HW2_game planning

Name: Jordan Wu Section R3 CRN: 69112
Shih-Jie Chang Section Q3 CRN:31425
date:2/22/2019

Section I:

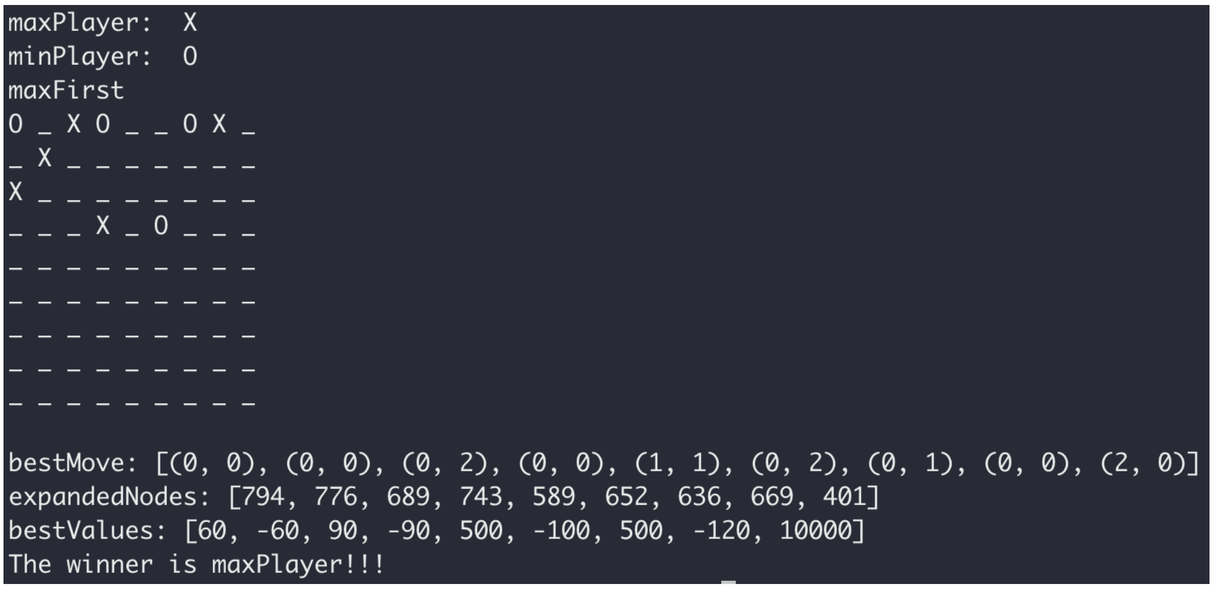
For this problem, we converted the problem to a matrix, where each row represents a certain “pentomino placement choice”, while each column represents the pentomino piece used and the linearized row-major position on the board it took. Columns -n to -1 (n is number of pentominoes) represents the pentomino piece, for example if a row[-1] == 1, then the first pentomino is chosen. Columns 0 to m-1 (m is the linearized board size) represents the positions taken by the pentomino. For example, if row[0:4] = 1, it means this pentomino is occupying (0,0), (0,1) (0,2)...(0,4). This sparse matrix is represented using 2 dictionaries. The first, Y, maps rows to the column indexes where the value is 1. While the second X, maps the columns to rows where the value is 1. This allows for Algorithm X to run much faster due to the sparse representation.

To initialize this representation, we first rotated and flipped the pentominoes in every different orientation, then tried to place the pentomino at every possible spot on the board. If a position and orientation is possible, we add to the row.

Algorithm X then choses a column (either pentomino choice or board position) to fill out first. This will be the column with the minimum number of “pentomino placement choice”, which is a Least Remaining Values heuristic. Then we iteratively chose a “pentomino placement choice” from the column, and remove all “pentomino placement choices” that conflicts, a sort of forward checking. Then we recurse and solve on the remaining “pentomino placement choices”. If were unable to fill out all of the columns at the bottom of the recursion, we backtrack and chose another “pentomino placement choice” in the underlined step.

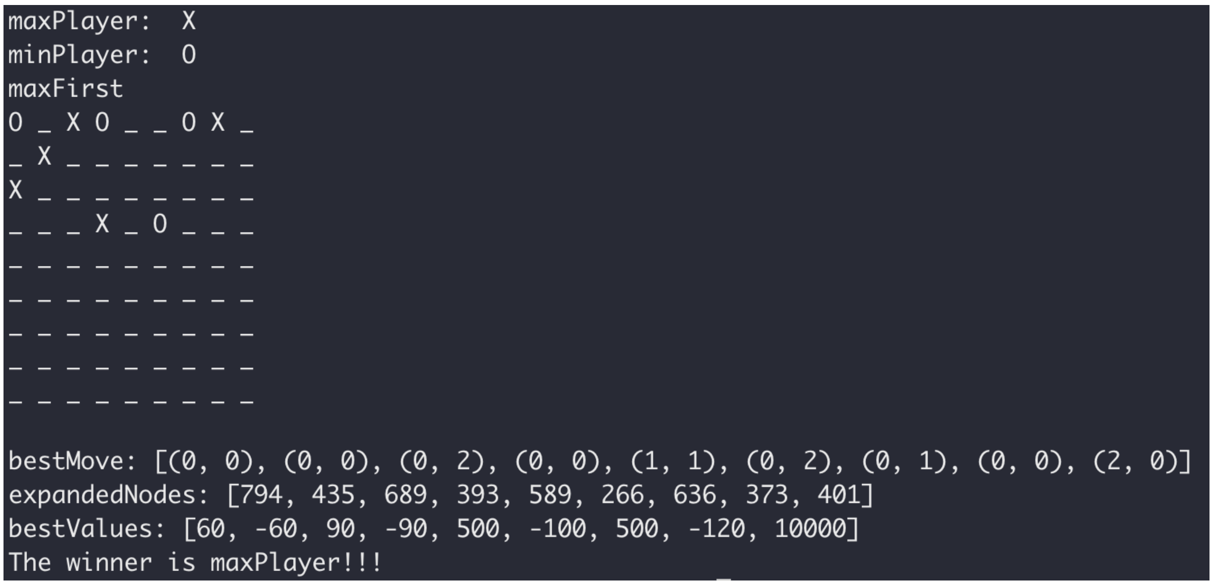
Section II:

1. Offensive(minimax) vs defensive(minimax), Offensive first



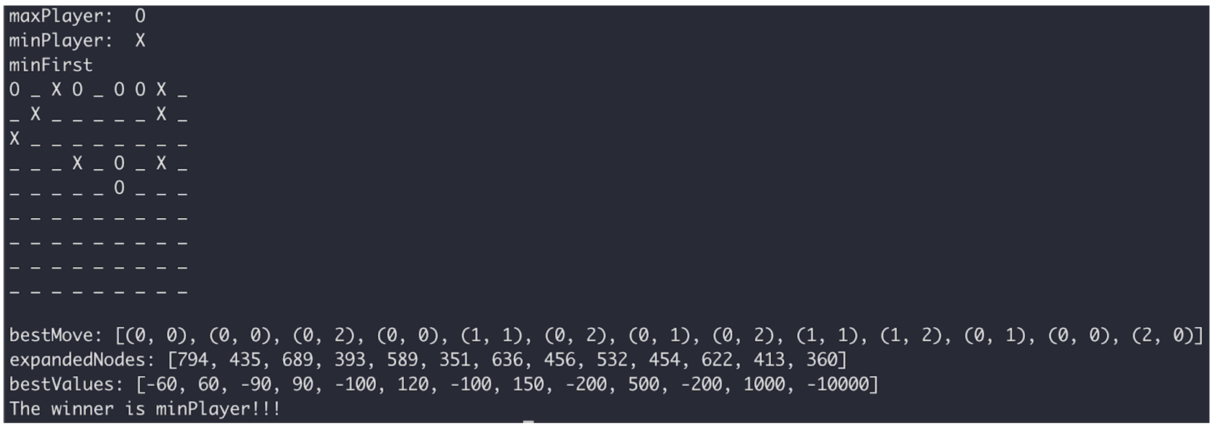
```
maxPlayer:  X
minPlayer:  O
maxFirst
O _ X O _ _ O X _
_ X _ _ _ _ _ _ _
X _ _ _ _ _ _ _ _
_ _ _ X _ O _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _

bestMove: [(0, 0), (0, 0), (0, 2), (0, 0), (1, 1), (0, 2), (0, 1), (0, 0), (2, 0)]
expandedNodes: [794, 776, 689, 743, 589, 652, 636, 669, 401]
bestValues: [60, -60, 90, -90, 500, -100, 500, -120, 10000]
The winner is maxPlayer!!!
```

2. Offensive(minimax) vs Defensive(alpha-beta), Offensive first



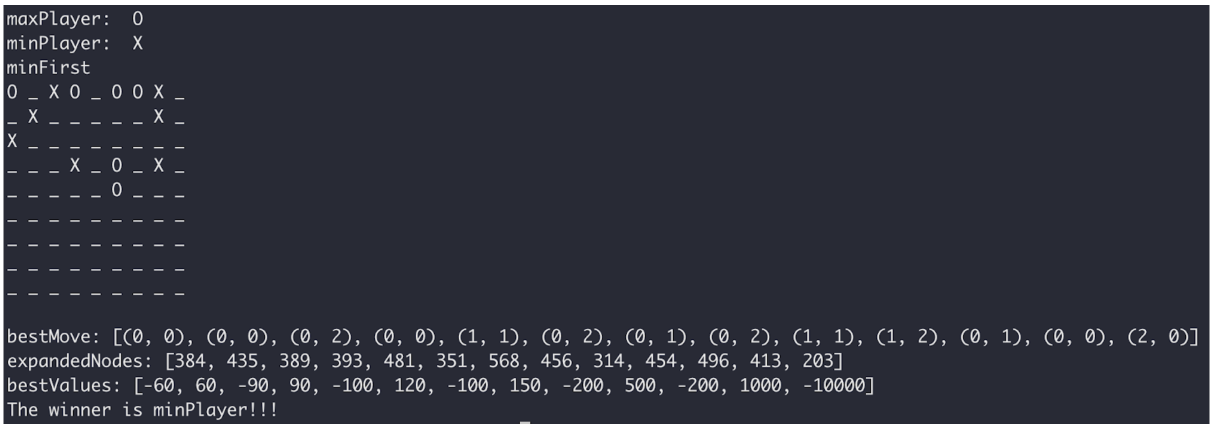
```
maxPlayer:  X
minPlayer:  O
maxFirst
O _ X O _ _ O X _
_ X _ _ _ _ _ _ _
X _ _ _ _ _ _ _ _
_ _ _ X _ O _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _

bestMove: [(0, 0), (0, 0), (0, 2), (0, 0), (1, 1), (0, 2), (0, 1), (0, 0), (2, 0)]
expandedNodes: [794, 435, 689, 393, 589, 266, 636, 373, 401]
bestValues: [60, -60, 90, -90, 500, -100, 500, -120, 10000]
The winner is maxPlayer!!!
```

3. Offensive(alpha-beta) vs Defensive(minimax), Defensive First



```
maxPlayer:  O
minPlayer:  X
minFirst
O _ X O _ O O X _
_ X _ _ _ _ _ X _
X _ _ _ _ _ _ _ _
_ _ _ X _ O _ X _
_ _ _ _ _ O _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _

bestMove: [(0, 0), (0, 0), (0, 2), (0, 0), (1, 1), (0, 2), (0, 1), (0, 2), (1, 1), (1, 2), (0, 1), (0, 0), (2, 0)]
expandedNodes: [794, 435, 689, 393, 589, 351, 636, 456, 532, 454, 622, 413, 360]
bestValues: [-60, 60, -90, 90, -100, 120, -100, 150, -200, 500, -200, 1000, -10000]
The winner is minPlayer!!!
```

4. Offensive(alpha-beta) vs Defensive(alpha-beta), Defensive First



```
maxPlayer:  O
minPlayer:  X
minFirst
O _ X O _ O O X _
_ X _ _ _ _ _ X _
X _ _ _ _ _ _ _ _
_ _ _ X _ O _ X _
_ _ _ _ _ O _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _

bestMove: [(0, 0), (0, 0), (0, 2), (0, 0), (1, 1), (0, 2), (0, 1), (0, 2), (1, 1), (1, 2), (0, 1), (0, 0), (2, 0)]
expandedNodes: [384, 435, 389, 393, 481, 351, 568, 456, 314, 454, 496, 413, 203]
bestValues: [-60, 60, -90, 90, -100, 120, -100, 150, -200, 500, -200, 1000, -10000]
The winner is minPlayer!!!
```

Section III:

Our evaluation function prioritizes:

1. Endgame scenarios. If opponent wins, return the “worst” utility (-10000 if player is MAX), if player wins, return the “best” utility (10000 if player is MAX). These two situations cannot both happen, since in our minimax/alphabeta, we prevent making moves if one player is detected to win.

2. Same utility per unblocked-two-in-a-row and prevent-three-in-a-row (300 if player is MAX)

3. Corner utility just like predefined (30 if player is MAX)

If player is MIN, we multiply the final util by -1.

By just checking to see if it will lose, our agent is able to prevent playing itself into a situation where it is guaranteed to lose, a trait which the predefined agents did not show.

<u>Our agent wins 18 out of 18 games (100% win rate) where the start board is empty at the beginning (9 possible startingboardIdx * 2 choices for who goes first).</u>

<u>In 2 more additional games where I “set” the board up to give MAX the advantage. However, MIN(our agent) still wins.</u>

<u>In Total: 100% win rate for our agent.</u>

<u>Results: (NOTE min is our designed agent)</u>

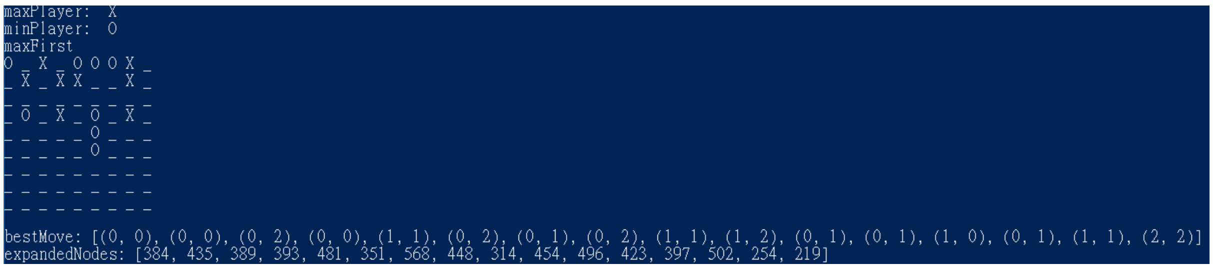
```
startBoardIdx: 0
maxFirst
expandedNodes: [448, 429, 405, 535, 372, 620, 455, 342, 451, 239]
winner minAgent

startBoardIdx: 1
maxFirst
expandedNodes: [384, 440, 453, 407, 530, 379, 634, 399, 381, 428, 389, 507, 481, 50]
winner minAgent

startBoardIdx: 2
maxFirst
expandedNodes: [383, 429, 391, 392, 442, 343, 484, 464, 588, 223, 372, 480, 220, 434,
474, 338, 450, 184]
winner minAgent

startBoardIdx: 3
maxFirst
expandedNodes: [384, 435, 389, 393, 481, 376, 550, 450, 311, 440, 537, 399, 362, 452,
232, 125]
winner minAgent

startBoardIdx: 4
```

```
maxFirst
expandedNodes: [384, 435, 389, 393, 481, 351, 568, 448, 314, 454, 496, 423, 397, 502,
254, 219]
winner minAgent

startBoardIdx: 5
maxFirst
expandedNodes: [384, 435, 389, 393, 481, 376, 550, 435, 303, 430, 453, 400, 357, 489,
352, 165]
winner minAgent

startBoardIdx: 6
maxFirst
expandedNodes: [383, 431, 390, 389, 481, 373, 550, 432, 303, 427, 468, 256, 353, 322,
171, 109]
winner minAgent

startBoardIdx: 7
maxFirst
expandedNodes: [384, 435, 389, 393, 480, 377, 550, 436, 304, 432, 461, 401, 373, 492,
363, 176]
winner minAgent

startBoardIdx: 8
maxFirst
expandedNodes: [383, 432, 390, 390, 481, 374, 550, 433, 303, 429, 461, 400, 365, 489,
354, 176]
winner minAgent

startBoardIdx: 0
minFirst
expandedNodes: [448, 429, 405, 535, 388, 620, 447, 342, 443, 523, 413, 419, 503, 408,
172]
winner minAgent

startBoardIdx: 1
minFirst
expandedNodes: [384, 440, 453, 400, 530, 356, 634, 399, 381, 436, 236]
winner minAgent

startBoardIdx: 2
minFirst
expandedNodes: [383, 429, 391, 392, 442, 280, 621, 375, 481, 608, 487, 336, 397, 316,
448, 344, 243, 418, 119, 306, 339, 335, 37]
winner minAgent

startBoardIdx: 3
minFirst
expandedNodes: [384, 435, 389, 393, 481, 362, 557, 439, 311, 436, 223]
winner minAgent
```

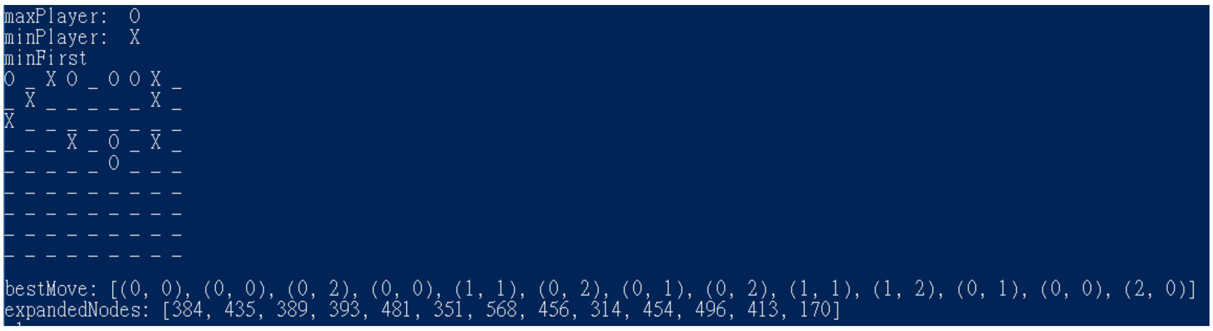
```
startBoardIdx: 4
minFirst
expandedNodes: [384, 435, 389, 393, 481, 351, 568, 456, 314, 454, 496, 413, 170]
winner minAgent

startBoardIdx: 5
minFirst
expandedNodes: [384, 435, 389, 393, 481, 362, 550, 443, 303, 438, 210]
winner minAgent

startBoardIdx: 6
minFirst
expandedNodes: [383, 431, 390, 389, 481, 361, 550, 444, 303, 435, 215]
winner minAgent

startBoardIdx: 7
minFirst
expandedNodes: [384, 435, 389, 393, 480, 362, 550, 445, 304, 439, 209]
winner minAgent

startBoardIdx: 8
minFirst
expandedNodes: [383, 432, 390, 390, 481, 361, 550, 443, 303, 437, 209]
winner minAgent

myAgentScore: 18
maxAgentScore: 0
```

Results for custom games where starting board gives max advantage:

```
startBoardIdx: 1
maxFirst
expandedNodes: [706, 400, 626, 439, 532, 395, 516, 191]
winner minAgent

startBoardIdx: 0
minFirst
expandedNodes: [331, 298, 605, 287, 270, 339, 85]
winner minAgent
```

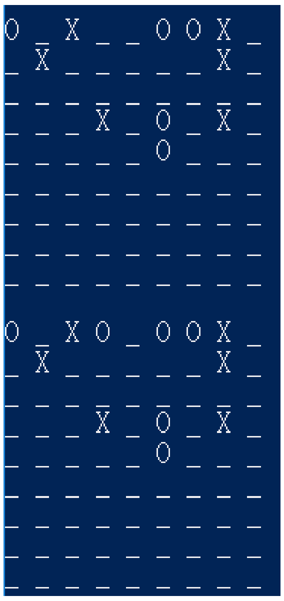


```
maxPlayer:  X
minPlayer:  O
maxFirst
O _ X _ O O O X _
_ X _ X X _ _ X _
_ _ _ _ _ _ _ _ _
_ O _ X _ O _ X _
_ _ _ _ _ O _ _ _
_ _ _ _ _ O _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _

bestMove: [(0, 0), (0, 0), (0, 2), (0, 0), (1, 1), (0, 2), (0, 1), (0, 2), (1, 1), (1, 2), (0, 1), (0, 1), (1, 0), (0, 1), (1, 1), (2, 2)]
expandedNodes: [384, 435, 389, 393, 481, 351, 568, 448, 314, 454, 496, 423, 397, 502, 254, 219]
```

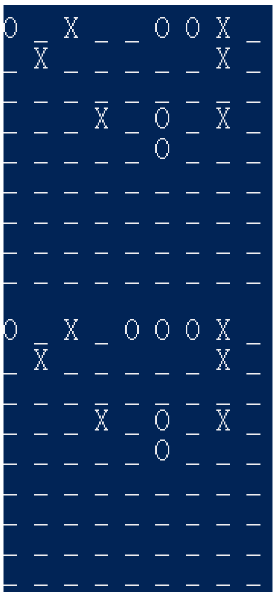


```
maxPlayer:  O
minPlayer:  X
minFirst
O _ X O _ O O X _
_ X _ _ _ _ _ X _
X _ _ _ _ _ _ _ _
_ _ _ X _ O _ X _
_ _ _ _ _ O _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _

bestMove: [(0, 0), (0, 0), (0, 2), (0, 0), (1, 1), (0, 2), (0, 1), (0, 2), (1, 1), (1, 2), (0, 1), (0, 0), (2, 0)]
expandedNodes: [384, 435, 389, 393, 481, 351, 568, 456, 314, 454, 496, 413, 170]
```

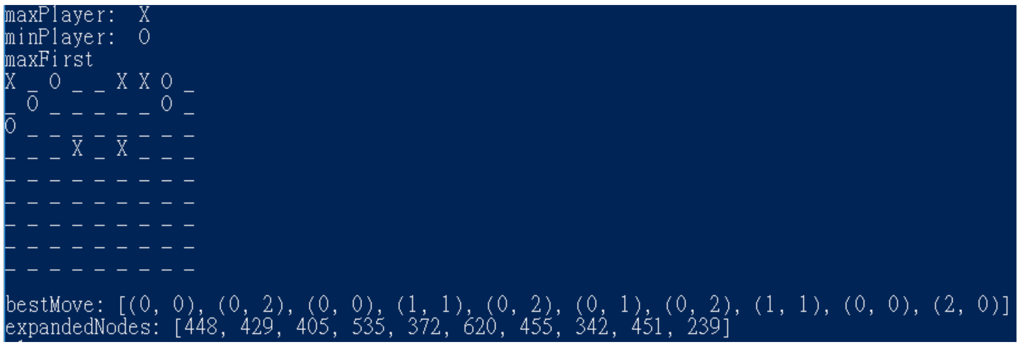
The above two games are all the same except that one is maxFirst(predefined agent first) and the other is minFirst(out agent first). Then we can see the difference appear at the step below(Left is maxFirst and right is minFirst). The offensive agent does not prevent its counterpart from winning in the first board. So it chooses the position of (0,0) in board(1). On the other hand, our agent chooses another position to prevent the counterpart from winning.



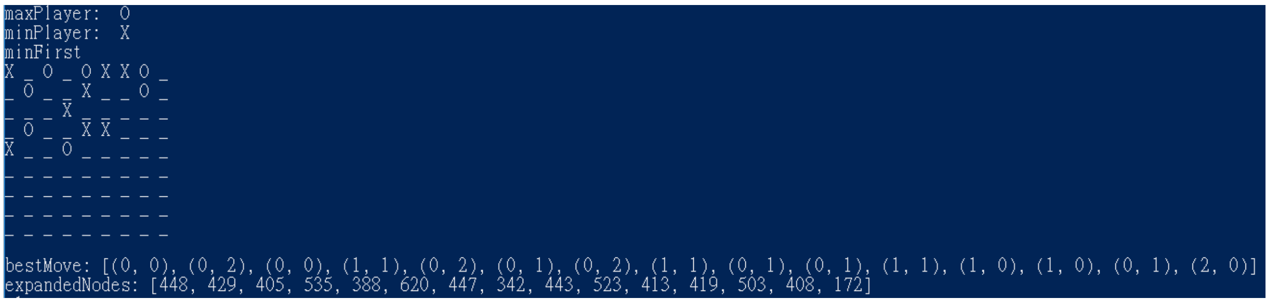
```
O _ X _ _ O O X _
_ X _ _ _ _ _ X _
_ _ _ _ _ _ _ _ _
_ _ _ X _ O _ X _
_ _ _ _ _ O _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _

O _ X O _ O O X _
_ X _ _ _ _ _ X _
_ _ _ _ _ _ _ _ _
_ _ _ X _ O _ X _
_ _ _ _ _ O _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
```



```
O _ X _ _ O O X _
_ X _ _ _ _ _ X _
_ _ _ _ _ _ _ _ _
_ _ _ X _ O _ X _
_ _ _ _ _ O _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _

O _ X _ O O O X _
_ X _ _ _ _ _ X _
_ _ _ _ _ _ _ _ _
_ _ _ X _ O _ X _
_ _ _ _ _ O _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
```

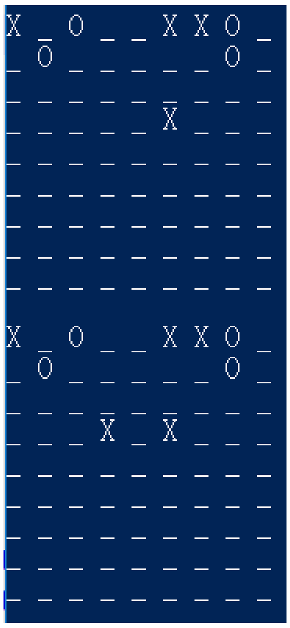


```
maxPlayer:  X
minPlayer:  O
maxFirst
X _ O _ _ X X O _
_ O _ _ _ _ _ O _
O _ _ _ _ _ _ _ _
_ _ _ X _ X _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _

bestMove: [(0, 0), (0, 2), (0, 0), (1, 1), (0, 2), (0, 1), (0, 2), (1, 1), (0, 0), (2, 0)]
expandedNodes: [448, 429, 405, 535, 372, 620, 455, 342, 451, 239]
```

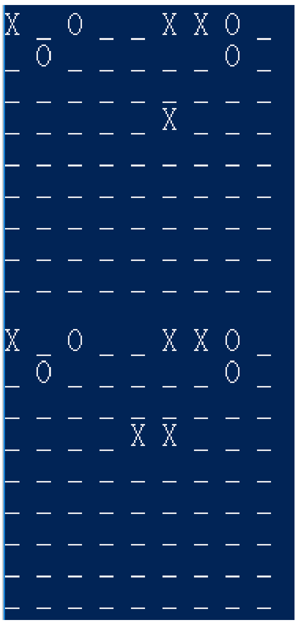


```
maxPlayer:  O
minPlayer:  X
minFirst
X _ O _ O X X O _
_ O _ _ X _ _ O _
_ _ _ X _ _ _ _ _
_ O _ _ X X _ _ _
X _ _ O _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _

bestMove: [(0, 0), (0, 2), (0, 0), (1, 1), (0, 2), (0, 1), (0, 2), (1, 1), (0, 1), (0, 1), (1, 1), (1, 0), (1, 0), (0, 1), (2, 0)]
expandedNodes: [448, 429, 405, 535, 388, 620, 447, 342, 443, 523, 413, 419, 503, 408, 172]
```

The above are two other examples that starts at board with index 0. From the example below, we can also see that the two agents also make different decision when it comes to preventing the counterpart from winning.



```
X _ O _ _ X X O _
_ O _ _ _ _ _ O _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ X _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _

X _ O _ _ X X O _
_ O _ _ _ _ _ O _
_ _ _ _ _ _ _ _ _
_ _ _ X _ X _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
```



```
X _ O _ _ X X O _
_ O _ _ _ _ _ O _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ X _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _

X _ O _ _ X X O _
_ O _ _ _ _ _ O _
_ _ _ _ _ _ _ _ _
_ _ _ _ X X _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
```

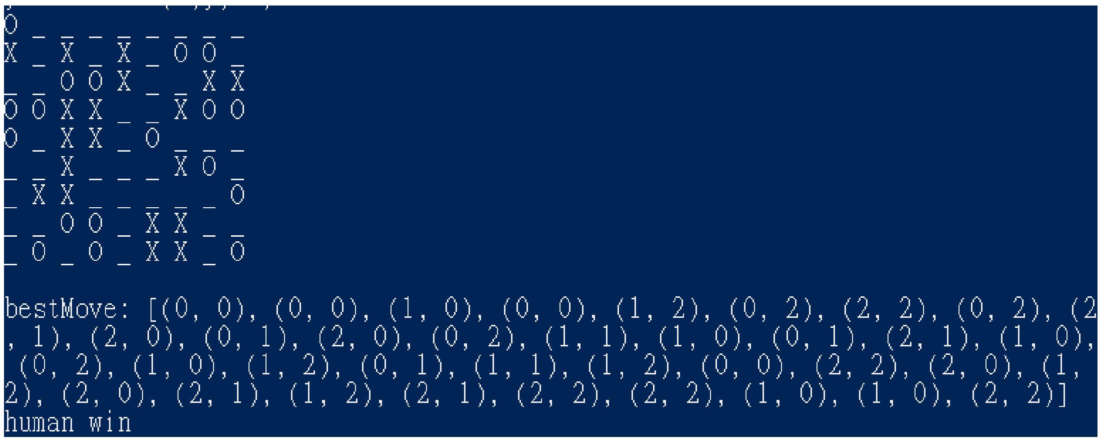
In conclusion, our designed agent has a 100% win rate against MAX agent due to it being able to give “worst” utility to paths that will lead to it losing.

Section IV:

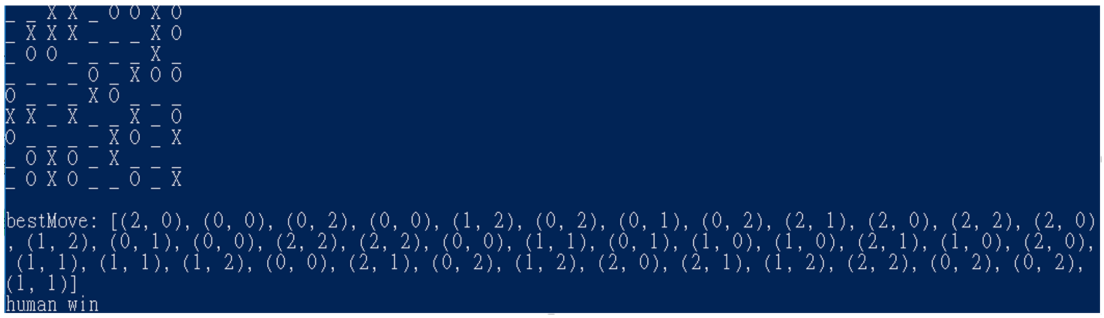
The biggest difference between me and the agent is that computer always choose the corner point, while I tend to choose the point that disables my opposite to make two points in a row. Therefore, I am always the first one to make two points in row.
The advantage of the agent’s algorithm is that is can occupy most corners, which can be

connected in row,col and diagonal. So it will be easier for the agent to make two points in row at the end of the game.
However, because it does not prevent its counterpart from composing two points in row, it may lose because of its late composition of two points in row.

Human First:

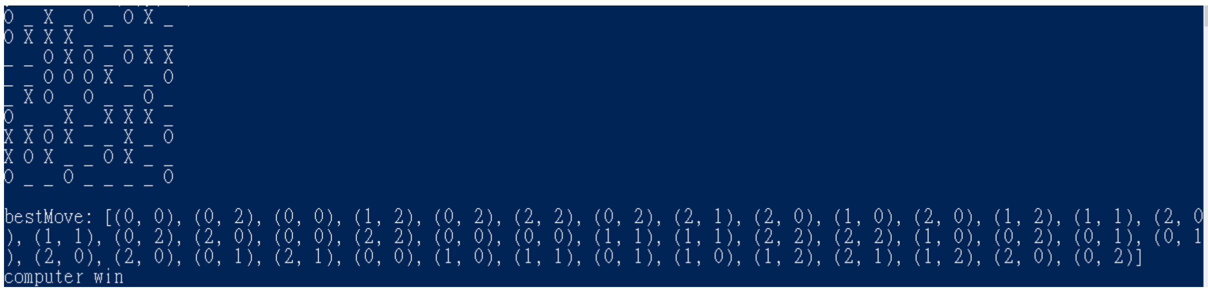


```
O _ _ _ _ _ _ _ _
X _ X _ X _ O O _
_ _ O O X _ _ X X
O O X X _ _ X O O
O _ X X _ O _ _ _
_ _ X _ _ _ X O _
_ X X _ _ _ _ _ O
_ _ O O _ X X _ _
_ O _ O _ X X _ O

bestMove: [(0, 0), (0, 0), (1, 0), (0, 0), (1, 2), (0, 2), (2, 2), (0, 2), (2
, 1), (2, 0), (0, 1), (2, 0), (0, 2), (1, 1), (1, 0), (0, 1), (2, 1), (1, 0),
 (0, 2), (1, 0), (1, 2), (0, 1), (1, 1), (1, 2), (0, 0), (2, 2), (2, 0), (1,
2), (2, 0), (2, 1), (1, 2), (2, 1), (2, 2), (2, 2), (1, 0), (1, 0), (2, 2)]
human win
```

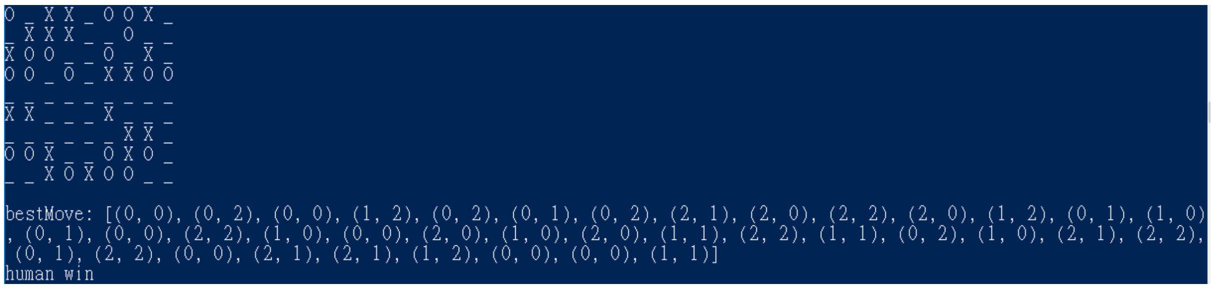


```
_ _ X X _ O O X O
_ X X X _ _ _ X O
_ O O _ _ _ _ X _
_ _ _ _ O _ X O O
O _ _ _ X O _ _ _
X X _ X _ _ X _ O
O _ _ _ _ X O _ X
_ O X O _ X _ _ _
_ O X O _ _ O _ X

bestMove: [(2, 0), (0, 0), (0, 2), (0, 0), (1, 2), (0, 2), (0, 1), (0, 2), (2, 1), (2, 0), (2, 2), (2, 0)
, (1, 2), (0, 1), (0, 0), (2, 2), (2, 2), (0, 0), (1, 1), (0, 1), (1, 0), (1, 0), (2, 1), (1, 0), (2, 0),
 (1, 1), (1, 1), (1, 2), (0, 0), (2, 1), (0, 2), (1, 2), (2, 0), (2, 1), (1, 2), (2, 2), (0, 2), (0, 2),
(1, 1)]
human win
```

Agent First:



```
O _ X _ O _ O X _
O X X X _ _ _ _ _
_ _ O X O _ O X X
_ _ O O O X _ _ O
_ X O _ O _ _ O _
O _ _ X _ X X X _
X X O X _ _ X _ O
X O X _ _ O X _ _
O _ _ O _ _ _ _ O

bestMove: [(0, 0), (0, 2), (0, 0), (1, 2), (0, 2), (2, 2), (0, 2), (2, 1), (2, 0), (1, 0), (2, 0), (1, 2), (1, 1), (2, 0
), (1, 1), (0, 2), (2, 0), (0, 0), (2, 2), (0, 0), (0, 0), (1, 1), (1, 1), (2, 2), (2, 2), (1, 0), (0, 2), (0, 1), (0, 1
), (2, 0), (2, 0), (0, 1), (2, 1), (0, 0), (1, 0), (1, 1), (0, 1), (1, 0), (1, 2), (2, 1), (1, 2), (2, 0), (0, 2)]
computer win
```



```
O _ X X _ O O X _
_ X X X _ _ O _ _
X O O _ _ O _ X _
O O _ O _ X X O O
_ _ _ _ _ _ _ _ _
X X _ _ _ X _ _ _
_ _ _ _ _ _ X X _
O O X _ _ O X O _
_ _ X O X O O _ _

bestMove: [(0, 0), (0, 2), (0, 0), (1, 2), (0, 2), (0, 1), (0, 2), (2, 1), (2, 0), (2, 2), (2, 0), (1, 2), (0, 1), (1, 0)
, (0, 1), (0, 0), (2, 2), (1, 0), (0, 0), (2, 0), (1, 0), (2, 0), (1, 1), (2, 2), (1, 1), (0, 2), (1, 0), (2, 1), (2, 2),
 (0, 1), (2, 2), (0, 0), (2, 1), (2, 1), (1, 2), (0, 0), (0, 0), (1, 1)]
human win
```

In conclusion, I win 7 games out of 10 games(70%winning rate), with five victories and two victories out of humanFirst and computerFirst respectively. So I have winning rate of 100% if human come first, but when it comes to computer come first, I only have winning rate of 40%.

Statement of Contribution:
Part1 is done by Jordan. In Part2, evaluate functions and minimax function are done by Jordan, and alpha-beta function is originally done by Shih-Jie and further modified by Jordan. PlayGame functions are also originally done by Shih-Jie and further modified by Jordan. Section I and Section II are done by Jordan. Section III and Section IV are done by Jordan and ShihJie.