

CS 696A Full-Stack Entrepreneurs App Development - Lab 3

Prof. Arya Boudaie - Fall 2025

Pace University

# OBJECTIVE

In this lab, you will create a fully functional Tic-Tac-Toe game using **HTML**, **CSS**, and **JavaScript** from scratch, and deploy it to your Github Pages created in Lab 2.

# DUE DATE

## Tuesday September 16th at 6:00 PM EST SUBMISSION INSTRUCTIONS

You may work together in your assigned groups, but do not copy anyone’s code. You may

consult LLMs about certain concepts if you are stuck, but ask it to not generate code. Try to figure things out on your own or using online resources so that you build core development fundamentals and learn from your mistakes. Do not submit any LLM generated code. It is suggested that you experiment with different things and see what they do; e.g. comment out some CSS and see why (or if) it is neeeded.

Follow the submission guidelines at the end of the lab.

# INSTRUCTIONS

Follow the steps below in order. Each step will build on the previous ones. I suggest you commit your code after every step in case you need to roll back.

## Part 1: Style

1. Initialize a git repository and connect it to GitHub.
2. Create three files: index.html, app.js, and style.css. Your HTML file should look like this - type it out by hand and note how I am linking the CSS and JS to the page. The viewport

is what allows the page to be responsive. You can use an LLM to help you understand any part of this that is confusing.

1

<!**doctype html**>

<**html**>

<**head**>

<**meta charset**="utf-8">

<**meta name**="viewport" **content**="width=device-width,initial-scale=1">

<**title**>Tic-Tac-Toe Game</**title**>

<**link rel**="stylesheet" **href**="style.css">

<**script src**="app.js"></**script**>

</**head**>

<**body**>

Your App Goes Here

</**body**>

</**html**>

2

3

4

5

6

7

8

9

10

11

12

13

14

1. In index.html body, remove the placeholder text and add:
   * A <div> for the page container.
   * A <div> for the board with id="gameBoard".
   * Three <div> elements for rows inside the board div, each with a CSS class.
   * Three <div> elements for cells in each row, each with a CSS class.
2. Save and run your project with this code:

python3 -m http.server

Open in the browser. Think about what you see and **why**. Hint: it should be blank.

1. In style.css, create a .cell class with these attributes:
   * width: 60px;
   * height: 60px;
   * border: 1px solid red;
   * background: cyan;

Refresh and think about what you see and why based on your CSS.

1. Using <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>, write CSS classes for .board and .row.
   * Each should have display: flex.
   * Think about the correct flex-direction for each, or try them randomly and see what happens.
2. Add a gap attribute between rows and between cells by adding it to the .board and .row

classes.

1. Add an X inside one of the cell divs temporarily. Use align-items and justify-content

to center it. Adjust the font size until it looks good. Remove the temporary X.

1. Use align-items and justify-content to center the whole board on the page.
2. Change any styling you want to make it look how you would like a tic-tac-toe board to look like. You can come back and do this.

## Step 2: Javascript Logic

1. In app.js, log "Hello World" to the console. Refresh the page and confirm it appears in the browser’s developer console.
2. In app.js, set up variables to keep track of the game state. Some values to include:
   * The game board array
   * Whose turn it is
   * Whether the game is over
   * The winner

Note that these should just be declared at the top of the file, no need to put it in a function.

1. Write a function initializeGame() that sets all variables to their default values.
2. Write a function renderBoard() that updates the HTML to match the board state. Hint: Use document.getElementById or getElementsByClassName and element.innerText = "X" to set values. You can modify the id’s of certain divs, or iterate through them.

Test it by setting the board array with some X’s and O’s, but don’t forget to revert these tests.

1. Write a function checkWinner() that checks if there is a winner. Update gameOver and winner variables accordingly. Test by passing in different board arrays and logging the results.
2. Write a function cellClicked(index):
   * If the cell is empty and the game is not over:
     + Update the board array
     + Update game state variables
     + Check for a winner
     + Render the board
   * Otherwise, return without doing anything. Test by calling it manually in the console.
3. In index.html, add onclick="cellClicked(INDEX)" to each cell, where INDEX matches that cell’s index in your board array (if you made it a 2D array update accordingly). Test by clicking cells.
4. Add a Reset button in HTML that calls initializeGame(). Hint - use <Button> and set the onclick accordingly.
5. Add a status text in HTML that shows:
   * Whose turn it is
   * The winner when the game is over Update it in renderBoard().
6. Add one custom modification:
   * Score tracking
   * Fun win animation
   * Auto generated board instead of hardcoding.
   * Sound effects
   * Emojis for X and O
   * Have one of the players be a bot
   * Anything else creative

## Deployment and Submission

1. Make sure your lastest code changes are on GitHub.
2. In the repository settings, go to pages, set source equal to "deploy from a branch", and set the branch as your main branch and folder as root, and save changes.
3. Your web application should be available at yourusername.github.io/repositoryname/ where ‘yourusername‘ is your username and ‘repositoryname‘ is your repository name.
4. Submit a word document to the classes page with the following:
   1. Link to your repository

**GitHub:** <https://deepesh-katudia.github.io/>

* 1. Link to your live demo

**Live Demo:** <https://drive.google.com/file/d/1w3vPKLPQ_kkTLSviVB2Fomebb5ubPe8D/view?usp=sharing>

* 1. What modification did you make?

I implemented several enhancements beyond the base requirements:

* **Game mode switcher:** players can choose **Two Players** or **Vs Bot** from the HUD.
* **Bot player (O):** simple strategy (win → block → center → corners → sides).
* **Win animation:** the three winning cells “pop” to celebrate a win.
* **Scoreboard & HUD:** running scores for X, O, and Draws with a prominent status line.
* **Layout polish:** centered page, clean card styling for the HUD, and accessible buttons.  
  *(Earlier in the build, I also explored auto-generating the 3×3 board from JavaScript.)*
  1. In a few sentences, reflect on this assignment. What was easy for you? What was difficult? What don’t you understand?
* **What was easy:** Structuring the HTML, centering the layout with Flexbox, and wiring up the status/scoreboard were straightforward. Rendering the board state to the DOM using innerText felt natural.
* **What was difficult:** Balancing UI updates with game logic—especially making sure the bot only moves when it’s truly its turn and stopping all moves after gameOver—took a bit of debugging. Designing the win animation so it only triggers on the winning cells also needed careful state handling.
* **What I still don’t fully understand:** I’d like to deepen my understanding of building a stronger AI (e.g., Minimax with pruning and difficulty levels) and the subtle differences between using Flexbox vs. Grid (and rules like display: contents) for grid-like UIs.
  1. AI Attestation.

I used AI (ChatGPT) only to understand concepts and brainstorm approaches for this assignment. I did not copy or submit any AI-generated code. All code and written work in my repository and live demo were authored and tested by me.