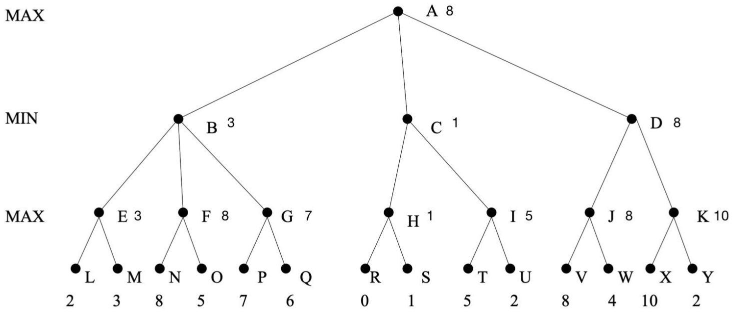
Problem assignment 3

Due: Wednesday, October 2, 2019

Problem 1

Part a.

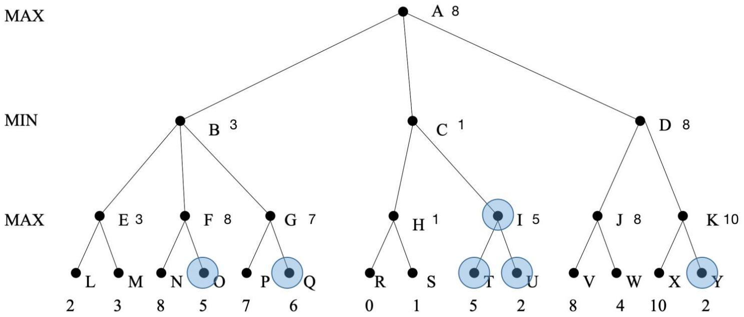


A-8, B-3, C-1, D-8, E-3, F-8, G-7, H-1, I-5, J-8, K-10, L-2, M-3, N-8, O-5, P-7, Q-6, R-0, S-1, T-5, U-2, V-8, W-4, X-10, Y-2.

The first player would choose the right now maximum utility. Among the 3 children of A, child D has the maximum values. So the first player would choose move D.

The solution path the rational players would play is A-D-J-V.

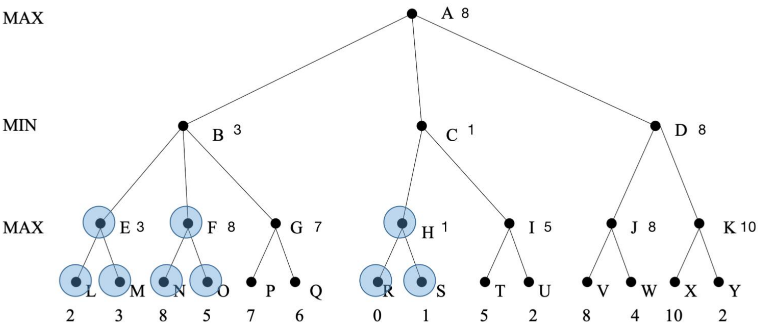
Parr b.



The nodes that are cut off from the tree and are never examined by the alpha beta procedure in left-to-right:

O, Q, I, T, U, Y.

Part c.



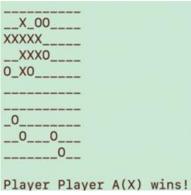
The nodes that are cut off from the tree and are never examined by the alpha beta procedure in right-to-left:

H, R, S, F, N, O, E, L, M.

Problem 2

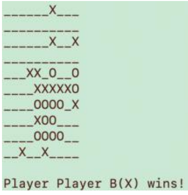
Part a.

Player A wins: 6, Player B wins: 4, Tied: 0.



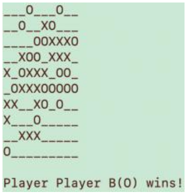
```
__________
__X_OO____
XXXXX_____
__XXXO____
O_XO______
__________
__________
_O________
__O___O___
_______O__

Player Player A(X) wins!
```



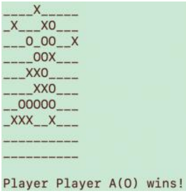
```
______X___
__________
______X__X
__________
___XX_O__O
____XXXXXO
____OOOO_X
____XOO___
____OOOO__
__X__X____

Player Player B(X) wins!
```



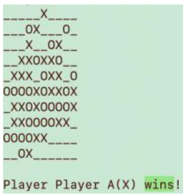
```
___O___O__
__O__XO___
____OOXXXO
__XOO_XXX_
X_OXXX_OO_
_OXXXOOOOO
XX__XO_O__
X___O_____
__XXX_____
O_________

Player Player B(O) wins!
```



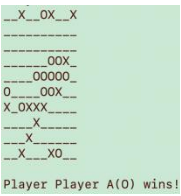
```
____X_____
_X___XO___
___O_OO__X
____OOX___
___XXO____
____XXO___
__OOOOO___
_XXX__X___
__________
__________

Player Player A(O) wins!
```



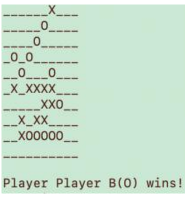
```
_____X____
___OX___O_
___X__OX__
__XXOXXO__
_XXX_OXX_O
OOOOXOXXOX
_XXOXOOOOX
_XXOOOOXX_
OOOOXX____
__OX______

Player Player A(X) wins!
```



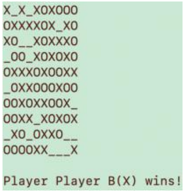
```
__X__OX__X
__________
__________
______OOX_
____OOOOO_
O____OOX__
X_OXXX____
____X_____
___X______
__X___XO__

Player Player A(O) wins!
```



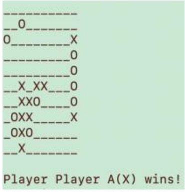
```
______X___
_____O____
____O_____
_O_O______
__O___O___
_X_XXXX___
_____XXO__
__X_XX____
__XOOOOO__
__________

Player Player B(O) wins!
```



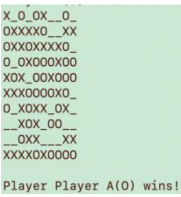
```
X_X_XOXOOO
OXXXXOX_XO
XO__XOXXXO
_OO_XOXOXO
OXXXOXOOXX
_OXXOOOXOO
OOXOXXOOX_
OOXX_XOXOX
_XO_OXXO__
OOOOXX___X

Player Player B(X) wins!
```



```
__________
__O_______
O________X
_________O
_________O
__X_XX___O
__XXO____O
_OXX_____X
_OXO______
__X_______

Player Player A(X) wins!
```



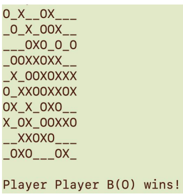
```
X_O_OX__O_
OXXXXO__XX
OXXOXXXXO_
O_OXOOOXOO
XOX_OOXOOO
XXXOOOOXO_
O_XOXX_OX_
__XOX_OO__
__OXX___XX
XXXXOXOOOO

Player Player A(O) wins!
```

As figures showed, player A wins slightly more than B. 'X' wins equal to 'O'. That's because A uses basic heuristics and B uses naive heuristics. The basic heuristics give 8 different kinds of estimate of patterns and naive only gives 3. So, the basic one can make more reasonable heuristic of each pattern and react to the 'danger' earlier than naive one.

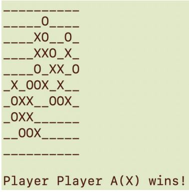
Part b.

Player A wins: 6, Player B wins: 4, Tied: 0.



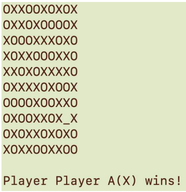
```
O_X__OX___
_O_X_OOX__
___OXO_O_O
_OOXXOXX__
_X_OOXOXXX
O_XXOOXXOX
OX_X_OXO__
X_OX_OOXXO
__XXOXO___
_OXO___OX_

Player Player B(O) wins!
```



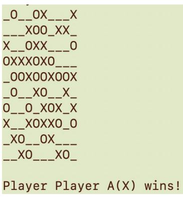
```
__________
_____O____
____XO__O_
____XXO_X_
____O_XX_O
_X_OOX_X__
_OXX__OOX_
_OXX______
__OOX_____
__________

Player Player A(X) wins!
```



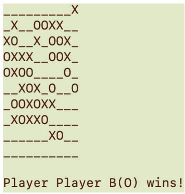
```
OXXOOXOXOX
OXXOXOOOOX
XOOOXXXOXO
XOXXOOOXXO
XXOXOXXXXO
OXXXXOXOOX
OOOOXOOXXO
OXOOXXOX_X
OXOXXOXOXO
XOXXOOXXOO

Player Player A(X) wins!
```



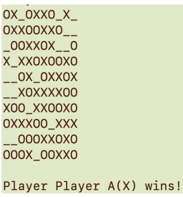
```
_O__OX___X
___XOO_XX_
X__OXX___O
OXXXOXO___
_OOXOOXOOX
_O__XO__X_
O__O_XOX_X
X__XOXXO_O
_XO__OX___
__XO___XO_

Player Player A(X) wins!
```



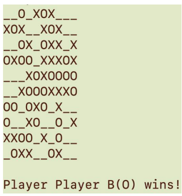
```
_________X
_X__OOXX__
XO__X_OOX_
OXXX__OOX_
OXOO____O_
__XOX_O__O
_OOXOXX___
_XOXXO____
______XO__
__________

Player Player B(O) wins!
```



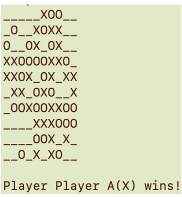
```
OX_OXXO_X_
OXXOOXXO__
_OOXXOX__O
X_XXOXOOXO
__OX_OXXOX
__XOXXXXOO
XOO_XXOOXO
OXXXOO_XXX
__OOOXXOXO
OOOX_OOXXO

Player Player A(X) wins!
```



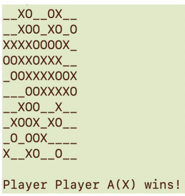
```
__O_XOX___
XOX__XOX__
__OX_OXX_X
OXOO_XXXOX
___XOXOOOO
__XOOOXXXO
OO_OXO_X__
O__XO__O_X
XXOO_X_O__
_OXX__OX__

Player Player B(O) wins!
```



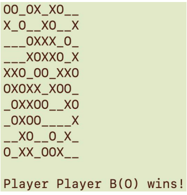
```
_____XOO__
_O__XOXX__
O__OX_OX__
XXOOOOXXO_
XXOX_OX_XX
_XX_OXO__X
_OOXOOXXOO
____XXXOOO
____OOX_X_
__O_X_XO__

Player Player A(X) wins!
```



```
__XO__OX__
__XOO_XO_O
XXXXOOOOX_
OOXXOXXX__
_OOXXXXOOX
___OOXXXXO
__XOO__X__
_XOOX_XO__
_O_OOX____
X__XO__O__

Player Player A(X) wins!
```



```
OO_OX_XO__
X_O__XO__X
___OXXX_O_
___XOXXO_X
XXO_OO_XXO
OXOXX_XOO_
_OXXOO__XO
_OXOO____X
__XO__O_X_
O_XX_OOX__

Player Player B(O) wins!
```

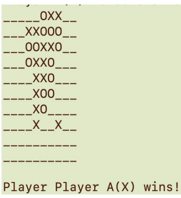
As figures showed, player A wins slightly more than B. Since player A and player B use the same heuristic method, the difference only lies in the starting. Player A always starts first. So starting first may somehow have advantage in the game.

Part c.

Players A using K-ply = 2

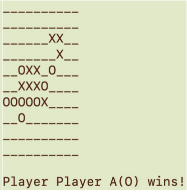
Players B using K-ply = 1

Player A wins: 8, Player B wins: 2, Tied: 0.



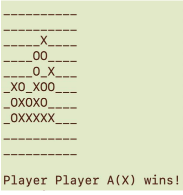
```
_____OXX__
___XXOOO__
___OOXXO__
___OXXO___
____XXO___
____XOO___
____XO____
____X__X__
__________
__________

Player Player A(X) wins!
```



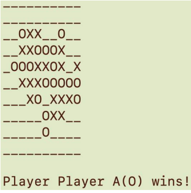
```
__________
__________
______XX__
_______X__
__OXX_O___
__XXXO____
OOOOOX____
__O_______
__________
__________

Player Player A(O) wins!
```



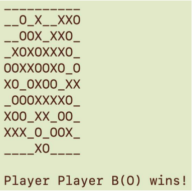
```
__________
__________
_____X____
____OO____
____O_X___
_XO_XOO___
_OXOXO____
_OXXXXX___
__________
__________

Player Player A(X) wins!
```



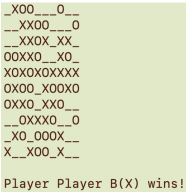
```
__________
__________
__OXX__O__
__XXOOOX__
_OOOXXOX_X
__XXXOOOOO
___XO_XXXO
_____OXX__
_____O____
__________

Player Player A(O) wins!
```



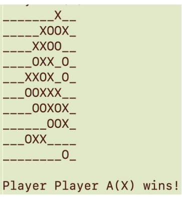
```
__________
__O_X__XXO
__OOX_XXO_
_XOXOXXXO_
OOXXOOXO_O
XO_OXOO_XX
_OOOXXXXO_
XOO_XX_OO_
XXX_O_OOX_
____XO____

Player Player B(O) wins!
```



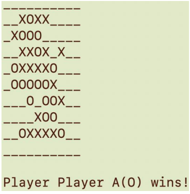
```
_XOO___O__
__XXOO___O
__XXOX_XX_
OOXXO__XO_
XOXOXOXXXX
OXOO_XOOXO
OXXO_XXO__
__OXXXO__O
_XO_OOOX__
X__XOO_X__

Player Player B(X) wins!
```



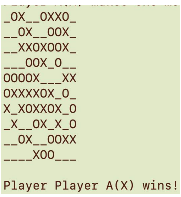
```
_______X__
_____XOOX_
____XXOO__
____OXX_O_
___XXOX_O_
___OOXXX__
____OOXOX_
______OOX_
___OXX____
________O_

Player Player A(X) wins!
```



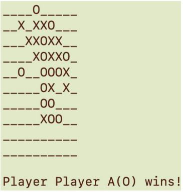
```
__________
__XOXX____
_XOOO_____
__XXOX_X__
_OXXXXO___
_OOOOOX___
___O_OOX__
____XOO___
__OXXXXO__
__________

Player Player A(O) wins!
```



```
_OX__OXXO_
__OX__OOX_
__XXOXOOX_
___OOX_O__
OOOOX___XX
OXXXXOX_O_
X_XOXXOX_O
_X__OX_X_O
__OX__OOXX
____XOO___

Player Player A(X) wins!
```



```
____O_____
__X_XXO___
___XXOXX__
____XOXXO_
__O__OOOX_
_____OX_X_
_____OO___
_____XOO__
__________
__________

Player Player A(O) wins!
```

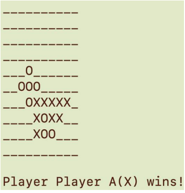
This time, player A nearly got the complete win of B. This indicates that when using the same heuristic method, the K plays an important role in the heuristic. As the K gets bigger, the adversarial search tree's depth is increasing. Therefore, when choosing where to put the next step, higher K means more reasonable heuristic for possible steps. The players tend to make more rational moves. That's why K-ply = 2 is better than K-ply = 1.

Players A using K-ply = 3

Players B using K-ply = 2

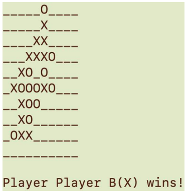
Player A wins: 5, Player B wins: 4, Tied: 1.

I waited for nearly whole long night to get the final results. The exponential explosion of the search space is really horrible.



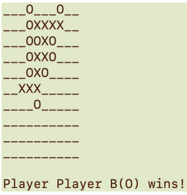
```
__________
__________
__________
__________
___O______
__OOO_____
___OXXXXX_
____XOXX__
____XOO___
__________

Player Player A(X) wins!
```



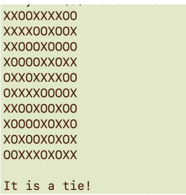
```
_____O____
_____X____
____XX____
___XXXO___
__XO_O____
_XOOOXO___
__XOO_____
__XO______
_OXX______
__________

Player Player B(X) wins!
```



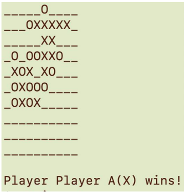
```
___O___O__
___OXXXX__
___OOXO___
___OXXO___
___OXO____
__XXX_____
____O_____
__________
__________
__________

Player Player B(O) wins!
```



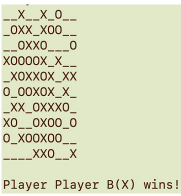
```
XXOOXXXXOO
XXXXOOXOOX
XXOOOXOOOO
XOOOOXXOXX
OXXOXXXXOO
OXXXXOOOOX
XXOOXOOXOO
XOOOOXOXXO
XOXOOXOXOX
OOXXXOXOXX

It is a tie!
```



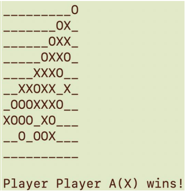
```
_____O____
___OXXXXX_
_____XX___
_O_OOXXO__
_XOX_XO___
_OXOOO____
_OXOX_____
__________
__________
__________

Player Player A(X) wins!
```



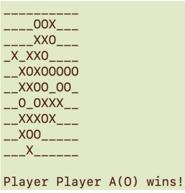
```
__X__X_O__
_OXX_XOO__
__OXXO___O
XOOOOX_X__
_XOXXOX_XX
O_OOXOX_X_
_XX_OXXXO_
XO__OXOO_O
O_XOOXOO__
____XXO__X

Player Player B(X) wins!
```



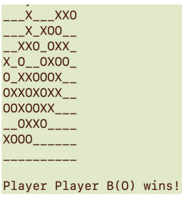
```
_________O
_______OX_
______OXX_
_____OXXO_
____XXXO__
__XXOXX_X_
_OOOXXXO__
XOOO_XO___
__O_OOX___
__________

Player Player A(X) wins!
```



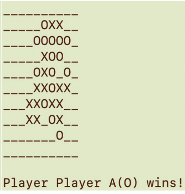
```
__________
____OOX___
____XXO___
_X_XXO____
__XOXOOOOO
__XXOO_OO_
__O_OXXX__
__XXXOX___
__XOO_____
___X______

Player Player A(O) wins!
```



```
___X___XXO
___X_XOO__
__XXO_OXX_
X_O__OXOO_
O_XXOOOX__
OXXOXOXX__
OOXOOXX___
__OXXO____
XOOO______
__________

Player Player B(O) wins!
```



```
__________
_____OXX__
____OOOOO_
_____XOO__
____OXO_O_
____XXOXX_
___XXOXX__
___XX_OX__
_______O__
__________

Player Player A(O) wins!
```

However, the result doesn't show great difference between K-ply=2 and K-ply=3, player A only win 1 more game than player B. This result means although with the adversarial search tree goes deeper, the estimation of each next-move is more reasonable, sometimes the depth of the adversarial search tree won't increase the performance a lot. What's worse, the time cost of the algorithm might rise greatly. As these time cost couldn't lead to remarkable performance improvement, we shouldn't always try to better the performance by just increase the K, some other methods should be applied, like pruning redundant branches.

Part d.

I've tried a lot of different patterns, however, none of them preformed very well at playing with basic heuristics.

My strategies are:

A. Try to maximize the utility of patterns '__iii__' or '_iiii_', these patterns may lead player to win the game.

B. Make great use of "active two". "active two" means the patterns like '__ii_' or '_ii_' or '__ii__'. If we put next step at each of these empties, they will become three consecutive patterns, which is desirable for the player.

C. Make wise defense. While defending, try to keep the opponent before it reach 3 or 4 consecutive patterns so that we can put more cost on offense rather than defense.

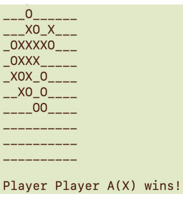
Here's the patterns:

```
'iiiii': 100,
'_iiii_': 80, #
'ii_ii': 80,    #
'_iii_': 20,
'__iii__': 30,    #
'iii__': 20,
'_i_ii_': 10,
'_ii__': 10,
'__ii_': 10,
'ii___': 3,       #
'__i__': 1,
'itttt_': 4,      #
'ittt_': 1,       #
'_ttt_': 5,       #
'_tt_t_': 2,      #
'itt_t_': 1,      #
't_t_t': 1        #
```

The Patterns with '#' mean new added or modified.

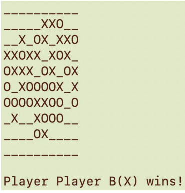
Here's the result.

Player A wins: 6, Player B wins: 3, Tied: 1.



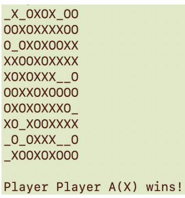
```
___O______
___XO_X___
_OXXXXO___
_OXXX_____
_XOX_O____
__XO_O____
____OO____
__________
__________
__________

Player Player A(X) wins!
```



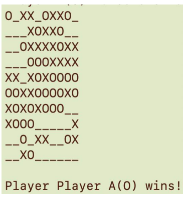
```
__________
_____XXO__
__X_OX_XXO
XXOXX_XOX_
OXXX_OX_OX
O_XOOOOX_X
OOOOXXOO_O
_X__XOOO__
____OX____
__________

Player Player B(X) wins!
```



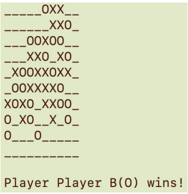
```
_X_OXOX_OO
OOXOXXXXOO
O_OXOXOOXX
XXOOXOXXXX
XOXOXXX__O
OOXXOXOOOO
OXOXOXXXO_
XO_XOOXXXX
_O_OXXX__O
_XOOXOXOOO

Player Player A(X) wins!
```



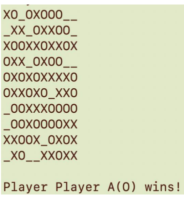
```
O_XX_OXXO_
___XOXXO__
__OXXXXOXX
___OOOXXXX
XX_XOXOOOO
OOXXOOOOXO
XOXOXOOO__
XOOO_____X
__O_XX__OX
__XO______

Player Player A(O) wins!
```



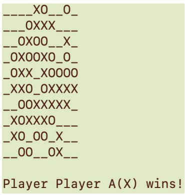
```
_____OXX__
______XXO_
___OOXOO__
___XXO_XO_
_XOOXXOXX_
_OOXXXXO__
XOXO_XXOO_
O_XO__X_O_
O___O_____
__________

Player Player B(O) wins!
```



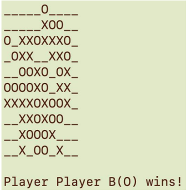
```
XO_OXOOO__
_XX_OXXOO_
XOOXXOXXOX
OXX_OXOO__
OXOXOXXXXO
OXXOXO_XXO
_OOXXXOOOO
_OOXOOOOXX
XXOOX_OXOX
_XO__XXOXX

Player Player A(O) wins!
```



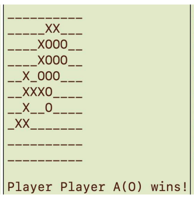
```
____XO__O_
___OXXX___
__OXOO__X_
_OXOOXO_O_
_OXX_XOOOO
_XXO_OXXXX
__OOXXXXX_
_XOXXXO___
_XO_OO_X__
__OO__OX__

Player Player A(X) wins!
```



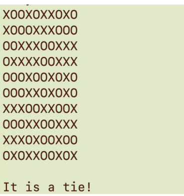
```
_____O____
_____XOO__
O_XXOXXXO_
_OXX__XXO_
__OOXO_OX_
OOOOXO_XX_
XXXXOXOOX_
__XXOXOO__
__XOOOX___
__X_OO_X__

Player Player B(O) wins!
```



```
__________
_____XX___
____XOOO__
____XOOO__
__X_OOO___
__XXXO____
__X__O____
_XX_______
__________
__________

Player Player A(O) wins!
```



```
XOOXOXXOXO
XOOOXXXOOO
OOXXXOOXXX
OXXXXOOXXX
OOOXOOXOXO
OOOXXOXOXO
XXXOOXXOOX
OOOXXOOXXX
XXXOXOOXOO
OXOXXOOXOX

It is a tie!
```

As shown above, my heuristics of patterns perform well when the board become crowded.