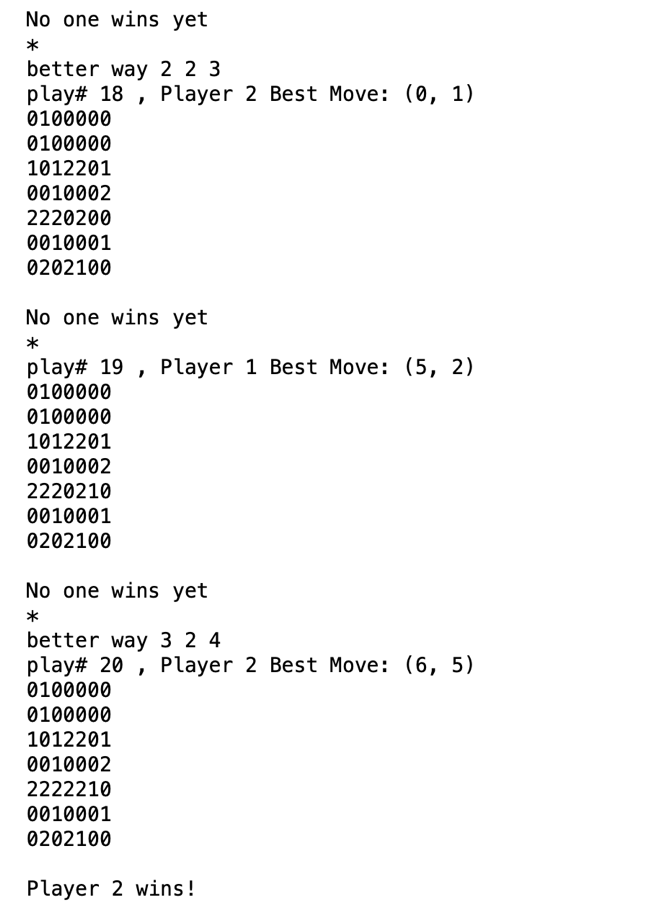
**assignment 3:** [**five\_in\_a\_line\_excercise**](https://eeclass.nthu.edu.tw/filedownload/697155)

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。

1. **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。

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:

0000000

0000000

0000000

0**11111**0

0000000

0000000

0000000

0000000

0000000

0000000

0**11111**0

0000000

0000000

0000000

0000000

0000000

0000000

0**11111**0

0000000

0000000

0000000

0000000

0000000

0000000

0**11111**0

0000000

0000000

0000000

0000000

0000000

0000000

0**11111**0

0000000

0000000

0000000

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：

00**1**0000

00**1**0000

00**1**0000

00**1**0000

00**1**0000

0000000

0000000

00**1**0000

00**1**0000

00**1**0000

00**1**0000

00**1**0000

0000000

0000000

,

00**1**0000

00**1**0000

00**1**0000

00**1**0000

00**1**0000

0000000

0000000

s

00**1**0000

00**1**0000

00**1**0000

00**1**0000

00**1**0000

0000000

0000000

x

00**1**0000

00**1**0000

00**1**0000

00**1**0000

00**1**0000

0000000

0000000

s

上斜對角：

0000000

00000**1**0

0000**1**00

000**1**000

00**1**0000

0**1**00000

0000000

0000000

00000**1**0

0000**1**00

000**1**000

00**1**0000

0**1**00000

0000000

0000000

00000**1**0

0000**1**00

000**1**000

00**1**0000

0**1**00000

0000000

0000000

00000**1**0

0000**1**00

000**1**000

00**1**0000

0**1**00000

0000000

0000000

00000**1**0

0000**1**00

000**1**000

00**1**0000

0**1**00000

0000000

下斜對角：

0000000

0**1**00000

00**1**0000

000**1**000

0000**1**00

00000**1**0

0000000

0000000

0**1**00000

00**1**0000

000**1**000

0000**1**00

00000**1**0

0000000

sa

0000000

0**1**00000

00**1**0000

000**1**000

0000**1**00

00000**1**0

0000000

0000000

0**1**00000

00**1**0000

000**1**000

0000**1**00

00000**1**0

0000000

0000000

0**1**00000

00**1**0000

000**1**000

0000**1**00

00000**1**0

0000000

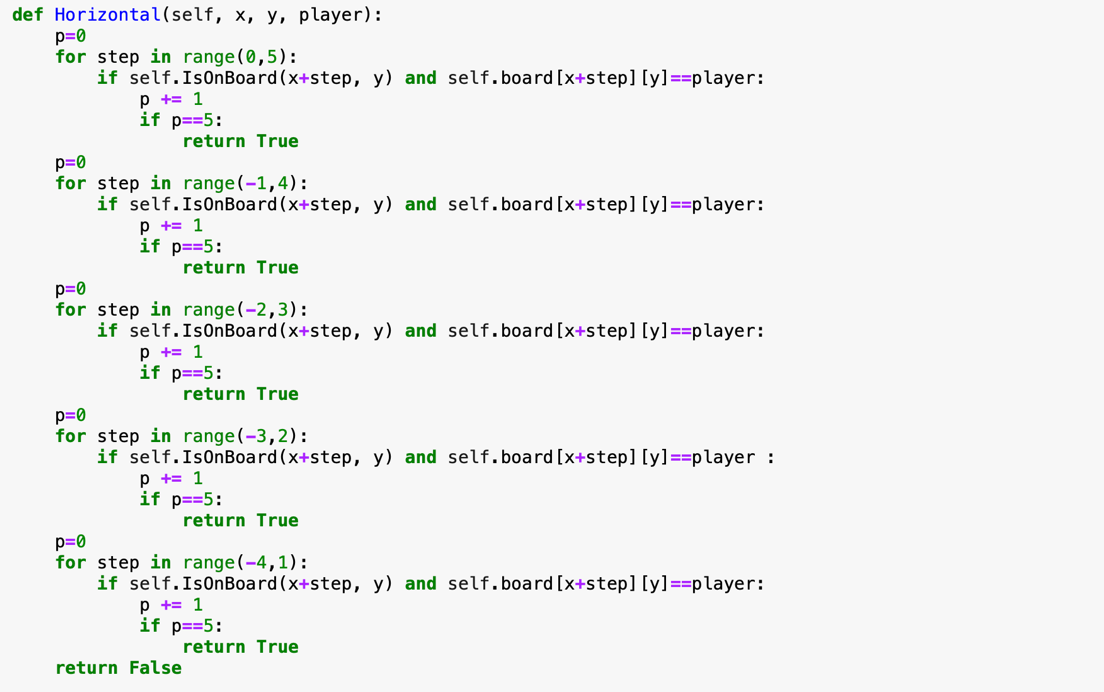
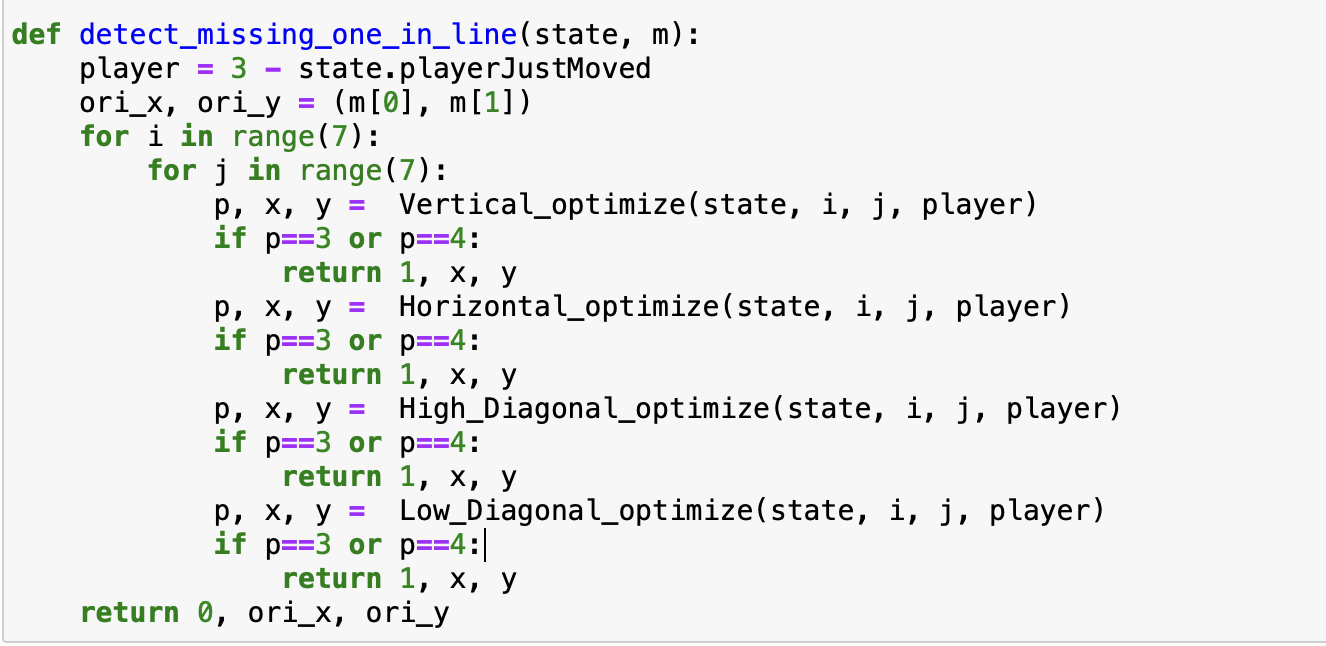


Fig. 利用for迴圈去偵測5種case

1. **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。

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。

0000000

0000000

0000000

0**01111**0

0000000

0000000

0000000

0000000

0000000

0000000

0**10111**0

0000000

0000000

0000000

0000000

0000000

0000000

0**11011**0

0000000

0000000

0000000

0000000

0000000

0000000

0**11101**0

0000000

0000000

0000000

0000000

0000000

0000000

0**11110**0

0000000

0000000

0000000

0000000

0000000

0000000

0**00111**0

0000000

0000000

0000000

0000000

0000000

0000000

0**10011**0

0000000

0000000

0000000

0000000

0000000

0000000

0**11001**0

0000000

0000000

0000000

0000000

0000000

0000000

0**11100**0

0000000

0000000

0000000



Fig. 計算board上面為0的位置有幾個