CMPUT 175 - Lab 3: Numerical Tic Tac Toe

Goal: Review classes/objects, import

Background Information:

Numerical Tic Tac Toe is a game for two players who take turns filling in a 3 by 3 grid with the numbers 1-9. The first player places an odd number in an empty square, and then the second player places an even number in an empty square. The players' turns continue to alternate until:

1) a player completes a line of three squares (horizontal, vertical, or diagonal) that adds up to 15 – in which case, the current player wins; or
2) all nine squares on the board are full, but no line adding to 15 is formed – in which case, it's a tie.

Note that player 1 must always place an odd number on the board, and player 2 an even number. In addition, each number from 1 to 9 can only appear on the board once. To view a complete sample game, please refer to **lab3_sampleOut.txt** on eClass.

Problem Description:

A colleague has already written a program to control the flow of this game, assuming that there was a Numerical Tic Tac Toe class. However, no such class currently exists. It is up to you to write the class, so that an instance of it can then be used in your colleague's program.

Part 1:

Download and save a copy of **lab3_NumTicTacToe.py**. This file contains the skeleton code for a class called NumTicTacToe, which you will complete.

1. The *__init__* method has been completed for you. Notice that it initializes two attributes: board and size. The size attribute defines how many rows (and columns) are on the board. The **board** attribute represents an empty 3 by 3 board – it is a list of lists, where each internal list represents a row on the board. The value 0 is used to represent an empty square.
   Run the test provided (i.e. by running lab3_NumTicTacToe.py) to test this method before moving on:
   `self.board` should be `[[0, 0, 0], [0, 0, 0], [0, 0, 0]]`

2. Complete the *drawBoard* method so that it displays the current state of the board, along with the column indices on top of the board, and the row indices to the left of the board. If the content of a given square is 0, an empty space should be displayed.
   Test your method before moving on:
   When `self.board` is `[[0, 0, 0], [0, 0, 0], [0, 0, 0]]`, the following should be printed:

```
   0   1   2
0    |   |
  -----------
1    |   |
  -----------
2    |   |
```

3. Complete the *squareIsEmpty(row, col)* method. This method checks the contents of the board at the given row index (integer) and column index (integer). If the board at that square is "empty", the method returns True. Otherwise, the method returns False. **Test your method before moving on.**

4. Complete the *update(row, col, num)* method. This method should check if the square at the provided row index (integer) and column index (integer) is "empty". (*Hint*: do you already have a method you can invoke to check this?) If it is "empty", the contents of the board at that square should be changed to num (integer). If the square is successfully updated, this method should return True. If the square is not empty, the contents of the board should not be changed, and the method should return False. **Test your method.**

5. Complete the *boardFull()* method. This method returns True if there are no "empty" squares, False otherwise. **Test this method. Which cases should you consider?**

6. Complete the *isWinner()* method. This method returns True if there is one line of three squares (horizontal, vertical, or diagonal) that adds up to 15. Otherwise, it returns False. **Test this method. Which cases should you consider?**

Demo your completed class to your TA, along with your tests. Be ready to explain which aspects you tested and why.

Part 2 (optional):

1. Once you have fully tested your NumTicTacToe class, download your colleague's game program (**lab3.py)** from eClass and save it in the same folder as your **lab3_NumTicTacToe.py**. Run lab3.py. Check against the sample output included in lab3_sampleOut.txt.

*Notice that all of the tests that you included under `if __name__ == '__main__':` in lab3_NumTicTacToe.py should <u>not</u> execute when you run lab3.py.

2. Run lab3.py again, and try entering an even number for Player 1. (Remember, Player 1 should only be able to place odd numbers on the board.) Does the program behave as you expect it should? Add code to validate a user's input in functions *getEntry(player, entries)* and *getCoord(player, dimension)*. What cases do you need to consider?