<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

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

<title>Tic Tac Toe - Player vs AI or Player vs Player</title>

<link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;700&display=swap" rel="stylesheet" />

<style>

/\* Reset and base \*/

\* {

box-sizing: border-box;

}

body {

margin: 0;

font-family: 'Inter', sans-serif;

background: #f9fafb;

color: #374151;

display: flex;

flex-direction: column;

min-height: 100vh;

}

.container {

max-width: 480px;

margin: 40px auto 64px;

padding: 0 24px;

display: flex;

flex-direction: column;

align-items: center;

}

h1 {

font-weight: 800;

font-size: 2.75rem;

margin-bottom: 8px;

text-align: center;

color: #111827;

}

p.subtitle {

font-weight: 500;

font-size: 1rem;

color: #6b7280;

margin-bottom: 32px;

text-align: center;

}

/\* Mode selection \*/

.mode-select {

display: flex;

justify-content: center;

gap: 20px;

margin-bottom: 32px;

}

.mode-button {

cursor: pointer;

background-color: #e0e7ff;

border-radius: 12px;

border: 1.5px solid transparent;

padding: 12px 28px;

font-weight: 600;

font-size: 1rem;

color: #3730a3;

transition: background-color 0.3s ease, border-color 0.3s ease;

user-select: none;

min-width: 160px;

text-align: center;

}

.mode-button:hover,

.mode-button:focus {

background-color: #c7d2fe;

outline: none;

}

.mode-button.active {

background-color: #3730a3;

color: #fff;

border-color: #4f46e5;

}

/\* Game board \*/

.board {

display: grid;

grid-template-columns: repeat(3, 1fr);

grid-gap: 12px;

width: 100%;

max-width: 480px;

aspect-ratio: 1/1;

background: #e0e7ff;

padding: 12px;

border-radius: 16px;

box-shadow: 0 8px 16px rgb(59 130 246 / 0.15);

}

.cell {

background-color: #fff;

border-radius: 12px;

box-shadow: 0 2px 5px rgb(0 0 0 / 0.1);

font-size: 4.5rem;

font-weight: 800;

color: #4338ca;

display: flex;

justify-content: center;

align-items: center;

cursor: pointer;

transition: background-color 0.1s ease;

user-select: none;

}

.cell:focus-visible {

outline: 3px solid #6366f1;

outline-offset: -3px;

}

.cell:hover:not(.disabled):not(.winner) {

background-color: #c7d2fe;

}

.cell.disabled {

cursor: default;

color: #9ca3af;

}

.cell.winner {

color: #2563eb;

background-color: #dbE0ff;

box-shadow: 0 0 12px 3px #2563ebaa;

}

/\* Info & buttons \*/

.info {

margin: 32px 0 24px;

font-size: 1.25rem;

font-weight: 600;

color: #1e3a8a;

text-align: center;

min-height: 1.5rem;

}

.btn-reset {

background-color: #4338ca;

border: none;

padding: 14px 32px;

border-radius: 12px;

color: white;

cursor: pointer;

font-weight: 700;

font-size: 1rem;

transition: background-color 0.3s ease;

user-select: none;

}

.btn-reset:hover,

.btn-reset:focus {

background-color: #3730a3;

outline: none;

}

/\* Responsive \*/

@media (max-width: 440px) {

.mode-button {

min-width: 140px;

padding: 10px 20px;

font-size: 0.9rem;

}

.cell {

font-size: 3.5rem;

}

}

</style>

</head>

<body>

<main class="container" role="main" aria-label="Tic Tac Toe Game">

<h1>Tic Tac Toe</h1>

<p class="subtitle">Play against another player or challenge the AI opponent</p>

<section class="mode-select" role="region" aria-label="Game Mode Selection">

<button id="mode-pvp" class="mode-button active" aria-pressed="true" type="button">Player vs Player</button>

<button id="mode-pvai" class="mode-button" aria-pressed="false" type="button">Player vs AI</button>

</section>

<section class="board" role="grid" aria-label="Game Board" tabindex="0" aria-live="polite">

<!-- 9 cells: will be populated by JS -->

</section>

<div class="info" aria-live="assertive" aria-atomic="true" id="game-status" role="status">Current Turn: Player X</div>

<button class="btn-reset" id="reset-btn" type="button" aria-label="Reset game">Restart Game</button>

</main>

<script>

(() => {

const boardElement = document.querySelector('.board');

const statusElement = document.getElementById('game-status');

const resetBtn = document.getElementById('reset-btn');

const modePvpBtn = document.getElementById('mode-pvp');

const modePvAiBtn = document.getElementById('mode-pvai');

// Game variables

let board = ['', '', '', '', '', '', '', '', ''];

let currentPlayer = 'X';

let gameActive = true;

let gameMode = 'pvp'; // 'pvp' or 'pvai'

let winnerIndices = [];

// Winning combinations

const winningCombos = [

[0,1,2], [3,4,5], [6,7,8], // rows

[0,3,6], [1,4,7], [2,5,8], // columns

[0,4,8], [2,4,6] // diagonals

];

// Initialize board in DOM

function createBoard() {

boardElement.innerHTML = '';

for(let i=0; i<9; i++) {

const cell = document.createElement('button');

cell.className = 'cell';

cell.setAttribute('data-cell', i);

cell.setAttribute('role', 'gridcell');

cell.setAttribute('aria-label', Cell ${i+1});

cell.addEventListener('click', handleCellClick);

cell.disabled = false;

boardElement.appendChild(cell);

}

}

// Update board UI for current board state

function updateBoard() {

const cells = boardElement.querySelectorAll('.cell');

cells.forEach((cell, idx) => {

cell.textContent = board[idx];

if (winnerIndices.includes(idx)) {

cell.classList.add('winner');

} else {

cell.classList.remove('winner');

}

cell.disabled = !gameActive || board[idx] !== '';

if (cell.disabled) {

cell.classList.add('disabled');

} else {

cell.classList.remove('disabled');

}

});

}

// Check for win or draw

function checkGameOver() {

winnerIndices = [];

for (const combo of winningCombos) {

const [a,b,c] = combo;

if (board[a] && board[a] === board[b] && board[b] === board[c]) {

winnerIndices = combo;

return board[a];

}

}

if (!board.includes('')) {

return 'draw';

}

return null;

}

// Switch player

function switchPlayer() {

currentPlayer = currentPlayer === 'X' ? 'O' : 'X';

}

// AI move logic: simple - tries to win, block, else random empty cell

function aiMove() {

if (!gameActive) return;

// Check if AI can win in next move

for (const combo of winningCombos) {

const marks = combo.map(i => board[i]);

if (marks.filter(m => m === 'O').length === 2 && marks.includes('')) {

const move = combo[marks.indexOf('')];

makeMove(move);

return;

}

}

// Check if AI needs to block player

for (const combo of winningCombos) {

const marks = combo.map(i => board[i]);

if (marks.filter(m => m === 'X').length === 2 && marks.includes('')) {

const move = combo[marks.indexOf('')];

makeMove(move);

return;

}

}

// Otherwise, pick random empty cell

const emptyIndices = board.flatMap((v,i)=>v === '' ? i : []);

if (emptyIndices.length === 0) return;

const move = emptyIndices[Math.floor(Math.random() \* emptyIndices.length)];

makeMove(move);

}

// Make a move at the chosen cell index

function makeMove(index) {

if (!gameActive || board[index] !== '') return;

board[index] = currentPlayer;

updateBoard();

const winner = checkGameOver();

if (winner) {

gameActive = false;

if (winner === 'draw') {

statusElement.textContent = 'Game ended in a draw.';

} else {

statusElement.textContent = Player ${winner} wins!;

}

return;

}

switchPlayer();

statusElement.textContent = Current Turn: Player ${currentPlayer};

if (gameMode === 'pvai' && currentPlayer === 'O' && gameActive) {

// Delay AI move for UX

setTimeout(aiMove, 450);

}

}

// Handle user click on cell

function handleCellClick(e) {

if (!gameActive) return;

if (gameMode === 'pvai' && currentPlayer === 'O') return; // Ignore user clicks on AI turn

const index = Number(e.target.getAttribute('data-cell'));

if (board[index] === '') {

makeMove(index);

}

}

// Reset game

function resetGame() {

board = ['', '', '', '', '', '', '', '', ''];

currentPlayer = 'X';

gameActive = true;

winnerIndices = [];

statusElement.textContent = 'Current Turn: Player X';

updateBoard();

// If AI starts first (optional), can trigger here

if (gameMode === 'pvai' && currentPlayer === 'O') {

setTimeout(aiMove, 350);

}

}

// Switch mode between pvp and pvai

function switchMode(mode) {

if (mode === gameMode) return;

gameMode = mode;

if (mode === 'pvp') {

modePvpBtn.classList.add('active');

modePvAiBtn.classList.remove('active');

modePvpBtn.setAttribute('aria-pressed', 'true');

modePvAiBtn.setAttribute('aria-pressed', 'false');

} else {

modePvAiBtn.classList.add('active');

modePvpBtn.classList.remove('active');

modePvAiBtn.setAttribute('aria-pressed', 'true');

modePvpBtn.setAttribute('aria-pressed', 'false');

}

resetGame();

}

// Initialize

createBoard();

updateBoard();

// Event listeners

resetBtn.addEventListener('click', resetGame);

modePvpBtn.addEventListener('click', () => switchMode('pvp'));

modePvAiBtn.addEventListener('click', () => switchMode('pvai'));

})();

</script>

</body>

</html>