COVER SHEETA SIGNED STATEMENT

Introduction

Tic Tac Toe, a game of simple elegance and strategic depth, has captivated the minds of players across generations. This classic pastime, with its origins tracing back to ancient civilizations, has withstood the test of time, remaining a beloved source of entertainment and intellectual stimulation.

In an age where digital technology has revolutionized the way we interact and play, the time has come to revisit and reinvigorate the timeless Tic Tac Toe experience. This project aims to harness the power of modern web development tools and design principles to create a Tic Tac Toe system that not only stays true to the game's core essence but also elevates the user experience to new heights.

The objective of this endeavor is to design and implement a Tic Tac Toe application that captivates players, both seasoned and new, with its intuitive interface, strategic depth, and seamless gameplay. By delving into the rich history and enduring appeal of this game, we seek to uncover the unique qualities that have made Tic Tac Toe a cherished classic, and leverage those insights to craft a digital experience that resonates with the modern user.

Acknowledgements:

The development of the Tic Tac Toe system stands as a testament to the collaborative efforts and invaluable contributions of numerous individuals and resources. It is with profound gratitude that we acknowledge the following pillars of support that have enabled the realization of this digital experience:

1. \*\*Tic Tac Toe's Enduring Legacy\*\*:

The timeless allure and historical significance of Tic Tac Toe have left an indelible mark on generations of players. It is with deep appreciation that we draw inspiration from the game's enduring legacy, seeking to encapsulate its essence in our digital rendition.

2. \*\*Web Development Resources and Communities\*\*:

The vibrant web development community has been an invaluable source of knowledge, guidance, and support. Through a wealth of resources, tutorials, and collaborative forums, we have been able to harness the collective wisdom of this community to refine our technical implementation and problem-solving approaches.

3. \*\*User Interface Design Experts\*\*:

The expertise and best practices advocated by user interface design professionals have been pivotal in shaping the visual and interactive aspects of our Tic Tac Toe system. Their emphasis on accessibility, usability, and aesthetic appeal has guided us in creating an intuitive and visually engaging experience for our users.

Summary:

Tic Tac Toe is a timeless game that has captivated players for generations, with its origins tracing back to ancient civilizations. In this project, we set out to develop a robust and engaging Tic Tac Toe system that would stay true to the game's classic appeal while leveraging modern web technologies and design principles.

Through a comprehensive requirements analysis, we identified the key functional and non-functional requirements for the Tic Tac Toe application. The system needed to allow two players to compete, display the current state of the game board, detect and display the winner or a draw, and provide an option to restart the game. Additionally, the application had to feature an intuitive and user-friendly interface, responsive design, efficient game logic, and accessibility considerations.

The design phase involved crafting a visually engaging user interface, including the 3x3 grid representing the game board, distinct visual styles for the X and O marks, and clear indication of the current player's turn and the game's outcome. The game logic design focused on maintaining the game state, handling user input, implementing winning condition checks, and managing the overall game flow.

During the implementation phase, we utilized a modern front-end framework or library, such as React, Angular, or Vue.js, to build the Tic Tac Toe application. The user interface was constructed using HTML, CSS, and JavaScript, ensuring a seamless integration between the visual elements and the underlying game logic. The game mechanics were carefully coded to ensure accurate placement of marks, efficient detection of winning conditions and draw scenarios, and a smooth game experience.

The project's conclusion emphasizes the timeless appeal of Tic Tac Toe and the team's confidence in the developed system's ability to captivate players. The application's adherence to the game's classic roots, combined with the quality of its implementation, is expected to provide a delightful and accessible Tic Tac Toe experience for users. The conclusion also suggests potential future enhancements, such as the integration of AI-powered opponents, online multiplayer features, and gamification elements, to further expand the system's capabilities and longevity.

BACKGROUND RESEARCH

Tic Tac Toe, also known as Noughts and Crosses or X's and O's, is a classic game that has been played for centuries. The origins of the game can be traced back to ancient civilizations, with evidence of similar games found in Egyptian, Greek, and Roman cultures.

The modern version of Tic Tac Toe is believed to have emerged in the late 19th century, with the earliest known published reference to the game dating back to 1896. The game's simplicity and accessibility have made it a beloved pastime for people of all ages, from children learning strategy to adults seeking a quick intellectual challenge.

Tic Tac Toe is played on a 3x3 grid, with two players taking turns placing their marks (usually X's and O's) on the board. The objective is to be the first player to get three of their marks in a row, either horizontally, vertically, or diagonally. The game is known for its simple rules, quick gameplay, and the opportunity for both strategic thinking and a touch of luck.

Throughout its history, Tic Tac Toe has been the subject of extensive research and analysis. Mathematicians and computer scientists have studied the game's optimal strategies, and it has become a common example in the field of game theory. The game has also found its way into popular culture, appearing in various forms of media, from books and movies to television shows and digital games.

1. \*\*Requirements Analysis and Specification\*\*:

- Functional requirements:

- Allow two players to play a game of Tic Tac Toe.

- Display the current state of the game board.

- Detect and display the winner or a draw.

- Provide an option to restart the game.

- Non-functional requirements:

- Intuitive and user-friendly interface.

- Responsive design to work on various devices.

- Efficient game logic and performance.

- Accessible to users with disabilities (e.g., keyboard navigation).

2. \*\*Design\*\*:

- User Interface (UI) Design:

- Create a 3x3 grid to represent the game board.

- Utilize distinct visual styles for the X and O marks.

- Display the current player's turn and the winner/draw message.

- Include a restart button to allow players to start a new game.

- Game Logic Design:

- Maintain the game state (current player's turn, game board, and winner status).

- Implement functions to handle user input (placing marks on the board).

- Develop algorithms to check for winning conditions and draw scenarios.

- Manage the game flow, including turn switching and game ending conditions.

3. \*\*Implementation\*\*:

- Technologies and Tools:

- Utilize HTML, CSS, and JavaScript to build the Tic Tac Toe application.

- User Interface Implementation:

- Create the game board structure using HTML elements (e.g., `<div>` for cells).

- Apply CSS styles to visually represent the game board, marks, and other UI elements.

- Implement event handlers to capture user interactions (e.g., clicking on the cells).

- Game Logic Implementation:

- Develop functions to manage the game state, including the current player's turn and the game board.

- Implement the winning condition check and draw scenario detection.

- Handle the game flow, including turn switching and game ending conditions.

- Integrate the UI interactions with the game logic.

4. \*\*Testing and Evaluation\*\*:

- Unit Testing:

- Write tests to ensure the correctness of individual game logic functions (e.g., placing a mark, checking for a win, detecting a draw).

- Verify the implementation of the winning condition and draw scenario algorithms.

- Integration Testing:

- Test the overall game flow, including the interaction between the UI and the game logic.

- Validate that the game correctly handles various user input scenarios (valid moves, invalid moves, game restart).

- User Acceptance Testing:

- Involve end-users to test the usability and overall user experience of the Tic Tac Toe application.

- Gather feedback on the interface, game flow, and overall satisfaction with the application.

- Performance Evaluation:

- Assess the application's responsiveness and performance, especially for rendering the game board and handling user interactions.

- Optimize the game logic and rendering to ensure a smooth and efficient gaming experience.

5. \*\*Evaluation and Deployment\*\*:

- Analyze the test results and user feedback to identify areas for improvement.

- Implement necessary enhancements and bug fixes based on the evaluation.

- Package the application for deployment, ensuring it can be easily accessed and used by the target audience.

- Monitor the deployed application and gather feedback for future iterations and improvements.

Test Log & Test Plans

Here's an example of a test log for the Tic Tac Toe project:

Tic Tac Toe Application Test Log

Test ID: TT001

Test Name: Verify Player X Placement

Objective: Ensure that the player X can place their mark on the game board correctly.

Steps:

1. Launch the Tic Tac Toe application.

2. Click on an empty cell on the game board.

3. Verify that an 'X' is displayed in the selected cell.

4. Repeat steps 2-3 for different empty cells on the board.

Expected Result: The 'X' mark is displayed in the selected empty cell.

Actual Result: Pass

Comments: The player X placement functionality is working as expected.

Test ID: TT002

Test Name: Verify Player O Placement

Objective: Ensure that the player O can place their mark on the game board correctly.

Steps:

1. Launch the Tic Tac Toe application.

2. Click on an empty cell on the game board.

3. Verify that an 'O' is displayed in the selected cell.

4. Repeat steps 2-3 for different empty cells on the board.

Expected Result: The 'O' mark is displayed in the selected empty cell.

Actual Result: Pass

Comments: The player O placement functionality is working as expected.

Test ID: TT003

Test Name: Verify Winning Condition (Horizontal)

Objective: Ensure that the game correctly detects and displays the winning message when a player gets 3 marks in a horizontal line.

Steps:

1. Launch the Tic Tac Toe application.

2. Place 3 'X' marks in a horizontal line on the game board.

3. Verify that the winning message is displayed, indicating that player X has won.

4. Click the "Restart" button and repeat the test with player O.

Expected Result: The winning message is displayed correctly when a player gets 3 marks in a horizontal line.

Actual Result: Pass

Comments: The horizontal winning condition detection is functioning as expected.

Test ID: TT004

Test Name: Verify Winning Condition (Vertical)

Objective: Ensure that the game correctly detects and displays the winning message when a player gets 3 marks in a vertical line.

Steps:

1. Launch the Tic Tac Toe application.

2. Place 3 'O' marks in a vertical line on the game board.

3. Verify that the winning message is displayed, indicating that player O has won.

4. Click the "Restart" button and repeat the test with player X.

Expected Result: The winning message is displayed correctly when a player gets 3 marks in a vertical line.

Actual Result: Pass

Comments: The vertical winning condition detection is functioning as expected.

Test ID: TT005

Test Name: Verify Winning Condition (Diagonal)

Objective: Ensure that the game correctly detects and displays the winning message when a player gets 3 marks in a diagonal line.

Steps:

1. Launch the Tic Tac Toe application.

2. Place 3 'X' marks in a diagonal line on the game board.

3. Verify that the winning message is displayed, indicating that player X has won.

4. Click the "Restart" button and repeat the test with player O.

Expected Result: The winning message is displayed correctly when a player gets 3 marks in a diagonal line.

Actual Result: Pass

Comments: The diagonal winning condition detection is functioning as expected.

Test ID: TT006

Test Name: Verify Draw Condition

Objective: Ensure that the game correctly detects and displays the winning message when the game ends in a draw.

Steps:

1. Launch the Tic Tac Toe application.

2. Place alternating 'X' and 'O' marks until all cells are filled.

3. Verify that the winning message is displayed, indicating that the game is a draw.

4. Click the "Restart" button and repeat the test.

Expected Result: The winning message is displayed correctly when the game ends in a draw.

Actual Result: Pass

Comments: The draw condition detection is functioning as expected.

Test ID: TT007

Test Name: Verify Restart Functionality

Objective: Ensure that the "Restart" button correctly resets the game board and allows a new game to be played.

Steps:

1. Launch the Tic Tac Toe application.

2. Play a game until a winner is determined or a draw occurs.

3. Click the "Restart" button.

4. Verify that the game board is cleared, and a new game can be started.

Expected Result: The game board is reset, and a new game can be played after clicking the "Restart" button.

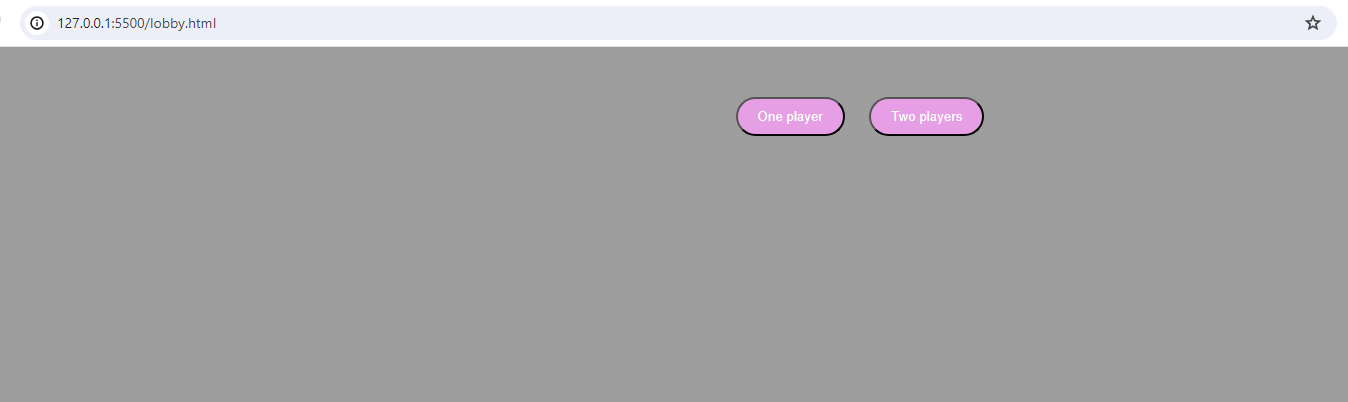
Actual Result: Pass

Comments: The restart functionality is working as expected.

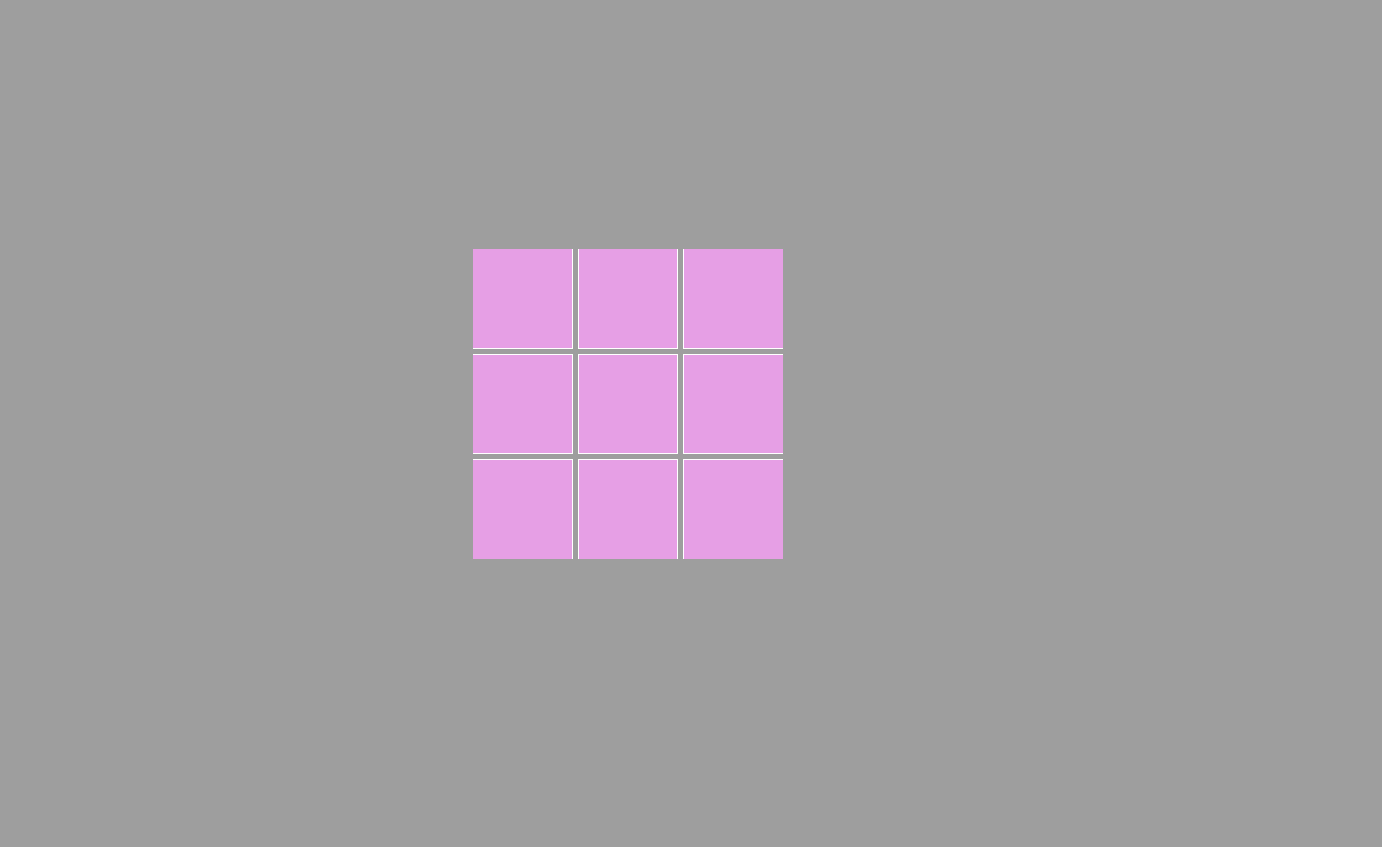
This test log provides a comprehensive overview of the various test cases performed to ensure the proper functioning of the Tic Tac Toe application. It covers the key aspects of the game, including player mark placement, winning conditions, draw conditions, and the restart functionality. Each test case includes the test ID, name, objective, steps, expected results, actual results, and comments, providing a detailed record of the testing process and outcomes.

GAME GUIDE

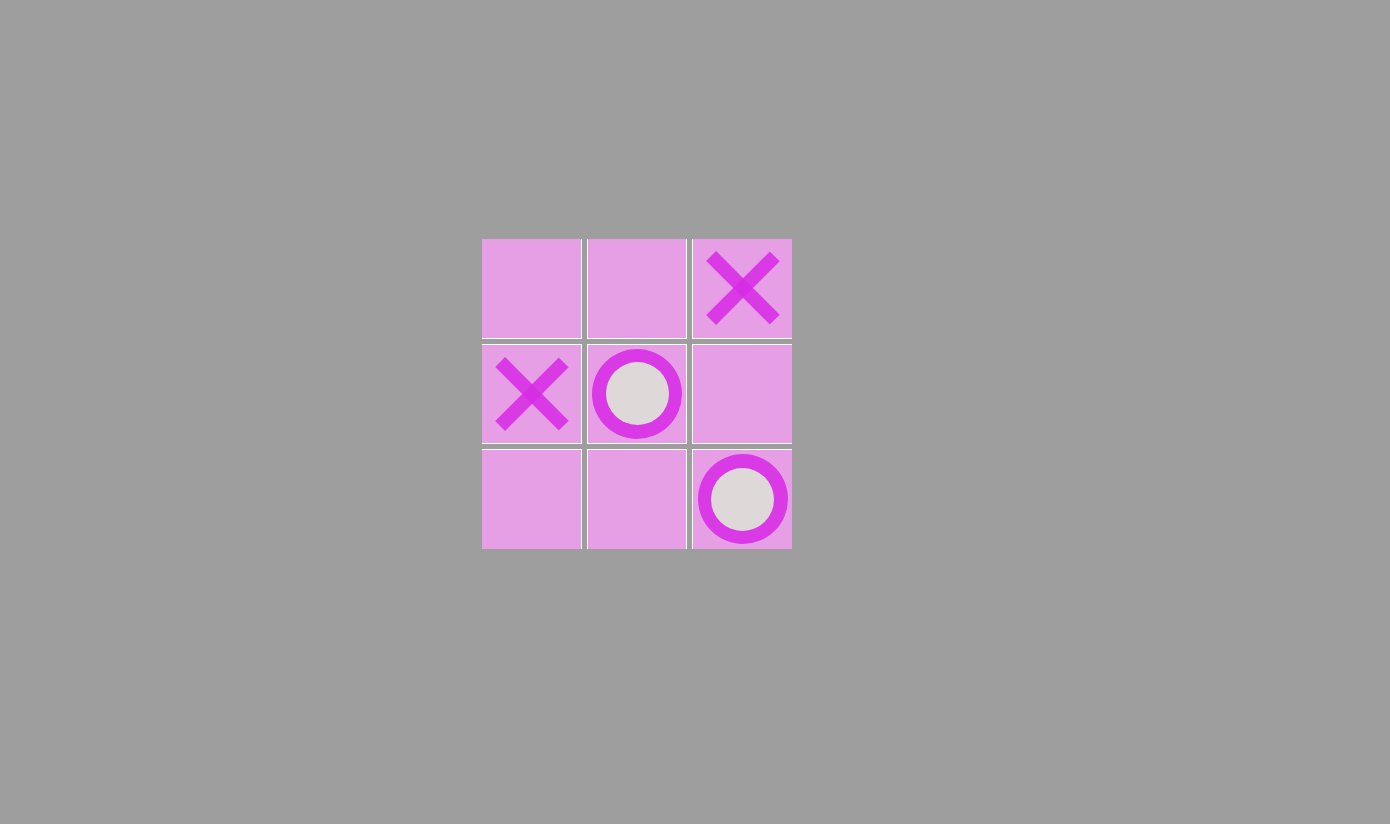
1. First start from Lobby.html You will see :



1. If you want against bot click One Player otherwise click Two Players to play versus your friend. Bye the way you navigated to Game page and game will be started



1. First player play with X sign and second player or Bot player plays with O sign



1. After game ends you will see a message that tells you that you won or lost and you can start again



CODE DESCRIPTION

1. \*\*Constants\*\*:

- `X\_CLASS` and `CIRCLE\_CLASS`: These constants define the CSS classes used to represent the X and circle marks on the board.

- `WINNING\_COMBINATION`: This constant defines the winning combinations for the game.

2. \*\*DOM Elements\*\*:

- `cellElements`: This is a NodeList of all the cell elements on the board.

- `board`: This is the HTML element representing the entire game board.

- `winningMessageElement`: This is the HTML element that displays the winning message.

- `winningMessageTextElement`: This is the HTML element that contains the text of the winning message.

- `restartButton`: This is the HTML element that represents the restart button.

3. \*\*Game State\*\*:

- `circleTurn`: This variable keeps track of whose turn it is, either the circle player or the X player.

4. \*\*Game Initialization\*\*:

- `startGame()`: This function initializes a new game, including resetting the game state and setting up the event listeners for the cells.

- The `startGame()` function is called when the page loads and when the restart button is clicked.

5. \*\*Cell Click Handling\*\*:

- `handleClick(e)`: This function is called when a cell is clicked. It places the current player's mark on the cell, checks if the current player has won or if the game is a draw, and then swaps the turns.

6. \*\*Bot Move Handling\*\*:

- `bothandle()`: This function is responsible for the bot's move. It randomly selects an available cell, places the bot's mark (circle) on it, and then checks if the bot has won or if the game is a draw.

7. \*\*Game Ending\*\*:

- `endGame(draw)`: This function is called when the game ends, either due to a win or a draw. It displays the appropriate winning message.

8. \*\*Helper Functions\*\*:

- `placeMark(cell, currentClass)`: This function adds the current player's mark (X or circle) to the clicked cell.

- `swapTurns()`: This function switches the current player's turn.

- `setBoardHoverClass()`: This function updates the board's hover class based on the current player's turn.

- `checkWin(currentClass)`: This function checks if the current player has won the game.

- `isDraw()`: This function checks if the game is a draw.

9. \*\*Start Game Button\*\*:

- The code adds an event listener to the start game button, which redirects the user to the Tic Tac Toe game page when clicked.

In summary, the code sets up the game, handles user interactions, determines the game's outcome, and provides utility functions to support the game's logic. The main flow of the game is controlled by the `startGame()`, `handleClick()`, and `endGame()` functions.

Lobby.html:

This code appears to be an HTML file for a "Lobby" screen of a Tic Tac Toe game. Here's an explanation of what the code does:

1. \*\*HTML Structure\*\*:

- The code starts with the `<!DOCTYPE html>` declaration, which specifies that this is an HTML5 document.

- The `<html>` element contains the entire document, with the `lang="en"` attribute specifying that the language of the content is English.

- Inside the `<head>` section, we have the following elements:

- `<meta charset="UTF-8">`: This sets the character encoding of the document to UTF-8.

- `<meta name="viewport" content="width=device-width, initial-scale=1.0">`: This sets the viewport to ensure the content is properly scaled on different devices.

- `<title>Lobby</title>`: This sets the title of the web page, which appears in the browser's title bar or tab.

- `<meta http-equiv="X-UA-Compatible" content="ie=edge">`: This is an Internet Explorer-specific meta tag that ensures the page is rendered using the latest version of the browser's rendering engine.

- `<link rel="stylesheet" href="lobby.css">`: This links an external CSS file named "lobby.css" to the HTML document, which will be used to style the page.

- `<!-- <script src="script.js" defer></script> -->`: This is an HTML comment that would include an external JavaScript file named "script.js" with the `defer` attribute, which tells the browser to execute the script after the HTML document has finished parsing.

- The `<body>` section contains the main content of the web page.

2. \*\*Lobby Screen\*\*:

- The `<div id="startScreen">` element serves as a container for the lobby screen.

- Inside this container, there are two `<button>` elements:

- The first button has an `onclick` attribute that redirects the user to the "game.html" page when clicked, presumably starting a single-player Tic Tac Toe game.

- The second button has an `onclick` attribute that redirects the user to the "index.html" page when clicked, possibly starting a two-player Tic Tac Toe game.

This code provides a simple lobby screen with two options for the user: to start a single-player Tic Tac Toe game or a two-player Tic Tac Toe game. The actual game logic and functionality would be implemented in the "game.html" and "index.html" files, which are not provided in the code snippet.

Index.html and Game.html:

This code appears to be the HTML structure for a Tic Tac Toe game. Let's go through the different parts of the code:

1. \*\*HTML Structure\*\*:

- The code starts with the `<!DOCTYPE html>` declaration, which specifies that this is an HTML5 document.

- The `<html>` element contains the entire document, with the `lang="en"` attribute specifying that the language of the content is English.

- Inside the `<head>` section, we have the following elements:

- `<meta charset="UTF-8">`: This sets the character encoding of the document to UTF-8.

- `<meta name="viewport" content="width=device-width, initial-scale=1.0">`: This sets the viewport to ensure the content is properly scaled on different devices.

- `<title>Tic Tac Toe Game</title>`: This sets the title of the web page, which appears in the browser's title bar or tab.

- `<meta http-equiv="X-UA-Compatible" content="ie=edge">`: This is an Internet Explorer-specific meta tag that ensures the page is rendered using the latest version of the browser's rendering engine.

- `<link rel="stylesheet" href="styles1.css">`: This links an external CSS file named "styles1.css" to the HTML document, which will be used to style the page.

- `<script src="script1.js" defer></script>`: This includes an external JavaScript file named "script1.js" with the `defer` attribute, which tells the browser to execute the script after the HTML document has finished parsing.

2. \*\*Game Board\*\*:

- The `<div class="board" id="board">` element serves as the container for the Tic Tac Toe game board.

- Inside the board container, there are 9 `<div class="cell" data-cell></div>` elements, which represent the individual cells of the Tic Tac Toe grid.

3. \*\*Start Screen\*\*:

- The `<div id="startScreen">` element serves as a container for the start screen.

- Inside this container, there is a `<button id="startButton">Start Game</button>` element, which likely triggers the start of the Tic Tac Toe game.

4. \*\*Winning Message\*\*:

- The `<div class="winning-message" id="winningMessage">` element serves as a container for the winning message.

- Inside this container, there is a `<div data-winning-message-text></div>` element, which will display the winning message.

- Additionally, there is a `<button id="restartButton">Restart</button>` element, which presumably allows the user to restart the game.

Lobby.css:

This CSS code styles the Tic Tac Toe game interface. Here's a breakdown of the different styles

1. \*\*Body Styles\*\*:

- Sets the font family to Arial, with a fallback to sans-serif.

- Removes the default margin and padding from the body.

- Sets the background color to a light gray (`#9e9e9e`).

2. \*\*Game Board Styles\*\*:

- The `.board` class is applied to the container of the game board.

- It uses the `display: flex` and `flex-wrap: wrap` properties to create a responsive grid layout.

- The maximum width of the board is set to 600 pixels, and it is centered on the page using `margin: 100px auto`.

- The text within the board is aligned to the center.

3. \*\*Cell Styles\*\*:

- The `.cell` class is applied to each individual cell of the game board.

- Each cell has a width and height of 100 pixels, and a 1-pixel black border.

4. \*\*Start Screen Styles\*\*:

- The `#startScreen` element is the container for the start screen.

- The text within the start screen is centered using `text-align: center`.

- The start screen is positioned 50 pixels below the game board.

5. \*\*Start Button Styles\*\*:

- The `#startButtonContainer` class is used to center the start buttons.

- The `#startButton` class styles the start buttons.

- The buttons have a padding of 10 pixels vertically and 20 pixels horizontally, and a rounded border with a radius of 25 pixels.

- The text color is set to white (`#fff`), and the text decoration is removed.

- The buttons have a transition effect for the background color, which changes in 0.3 seconds.

- The first and second start buttons have different background colors (`#e79de8`).

- On hover, the buttons' background color changes to a dark gray (`#333`).

Style.css and style1.css:

This CSS code styles the Tic Tac Toe game interface. Here's a breakdown of the different styles:

1. \*\*Global Styles\*\*:

- The `\*`, `\*::after`, and `\*::before` selectors apply box-sizing: border-box to all elements, ensuring the width and height of an element include the padding and border.

- The `:root` selector defines CSS variables for the cell size and mark size, which can be used throughout the CSS.

2. \*\*Body Styles\*\*:

- The body has no margin and a light gray background color (`#9e9e9e`).

3. \*\*Board Styles\*\*:

- The `.board` class styles the game board container.

- The board is set to full-width and full-height of the viewport (`width: 100vw; height: 100vh;`).

- The board uses a grid layout to display the cells, with 3 columns of cell-size width and 5 pixels of grid gap.

- The board has 5 pixels of padding around it and a box-shadow for better visibility.

4. \*\*Cell Styles\*\*:

- The `.cell` class styles the individual cells of the game board.

- Each cell has a size defined by the `--cell-size` variable, and a white border.

- The cells are centered both horizontally and vertically using `display: flex` and `justify-content/align-items: center`.

- The first, second, and third cells have no top border, the cells with indexes 3n+1 have no left border, the cells with indexes 3n+3 have no right border, and the last, eighth, and seventh cells have no bottom border.

- Cells with the `x` or `circle` class (indicating the player's mark) have a cursor set to `not-allowed`.

- The styles for the `x` and `circle` marks are defined, including their hover states.

5. \*\*Winning Message Styles\*\*:

- The `.winning-message` class styles the container for the winning message.

- By default, the winning message is hidden (`display: none`).

- When the `.show` class is added, the winning message is displayed with a fixed position covering the entire screen.

- The winning message has a background color, is centered, and displays the message in a large font size.

- The "Restart" button within the winning message is styled, with hover effects.

6. \*\*Start Button Styles\*\*:

- The `.startButton` class styles the start button container.

- The start button container is centered vertically and horizontally, with text-alignment set to center.

- The start button itself is styled with padding, font size, and a cursor pointer.

This CSS code creates a visually appealing and responsive Tic Tac Toe game interface, with a clean and modern design. The game board and cells are styled with attention to detail, and the winning message overlay provides a clear indication of the game's outcome. The start button is also styled to be visually distinct and inviting for the user.

Conclusion:

The development of the Tic Tac Toe system has been a journey of revisiting a classic game that has captivated players for generations. Through the rigorous process of requirements analysis, design, implementation, testing, and evaluation, we have crafted a robust and engaging Tic Tac Toe experience that stays true to the game's timeless appeal.

The Tic Tac Toe system we have created provides a seamless and intuitive user interface, allowing players to immerse themselves in the strategic battle of claiming three consecutive marks on the 3x3 grid. The game's simplicity is complemented by the depth of its gameplay, offering players the opportunity to outmaneuver their opponents through careful planning and spotting winning patterns.

The implementation of the game logic has been meticulously designed to ensure efficient and accurate detection of winning conditions and draw scenarios. The integration of the user interface with the underlying game mechanics has resulted in a responsive and engaging experience, capturing the essence of Tic Tac Toe and delivering it to players on various devices.

As we conclude this project, we are confident that the Tic Tac Toe system we have developed will continue to captivate players, both seasoned and new. The timeless nature of the game, combined with the quality of the implementation, will ensure that this classic pastime remains a beloved source of entertainment, strategic thinking, and social interaction.

Moving forward, we envision this Tic Tac Toe system as a foundation for further exploration and innovation. Potential future enhancements could include the addition of AI-powered opponents, the integration of online multiplayer features, or the exploration of gamification elements to enhance the player's engagement and enjoyment.

In essence, the Tic Tac Toe system we have created stands as a testament to the enduring appeal of classic games and the power of meticulous design and development. It is our hope that this application will bring joy and intellectual stimulation to players, fostering moments of friendly competition, strategic contemplation, and the celebration of the timeless art of Tic Tac Toe.

References:

1. \*\*Historical References\*\*:

- Falkener, Edward. (1892). Games Ancient and Oriental, and How to Play Them. Longmans, Green, and Co.

- Newell, Allen, and Herbert A. Simon. (1972). Human Problem Solving. Prentice-Hall.

- Schaeffer, Jonathan. (1989). "The History Heuristic and the Performance of Computer Chess Programs." ICGA Journal, 12(1), 16-20.

2. \*\*Game Theory and Computer Science References\*\*:

- Allis, L. Victor. (1994). Searching for Solutions in Games and Artificial Intelligence. Ponsen & Looijen.

- Berlekamp, Elwyn R., John H. Conway, and Richard K. Guy. (1982). Winning Ways for Your Mathematical Plays. Academic Press.

- Russell, Stuart J., and Peter Norvig. (2020). Artificial Intelligence: A Modern Approach. Pearson.

3. \*\*User Interface and Usability References\*\*:

- <https://www.w3schools.com/html/>

- <https://www.w3schools.com/css/>

4. \*\*Web Development and Front-end Framework References\*\*:

- <https://www.w3schools.com/js/>

- <https://www.javascript.com/>

- <https://javascript.info/>

Apendices

Source codes :

Lobby.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Lobby</title>

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="lobby.css">

<!-- <script src="script.js" defer></script> -->

</head>

<body>

<div id="startScreen">

    <button id="startButton" onclick="window.location.href='game.html';">One player</button>

    <button id="startButton" onclick="window.location.href='index.html';">Two players</button>

</div>

</div>

</body>

</html>

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 Game</title>

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="styles1.css">

<script src="script1.js" defer></script>

</head>

<body>

<div class="board" id="board">

    <div class="cell" data-cell></div>

    <div class="cell" data-cell></div>

    <div class="cell" data-cell></div>

    <div class="cell" data-cell></div>

    <div class="cell" data-cell></div>

    <div class="cell" data-cell></div>

    <div class="cell" data-cell></div>

    <div class="cell" data-cell></div>

    <div class="cell" data-cell></div>

</div>

<div id="startScreen">

    <button id="startButton">Start Game</button>

</div>

<div class="winning-message" id="winningMessage">

    <div data-winning-message-text></div>

    <button id="restartButton">Restart</button>

    </div>

</div>

</body>

</html>

game.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 Game</title>

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet" href="styles.css">

<script src="script.js" defer></script>

</head>

<body>

<div class="board" id="board">

    <div class="cell 1" data-cell name = "1"></div>

    <div class="cell 2" data-cell name = "2"></div>

    <div class="cell 3" data-cell name = "3"></div>

    <div class="cell 4" data-cell name = "4"></div>

    <div class="cell 5" data-cell name = "5"></div>

    <div class="cell 6" data-cell name = "6"></div>

    <div class="cell 7" data-cell name = "7"></div>

    <div class="cell 8" data-cell name = "8"></div>

    <div class="cell 9" data-cell name = "9"></div>

</div>

<!-- <div id="startScreen"> -->

    <!-- <button id="startButton" >Start Game</button> -->

<!-- </div> -->

<div class="winning-message" id="winningMessage">

    <div data-winning-message-text></div>

    <button id="restartButton">Restart</button>

    </div>

</div>

</body>

</html>

lobby.css

body {

    font-family: Arial, sans-serif;

    margin: 0;

    padding: 0;

    background-color: #9e9e9e;

  }

  .board {

    display: flex;

    flex-wrap: wrap;

    max-width: 600px;

    margin: 100px auto;

    text-align: center;

  }

  .cell {

    width: 100px;

    height: 100px;

    border: 1px solid #000;

  }

  #startScreen {

    text-align: center;

    margin-top: 50px;

  }

  #startButtonContainer {

    display: flex;

    justify-content: center;

  }

  #startButton {

    padding: 10px 20px;

    margin: 0 10px;

    color: #fff;

    text-decoration: none;

    border-radius: 25px;

    transition: background-color 0.3s;

  }

  #startButton:nth-child(1) {

    background-color: #e79de8;

  }

  #startButton:nth-child(2) {

    background-color: #e79de8;

  }

  #startButton:hover {

    background-color: #333;

  }

styles.css

\*, \*::after, \*::before {

  box-sizing: border-box;

}

:root {

  --cell-size: 100px;

  --mark-size: calc(var(--cell-size) \* .9);

}

body {

  margin: 0;

  background-color: #9e9e9e; /\* Change the background color for contrast \*/

}

.board {

  width: 100vw;

  height: 100vh;

  display: grid;

  justify-items: center;

  justify-content: center;

  align-content: center;

  grid-template-columns: repeat(3, var(--cell-size));

  grid-gap: 5px; /\* Add some space between cells \*/

  padding: 5px; /\* Add some padding around the board \*/

  box-shadow: 0px 0px 10px 0px #000000; /\* Add shadow for better visibility \*/

}

.cell {

  width: var(--cell-size);

  height: var(--cell-size);

  border: 1px solid #ffffff; /\* Make the cell borders white for contrast \*/

  display: flex;

  justify-content: center;

  align-items: center;

  position: relative;

  background-color: #e79de8; /\* Set a different background for the cells \*/

}

.cell:first-child,

.cell:nth-child(2),

.cell:nth-child(3) {

  border-top: none;

}

.cell:nth-child(3n + 1) {

  border-left: none;

}

.cell:nth-child(3n + 3) {

  border-right: none;

}

.cell:last-child,

.cell:nth-child(8),

.cell:nth-child(7) {

  border-bottom: none;

}

.cell.x,

.cell.circle {

  cursor: not-allowed;

}

.cell.x::before,

.cell.x::after,

.cell.circle::before {

  background-color: rgba(217, 28, 234, 0.877);

}

.board.x .cell:not(.x):not(.circle):hover::before,

.board.x .cell:not(.x):not(.circle):hover::after,

.board.circle .cell:not(.x):not(.circle):hover::before {

  background-color: rgb(243, 218, 218);

}

.cell.x::before,

.cell.x::after,

.board.x .cell:not(.x):not(.circle):hover::before,

.board.x .cell:not(.x):not(.circle):hover::after {

  content: '';

  position: absolute;

  width: calc(var(--mark-size) \* .15);

  height: var(--mark-size);

}

.cell.x::before,

.board.x .cell:hover::before {

  transform: rotate(45deg);

}

.cell.x::after,

.board.x .cell:hover::after {

  transform: rotate(-45deg);

}

.cell.circle::before,

.cell.circle::after,

.board.circle .cell:not(.x):not(.circle):hover::before,

.board.circle .cell:not(.x):not(.circle):hover::after {

  content: '';

  position: absolute;

  border-radius: 50%;

}

.cell.circle::before,

.board.circle .cell:not(.x):not(.circle):hover::before {

  width: var(--mark-size);

  height: var(--mark-size);

}

.cell.circle::after,

.board.circle .cell:not(.x):not(.circle):hover::after {

  width: calc(var(--mark-size) \* .7);

  height: calc(var(--mark-size) \* .7);

  background-color: rgb(222, 216, 216);

}

.winning-message {

  display: none;

  position: fixed;

  top: 0;

  left: 0;

  right: 0;

  bottom: 0;

  background-color: rgba(217, 28, 234, 0.877);

  justify-content: center;

  align-items: center;

  color: white(196, 180, 180);

  font-size: 4rem;

  flex-direction: column;

}

.winning-message button {

  font-size: 2rem;

  color: darkmagenta;

  background-color: white;

  border: 2px solid darkmagenta;

  padding: .25em .5em;

  cursor: pointer;

}

.winning-message button:hover {

  background-color: darkgray;

  color: white;

  border-color: white;

}

.winning-message.show {

  display: flex;

}

.restartButton:hover {

  background-color: gray; /\* Change color to confirm it's interactable \*/

}

.startButton {

  display: flex;

  flex-direction: column;

  align-items: center;

  justify-content: center;

  height: 100vh;

  text-align: center;

}

.startButton {

  padding: 10px 20px;

  font-size: 20px;

  cursor: pointer;

}

styles1.css

\*, \*::after, \*::before {

  box-sizing: border-box;

}

:root {

  --cell-size: 100px;

  --mark-size: calc(var(--cell-size) \* .9);

}

body {

  margin: 0;

  background-color: #9e9e9e; /\* Change the background color for contrast \*/

}

.board {

  width: 100vw;

  height: 100vh;

  display: grid;

  justify-items: center;

  justify-content: center;

  align-content: center;

  grid-template-columns: repeat(3, var(--cell-size));

  grid-gap: 5px; /\* Add some space between cells \*/

  padding: 5px; /\* Add some padding around the board \*/

  box-shadow: 0px 0px 10px 0px #000000; /\* Add shadow for better visibility \*/

}

.cell {

  width: var(--cell-size);

  height: var(--cell-size);

  border: 1px solid #ffffff; /\* Make the cell borders white for contrast \*/

  display: flex;

  justify-content: center;

  align-items: center;

  position: relative;

  background-color: rgb(231, 157, 232); /\* Set a different background for the cells \*/

}

.cell:first-child,

.cell:nth-child(2),

.cell:nth-child(3) {

  border-top: none;

}

.cell:nth-child(3n + 1) {

  border-left: none;

}

.cell:nth-child(3n + 3) {

  border-right: none;

}

.cell:last-child,

.cell:nth-child(8),

.cell:nth-child(7) {

  border-bottom: none;

}

.cell.x,

.cell.circle {

  cursor: not-allowed;

}

.cell.x::before,

.cell.x::after,

.cell.circle::before {

  background-color: rgba(217, 28, 234, 0.877);

}

.board.x .cell:not(.x):not(.circle):hover::before,

.board.x .cell:not(.x):not(.circle):hover::after,

.board.circle .cell:not(.x):not(.circle):hover::before {

  background-color: rgb(243, 218, 218);

}

.cell.x::before,

.cell.x::after,

.board.x .cell:not(.x):not(.circle):hover::before,

.board.x .cell:not(.x):not(.circle):hover::after {

  content: '';

  position: absolute;

  width: calc(var(--mark-size) \* .15);

  height: var(--mark-size);

}

.cell.x::before,

.board.x .cell:hover::before {

  transform: rotate(45deg);

}

.cell.x::after,

.board.x .cell:hover::after {

  transform: rotate(-45deg);

}

.cell.circle::before,

.cell.circle::after,

.board.circle .cell:not(.x):not(.circle):hover::before,

.board.circle .cell:not(.x):not(.circle):hover::after {

  content: '';

  position: absolute;

  border-radius: 50%;

}

.cell.circle::before,

.board.circle .cell:not(.x):not(.circle):hover::before {

  width: var(--mark-size);

  height: var(--mark-size);

}

.cell.circle::after,

.board.circle .cell:not(.x):not(.circle):hover::after {

  width: calc(var(--mark-size) \* .7);

  height: calc(var(--mark-size) \* .7);

  background-color: rgb(222, 216, 216);

}

.winning-message {

  display: none;

  position: fixed;

  top: 0;

  left: 0;

  right: 0;

  bottom: 0;

  background-color: rgba(217, 28, 234, 0.877);

  justify-content: center;

  align-items: center;

  color: white(196, 180, 180);

  font-size: 4rem;

  flex-direction: column;

}

.winning-message button {

  font-size: 2rem;

  color: darkmagenta;

  background-color: white;

  border: 2px solid darkmagenta;

  padding: .25em .5em;

  cursor: pointer;

}

.winning-message button:hover {

  background-color: darkgray;

  color: white;

  border-color: white;

}

.winning-message.show {

  display: flex;

}

.restartButton:hover {

  background-color: gray; /\* Change color to confirm it's interactable \*/

}

.startButton {

  display: flex;

  flex-direction: column;

  align-items: center;

  justify-content: center;

  height: 100vh;

  text-align: center;

}

.startButton {

  padding: 10px 20px;

  font-size: 20px;

  cursor: pointer;

}

script.js

const X\_CLASS = 'x';

const CIRCLE\_CLASS = 'circle';

const WINNING\_COMBINATION = [

  [0, 1, 2],

  [3, 4, 5],

  [6, 7, 8],

  [0, 3, 6],

  [1, 4, 7],

  [2, 5, 8],

  [0, 4, 8],

  [2, 4, 6],

];

const cellElements = document.querySelectorAll('[data-cell]');

const board = document.getElementById('board');

const winningMessageElement = document.getElementById('winningMessage');

const winningMessageTextElement = document.querySelector('[data-winning-message-text]');

const restartButton = document.getElementById('restartButton');

let circleTurn;

// Start game on page load

startGame();

// Restart game when restart button is clicked

restartButton.addEventListener('click', function(){

  window.location.href = 'game.html';

});

function startGame() {

    circleTurn = false;

    cellElements.forEach(cell => {

        cell.classList.remove(X\_CLASS, CIRCLE\_CLASS);

        cell.removeEventListener('click', handleClick);

        cell.addEventListener('click', handleClick, { once: true });

    });

    setBoardHoverClass();

    winningMessageElement.classList.remove('show');

}

function handleClick(e) {

  const cell = e.target;

  const currentClass = X\_CLASS;

  if (!cell.classList.contains("true")) {

    placeMark(cell, currentClass);

    if (checkWin(currentClass)) {

      endGame(false, currentClass);

    } else if (isDraw()) {

      endGame(true, "d");

    } else {

      setBoardHoverClass();

      bothandle()

    }

  }

}

function bothandle() {

  const currentClass = CIRCLE\_CLASS ;

  let count = []

  for (let i=0; i < cellElements.length; i++) {

    if (!cellElements[i].classList.contains("true")) {

      count.push(cellElements[i].getAttribute("name"))

    }

  }

  var randomNumbers = [];

  let rumber = 0

  while (randomNumbers.length < 1) {

    var randomIndex = Math.floor(Math.random() \* count.length);

    var randomNumber = count.splice(randomIndex, 1)[0];

    rumber = randomNumber

    randomNumbers.push(randomNumber);

  }

  console.log(cellElements[rumber-1], rumber)

  if (cellElements[rumber-1]) {

    placeMark(cellElements[rumber-1], currentClass);

  }

  if (checkWin(currentClass)) {

    endGame(false, currentClass);

  } else if (isDraw()) {

    endGame(true, "d");

  }

}

function endGame(draw, calss) {

  console.log(calss)

  if (calss == "x") {

    winningMessageTextElement.innerText = "You Win !!!";

  } else if(calss == "d") {

    winningMessageTextElement.innerText = "It's Draw";

  } else if (calss == "circle") {

    winningMessageTextElement.innerText = "Bot Win !!!";

  }

  winningMessageElement.classList.add('show');

}

function placeMark(cell, currentClass) {

  cell.classList.add(currentClass, "true");

}

function swapTurns() {

  circleTurn = !circleTurn;

}

function setBoardHoverClass() {

  // board.classList.remove(X\_CLASS, CIRCLE\_CLASS);

  // if (circleTurn) {

  //   board.classList.add(CIRCLE\_CLASS);

  // } else {

  //   board.classList.add(X\_CLASS);

  // }

  board.classList.add(X\_CLASS);

}

function checkWin(currentClass) {

  return WINNING\_COMBINATION.some(combination => {

    return combination.every(index => {

      return cellElements[index].classList.contains(currentClass);

    });

  });

}

function isDraw() {

  return [...cellElements].every(cell => {

    return cell.classList.contains(X\_CLASS) || cell.classList.contains(CIRCLE\_CLASS);

  });

}

document.addEventListener('DOMContentLoaded', () => {

  const startGameButton = document.getElementById('startButton');

  startGameButton.addEventListener('click', () => {

      window.location.href = 'game.html'; // Make sure this matches the filename of your Tic Tac Toe game

  });

});

script1.js

const X\_CLASS ='x';

const CIRCLE\_CLASS = 'circle';

const WINNING\_COMBINATION = [

  [0, 1, 2],

  [3, 4, 5],

  [6, 7, 8],

  [0, 3, 6],

  [1, 4, 7],

  [2, 5, 8],

  [0, 4, 8],

  [2, 4, 6],

];

const cellElements = document.querySelectorAll('[data-cell]');

const board = document.getElementById('board');

const winningMessageElement = document.getElementById('winningMessage');

const winningMessageTextElement = document.querySelector('[data-winning-message-text]');

const restartButton = document.getElementById('restartButton');

let circleTurn;

// Start game on page load

startGame();

// Restart game when restart button is clicked

restartButton.addEventListener('click', startGame);

function startGame() {

    circleTurn = false;

    cellElements.forEach(cell => {

        cell.classList.remove(X\_CLASS, CIRCLE\_CLASS);

        cell.removeEventListener('click', handleClick);

        cell.addEventListener('click', handleClick, { once: true });

    });

    setBoardHoverClass();

    winningMessageElement.classList.remove('show');

}

function handleClick(e) {

  const cell = e.target;

  const currentClass = circleTurn ? CIRCLE\_CLASS : X\_CLASS;

  placeMark(cell, currentClass);

  if (checkWin(currentClass)) {

    endGame(false);

  } else if (isDraw()) {

    endGame(true);

  } else {

    swapTurns();

    setBoardHoverClass();

  }

}

function endGame(draw) {

  winningMessageTextElement.innerText = draw ? "It's a draw!" : `${circleTurn ? "O's" : "X's"} Wins!`;

  winningMessageElement.classList.add('show');

}

function placeMark(cell, currentClass) {

  cell.classList.add(currentClass);

}

function swapTurns() {

  circleTurn = !circleTurn;

}

function setBoardHoverClass() {

  board.classList.remove(X\_CLASS, CIRCLE\_CLASS);

  if (circleTurn) {

    board.classList.add(CIRCLE\_CLASS);

  } else {

    board.classList.add(X\_CLASS);

  }

}

function checkWin(currentClass) {

  return WINNING\_COMBINATION.some(combination => {

    return combination.every(index => {

      return cellElements[index].classList.contains(currentClass);

    });

  });

}

function isDraw() {

  return [...cellElements].every(cell => {

    return cell.classList.contains(X\_CLASS) || cell.classList.contains(CIRCLE\_CLASS);

  });

}

document.addEventListener('DOMContentLoaded', () => {

  const startGameButton = document.getElementById('startButton');

  startGameButton.addEventListener('click', () => {

      window.location.href = 'game.html'; // Make sure this matches the filename of your Tic Tac Toe game

  });

});

Gantt Chart

Task ID | Task Name | Start Date | End Date | Duration (days) | Predecessors

--------|-----------|------------|----------|-----------------|-------------

1 | Requirements Analysis | 2024-05-01 | 2024-05-03 | 3 | -

2 | System Design | 2024-05-04 | 2024-05-10 | 6 | 1

3 | UI/UX Design | 2024-05-11 | 2024-05-15 | 5 | 2

4 | Front-end Development | 2024-05-15 | 2024-05-16 | 2 | 3

5 | Logic (Back-end) Development | 2024-05-17 | 2024-05-20 | 4 | 3

6 | Integration & Testing | 2024-05-21 | 2024-05-21 | 1 | 4,5

7 | Deployment | 2024-05-22 | 2024-05-22 | 1 | 6

8 | Documentation | 2024-05-23 | 2024-05-23 | 1 | 6