CS 4100 Homework 04: Connect Four

Due Tuesday 3/14 at midnight (1 minute after 11:59 pm) in Gradescope (with a grace period of 6 hours)

You may submit the homework up to 24 hours late (with the same grace period) for a penalty of 10%.

You must submit the homework in Gradescope as a zip file containing **two files:**

- The `.ipynb` file (be sure to `Kernel -> Restart and Run All` before you submit); and
- A `.pdf` file of the notebook.

For best results obtaining a clean PDF file on the Mac, select `File -> Print Review` from the Jupyter window, then choose `File-> Print` in your browser and then `Save as PDF`. Something similar should be possible on a Windows machine.

All homeworks will be scored with a maximum of 100 points; if point values are not given for individual problems, then all problems will be counted equally.

An Appendix is provided with examples of output for cases where the expected output is not able to be explained in comments.

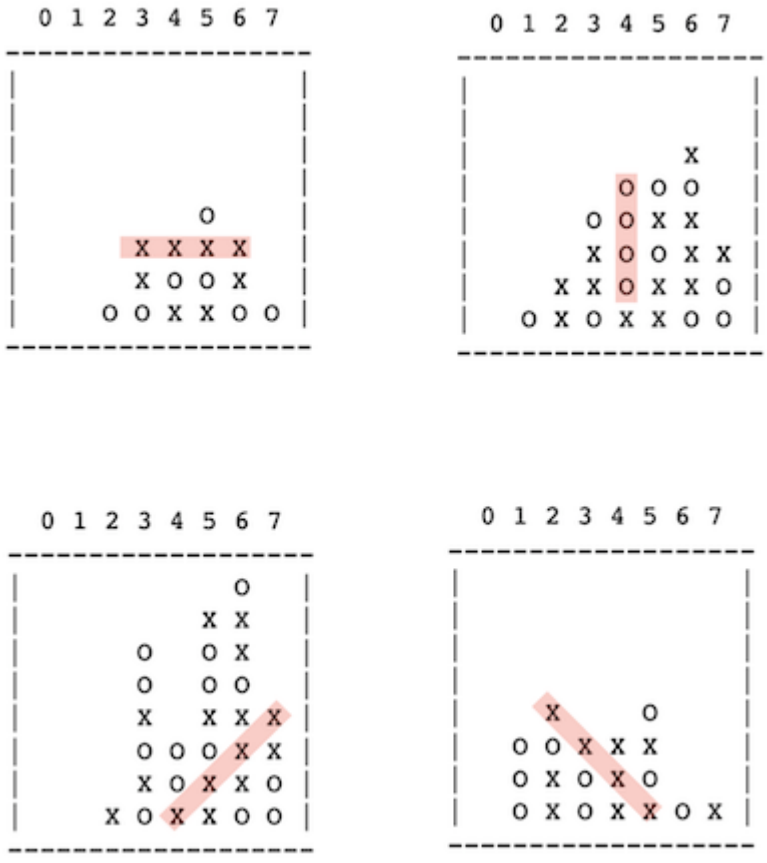
Problem One: Interactive Connect 4 (30 pts)

In this first problem, you will create the basic functionality for an interactive version of the Connect 4 game, in which you play against a naive player `player(...)` which simply chooses a random move. In the rest of the homework, you will write an improved `player(...)` which uses minmax to search for the best move.

Setup and Rules of Connect 4

Connect 4 is a children's game consisting of an 8x8 frame in which you can drop either red or yellow disks in a column; the disks fall to the bottom and can not be moved afterwards:

Two players (one taking red, the other yellow) take turns dropping the disks into the frame. We shall use `X` and `0` instead of red and yellow. The first player to get four across, down, or on a diagonal wins:



Utility Code

You must use this data structure for the board.

In [1]:
```
import numpy as np

# Board is 8x8 numpy array
# 0 = no piece
# 1 = X piece
# 2 = O piece

blank = 0
X = 1
O = 2

symbol = [' ','X','O']

N = 8

def getEmptyBoard():                                    # use this function to create a fre
    return np.zeros((N,N)).astype(int)

# This will be used to indicate an error when you try to make a move in a column that

ERROR = -1

# Check for error: use this function ONLY, since numpy arrays work strangely with com

def isError(B):
    if type(B) == int:
        return B == ERROR
```

```
        return False

# Print out a human-readable version of the board, can indent if want to trace throug

def printBoard(B,ind=0):
    indent = '\t'*ind
    if isError(B):
        print(indent,"ERROR: Overflow in column.")
        return
    print(indent,'  0 1 2 3 4 5 6 7')
    print(indent,'-------------------')
    for row in range(N):
        print(indent,'|',end='')
        for col in range(N):
            print(' '+ symbol[B[row][col]],end='')
        print(' |')
    print(indent,'-------------------')

printBoard(getEmptyBoard())
print()
printBoard(ERROR)
```

```
   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 -------------------

 ERROR: Overflow in column.
```

Part A

Fill in the following template for `dropPiece` to allow players to drop pieces into the frame to make a move.

Consult the Appendix to see the detailed outputs from the tests.

In [2]:
```
# This function should make the indicated move on the input board, and return that bo
# if there is no room in the column of the move.  Note that you are changing the orig
# IN PLACE, but also returning it, so you can indicate the error by returning ERROR (
# Do NOT make a copy, as that is very inefficient!

# player is 1 (X) or 2 (O); 0 <= move <= 7; board is 8x8 numpy array as shown in firs
# If move is illegal (either outside range 0..7) or there is no room in that column,

def illegalMove(m):
    return not(0 <= m <= 7)

def noRoomInColumn(move,board):
    return (board[0][move] != 0)

def dropPiece(player,move,board):
    if illegalMove(move) or noRoomInColumn(move, board):
        return ERROR
    for c in range(N):
        if(board[N-(c+1)][move] == 0):
            board[N-(c+1)][move] = player
            return board
```

```
    return ERROR                    # just to get it to compile, you must write this


# tests

# makeExample takes a list of X,O,X,O etc. moves and create a board.
# May be useful for testing.

def makeExample(moves):
    B = getEmptyBoard()
    player = X
    nextPlayer = O
    for m in moves:
        B = dropPiece(player,m,B)
        if isError(B):              # NOTE: This is the way to check for an error re
            return ERROR
        player,nextPlayer = nextPlayer,player
    return B


# Test out of range error -- See Appendix for what you should produce

if(dropPiece(X,100,getEmptyBoard())):
    print("Move outside range 0..7!")
else:
    print("Range test did not work. ")
print()

# Test dropPiece

B = dropPiece(X,3,getEmptyBoard())
B = dropPiece(O,4,B)
B = dropPiece(X,0,B)
B = dropPiece(O,7,B)
B = dropPiece(X,5,B)
B = dropPiece(O,3,B)
B = dropPiece(X,4,B)
B = dropPiece(O,5,B)
B = dropPiece(X,5,B)
printBoard(B)
print()


L2R = list(range(8))
R2L = L2R[::-1]
M = (L2R + R2L) * 4


fullBoard = makeExample(M)
printBoard(fullBoard)
print()


# next one should return error message for any 0 <= m <= 7, since there is no room in

m = 4

print("No room in column "+str(m)+":",noRoomInColumn(m,fullBoard),'\n')

printBoard( dropPiece(X,m,fullBoard) )
```

```
Move outside range 0..7!

  0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |           X     |
 |       O X O     |
 | X     X O X   O |
 -------------------

  0 1 2 3 4 5 6 7
 -------------------
 | O X O X O X O X |
 | X O X O X O X O |
 | O X O X O X O X |
 | X O X O X O X O |
 | O X O X O X O X |
 | X O X O X O X O |
 | O X O X O X O X |
 | X O X O X O X O |
 -------------------

No room in column 4: True

 ERROR: Overflow in column.
```

Part B

Next, you must write the function `checkWin`, which determines whether one of the players has a winning configuration.

In [3]:
```python
# player = 1 (X) or 2 (O)
# checkWin(X,board) returns X=1 if X wins,  else 0
# checkWin(O,board) returns O=2 if O wins,  else 0

# No need to check if X and O both have winning sequences, since this will be used af

def checkWin(player,board):
    for c in range(N):
        for r in range(N):
            # Horizontal check
            counter = 0
            for i in range(4):
                if illegalMove(c+i):
                    break
                elif board[r][c+i] == player:
                    counter += 1
                else:
                    break
            if counter == 4:
                return player
            # Vertical check
            counter = 0
            for j in range(4):
                if illegalMove(r+j):
                    break
                elif board[r+j][c] == player:
                    counter += 1
                else:
```

```
            if counter == 4:
                return player
            # LDiag check
            counter = 0
            for k in range(4):
                if illegalMove(r+k) or illegalMove(c-k):
                    break
                elif board[r+k][c-k] == player:
                    counter += 1
                else:
                    break
            if counter == 4:
                return player
            # RDiag check
            counter = 0
            for l in range(4):
                if illegalMove(r+l) or illegalMove(c+l):
                    break
                elif board[r+l][c+l] == player:
                    counter += 1
                else:
                    break
            if counter == 4:
                return player
    return 0
```

In [4]:
```
# tests

NoWins = [
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 1, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 2, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[2, 0, 0, 0, 0, 0, 0, 0],[2, 0, 0, 0, 1, 0, 0, 0]

XWins = [
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]

OWins = [
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
 np.array([[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0]
```

In [5]:
```
for b in XWins:
    print(checkWin(X,b),end='')       # 1111111111111
print()
for b in OWins:
    print(checkWin(O,b),end='')       # 2222222222222
print()
for b in NoWins:
    print(checkWin(X,b),end='')        # 00000
print()
```

```
1111111111111
2222222222222
00000
```

Part C

This last part of Problem One will enable you to play interactively against a random player. You should play the game sufficiently to understand the rules and some basic strategy before starting on the minmax version of the player.

Since I/O is always the most frustrating and least interesting part of any program, the template below provides some basic interaction to build on.

You should provide an interaction approximately as shown in the Appendix at the bottom of this notebook.

Note carefully:

- You must check for a win after each move;
- Code your main loop as a `for` loop with a maximum of 64, so that if the board were to fill up, the game would terminate with the message "Tie game!" (just check if the `for` loop variable == 64 after the loop ends);
- Terminate the game with an appropriate error message (as shown in the Appendix) if your move is an error, i.e.,
    - Move is not in the range 0..7; or
    - Move is in a column that is already full.

Note that the random player will never make an illegal move.

The graders will play your game to verify that it works as expected.

In [6]:
```
### Interactive version

from numpy.random import randint

def randomPlayer(board):
    move = randint(8)
    while noRoomInColumn(move,board):                # no move in this column, try ag
        move = randint(8)
    return move

# following is just to show how to accept input from keyboard, you will rewrite all o

board = getEmptyBoard()
for k in range(64):
    move = int(input('X\'s move: '))  # convert string to int
    if illegalMove(move) or noRoomInColumn(move,board):
        print("Illegal move: not in range 0..7.")
```

```
        break
    else:
        dropPiece(X,move,board)
        print("You entered",move)
        printBoard(board)
        if checkWin(X, board) == 1:
            printBoard(board)
            print('X wins!')
            break
        dropPiece(O, randomPlayer(board),board)
        print('O\'s move: '+str(m))
        printBoard(board)
        if checkWin(O, board) == 2:
            print('O wins!')
            break
if k == 63:
    print("Tie game!")
print("Bye!")
```

```
X's move: 1
You entered 1
   0 1 2 3 4 5 6 7
 --------------------
 |                  |
 |                  |
 |                  |
 |                  |
 |                  |
 |                  |
 |                  |
 |   X              |
 --------------------
O's move: 4
   0 1 2 3 4 5 6 7
 --------------------
 |                  |
 |                  |
 |                  |
 |                  |
 |                  |
 |                  |
 |                  |
 |   X O            |
 --------------------
X's move: 1
You entered 1
   0 1 2 3 4 5 6 7
 --------------------
 |                  |
 |                  |
 |                  |
 |                  |
 |                  |
 |                  |
 |   X              |
 |   X O            |
 --------------------
O's move: 4
   0 1 2 3 4 5 6 7
 --------------------
 |                  |
 |                  |
 |                  |
 |                  |
 |                  |
 |                  |
 |                  |
 |   X              |
 |   X O     O      |
 --------------------
X's move: 1
You entered 1
   0 1 2 3 4 5 6 7
 --------------------
 |                  |
 |                  |
 |                  |
 |                  |
 |                  |
 |   X              |
 |   X              |
 |   X O     O      |
 --------------------
O's move: 4
   0 1 2 3 4 5 6 7
```

```
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |   X             |
 |   X       0     |
 |   X 0     0     |
 -------------------
X's move: 1
You entered 1
   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |   X             |
 |   X             |
 |   X       0     |
 |   X 0     0     |
 -------------------
   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |   X             |
 |   X             |
 |   X       0     |
 |   X 0     0     |
 -------------------
X wins!
Bye!
```

Problem Two (60 pts)

You will now create an automated player. The basic ideas have been presented in lecture on 2/8 and 2/13 and we will not repeat them here.

Part A

The first task is to write the evaluation (heuristic) function which tells you how good a board position is for you.

There are two parameters which you can experiment with to provide the best behavior:

- THREE_SCORE
- TWO_SCORE

The evaluation method returns an integer value calculated from O's point of view as follows:

- If the board is a win for O, return `sys.maxsize` = 9223372036854775807 (you will need to `import sys`)
- If the board is a win for X, return `-sys.maxsize` = -9223372036854775807

Otherwise, let `O_SCORE` be the sum of the following:

- For any sequence of 3 O's (in a row, column, or diagonal) which could potentially be extended later

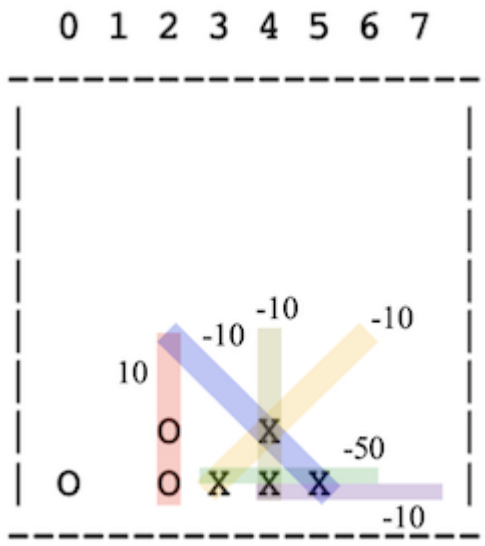
- For any sequence of 2 O's (in a row, column, or diagonal) which could potentially be extended later to a win, add `TWO_SCORE`

and let `X_SCORE` be the sum of the following:

- For any sequence of 3 X's (in a row, column, or diagonal) which could potentially be extended later to a win, subtract `THREE_SCORE`
- For any sequence of 2 X's (in a row, column, or diagonal) which could potentially be extended later to a win, subtract `TWO_SCORE`

Return `O_SCORE + X_SCORE`

For example, in the following board `B`, if we set `THREE_SCORE = 50` and `TWO_SCORE = 10`, `eval(B)` should return -80, not a good position for O!



(Note that the two winning configurations for X in the lower right corner are not mutually exclusive, and in fact an intelligent player would always choose to move to 6, so that a move to 7 is irrelevant; however, considering all such interactions is too complicated, and in the eval function described below, we do not account for these complications.)

Complete the following template and test it as shown.

HINT: The best way to write this is to adapt your `checkWin` function, which already enumerates all possible sequences which could produce a win. For each such sequence, count the number of 0's, 1's, and 2's. Then:

- If there are four 2's, then return `sys.maxsize` (a win for O);
- If there are four 1's, then return `-sys.maxsize` (a win for X);
- If there are two 0's and two 2's, then this is a sequence which should count for `TWO_SCORE`;
- If there is one 0 and three 2's, then this is a sequence which should count for `THREE_SCORE`;
- If there are two 0's and two 1's, then this is a sequence which should count for `-TWO_SCORE`; and
- If there is one 0 and three 1's, then this is a sequence which should count for `-THREE_SCORE`.

Note that for the first two cases, as an alternative you could simply call your original `checkWin` before checking the other cases.

In [7]:
```
import sys

OWIN = sys.maxsize
```

```
THREE_SCORE = 50                    # just for testing, you may want to experiment with
TWO_SCORE = 10                      # for these two parameters


# Return evaluation of the board from O's point of view

XTHREE_LIST = [[1,0,1,1], [1,1,0,1], [1,1,1,0], [0,1,1,1]]
XTWO_LIST = [[0,1,1,0], [0,0,1,1], [1,1,0,0], [0,1,0,1], [1,0,1,0], [1,0,0,1]]
OTHREE_LIST = [[2,0,2,2], [2,2,0,2], [2,2,2,0], [0,2,2,2]]
OTWO_LIST = [[0,2,2,0], [0,0,2,2], [2,2,0,0], [0,2,0,2], [2,0,2,0], [2,0,0,2]]

def scoreUp(line):
    for a in range(3):
        if line[a] == -1:
            return 0
    if line in XTHREE_LIST:
        return -1 * THREE_SCORE
    elif line in OTHREE_LIST:
        return THREE_SCORE
    elif line in XTWO_LIST:
        return -1 * TWO_SCORE
    elif line in OTWO_LIST:
        return TWO_SCORE
    else:
        return 0


def eval(board):
    if checkWin(O, board) == 2:
        return OWIN
    if checkWin(X, board) == 1:
        return XWIN
    sumVal = 0
    # Checks all blank spaces
    for c in range(N):
        for r in range(N):
            # Horizontal case
            line = []
            for i in range(4):
                if illegalMove(c+i):
                    line.append(-1)
                else:
                    line.append(board[r][c+i])
            sumVal += scoreUp(line)
            # Vertical case
            line = []
            for j in range(4):
                if illegalMove(r+j):
                    line.append(-1)
                else:
                    line.append(board[r+j][c])
            sumVal += scoreUp(line)
            # LDiag case
            line = []
            for k in range(4):
                if illegalMove(r+k) or illegalMove(c-k):
                    line.append(-1)
                else:
                    line.append(board[r+k][c-k])
            sumVal += scoreUp(line)
            # RDiag case
            line = []
            for l in range(4):
                if illegalMove(r+l) or illegalMove(c+l):
                    line.append(-1)
```

```
                else:
                    line.append(board[r+l][c+l])
            sumVal += scoreUp(line)

    return sumVal
```

In [8]:

```
# tests with THREE_SCORE = 50 and TWO_SCORE = 10

def testEval(board):
    printBoard(board)
    print()
    print('eval(B) =',eval(board))

testEval(makeExample([3,0,4,2,5,2,4])); print()                                  # eval(
testEval(makeExample([6, 5, 1])); print()                                        # eval(
testEval(makeExample([0, 2, 6, 1, 2, 1])); print()                               # eval(
testEval(makeExample([2, 6, 6, 1, 4, 0, 3, 3, 3, 2])); print()                  # eval(
testEval(makeExample([4, 3, 2, 6, 5, 3, 7, 5, 4, 4, 1, 2])); print()            # eval(
testEval(makeExample([3, 5, 6, 2, 2, 5, 7, 3, 6, 6, 5, 4, 6, 4, 4])); print() # eval(
testEval(makeExample([5, 1, 7, 0, 3, 6, 1, 4, 2, 2, 5, 0, 4, 5, 4, 2,
                      3, 6, 6, 1, 1, 2, 2, 7, 6, 7, 2, 0, 0, 5, 4, 7,
                      7, 4, 2])); print()                                        # eval(
r1 = [2, 1]*4; r2 = [1, 2]*4
testEval(np.array([r1,r2,r2,r1,r2,r2,r1,r2])); print()                           # eval(
testEval(makeExample([7, 1, 1, 5, 6, 1, 7, 6, 1, 3, 1, 2, 6, 0, 6])); print() # eval(
testEval(makeExample([7, 7, 7, 4, 0, 0, 7, 6, 7, 2, 6, 6, 7, 6, 1])); print() # eval(

print()
```

```
  0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |     O   X       |
 | O   O X X X     |
 -------------------
```

eval(B) = -80

```
  0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |   X       O X   |
 -------------------
```

eval(B) = 0

```
  0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |   O X           |
 | X O O       X   |
 -------------------
```

eval(B) = 20

```
  0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |       X         |
 |     O O     X   |
 | O O X X X   O   |
 -------------------
```

eval(B) = -20

```
  0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |         O       |
 |     O O X O     |
```

```
  ------------------

eval(B) = 90

    0 1 2 3 4 5 6 7
  ------------------
  |                |
  |                |
  |                |
  |                |
  |             X  |
  |          X X O |
  |     X O O O X  |
  |     O X O O X X|
  ------------------

eval(B) = -10

    0 1 2 3 4 5 6 7
  ------------------
  |                |
  |     X          |
  |     X          |
  |     X   O    X |
  | X X O   X O X O|
  | O O O   X O X O|
  | O X O X X X O O|
  | O O X X O X O X|
  ------------------

eval(B) = -30

    0 1 2 3 4 5 6 7
  ------------------
  | O X O X O X O X|
  | X O X O X O X O|
  | X O X O X O X O|
  | O X O X O X O X|
  | X O X O X O X O|
  | X O X O X O X O|
  | O X O X O X O X|
  | X O X O X O X O|
  ------------------

eval(B) = 0

    0 1 2 3 4 5 6 7
  ------------------
  |                |
  |                |
  |                |
  |   X            |
  |   X         X  |
  |   O         X  |
  |   X         O X|
  | O O O O   O X X|
  ------------------

eval(B) = 9223372036854775807

    0 1 2 3 4 5 6 7
  ------------------
  |                |
  |                |
  |               X|
```

```
|             O X |
|             O X |
| O           X O |
| X X O   O   O X |
 ------------------

eval(B) = -9223372036854775807
```

Part B

Now you must implement the `minMax` algorithm as described in lecture, with the following changes and additions:

- `minMax` as shown in the template must take the following parameters:
  - board -- the current board being evaluated
  - player -- either 1 (X) or 2 (O); the O player is the maximizing player (board is a max node) and X is the minimizing player (board is a min node)
  - depth -- level of this call: the first call in `player` starts at level 0, and you should increase the depth by 1 for each recursive call to `minMax`
  - alpha,beta -- cutoff bounds as described in lecture.
- `minMax` must return a pair `(score,move)` giving the min-max score calculated for the `board` and the move that corresponds to this score. The move will only be used at the top level by `player`, and will be ignored by recursive calls to `minMax`. (However, it might be useful for tracing execution.) By this arrangement, you will not need a separate "chooseMove" function as shown in the lecture slides, and can simply use the first call to `minMax` to generate the move.
- You must count the number of nodes examined (or, following the pseudo-code below, the number of calls to `minMax`); you may examine at most 10,000 nodes in any single call to `player`; as shown in the pseudo-code, a new call to `minMax` above this limit should immediately return `(0,None)`.
- It is *strongly recommended* that you do *not* make multiple copies of the board (e.g., when creating child nodes); instead, use the "recursive backtracking" trick of making a move on the board before each recursive call, then *undoing* the move before the next call:

```
    for move in range(8):
        row = row that this move would be placed in
        board[row][move] =  player that is making this move          #
make the move
        val = minMax(board, <other player>,  ....)
        board[row][move] = 0                                         #
undo the move
```

In [9]:
```
# Code for Part B

maxNodeLimit = 10000           # You can not change this
maxDepth = 5                   # 8^4 < 10000, 8^5 > 10000

countNodes = 0


def minMax(board, player, depth, alpha, beta):

    global countNodes
    countNodes += 1
```

```
    if countNodes > maxNodeLimit:
        countNodes += -1
        return (0,None)

    boardScore = eval(board)

    if boardScore == alpha or boardScore == beta or depth == maxDepth:
        return (boardScore,None)

    scoreList = []

    for move in range(8):
        if noRoomInColumn(move, board):
            continue
        else:
            for i in range(8):
                if board[7-i][move] == 0:
                    row = 7-i
                    break
        board[row][move] = player
        scoreList.append((minMax(board, (player%2)+1, depth+1, alpha, beta)[0],move))
        board[row][move] = 0


    combo = ()
    if player == O:
        scoreVal = alpha
        for j in range(len(scoreList)):
            if scoreVal <= scoreList[j][0]:
                scoreVal = scoreList[j][0]
                combo = scoreList[j]
        return combo
    else:
        scoreVal = beta
        for k in range(len(scoreList)):
            if scoreVal >= scoreList[k][0]:
                scoreVal = scoreList[k][0]
                combo = scoreList[k]
        return combo

# You will use this function in your interactive version below

def player(board):
    (_,move) = minMax(board,O,0,-sys.maxsize,sys.maxsize)    # only place we need the
    return move
```

In [10]:

```
# Some simple tests:  better testing can be done by running the interactive version f
# Your results may vary slight from what is shown here, but should be similar

maxDepth = 1          # minMax will call eval on all children of root node

board1 = makeExample([3,4,2,5,2,6,2])
print()
printBoard(board1)
print("minMax:", minMax(board1,O,0,-sys.maxsize,sys.maxsize) )  # (922337203685477580

board2 = makeExample([3,4,2,5,2,0,2])
print()
printBoard(board2)
print("minMax:", minMax(board2,O,0,-sys.maxsize,sys.maxsize) )  # (10, 2)

maxDepth = 2

board2 = makeExample([3,4,2,5,2,0,2])
print()
```

```
printBoard(board2)
print("minMax:", minMax(board2,0,0,-sys.maxsize,sys.maxsize) )  # (-50, 2)

board3 = makeExample([3,0,4,4,3,4,5])
print()
printBoard(board3)
print("minMax:", minMax(board3,0,0,-sys.maxsize,sys.maxsize) )  # (-92233720368547758
```

```
   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |     X           |
 |     X           |
 |     X X 0 0 0   |
 -------------------
minMax: (9223372036854775807, 7)

   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |     X           |
 |     X           |
 | 0   X X 0 0     |
 -------------------
minMax: (10, 2)

   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |     X           |
 |     X           |
 | 0   X X 0 0     |
 -------------------
minMax: (-50, 2)

   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |         0       |
 |       X 0       |
 | 0     X X X     |
 -------------------
minMax: (-9223372036854775807, 7)
```

Part C

Now you must copy your code from Problem 1 C down to the next cell, but use `player` instead of `randomPlayer`, and verify that your game works as you expect.

Note the following requirements:

- You must print out the number of nodes examined (= number of times `minMax` is called)
- You must print out the elapsed time to make the call to `player`, you can use the `time` library as follows:

```
  import time

  t_start = time.perf_counter()
  code to be timed
  t_end = time.perf_counter()

  print("Time elapsed:", np.around(t_end-t_start,2), "secs.")       #
will print out to 2 decimal places
```

A typical session is shown in the Appendix.

In [11]:

```
import time
maxDepth = 4

# Your code here
from numpy.random import randint

board = getEmptyBoard()
for k in range(64):
    move = int(input('X\'s move: '))  # convert string to int
    if illegalMove(move) or noRoomInColumn(move,board):
        print("Illegal move: not in range 0..7.")
        break
    else:
        dropPiece(X,move,board)
        if checkWin(X, board) == 1:
            printBoard(board)
            print('X wins!')
            break
        countNodes = 0
        t_start = time.perf_counter()
        cpuMove = player(board)
        t_end = time.perf_counter()
        dropPiece(O, cpuMove,board)
        print('O\'s move: '+str(cpuMove))
        printBoard(board)
        print("Number of nodes examined: "+str(countNodes))
        print("Time elapsed:", np.around(t_end-t_start,2), "secs.")       # will prin
        if checkWin(O, board) == 2:
            print('O wins!')
            break
if k == 63:
    print("Tie game!")
print("Bye!")
```

```
X's move: 2
O's move: 4
   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |     X   O       |
 -------------------
Number of nodes examined: 4681
Time elapsed: 10.12 secs.
X's move: 2
O's move: 7
   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |     X           |
 |     X   O     O |
 -------------------
Number of nodes examined: 4681
Time elapsed: 10.05 secs.
X's move: 5
O's move: 7
   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |     X         O |
 |     X   O X   O |
 -------------------
Number of nodes examined: 4681
Time elapsed: 10.07 secs.
X's move: 7
O's move: 5
   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |               X |
 |     X     O   O |
 |     X   O X   O |
 -------------------
Number of nodes examined: 4681
Time elapsed: 10.39 secs.
X's move: 2
O's move: 2
   0 1 2 3 4 5 6 7
 -------------------
 |                 |
```

```
 |                 |
 |                 |
 |     O           |
 |     X         X |
 |     X     O   O |
 |     X   O X   O |
 -------------------
Number of nodes examined: 4169
Time elapsed: 8.33 secs.
X's move: 3
O's move: 4
   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |     O           |
 |     X         X |
 |     X   O O   O |
 |     X X O X   O |
 -------------------
Number of nodes examined: 4673
Time elapsed: 9.94 secs.
X's move: 3
O's move: 4
   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |     O           |
 |     X   O     X |
 |     X X O O   O |
 |     X X O X   O |
 -------------------
Number of nodes examined: 4513
Time elapsed: 9.55 secs.
X's move: 4
O's move: 3
   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |     O   X       |
 |     X O O     X |
 |     X X O O   O |
 |     X X O X   O |
 -------------------
Number of nodes examined: 4553
Time elapsed: 9.47 secs.
X's move: 5
O's move: 6
   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |     O   X       |
 |     X O O X   X |
 |     X X O O   O |
```

```
 -------------------
Number of nodes examined: 4345
Time elapsed: 9.2 secs.
X's move: 6
O's move: 7
   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |     O   X     O |
 |     X O O X   X |
 |     X X O O X O |
 |     X X O X O O |
 -------------------
Number of nodes examined: 3593
Time elapsed: 7.47 secs.
X's move: 3
O's move: 6
   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |     O X X     O |
 |     X O O X O X |
 |     X X O O X O |
 |     X X O X O O |
 -------------------
Number of nodes examined: 3321
Time elapsed: 6.51 secs.
O wins!
Bye!
```

Problem Three (Connect4 Contest) (10 pts)

For this problem, you will provide a complete listing of all necessary code in a single code cell, so that we can copy it into a master notebook to run a contest among all the submissions, plus Prof Snyder's implementation, and the random player.

You may examine at most 10,000 nodes, and you may take no longer than 30 seconds to make a move (on my power Mac, which is quite fast, so this is just to make sure that you don't do something completely crazy that takes absurd amounts of time). There is no other restriction on the depth, just that you may call `minMax` at most 10,000 times.

We will run multiple versions of the contest, perhaps with slightly randomized starting points (all with eval scores of 0 to "level the playing field"). At the end we will have a ranking of all submissions.

For this part of the homework, you will receive points as follows:

- If you rank below the random player, you will receive 0 points;
- If you rank above Prof Snyder's player, you will receive 10 points and definite bragging rights;
- Otherwise, we will divide the remaining players into 10 intervals (we'll be generous), receiving from 1 to 10 points (thus you can still get 10 points even if you can't beat Snyder's player).

Note: I have run similar contests in the past, and my experience has been that lots of students were able to beat my player!

In the next code cell, keep the first line, add your name in the second line, and then copy ALL CODE from the previous cells which would be necessary to run your `player` function.

DO NOT copy down all the tests, just the code!

You do not have to copy down your interactive code, just the code necessary to run `player`; however you may do so if you want to verify that everything works as expected.

It would be an **excellent** idea to Restart and then just run this one cell, to make sure you have copied down everything necessary.

In [12]:
```
# Solution for Problem Three
# Your name:
# Max Correia

# your code here

# Imports

import time
import sys
from numpy.random import randint

# win condition function

def checkWin(player,board):
    for c in range(N):
        for r in range(N):
            # Horizontal check
            counter = 0
            for i in range(4):
                if illegalMove(c+i):
                    break
                elif board[r][c+i] == player:
                    counter += 1
                else:
                    break
            if counter == 4:
                return player
            # Vertical check
            counter = 0
            for j in range(4):
                if illegalMove(r+j):
                    break
                elif board[r+j][c] == player:
                    counter += 1
                else:
                    break
            if counter == 4:
                return player
            # LDiag check
            counter = 0
            for k in range(4):
                if illegalMove(r+k) or illegalMove(c-k):
                    break
                elif board[r+k][c-k] == player:
                    counter += 1
                else:
                    break
            if counter == 4:
                return player
            # RDiag check
            counter = 0
```

```
                if illegalMove(r+l) or illegalMove(c+l):
                    break
                elif board[r+l][c+l] == player:
                    counter += 1
                else:
                    break
            if counter == 4:
                return player
    return 0

# eval function + variables

OWIN = sys.maxsize
XWIN = -OWIN

THREE_SCORE = 50                    # just for testing, you may want to experiment with
TWO_SCORE = 10                      # for these two parameters


# Return evaluation of the board from O's point of view

XTHREE_LIST = [[1,0,1,1], [1,1,0,1], [1,1,1,0], [0,1,1,1]]
XTWO_LIST = [[0,1,1,0], [0,0,1,1], [1,1,0,0], [0,1,0,1], [1,0,1,0]]
OTHREE_LIST = [[2,0,2,2], [2,2,0,2], [2,2,2,0], [0,2,2,2]]
OTWO_LIST = [[0,2,2,0], [0,0,2,2], [2,2,0,0], [0,2,0,2], [2,0,2,0]]

def scoreUp(line):
    for a in range(3):
        if line[a] == -1:
            return 0
    if line in XTHREE_LIST:
        return -1 * THREE_SCORE
    elif line in OTHREE_LIST:
        return THREE_SCORE
    elif line in XTWO_LIST:
        return -1 * TWO_SCORE
    elif line in OTWO_LIST:
        return TWO_SCORE
    else:
        return 0


def eval(board):
    if checkWin(O, board) == 2:
        return OWIN
    if checkWin(X, board) == 1:
        return XWIN
    sumVal = 0
    # Checks all blank spaces
    for c in range(N):
        for r in range(N):
            # Horizontal case
            line = []
            for i in range(4):
                if illegalMove(c+i):
                    line.append(-1)
                else:
                    line.append(board[r][c+i])
            sumVal += scoreUp(line)
            # Vertical case
            line = []
            for j in range(4):
                if illegalMove(r+j):
                    line.append(-1)
                else:
                    line.append(board[r+j][c])
```

```
            sumVal += scoreUp(line)
            # LDiag case
            line = []
            for k in range(4):
                if illegalMove(r+k) or illegalMove(c-k):
                    line.append(-1)
                else:
                    line.append(board[r+k][c-k])
            sumVal += scoreUp(line)
            # RDiag case
            line = []
            for l in range(4):
                if illegalMove(r+l) or illegalMove(c+l):
                    line.append(-1)
                else:
                    line.append(board[r+l][c+l])
            sumVal += scoreUp(line)

    return sumVal

# minMax function + variables

maxNodeLimit = 10000           # You can not change this
maxDepth = 5                   # 8^4 < 10000, 8^5 > 10000

countNodes = 0


def minMax(board, player, depth, alpha, beta):

    global countNodes
    countNodes += 1

    if countNodes > maxNodeLimit:
        countNodes += -1
        return (0,None)

    boardScore = eval(board)

    if boardScore == alpha or boardScore == beta or depth == maxDepth:
        return (boardScore,None)

    scoreList = []

    for move in range(8):
        if noRoomInColumn(move, board):
            continue
        else:
            for i in range(8):
                if board[7-i][move] == 0:
                    row = 7-i
                    break
        board[row][move] = player
        scoreList.append((minMax(board, (player%2)+1, depth+1, alpha, beta)[0],move))
        board[row][move] = 0


    combo = ()
    if player == O:
        scoreVal = alpha
        for j in range(len(scoreList)):
            if scoreVal <= scoreList[j][0]:
                scoreVal = scoreList[j][0]
                combo = scoreList[j]
        return combo
```

```
        scoreVal = beta
        for k in range(len(scoreList)):
            if scoreVal >= scoreList[k][0]:
                scoreVal = scoreList[k][0]
                combo = scoreList[k]
        return combo


# Player function

def player(board):
    (_,move) = minMax(board,0,0,-sys.maxsize,sys.maxsize)    # only place we need the
    return move
```

Appendix: Expected Outputs

Problem 1 B

```
Move outside range 0..7!

  0 1 2 3 4 5 6 7
-------------------
|                 |
|                 |
|                 |
|                 |
|                 |
|           X     |
|       0 X 0     |
| X     X 0 X   0 |
-------------------

  0 1 2 3 4 5 6 7
-------------------
| 0 X 0 X 0 X 0 X |
| X 0 X 0 X 0 X 0 |
| 0 X 0 X 0 X 0 X |
| X 0 X 0 X 0 X 0 |
| 0 X 0 X 0 X 0 X |
| X 0 X 0 X 0 X 0 |
| 0 X 0 X 0 X 0 X |
| X 0 X 0 X 0 X 0 |
-------------------

No room in column 4: True

ERROR: Overflow in column.
```

Problem 1 C

```
-------- Win for X --------------------------------

X's move: 3

  0 1 2 3 4 5 6 7
-------------------
|                 |
|                 |
|                 |
```

```
|                 |
|                 |
|                 |
|       X         |
-------------------

O's move: 2

  0 1 2 3 4 5 6 7
-------------------
|                 |
|                 |
|                 |
|                 |
|                 |
|                 |
|                 |
|     O X         |
-------------------

X's move: 3

  0 1 2 3 4 5 6 7
-------------------
|                 |
|                 |
|                 |
|                 |
|                 |
|                 |
|       X         |
|     O X         |
-------------------

O's move: 2

  0 1 2 3 4 5 6 7
-------------------
|                 |
|                 |
|                 |
|                 |
|                 |
|                 |
|     O X         |
|     O X         |
-------------------

X's move: 3

  0 1 2 3 4 5 6 7
-------------------
|                 |
|                 |
|                 |
|                 |
|                 |
|       X         |
|     O X         |
|     O X         |
-------------------
```

```
O's move: 4

  0 1 2 3 4 5 6 7
------------------
|                |
|                |
|                |
|                |
|                |
|       X        |
|     O X        |
|     O X O      |
------------------

X's move: 3

  0 1 2 3 4 5 6 7
------------------
|                |
|                |
|                |
|                |
|       X        |
|       X        |
|     O X        |
|     O X O      |
------------------

Win for X!
Bye!


---------- Win for O ------------------------------

X's move: 3

  0 1 2 3 4 5 6 7
------------------
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|       X        |
------------------

O's move: 2

  0 1 2 3 4 5 6 7
------------------
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|     O X        |
------------------
```

```
.... Many moves later .......


O's move: 3

  0 1 2 3 4 5 6 7
------------------
|                |
|                |
|                |
|     X          |
|     X   X     0 |
|     0 0 0   0 0 |
| X 0 X X 0   X X |
| 0 X 0 X X X 0 0 |
------------------

X's move: 0

  0 1 2 3 4 5 6 7
------------------
|                |
|                |
|                |
|     X          |
|     X   X     0 |
| X   0 0 0   0 0 |
| X 0 X X 0   X X |
| 0 X 0 X X X 0 0 |
------------------

O's move: 1

  0 1 2 3 4 5 6 7
------------------
|                |
|                |
|                |
|     X          |
|     X   X     0 |
| X 0 0 0 0   0 0 |
| X 0 X X 0   X X |
| 0 X 0 X X X 0 0 |
------------------

Win for O!
Bye!


---------- Error: Move not in range ----------------------------

X's move: 3

  0 1 2 3 4 5 6 7
------------------
|                |
|                |
|                |
|                |
|                |
```

```
|                  |
|       X          |
------------------

O's move: 7

  0 1 2 3 4 5 6 7
------------------
|                  |
|                  |
|                  |
|                  |
|                  |
|                  |
|                  |
|       X       O  |
------------------

X's move: 10
Illegal move: not in range 0..7.
Bye!


---------- Error: move to column already filled ------------

X's move: 4

  0 1 2 3 4 5 6 7
------------------
|                  |
|                  |
|                  |
|                  |
|                  |
|                  |
|                  |
|          X       |
------------------

O's move: 3

  0 1 2 3 4 5 6 7
------------------
|                  |
|                  |
|                  |
|                  |
|                  |
|                  |
|                  |
|       O X        |
------------------


.... Many moves later.....



X's move: 2

  0 1 2 3 4 5 6 7
```

```
|                 |
|       X         |
|       X         |
|       X         |
|       O         |
|     X X X       |
|   O X X X O   O |
| O O X O X O O O |
-------------------

O's move: 3

  0 1 2 3 4 5 6 7
-------------------
|       O         |
|       X         |
|       X         |
|       X         |
|       O         |
|     X X X       |
|   O X X X O   O |
| O O X O X O O O |
-------------------

X's move: 3
Illegal move: column 3 is already full.
Bye!
```

Problem 2 Part A

```
  0 1 2 3 4 5 6 7
-------------------
|                 |
|                 |
|                 |
|                 |
|                 |
|                 |
|     O   X       |
| O   O X X X     |
-------------------

eval(B) = -80

  0 1 2 3 4 5 6 7
-------------------
|                 |
|                 |
|                 |
|                 |
|                 |
|                 |
|                 |
|   X       O X   |
-------------------

eval(B) = 0

  0 1 2 3 4 5 6 7
-------------------
```

```
|                  |
|                  |
|                  |
|                  |
|                  |
|   O X            |
| X O O       X    |
--------------------
```

eval(B) = 20

```
  0 1 2 3 4 5 6 7
--------------------
|                  |
|                  |
|                  |
|                  |
|                  |
|       X          |
|     O O     X    |
| O O X X X   O    |
--------------------
```

eval(B) = -20

```
  0 1 2 3 4 5 6 7
--------------------
|                  |
|                  |
|                  |
|                  |
|                  |
|         O        |
|     O O X O      |
|   X X O X X O X  |
--------------------
```

eval(B) = 90

```
  0 1 2 3 4 5 6 7
--------------------
|                  |
|                  |
|                  |
|                  |
|             X    |
|         X X O    |
|     X O O O X    |
|     O X O O X X  |
--------------------
```

eval(B) = -10

```
  0 1 2 3 4 5 6 7
--------------------
|                  |
|     X            |
|     X            |
|     X   O     X  |
| X X O   X O X O  |
| O O O   X O X O  |
```

```
| O O X X O X O X |
------------------

eval(B) = -30

  0 1 2 3 4 5 6 7
------------------
| O X O X O X O X |
| X O X O X O X O |
| X O X O X O X O |
| O X O X O X O X |
| X O X O X O X O |
| X O X O X O X O |
| O X O X O X O X |
| X O X O X O X O |
------------------

eval(B) = 0

  0 1 2 3 4 5 6 7
------------------
|                |
|                |
|                |
|   X            |
|   X        X   |
|   O        X   |
|   X        O X |
| O O O O  O X X |
------------------

eval(B) = 9223372036854775807

  0 1 2 3 4 5 6 7
------------------
|                |
|                |
|              X |
|              X |
|            O X |
|            O X |
| O          X O |
| X X O  O   O X |
------------------

eval(B) = -9223372036854775807
```

Problem Two Part B

```
  0 1 2 3 4 5 6 7
 ------------------
 |                |
 |                |
 |                |
 |                |
 |                |
 |    X           |
 |    X           |
 |    X X O O O   |
```

```
minMax: (9223372036854775807, 7)

  0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |     X           |
 |     X           |
 | O   X X O O     |
 -------------------
minMax: (10, 2)

  0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |     X           |
 |     X           |
 | O   X X O O     |
 -------------------
minMax: (-50, 2)

  0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |         O       |
 |       X O       |
 | O     X X X     |
 -------------------
minMax: (-9223372036854775807, 7)
```

Problem Two Part C

This trace was performed with a vanilla-flavored minMax (nothing other than alpha-beta pruning), with a maxDepth of 5.

```
X's move: 3

  0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |       X         |
```

```
O's move: 7

   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |       X       O |
 -------------------
Number of nodes examined: 10000
Elapsed time: 13.9 secs.

X's move: 4

   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |       X X     O |
 -------------------

O's move: 5

   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |       X X O   O |
 -------------------
Number of nodes examined: 9111
Elapsed time: 12.5 secs.

X's move: 4

   0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |         X       |
 |       X X O   O |
 -------------------
```

```
  0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |         O       |
 |         X       |
 |       X X O   O |
 -------------------
Number of nodes examined: 9050
Elapsed time: 12.53 secs.

X's move: 2

  0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |         O       |
 |         X       |
 |     X X X O   O |
 -------------------

O's move: 1

  0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |         O       |
 |         X       |
 |   O X X X O   O |
 -------------------
Number of nodes examined: 2899
Elapsed time: 4.01 secs.

X's move: 3

  0 1 2 3 4 5 6 7
 -------------------
 |                 |
 |                 |
 |                 |
 |                 |
 |                 |
 |         O       |
 |       X X       |
 |   O X X X O   O |
 -------------------

O's move: 5
```

```
 ------------------
 |                |
 |                |
 |                |
 |                |
 |                |
 |        O       |
 |      X X O     |
 |  O X X X O   O |
 ------------------
Number of nodes examined: 6609
Elapsed time: 9.29 secs.

X's move: 3

   0 1 2 3 4 5 6 7
 ------------------
 |                |
 |                |
 |                |
 |                |
 |                |
 |      X O       |
 |      X X O     |
 |  O X X X O   O |
 ------------------

O's move: 3

   0 1 2 3 4 5 6 7
 ------------------
 |                |
 |                |
 |                |
 |                |
 |      O         |
 |      X O       |
 |      X X O     |
 |  O X X X O   O |
 ------------------
Number of nodes examined: 6912
Elapsed time: 8.96 secs.

X's move: 2

   0 1 2 3 4 5 6 7
 ------------------
 |                |
 |                |
 |                |
 |                |
 |      O         |
 |      X O       |
 |    X X X O     |
 |  O X X X O   O |
 ------------------

O's move: 6

   0 1 2 3 4 5 6 7
 ------------------
```

```
 |                 |
 |                 |
 |                 |
 |       O         |
 |       X O       |
 |     X X X O     |
 |   O X X X O O O |
 -------------------
Number of nodes examined: 3953
Elapsed time: 4.87 secs.

Win for O!
Bye!
```