Lab 3:
Exes and Ohs

Due date: February 12

Overview

Yay for more games! In this lab you will implement a simple Tic-Tac-Toe game using multidimensional arrays and functions in C++. Your game will **not** detect winners (unless you're ambitious...) but will instead terminate after 9 valid moves are played. The general flow of your program will go as such:

1. Use a two-dimensional 3x3 array to represent the game board. Identify the **smallest** integral type in C++ that you can use to hold the game's state, and make your board a 3x3 array of that type. (**Hint:** lots of things are "numbers" even if you don't think of them that way...) The board will hold one of three distinct values at each position: `0` if the space is empty; `1` if the space is taken by the "X" player (player 1); `-1` if the space is taken by the "O" player (player 2). Since the board is initially empty, it should be initialized to contain only `0` values. Note that `0` is different than `'0'`.

2. Keep track of which player's turn it is (X goes first). Output the current state of the board (see below) and then ask the current player to choose a move. Moves need to be inputted in the form `r,c` where `r` is a 0-based index for the row and `c` is a 0-based index for the column. In this format, `0,0` is the upper left corner of the board and `1,1` is the "middle" space.
   Hint: you will need to use `cin` to scan these values from the user, but the comma gets in the way. Use a "dummy" variable appropriate for storing a comma, and `cin` that variable to remove the comma from the keyboard buffer.

   In this step, you will use a function called `GetMove`, which takes two integer pointers and fills them in with the inputs from the user. The `main` function will pass in pointers to local variables to the function `GetMove`, and when `GetMove` is done, `main`'s variables should be updated with the input.

3. Check the square the user chose to make sure it is in bounds and currently empty (what value will be in the array if it's empty?). If the square is occupied, loop and ask the user to choose a valid move again. **Warning**: what will happen if you access the matrix at a square that is not in-bounds?

4. Once a valid square is chosen, update the game board by writing a 1 or -1 to the chosen space according to whose turn it is.
   Hint: you can do this **without** an if statement if you make an intelligent choice for how to keep track of whose turn it is.

5. Change your variable for whose turn it is, then loop back to step 2. (Output the board; ask for a move; verify the move; update the game.) Complete a total of **nine** game loops, one for each square in the board.
   Hint: if you want to quit your program in the middle of a loop, press Control+C.

6. Don't worry about checking for a winner – you can get full points on the lab without detecting a winner. If you are ambitious, you should try writing a function to check for a winner. Write the function using loops instead of manually checking all possible direction combinations.
   Hint: code like "`if (board[0][0] == board[0][1] && board[0][1] == board[0][2])`" is a poor solution to this problem and won't help you when we start on our big projects.

7. That's it!

Starting Off

1. Download the file `TicTacToe.h` from the course website. This header contains declarations for functions you will need to write and use in your implementation, along with comments about how they should

work. **You must follow the design laid out in this .h file.**

2. Create a project and copy the `.h` file to the project's folder, then add it to the Header files of the project. Add a new file `TicTacToe.cpp` to the Source files of the project. In this file, you will implement the methods declared in `TicTacToe.h`, **as well as** the main for your program.

Output Formatting

User input is in *italics*.

When outputting the game board, make it look like this:

```
- 0 1 2
0 . . .
1 . X .
2 O . .
X's turn:  1,1
That space is already taken!
X's turn:  2,2
That space is already taken!
X's turn:  1,0

- 0 1 2
0 . . .
1 X X .
2 O . .
O's turn:
```

Put the 0-based indices of each row/column around the perimeter. Use periods to show an "empty" square. Put a capital "X" or "O" depending on who is in each space.

Deliverables

Turn in the following when the lab is due:

1. A printed copy of your code, **printed from Visual Studio or your IDE when possible.** If you cannot print from your editor, copy your code into Notepad or another program with a fixed-width (monospace) font and print from there.
2. Because the output of this program is lengthy and verbose, you do not need to print a copy of a full run of the program. Simply print the output of the program after **2 moves**: one for X and one for O.