Assignment 9

Simulating the Connect 4 Board

Due before lecture November 14th
Problem 9: Connect 4 Board

• Connect Four is a variation of tic-tac-toe played on a 7x6 rectangular board:

• The game is played by two players, alternating turns, with each trying to place four checkers in a row vertically, horizontally, or diagonally.

• One constraint in the game is because the board stands vertically, the checkers cannot be placed in any arbitrary position. A checker may only be placed at the top of one of the currently existing columns (or it may start a new column).
So how do we represent a connect 4 board in python???

Well, we would need a 2d data structure (i.e. list o’ lists) to symbolize the board and width and height variables (assuming these values could be something other than 6 x 7)

So our board class should have.....

• A variable data storing the two-dimensional array (list of lists), which is the game board

• A variable height storing the number of rows on the game board

• A variable width storing the number of columns on the game board
Problem 9: Connect 4 Board

And we’ll need methods to perform actions...

__init__, the constructor

```python
def __init__( self, width, height ):
    self.width = width
    self.height = height
    self.data = []   # this will be the board

    for row in range( self.height ):
        boardRow = []
        for col in range( self.width ):
            boardRow += [' ']
        self.data += [boardRow]
```

# do not need to return anything from inside a constructor!
This is a constructor for Board objects that takes two arguments. This constructor takes in a number of columns and rows. The constructor will set the values of the data members of the object.
And we’ll need methods to perform actions…
\_\_\_repr\_, for printing or any string representation

```python
def \_\_repr\_(self):
    # print out rows & cols
    s = ''  # the string to return
    for row in range( self.height ):
        s += '|'  # add the separator character
        for col in range( self.width ):
            s += self.data[row][col] + '|'
        s += '

    # print out the horizontal separator
    (your code goes here)

    # print out indices of each column
    # using mod if greater than 9,
    # for spacing issues
    (your code goes here)

    return s  # return string
```

6 X 7 grid
Problem 9: Connect 4 Board

And we’ll need methods to perform actions...

• This method takes two inputs: the first input col represents the index of the column to which the checker will be added; the second input ox either an ‘X’ or ‘O’

• In addMove you do not have to check that col is a legal column number or that there is space in column col. That checking is important, however. The next method, which is called allowsMove, will do just that.

```python
def addMove(self, col, ox):
    #find the first row in the column
    #without a checker in it and
    #then add the ox checker there...

    #do this by checking values
    #in self.data...
```

```python
>>> b = Board(7, 6)
>>> b.addMove(0, 'X')
>>> b.addMove(0, 'O')
>>> b.addMove(3, 'O')
```

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
|X| | | | | | |
|X|O|O| | | | |
---------------
0 1 2 3 4 5 6
```
And we’ll need methods to perform actions...

**clear**(*self*), should clear the board that invokes it. Not much to say about clear(*self*). It may be useful when or starting a new game.

**delMove**(*self, c*) removes a checker from the board:

This method should do the opposite of addMove. It should remove the top checker from the column c. If the column is empty, then delMove should do nothing. This function may not seem useful now, but it will become very useful when you try to implement your own Connect Four player.
And we’ll need methods to perform actions…

`allowsMove(self, c)`, for checking if a column is a legal move:

This method should `return True` if the calling object (of type Board) `does` allow a move into column `c`. It `returns False` if column `c` is not a legal column number for the calling object. It also `returns False` if column `c` is full.

`isFull(self)`, checks if the board is full:

This method should `return True` if the calling object (of type Board) is completely full of checkers. It should `return False` otherwise. Notice that you can leverage `allowsMove` to make this method very concise! Unless you're supernaturally patient, you'll want to test this on small boards:
Problem 9: Connect 4 Board

And we’ll need methods to perform actions…

winsFor (self, ox): checks if someone has won the game. It should return True if there are four checkers of type ox in a row on the board. It should return False otherwise.

One way to approach this is to consider each possible anchor checker that might start a four-in-a-row run. for example, all of the "anchors" that might start a horizontal run (going from left to right) must be in the columns at least four places from the end of the board.

```python
# check for horizontal wins
for row in range(0, self.height):
    for col in range(0, self.width-3):
        if self.data[row][col] == ox and 
            self.data[row][col+1] == ox and 
            self.data[row][col+2] == ox and 
            self.data[row][col+3] == ox:
            return True
```

Important – Must check
1) Horizontally (shown here)
2) Vertically
3) Diagonally NE – SW
4) Diagonally NW - SE
And we’ll need methods to perform actions...

**hostGame**(self): This method brings everything together into the familiar game. It should alternate turns between 'X' and 'O'. It should ask the user to select a column number for each move.

*Here are a few important points to keep in mind:*

1) This method should print the board before prompting for each move.

2) After obtaining a move, you should check if the column chosen is a valid one (using `allowsMove()`). If invalid, prompt for another move instead.

3) This method should place the checker into its (valid) column. Then it should check if that player has won the game or if the board is now full.

4) If the game is over for either reason, the game should stop, the board should be printed out one last time, and the program should report who won (or that it was a tie.)
Problem 9: Connect 4 Board

hostGame example...

```

0 1 2 3 4 5 6
X's choice: 3

0 1 2 3 4 5 6
O's choice: 4

0 1 2 3 4 5 6
X's choice: 0

X wins -- Congratulations!
```