SI 507: Tic Tac Toe Homework #1

Overview

In this assignment, you will implement a tic-tac-toe game for two players. If you are unfamiliar with the game, you should look it up so that you understand each piece that you will need to program.

The program will prompt each player for their move and then check to see if anyone has won after each move. If the board fills up and nobody has won, the program will declare the game a tie. You will start with a partially implemented game that has some functions implemented and documented, as well as some functions that need to be implemented, documented, or both.

This assignment will give you experience working with functions and documentation, especially Python docstrings using the NumPy docstring format. You will also walk through the process of creating the program from scratch in your Discussion Section, so you will gain some experience with using functions to break a complex problem down into smaller problems that you can solve one at a time.

Steps

1. Copy the file `TicTacToeStarter.py` to your computer and rename it to `TicTacToe.py`.
2. Find all of the `# TODO` comments and implement the missing pieces (documentation and/or function implementations).
3. Play-test your game as you develop to make sure that it works properly and that it handles invalid inputs gracefully (i.e., the program does not crash).
4. Your program should behave as shown in the file: `TicTacToe Sample Output.pdf`.
    - Note: The sample code we provided does not display the winner (Nobody, X, or O) when the game ends as shown in the sample output. You should add this where appropriate.

Submission

What to turn in: `TicTacToe.py` in Gradescope

Function Specifications

Do not copy the specification of the methods as docstring since some of them do not follow the Numpy docstring format.

Functions

```python
def make_move(player, board) -> None:
    """
    Allows a player to make a move in the game and displays whose move it is (X or O).
    Prompts the user to enter a number 1-9, validates the input, repeats until valid input is entered, checks move is valid (space is unoccupied).
    It should keep taking input (using input()) until a valid move is made.
    Updates/modifies the board in place when a valid move is entered.
    Use the player ID (index) INT to modify the board, not char or string.
    Parameters:
    ----------
    player: int
        The id (index) of the player to move (1 = X, 2 = O).
    board: list
        The board upon which to move, the board is modified in place when a valid
move is entered.
    """
    pass

def check_win_horizontal(board) -> int:
    pass

def check_win_vertical(board) -> int:
    pass

def check_win_diagonal(board) -> int:
    pass

def next_player(current_player) -> int:
    """
    Switch player

    Parameters:
    ----------
    current_player: int
        The id (index) of the player whose turn it is now.

    Returns:
    -------
    int
        The id of the player to go next.
    """
    pass

def main():
    """
    The main flow of the game and the return of the winner id (index).

    Returns:
    -------
    int
        The id of the winner, if none return 0.
    """
    pass
```

Global Variables

```
board = [0, 0, 0, 0, 0, 0, 0, 0, 0]   # 3x3 game board
player = 1                            # X goes first
```

```
moves_left = 9                      # Number of moves so far
winner = 0                          # "Nobody" is winning to start
```

Program Behaviors

- The program displays a 3x3 game board with numbers in cells and prompts the user for X's move.
- The program rejects invalid inputs (non-numeric, numbers 1 to 9, and other ambiguous input) gracefully and prompts the user for valid input as shown in the sample output.
- The program rejects moves into occupied spaces and prompts the user for a different move.
- When a player makes a valid move, the board is displayed with their player name in the correct cell. Their name is displayed in that cell for the remainder of the game.
- The program correctly determines vertical, horizontal, and diagonal wins.
- The program displays the correct winner (Nobody, X, or O) when the game is over.

PROGRAM CODE

- `make_move()` is implemented to match functionality specified in the docstring.
- Docstring is correctly added for `check_win_horizontal()` to match function implementation.
- Docstring is correctly added to `check_win_vertical()`.
- `check_win_vertical()` is correctly implemented and matches the docstring.
- The docstring is correctly added to `check_win_diagonal()`.
- `check_win_diagonal()` is correctly implemented and matches the docstring.
- `next_player()` is implemented to match functionality specified in the docstring.
- The code is readable and interpretable.

IMPORTANT NOTE ON DOCSTRINGS

You should use the Numpy format for ALL docstrings in this assignment and may lose points if you do not. The general format is below and there are examples you may follow. You can also read more about Numpy string formatting at: Napoleon Sphinx Example

```
def function_with_types_in_docstring(param1, param2):
    """
    Example function with types documented in the docstring.

    This is where you would describe in plain English what your function is
supposed to do.

    Parameters
    ----------
    param1 : int
        The first parameter. You should give some helpful information about this
if necessary.
    param2 : str
        The second parameter. You should give some helpful information about this,
if necessary.

    Returns
```

```
        -------
        bool
            True if successful, False otherwise.
        """
```