Web Development – Mr. Turner

Project – Two-Layer Tic Tac Toe

Project Overview

Tic Tac Toe is the game where you have to get your X's and O's 3 in a row. It's the game that usually ends in a tie, but what happens if you need to place a letter twice in order to actually capture the space?

This version of Tic Tac Toe plays mechanically like regular Tic Tac Toe. One player is **X** and the other is **O.** Turn by turn, each player puts his or her letter into one of the squares on the 3X3 grid. If there are 3 in a row (vertically, horizontally, or diagonally) that player is the winner.

Two Layer Tic Tac Toe differs in that players can take spaces away from one another. A space can only be locked if you take it twice. For example, if you put your **X** on the center square, the other player can place his or her **O** on the center square later and take it away from you. If you place a second **X** on the center square before your opponent takes it away, you've claimed it for good.

Because of the layered nature of the game, there are a couple of new rules.

If there is a letter on the space, whether once or twice, the space is considered held by that player. A player *can* win the game with connected single-layer captures.

A player may not move on the same space his or her opponent took on the previous turn. ***For example, if I take the center square, you cannot move onto the center square on your very next turn. You can only do it on a later turn.***

Display

The list below includes the essential elements of the page.

- An interface for the player choosing **X** or **O**.
- The board.
    - The board consists of 9 squares.
    - Each of the 9 squares will start empty, but may reflect a first layer capture or a lock (a player capturing the square for a second time).
- A reset button.
- Use CSS to dress up the page nicely.

Functionality

A human will play against the computer.

At the very beginning of the game, the human player will choose **X** or **O**.

X will always go first.

On the player's turn, they must choose a space by clicking on it. The player may choose a space that is:

- Empty.
- Occupied as a single layer by them.
- Occupied as a single layer by the computer ***if*** the computer did not capture it with its last move.

All other moves are illegal and should not be allowed.

On the computer's turn it will choose a space at random. The computer must adhere to the same restrictions as the player.

After each turn, check to see if a win condition exists. A player must connect 3 in a row vertically, horizontally, or diagonally. First layer and locks can string together for a win.

If the player clicks the reset button, clear the board and let him or her choose **X** or **O** again.

Enhancements

- Computer AI
    - Try to give the computer some semblance of strategy. There are a number of strategies you might employ.
    - An aggressive player will look to take spaces and worry less about blocking.
    - A defensive player will always try to prevent the human from grabbing too many spaces.
    - A strategic player will try to anticipate moves, calculating board position in advance.
    - Start small and see what opportunities open up with this.

Necessary Programming Skills

- Comprehension of the specifications sheet.
- Design Document
    - Figure out the information you need to keep track of.
        - This information will become your global variables.
    - Plan out the individual tasks your program must perform.
        - Think through the steps for each task.
        - Think through the information your task needs (where does it come from?).
        - These will become your functions.
    - Plan out the user interface.
        - You can start with the barest interface, but you should have an idea what you want the final product to look like.
- Managing your variables
    - What's global, local, and passed through as parameters (hint - this program can use all three)?

    - Are you making groups of variables into arrays?
    - Do you have a complete back end design (variables and functions that work the program)?
    - Does your back end inform your display?
- Sequencing
    - Does your program sort out the different tasks into their own functions?
    - Does your program sequence from the user interaction into the necessary functions?
    - Is there an efficiency to your code that flows from the design document?
- An intuitive user experience
    - Is your display appropriate to the program (what's viewable and what scrolling has to be done)?
    - Is your display adaptable to other resolutions?
    - Is the interface intuitive?