EXPERIMENT

AIM

To build a responsive and interactive user interface using Tailwind CSS.

THEORY

Tailwind CSS is a modern utility-first CSS framework designed to help developers build fast, responsive, and visually appealing websites without writing custom CSS from scratch. Instead of creating long CSS files with separate classes, Tailwind offers thousands of pre-defined **utility classes** that can be directly applied to HTML elements. These classes handle properties like spacing, colors, typography, borders, shadows, and more.

One of its biggest advantages is **mobile-first responsiveness.** Tailwind uses a responsive design approach where layouts are automatically optimized for smaller screens first, and can then be adjusted for larger devices using built-in responsive prefixes (e.g., sm:, md:, lg:).

Tailwind also supports **hover effects, animations, and transitions** through simple class names. For example, applying hover:bg-blue-500 instantly changes a button's background on hover, and transition duration-300 makes the effect smooth.

Additionally, Tailwind can be easily customized through its configuration file (tailwind.config.js), allowing developers to define their own color palette, font sizes, breakpoints, and other design settings, ensuring brand consistency.

Because Tailwind eliminates the need to switch between HTML and CSS files, it speeds up the development process and helps maintain clean, maintainable, and scalable code. This makes it a preferred choice for building both small prototypes and large-scale production applications.

What we have done :

Tic-Tac-Toe is a simple and popular two-player game played on a 3×3 grid.

The objective is to place three marks (X or O) in a horizontal, vertical, or diagonal row before the opponent.

This project implements **a responsive and interactive Tic-Tac-Toe game** using **HTML, CSS (Tailwind CSS), and JavaScript**.

The UI is designed with **Tailwind CSS** to make it visually appealing and fully responsive across devices.
The game supports multiple modes:

- **Player vs Player**
- **Player vs AI**
- **AI vs AI**

Game logic is handled using JavaScript, including win detection, draw detection, and automatic reset.
Additional features include:

- Animated backgrounds
- Winner celebration effects
- Smooth navigation between screens

This experiment demonstrates the integration of **front-end design** with **interactive JavaScript functionality** to create a complete browser-based game.

Technology Used

Technology	Purpose
HTML5	Structure of the game pages
CSS3/Tailwind	Styling and responsive design
JavaScript	Game logic and AI implementation
LocalStorage	Store game settings and winner data

Tools Used

Tool	Description
VS Code	Code editor for development
Chrome/Browser	Platform to run and test the game
Tailwind CDN	For styling components with utility classes
Canvas Confetti	Used in celebration for visual effects

Code Structure

1. **welcome.html** – Collect player names and game mode.
2. **index.html** – Game logic, grid layout, AI implementation.
3. **celebration.html** – Display winner and fun celebratory UI.

30% EXTRA FEATURE THEORY

Apart from building a responsive UI, this project integrates Tailwind CSS animations and transitions to enhance interactivity. Using classes such as animate-bounce, transition, ease-in-out, and duration-500, interactive winner popups and celebration effects were implemented. This not only improves the user experience but also demonstrates advanced Tailwind concepts for creating engaging, dynamic components.

SOURCE CODE

welcome.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Welcome to Tic-Tac-Toe</title>
    <script src="https://cdn.tailwindcss.com"></script>
    <style>
        body {
            margin: 0;
```

```html
<!-- Floating X/O Symbols -->
<div class="floating-symbol symbol-x">X</div>
<div class="floating-symbol symbol-o">O</div>
<div class="floating-symbol symbol-x2">X</div>
<div class="floating-symbol symbol-o2">O</div>

<div class="welcome-box">
    <h1 class="welcome-title">✨ Welcome to the Ultimate Tic-Tac-Toe Battle! ✨</h1>
    <div class="input-group">
        <label class="input-label" for="playerX">Player X Name:</label>
        <input id="playerX" class="input-field" type="text" placeholder="Enter Player X Name" />
    </div>
    <div class="input-group">
        <label class="input-label" for="playerO">Player O Name:</label>
        <input id="playerO" class="input-field" type="text" placeholder="Enter Player O Name" />
    </div>

    <div class="mt-4" id="modeButtons">
        <button class="game-mode-btn" onclick="setGameMode(this, 'pvp')">Player vs Player</button>
        <button class="game-mode-btn" onclick="setGameMode(this, 'pve')">Player vs AI</button>
        <button class="game-mode-btn" onclick="setGameMode(this, 'eve')">AI vs AI</button>
    </div>

    <button class="game-mode-btn start-btn" onclick="startGame()">Start Game</button>
</div>
```

```javascript
function startGame() {
    const playerX = document.getElementById('playerX').value || 'Player X';
    const playerO = document.getElementById('playerO').value || 'Player O';

    localStorage.setItem('playerXName', playerX);
    localStorage.setItem('playerOName', playerO);
    localStorage.setItem('gameMode', selectedMode);

    window.location.href = 'index.html';
}
```

index.html

```
function updateStatus() {
  statusText.textContent = `${currentPlayer === 'X' ? playerXName : playerOName}'s Turn`;
}

function handleClick(e) {
  const index = e.target.getAttribute('data-index');
  if (gameBoard[index] !== '') return;
  if (mode === 'pve' && (currentPlayer === aiX || currentPlayer === aiO)) return;
  if (mode === 'eve') return;
  makeMove(index, currentPlayer);
}

function makeMove(index, symbol) {
  gameBoard[index] = symbol;
  cells[index].textContent = symbol;
  cells[index].classList.add(symbol === 'X' ? 'text-blue-600' : 'text-pink-500');

  if (checkWinner(symbol)) {
    endGame(symbol);
    return;
  }
  if (!gameBoard.includes('')) {
    endGame('Draw');
    return;
  }

  currentPlayer = symbol === 'X' ? 'O' : 'X';
  updateStatus();

  if ((mode === 'pve' && (currentPlayer === aiX || currentPlayer === aiO)) || mode === 'eve') {
    setTimeout(aiMoveMedium, 500);
  }
```

```
function findWinningMove(sym) {
  const winPatterns = [
    [0, 1, 2], [3, 4, 5], [6, 7, 8],
    [0, 3, 6], [1, 4, 7], [2, 5, 8],
    [0, 4, 8], [2, 4, 6]
  ];
  for (let pattern of winPatterns) {
    const [a, b, c] = pattern;
    const vals = [gameBoard[a], gameBoard[b], gameBoard[c]];
    if (vals.filter(v => v === sym).length === 2 && vals.includes('')) {
      return pattern[vals.indexOf('')];
    }
  }
  return null;
}

function checkWinner(symbol) {
  const winPatterns = [
    [0, 1, 2], [3, 4, 5], [6, 7, 8],
    [0, 3, 6], [1, 4, 7], [2, 5, 8],
    [0, 4, 8], [2, 4, 6]
  ];
  return winPatterns.some(pattern => pattern.every(i => gameBoard[i] === symbol));
}

function endGame(winner) {
  localStorage.setItem('winner', winner);
  localStorage.setItem('winnerName', winner === 'Draw' ? 'Draw' : (winner === 'X' ? playerXName : playerOName));
  window.location.href = 'celebration.html';
}
```

celebration.html

```
if (winner === 'Draw') {
  message = "It's a Draw!";
  quote = drawQuotes[Math.floor(Math.random() * drawQuotes.length)];
} else {
  if (mode === 'eve') {
    message = `AI ${winner === 'X' ? '1' : '2'} Wins! 🤖`;
    quote = aiQuotes[Math.floor(Math.random() * aiQuotes.length)];
  } else if (mode === 'pve' && winner === 'O') {
    message = "AI Wins! 🤖";
    quote = aiQuotes[Math.floor(Math.random() * aiQuotes.length)];
  } else {
    const playerName = winner === 'X' ? playerX : playerO;
    message = `${playerName} Wins! 🎉`;
    quote = playerQuotes[Math.floor(Math.random() * playerQuotes.length)];
  }
}
```

OUTPUT

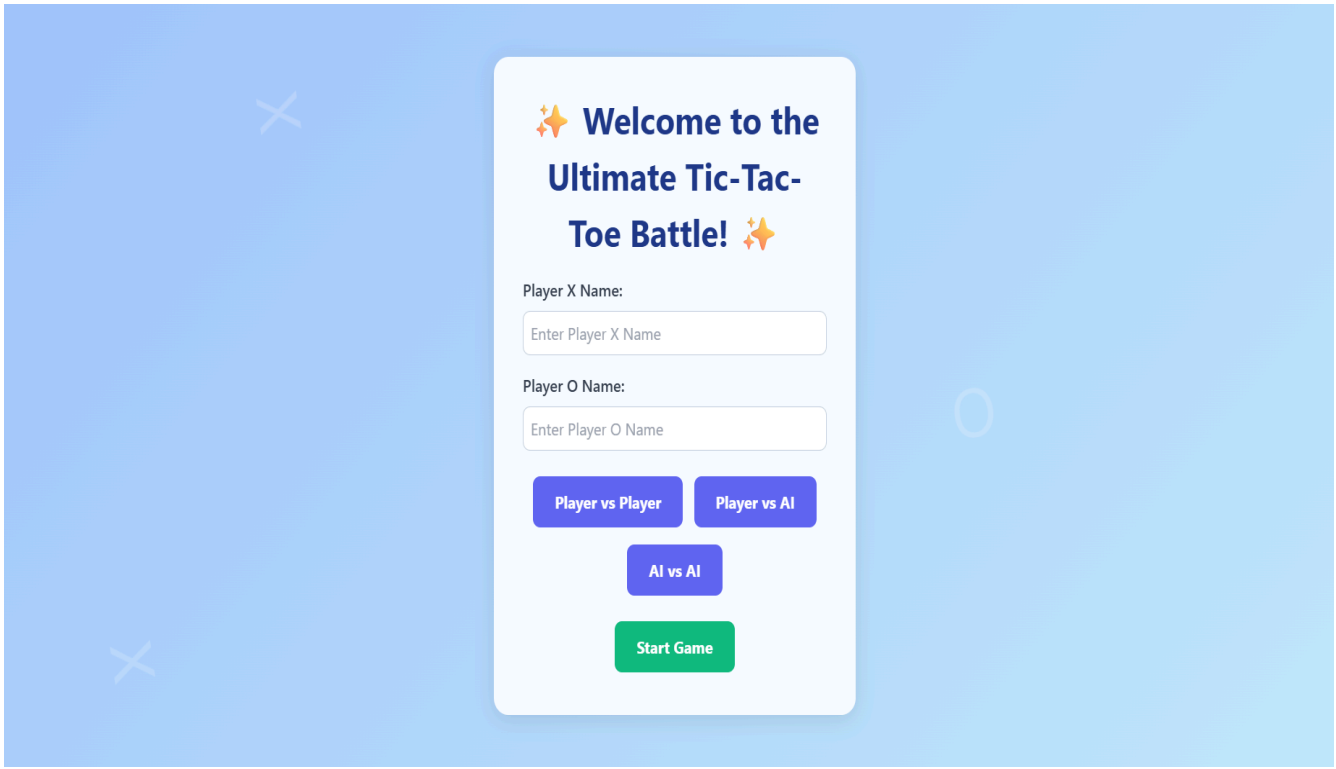


Figure 1: Welcome Page – Player name input and mode selection.

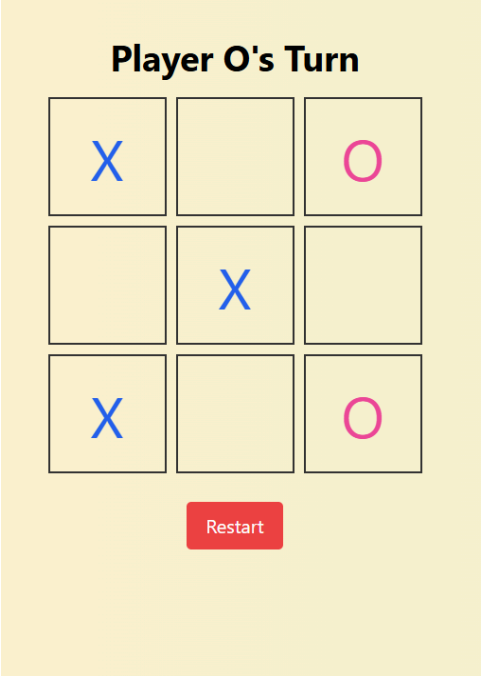


Figure 2: Gameplay Screen

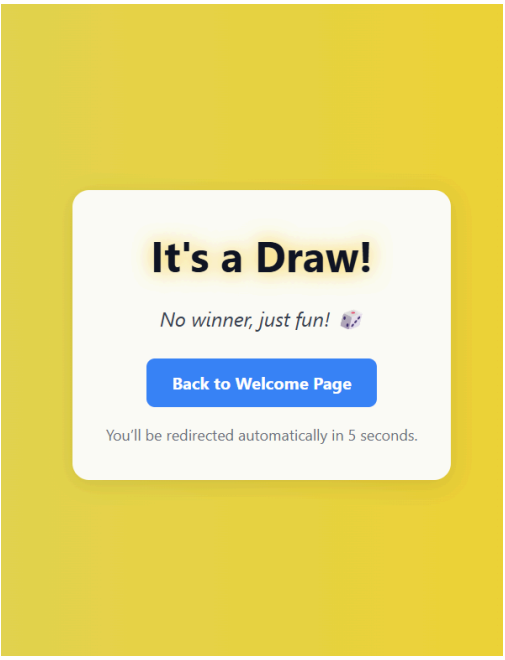
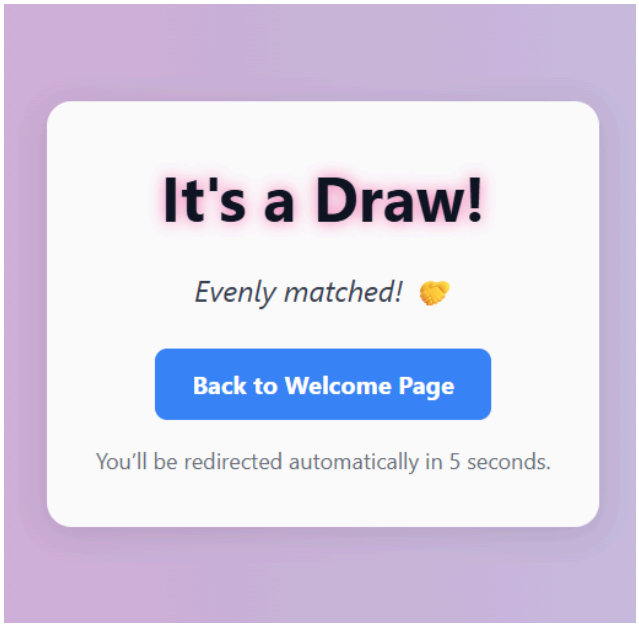


Figure 3: Draw Screen (Celebration page)

CONCLUSION

The experiment successfully demonstrated how Tailwind CSS can be used to build a responsive and interactive UI. The 30% extra feature of the animated winner celebration screen showcased the use of Tailwind's advanced animation utilities, enhancing user engagement and visual appeal.



Change the colors using the utilities!!