**CS 20 Web Programming  
Assignment 4: Tic Tac Toe**

## Summary: Your assignment is to create a single page app that implements Tic Tac Toe using Javascript.

## Overview

* Create a 3 x 3 grid to enter X’s and O’s
* Play will proceed as follows:
  + Indicate whether it is the turn for X or O
  + When the user clicks on one of the grid squares, place an X or O on that square (depending on whose turn it is)
  + Check for a “win” condition (3 of the same symbol in a row)
    - If so, congratulate the winner. Game over.
  + If there is no win yet, check if all squares are occupied.
    - If so, announce the stalemate. Game over.
  + Otherwise switch to the other player/symbol (X or O) and repeat.
* Add a button to start a new game
* The game will be broken into several parts – you must complete each in order.

## Ground Rules

* You may use HTML, CSS, Javascript and JQuery only
* We will be judging the quality of the assignment on your use of loops, functions and arrays.
* **You must follow the instructions to complete the assignment one step at a time for credit. All files must be submitted!**
* You may host anywhere you want, but all files must work online.
* You may add enhancements or variations after meeting the requirements.
* **DO NOT look up how to do this online.** If we detect that you adopted a solution from elsewhere, you will receive a 0

## Rubric (up to 10 points extra credit)

Part 1 20 points

Part 2 25 points

Part 3 25 points

Part 4 25 points

Effort/deliverables 15 points

**Part 1 - ttt1.html**

## Build a tic tac toe board using HTML, CSS. Add CSS as needed to create the 3 x 3 grid of squares. Each square should be created using a <div> element. Do not use <table>

The squares should end up looking similar to this- note that these are 100px by 100px with a 5px margin. You may size differently.  
  
A grid of squares with black lines

Description automatically generated

## Hard code two X’s and one O in 3 different squares of your choice. This will help you to visualize the position and formatting of the X’s and O’s correct. The X or O should be large enough to fill most of the square.

## Add CSS to change the background of a square when you hover over it. All CSS should be within an internal style sheet.

## **Part 2 – ttt2.html**

* Create a second file (ttt2.html) to redo this problem. You may reuse any code from ttt1.
* You must use the following JavaScript code *AS IS* to create the <div>’s. You may need to adjust your CSS.

const NUM\_SQUARES = 9;

for (i=0; i<NUM\_SQUARES; i++)

{

id = "sq"+i;

document.write( "<div class='square' id='" + id + "'></div>" );

}

* Add a cool title on the page above the squares using an H1
* Add a <div> section on the page above the board to indicate whether it is X’s turn or O’s turn – put a border around the div so that we can see its placement
* Add a <div> on the page to display a win condition below the board – put a border around it so that we can see its placement
* Using JavaScript, place an X in row 1, column 3 and place an O in row 2, column 1
* The background of each square should change when you hover over it.

**Part 3 – ttt3.html**

* Add an event handler for each <div> that represents the squares such that when you click in a square it will put an X in the square and a popup will display the number of the square. 0 is the first square up to 8 for the last square. Use one event handler function for all squares. You may add JavaScript, but you may not change the code from ttt2
* Once you have the event handlers working, alternate between X’s turn or O’s turn. Put the appropriate symbol (X or O) in the square when clicked. The popup should still be working.
* Indicate whose turn it is in the message area above the board as created in part 1
* A square should not be allowed to be changed once an X or O has been placed.
* Randomly choose whether X or O will start.
* Add a button called “Start Game” that will clear the board and allow squares to be filled

**Part 4 – ttt4.html (The Finale!)**

* After a user clicks on a square, place the X or O in the square and then do the following:
  + Using forEach(), create a new array called *moves* with the current X or O value that is in the corresponding div- ie, moves[0] will have the value inside of the div with   
    id = sq0. Represent empty squares with a hyphen (-).  
    (hint: use the innerText or innerHTML property of the <div> to get the value inside)  
      
    for example, if the board is:

X | O | O

X | X |

O | |

the array elements would be: X, O, O, X, X, -, O, -, -

* + Create an array called *winConditions* of 8 strings (based on the array) to represent the 8 possible win conditions (3 across, 3 down, 2 diagonal), Only put X’s and O’s in the strings.  
    In this case, the first 4 elements are: “XOO” , “XX-” , “O--“ , “XXO” , etc.
  + Identify any that are XXX or OOO to determine if there is a win
  + If none, look for any hyphens in your original array- if there are none, it’s a stalemate. (stalemate means all squares full and no winner)
  + If there is a win or stalemate condition, display the status and disallow any additional “moves” until the button is clicked to restart the game
  + Otherwise the game continues

## Deliverables Worksheet

* This worksheet in a Word document
* All files: ttt1.html through ttt4.html compressed in a zip file and uploaded to canvas

URL of each page working online

ttt1.html \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ttt2.html \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ttt3.html \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ttt4.html \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Answer:

What was the most challenging part of this assignment?

The most challenging part was setting up the 3 by 3 grid! For the longest time, I couldn’t figure out the CSS, and I was making a 1 x 9 column instead!

What is the section of code you are most proud of (include the code in your answer).  
I am proudest of the section of code used to create the winConditions array.

let winConditions = winConditionIndices.map(condition => {

                        return condition.map(index => moves[index]).join('');

                    });

I use mapping functions to connect an array of sets of 3 ints (indices representing a row, column, or diagonal) to the moves array (which stores moves in the form of characters). Then, I join the moves that were played on the sets of 3 indices to quickly generate the winConditions array, which contains strings that are 3 characters long.