Tic-Tac-Toe Game
Project Report

Project Title: Tic-Tac-Toe with Html, CSS and JavaScript

Your Name: Swarnil Bhowmick

Course/Subject: Masters In Computer Application Graduate

Date of Submission: 8th October, 2024

Abstract

This project is a web-based Tic-Tac-Toe game developed using HTML, CSS, and JavaScript, enhanced with the Bootstrap framework for responsive design. The game allows two players to play against each other, taking turns to place their marks (X or O) in a 3x3 grid. The game detects wins, tracks the current player, and provides a restart functionality. The main goal was to create an interactive and visually appealing game that demonstrates fundamental web development skills.

INDEX

Introduction

Background

Tic-Tac-Toe is a classic paper-and-pencil game often played by children. It serves as an excellent introductory project for those learning programming and game development concepts.

Objectives

The primary objectives of this project were to:

- Implement the game logic for Tic-Tac-Toe.
- Create an engaging user interface using HTML and CSS.
- Enhance the interface with Bootstrap for responsiveness.
- Allow users to restart the game easily.

Scope

This project covers the implementation of a two-player Tic-Tac-Toe game with functionalities including:

- Player turns and marking of the grid.
- Detection of winning conditions.
- A reset button to restart the game.

System Analysis & Design

Functional Requirements

- Players can take turns placing their marks in the grid.
- The game detects when a player has won or if the game ends in a draw.
- A restart button to reset the game state.

Non-Functional Requirements

- The application should be responsive and visually appealing on various screen sizes.

Architecture/Design

The game's user interface consists of a grid of nine boxes arranged in a 3x3 layout. Each box can display either an "X" or an "O", depending on the current player's turn. The user interface is styled using CSS with Bootstrap to ensure a responsive layout.

Development Environment

- **Languages Used:** HTML, CSS, JavaScript
- **Framework:** Bootstrap

Implementation

The game consists of three main files: index.html, style.css, and script.js which contains html, CSS and JS code for the project, accordingly.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tic-Tac-Toe</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
crossorigin="anonymous">
    <link rel="stylesheet" href="style.css">
</head>
<body>

    <div class="myContainer">
        <h1 id="playerText">Tic-Tac-Toe</h1>
        <button id="restartBtn">Restart</button>

        <div class="container adjustBox">
            <div class="row">
                <div class="box" id="0"></div>
                <div class="box" id="1"></div>
                <div class="box" id="2"></div>
                <div class="box" id="3"></div>
                <div class="box" id="4"></div>
                <div class="box" id="5"></div>
                <div class="box" id="6"></div>
                <div class="box" id="7"></div>
                <div class="box" id="8"></div>
            </div>
        </div>
    </div>



    <!-- script part -->
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min
.js" integrity="sha384-
```

```
YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
crossorigin="anonymous"></script>
    <script src="script.js"></script>
</body>
</html>
```

style.css

```
@font-face {
    font-family: "Short Plan";
    src: url(./Fonts/Short\ Plan.otf);
    src: url(./Fonts/Short\ Plan.ttf);
}

*{
    padding: 0;
    margin: 0;
    box-sizing: border-box;
}

/* css variables */
:root {
    --winningBoxColor: rgb(56, 65, 92);
}


body{
    background: rgb(73, 81, 105);
}

h1 {
    font-size: 80px;
    font-family: "Short Plan", Arial, sans-serif;
    color: yellow;
}

.myContainer {
    display: flex;
    align-items: center;
    justify-content: center;
    padding: 20px 40px 40px 40px;
    flex-direction: column;
}

.box{
    height: 150px;
    width: 150px;
```

```
    display: flex;
    align-items: center;
    justify-content: center;
    font-family: "Short Plan", Arial, sans-serif;
    font-size: 120px;
    color: yellow;
    border-right: 2px solid orange;
    border-bottom: 2px solid orange;
}

.box:nth-child(3n) {
    border-right: none;
}

.box:nth-child(n+7) {
    border-bottom: none;
}


.adjustBox{
    width: 450px;
    margin-top: 40px;
}

button {
    padding: 10px 20px;
    border-radius: 20px;
    background-color: orange;
    border-color: orange;
    color: rgb(73, 81, 105);
    font-weight: bold;
    font-size: 18px;
    transition: 200ms transform;
}

button:hover {
    cursor: pointer;
    transform: translateY(-2px);
}


@media (max-width: 575.98px) {

    .myContainer {
        min-height: 100vh
    }

    #playerText  {
        font-size: 60px;
```

```
    }

    .adjustBox {
        width: 300px;
        margin-bottom: 40px;
    }

    .box{
        height: 100px;
        width: 100px;
        font-size: 80px;
    }

}
```

script.js

```
let playerText = document.getElementById('playerText')
let restartBtn = document.getElementById('restartBtn')
let boxes = Array.from(document.getElementsByClassName('box'))

// briging css values
let winIndicator = getComputedStyle(document.body).getPropertyValue('--
winningBoxColor')

// console.log(boxes)

const round_O = "O"
const cross_X = "X"

let currentPlayer = cross_X

// click tracker, track the clicks on div
// keeping records of the X and O and empty spaces

let space_arr = Array(9).fill(null)

// game start

// game over state
let gameOver = false;

const startGame = ()=>{
    boxes.forEach(box => box.addEventListener('click', boxClicked))
}
```

```
function boxClicked(e){
    // if game over ture, stop clicks
    if (gameOver) return

    const id = e.target.id

// if space array position null, fill with current player

    if (space_arr[id] == null) {
        space_arr[id] = currentPlayer
        e.target.innerText = currentPlayer

        // if not == false, cause false by default
        const winResult = playerWin()
        if (winResult !== false) {
            playerText.innerHTML = `${currentPlayer} has won!`
            let winningBoxCross = winResult

            // color change on win
            winningBoxCross.map(box => boxes[box].style.backgroundColor =
winIndicator)

            // game over state
            gameOver = true;
            return
        }
        // if X then O else X
        currentPlayer = currentPlayer == cross_X ? round_O : cross_X
    }
}

// player win condition

const winConditions = [
    [0,1,2],
    [3,4,5],
    [6,7,8],
    [0,3,6],
    [1,4,7],
    [2,5,8],
    [0,4,8],
    [2,4,6]
]

// travers through winConditions, if space arr matches with winConditions,
// send it else send false

function playerWin(){
```

```
    for(const cond of winConditions){
        let [a,b,c] = cond

        if (space_arr[a] && (space_arr[a]==space_arr[b] &&
space_arr[a]==space_arr[c])) {
            return [a,b,c]
        }
    }
    return false
}

// restart the game

restartBtn.addEventListener('click', restart)

// space_arr will be null, box innertext values will ease, current player will
be default
function restart(){
    space_arr.fill(null)

    boxes.forEach(box => {
        box.innerText = ''
        box.style.backgroundColor = ''
    })

    gameOver = false

    playerText.innerHTML = "Tic-Tac-Toe"

    currentPlayer = cross_X
}

startGame()
```

Code Explanation

1. Overview

The Tic-Tac-Toe project is a simple web-based game built using **HTML**, **CSS**, and **JavaScript**, with **Bootstrap** for responsive design. The game features a 3x3 grid where two players take turns marking spaces with "X" or "O." The game detects a win or draw state, provides visual feedback, and allows players to restart the game.

2. Structure

The project consists of three main files:

- **index.html**: Defines the structure of the webpage and the game grid.
- **style.css**: Adds styling and visual elements to the game, including layout, colors, and fonts.
- **script.js**: Implements the game logic, handles player interactions, and updates the game state.

3. Key Components

a. HTML (index.html)

- **Grid Layout**: The grid is structured using Bootstrap's grid system ( ), where each cell (box) has a unique ID (id="0", id="1", etc.). This allows JavaScript to target specific cells when players click on them.
- **Player Information and Restart Button**:
    - **<h1 id="playerText">**: Displays the game status, such as which player's turn it is or if someone has won.
    - **<button id="restartBtn">Restart</button>**: A button that allows players to reset the game.

b. CSS (style.css)

- **Styling and Fonts**: The @font-face rule adds a custom font called "Short Plan" for a unique look. The CSS defines the colors, sizes, and alignment of various elements like the grid, buttons, and text.

- **CSS Variables**: The CSS uses a variable (--winningBoxColor) for the winning box color, allowing JavaScript to access and change the color dynamically.
- **Responsive Design**: Media queries ensure the grid and text scale down for smaller devices.

c. JavaScript (script.js)

- **Variables and Constants**:
    - **playerText:** Displays the current player's turn or the winner.
    - **restartBtn:** Represents the restart button.
    - boxes: An array that holds all the grid boxes ( ), allowing JavaScript to attach click events.
    - **winIndicator:** Gets the value of the CSS variable for the winning box color.
    - **cross_X and round_O:** Constants representing the two players (X and O).
    - **currentPlayer:** Tracks whose turn it is (initially set to X).
    - **space_arr:** An array of 9 elements initialized to null, tracking the state of each grid cell (whether it's occupied by X or O).
    - **gameOver:** A flag indicating if the game is finished.
- **Game Initialization (startGame Function)**:
    - Attaches a click event listener to each grid box, which calls the boxClicked function whenever a player clicks a cell.
- **boxClicked Function**:
    - Checks if the game is already over; if so, it returns early.
    - Gets the ID of the clicked box and updates space_arr with the current player’s symbol (X or O).
    - Updates the box's inner text to display the current player’s mark.
    - Calls the playerWin function to check for any winning combinations. If a player wins:

        - Updates playerText to announce the winner.
        - Highlights the winning combination by changing the background color of the winning boxes.
        - Sets the gameOver flag to true.
    - Switches the currentPlayer from X to O or vice versa.
- **playerWin Function**:
    - Defines all possible winning combinations in the grid (rows, columns, and diagonals).
    - Iterates through these combinations to check if any match the current state of space_arr.
    - If a winning combination is found, it returns the indices of the winning boxes; otherwise, it returns false.
- **Restart Functionality (restart Function)**:
    - Resets space_arr to null for all positions.
    - Clears the inner text and background color of each grid box.
    - Resets playerText to "Tic-Tac-Toe" and currentPlayer back to X.
    - Sets gameOver to false, allowing the game to start fresh.

4. Interactions

- **HTML and JavaScript**: The HTML structure (grid and button) interacts with JavaScript functions (boxClicked and restart) via event listeners. The state updates dynamically based on player actions.
- **CSS and JavaScript**: JavaScript accesses CSS variables and classes to update the visual state of the game (e.g., highlighting winning combinations).
- **Bootstrap Integration**: The grid layout and styling are responsive, ensuring the game looks good across different screen sizes.

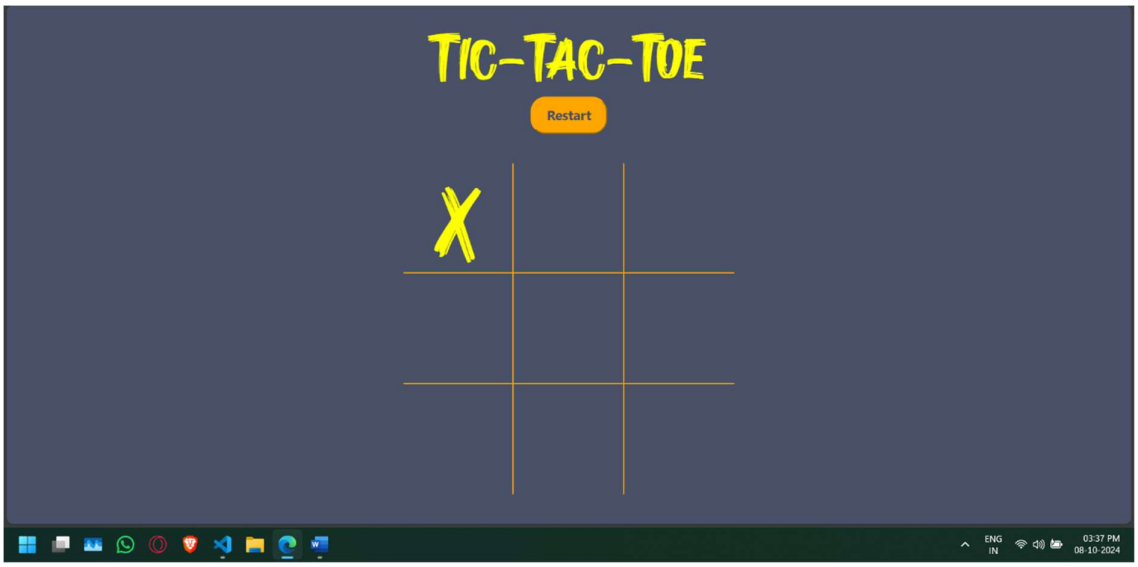
5. Challenges and Solutions

- **Tracking Game State**: The game uses an array (space_arr) to track the state of each cell, ensuring the program knows when a player wins or when to switch turns.
- **Winning Condition Detection**: The playerWin function efficiently checks all possible win conditions using a loop, ensuring the game identifies a win as soon as it occurs.
- **Responsive Design**: CSS media queries and Bootstrap classes ensure that the game layout adjusts correctly on smaller screens.

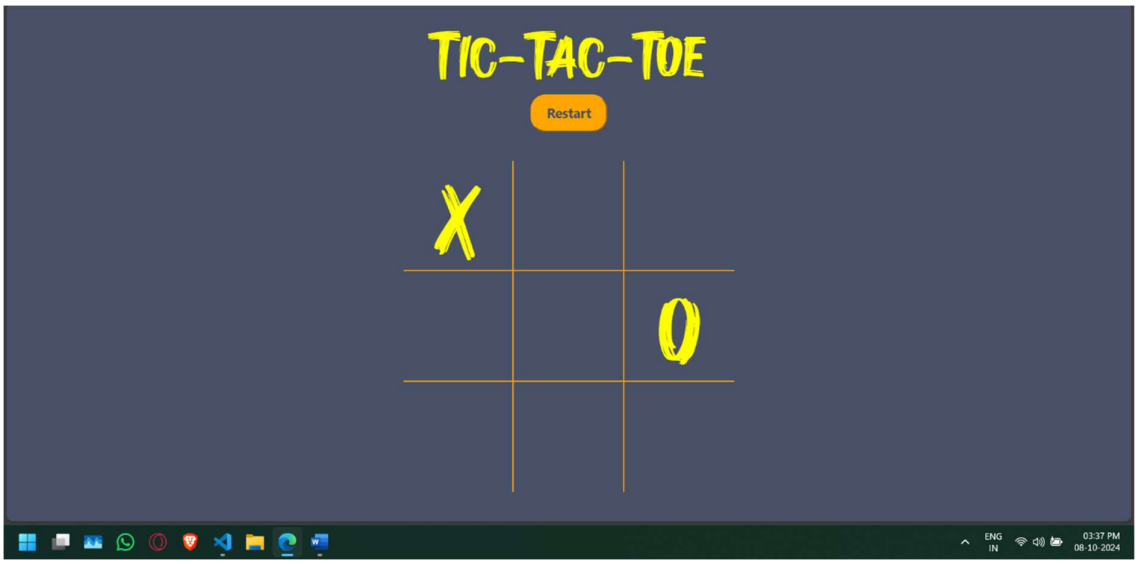
Project Screenshots

Here we can see the empty board of the game

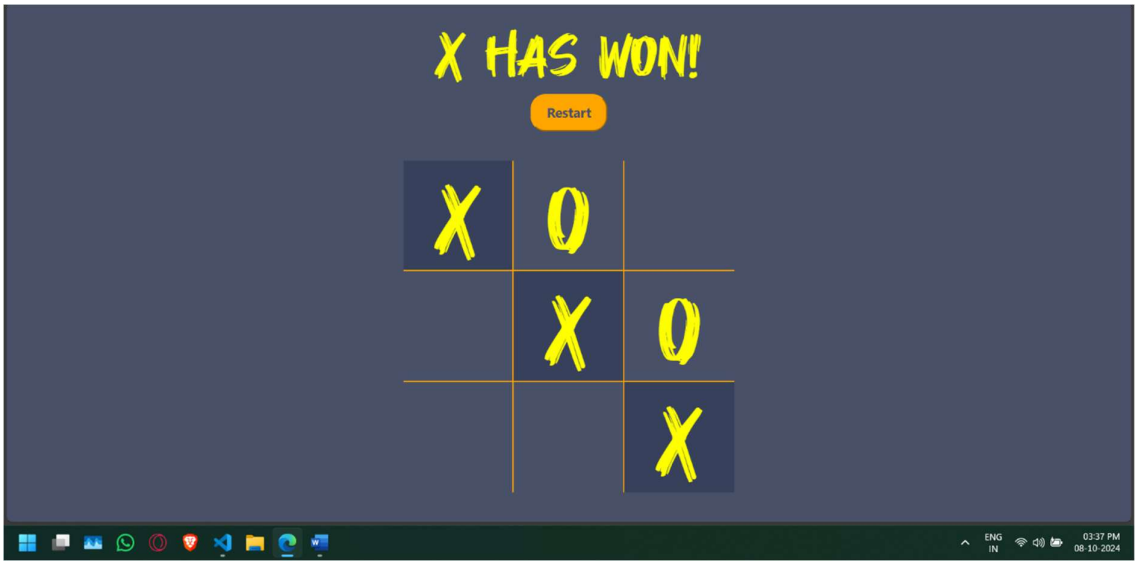
On clicking a block, we get the X symbol. X is the first move



Same way, the second click gets us the O symbol, which is the second move.

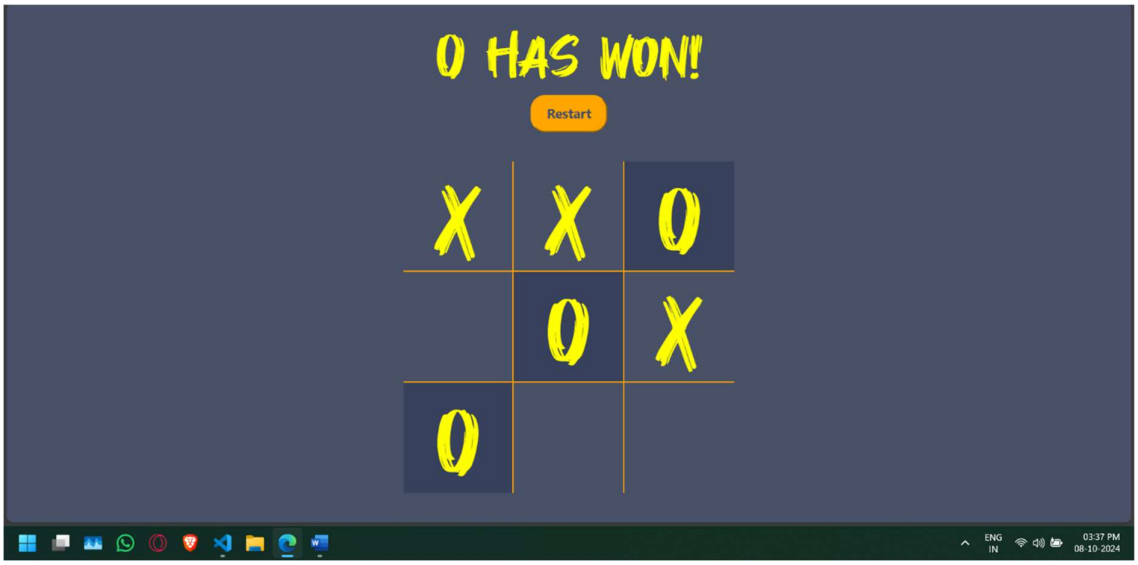


This way, the game goes on, if X player matches a win move, the boxes background changes and it shows that X has won.



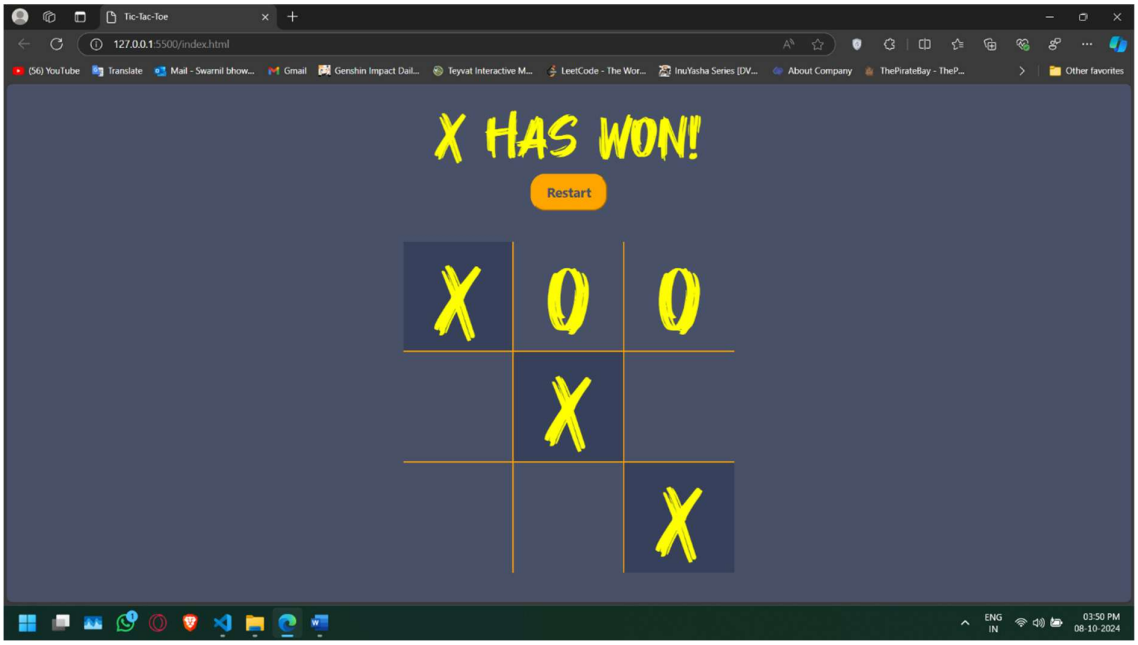
Same way, if O player matches a win move, the boxes background changes and it shows that O has won.

It is notable that when the winner is declared , no more movements on the empty square boxes can be made.

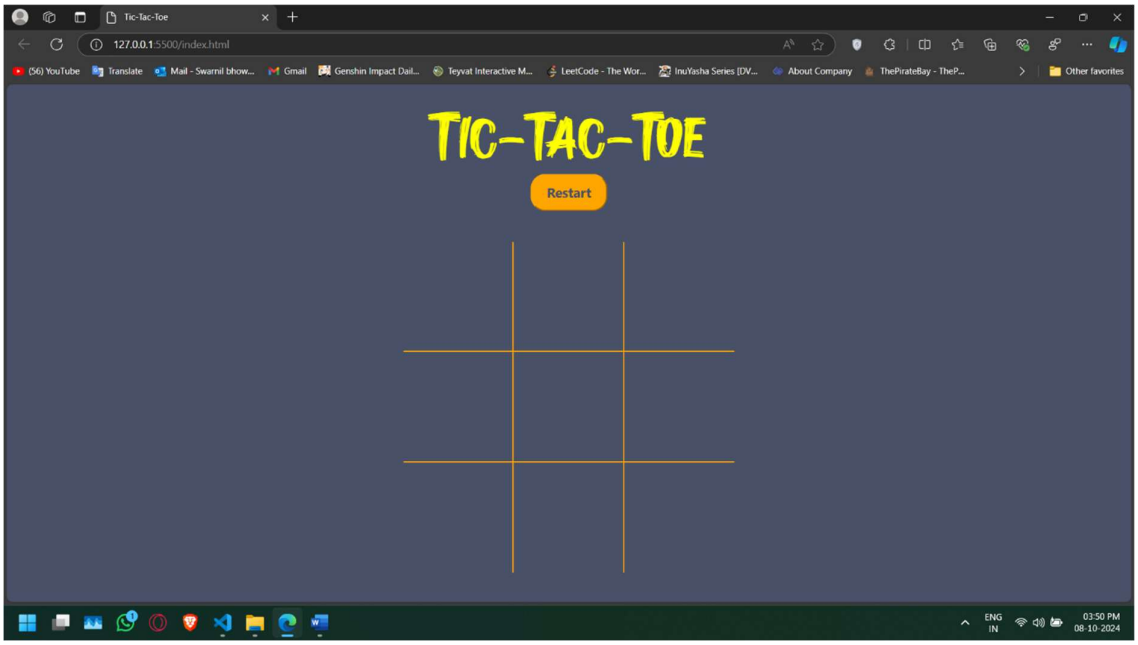


Upon pressing the restart button, the board clears and game starts form beginning.

Before pressing restart

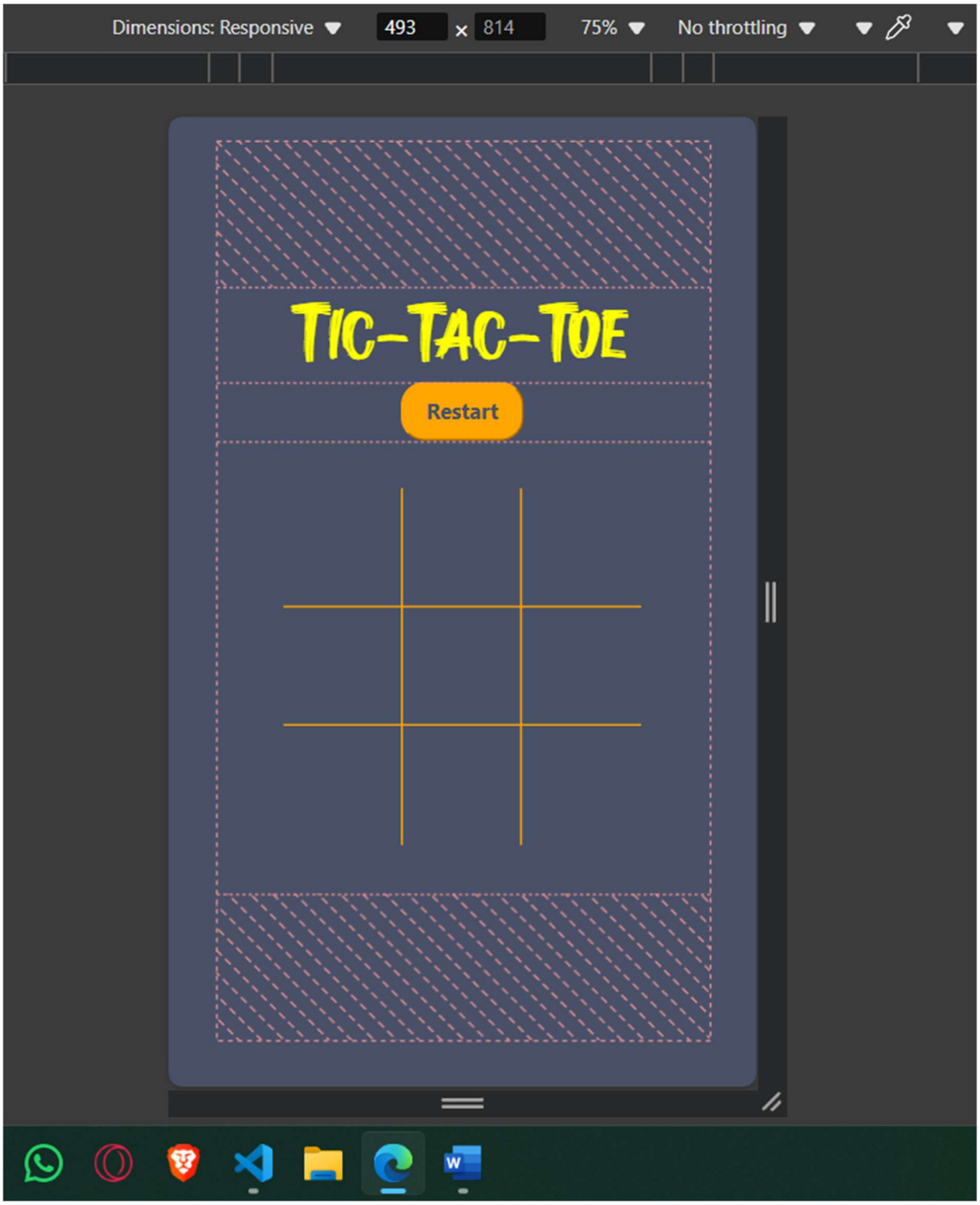


After pressing restart



Mobile View

On mobile screen, this project looks something like this.



Testing

The game was tested through various scenarios to ensure functionality:

- Valid moves were made, alternating between players.
- Winning conditions were detected correctly.
- The game reset function operated as expected.

Results and Discussion

The final implementation meets the project's objectives, providing a fully functional Tic-Tac-Toe game. The user interface is responsive, ensuring a good user experience on both desktop and mobile devices. Challenges encountered included ensuring that win conditions were accurately detected, which was resolved through careful condition checking.

Conclusion

The Tic-Tac-Toe project successfully demonstrates the application of HTML, CSS, and JavaScript in creating an interactive web-based game. The implementation of Bootstrap contributed to a modern, responsive design. Future improvements could include adding features like a single-player mode against an AI opponent and score tracking.

References

- W3Schools - HTML
- MDN Web Docs - CSS
- JavaScript.info