# Grid Glider Docs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">

    <title>Snake Game - Ultimate Edition</title>

    <!-- PWA Meta Tags -->

    <meta name="theme-color" content="#0c0a18"/>

    <meta name="apple-mobile-web-app-capable" content="yes">

    <meta name="apple-mobile-web-app-status-bar-style" content="black-translucent">

    <meta name="apple-mobile-web-app-title" content="Snake Game">

    <link id="manifestLink" rel="manifest">

    <script src="https://cdn.tailwindcss.com"></script>

    <link rel="preconnect" href="https://fonts.googleapis.com">

    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

    <link href="https://fonts.googleapis.com/css2?family=Press+Start+2P&family=Roboto+Mono:wght@400;700&display=swap" rel="stylesheet">

    <style>

        body {

            font-family: 'Roboto Mono', monospace;

            overflow: hidden;

            background-color: #0c0a18;

            background-image: radial-gradient(ellipse at center, #2a2a4a 0%, #0c0a18 70%);

        }

        .font-pixel { font-family: 'Press-Start-2P', cursive; }

        .modal { display: flex; position: fixed; z-index: 1000; left: 0; top: 0; width: 100%; height: 100%; background-color: rgba(0, 0, 0, 0.5); align-items: center; justify-content: center; }

        .modal-content {

            background-color: rgba(20, 20, 30, 0.85);

            backdrop-filter: blur(10px);

            border: 2px solid #a855f7; /\* Purple border \*/

            box-shadow: 0 0 20px rgba(168, 85, 247, 0.5);

        }

        .hidden { display: none; }

        canvas { display: block; image-rendering: pixelated; image-rendering: crisp-edges; }

        .touch-controls button { width: 60px; height: 60px; background-color: rgba(255, 255, 255, 0.2); border-radius: 50%; display: flex; align-items: center; justify-content: center; }

        .btn-active { box-shadow: inset 0 0 10px rgba(255, 255, 100, 0.8); }

        .theme-btn.locked { filter: grayscale(1) brightness(0.5); cursor: not-allowed; }

    </style>

</head>

<body class="bg-gray-900 text-white flex flex-col items-center justify-center min-h-screen p-2 md:p-4">

    <div id="gameContainer" class="hidden relative">

        <div class="w-full max-w-2xl text-center mb-4">

            <div class="flex justify-between items-center">

                <div class="font-pixel text-lg">SCORE: <span id="score" class="text-red-400">0</span></div>

                <h1 class="font-pixel text-3xl md:text-4xl text-yellow-400">SNAKE</h1>

                <div id="timerDisplay" class="font-pixel text-lg text-blue-400 w-28 text-right"></div>

            </div>

        </div>

        <div id="canvas-container" class="bg-gray-800 border-4 border-yellow-500 shadow-lg rounded-lg overflow-hidden relative">

            <canvas id="gameCanvas"></canvas>

            <div id="comboDisplay" class="absolute top-2 left-2 font-pixel text-yellow-300 text-lg hidden">COMBO x2</div>

            <div id="pauseScreen" class="absolute inset-0 modal bg-black bg-opacity-50 hidden"><h2 class="font-pixel text-4xl text-white">PAUSED</h2></div>

            <button id="playPauseButton" class="absolute top-2 right-2 w-10 h-10 bg-black bg-opacity-40 rounded-full text-white flex items-center justify-center z-20 hover:bg-opacity-60 transition">

                <svg id="pauseIcon" xmlns="http://www.w3.org/2000/svg" class="h-6 w-6" fill="none" viewBox="0 0 24 24" stroke="currentColor"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M10 9v6m4-6v6" /></svg>

                <svg id="playIcon" xmlns="http://www.w3.org/2000/svg" class="h-6 w-6 hidden" fill="none" viewBox="0 0 24 24" stroke="currentColor"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M14.752 11.168l-3.197-2.132A1 1 0 0010 9.87v4.263a1 1 0 001.555.832l3.197-2.132a1 1 0 000-1.664z" /></svg>

            </button>

        </div>

        <div class="mt-4 text-center text-gray-400 text-sm">Use Arrow Keys or WASD to move. Press 'P' to Pause.</div>

    </div>

    <div id="homeScreen" class="modal"><div class="modal-content rounded-lg p-6 md:p-8 text-center shadow-2xl max-w-md w-full"><h2 class="font-pixel text-4xl text-yellow-400 mb-4">SNAKE</h2><div class="mb-6"><h3 id="leaderboardTitle" class="font-pixel text-xl text-yellow-300 mb-3">High Scores</h3><ol id="leaderboardList" class="text-left text-lg space-y-1"></ol></div><p id="loadingText" class="text-lg mb-6">Loading Assets...</p><div id="homeButtons" class="hidden"><button id="playButton" class="font-pixel bg-yellow-500 hover:bg-yellow-600 text-gray-900 font-bold py-3 px-8 rounded-lg transition duration-300 text-xl w-full">Play</button><button id="settingsButton" class="font-pixel mt-4 bg-gray-600 hover:bg-gray-700 text-white font-bold py-2 px-6 rounded-lg transition duration-300 w-full">Settings</button></div></div></div>

    <div id="gameModeScreen" class="modal hidden"><div class="modal-content rounded-lg p-6 md:p-8 text-center shadow-2xl max-w-md w-full"><h2 class="font-pixel text-3xl text-yellow-400 mb-6">Select Mode</h2><div class="space-y-4"><button data-mode="classic" class="game-mode-btn font-pixel bg-green-600 hover:bg-green-700 p-3 rounded-lg w-full">Classic</button><button data-mode="maze" class="game-mode-btn font-pixel bg-blue-600 hover:bg-blue-700 p-3 rounded-lg w-full">Hurdles</button><button data-mode="timeAttack" class="game-mode-btn font-pixel bg-purple-600 hover:bg-purple-700 p-3 rounded-lg w-full">Time Attack</button></div><button id="modeBackButton" class="font-pixel mt-6 bg-gray-600 hover:bg-gray-700 text-white font-bold py-2 px-6 rounded-lg">Back</button></div></div>

    <div id="settingsScreen" class="modal hidden"><div class="modal-content rounded-lg p-6 md:p-8 text-center shadow-2xl max-w-md w-full"><h2 class="font-pixel text-3xl text-yellow-400 mb-6">Settings</h2><div class="mb-4"><h3 class="font-pixel text-xl mb-3">Difficulty</h3><div class="flex justify-center space-x-2"><button data-difficulty="easy" class="difficulty-btn font-pixel bg-green-600 hover:bg-green-700 p-2 rounded-lg flex-1">Easy</button><button data-difficulty="medium" class="difficulty-btn font-pixel bg-yellow-600 hover:bg-yellow-700 p-2 rounded-lg flex-1">Medium</button><button data-difficulty="hard" class="difficulty-btn font-pixel bg-red-600 hover:bg-red-700 p-2 rounded-lg flex-1">Hard</button></div></div><div class="mb-4"><h3 class="font-pixel text-xl mb-3">Sound</h3><button id="soundToggle" class="font-pixel bg-gray-600 hover:bg-gray-700 p-2 rounded-lg w-full">Sound: ON</button></div><div class="mb-6"><h3 class="font-pixel text-xl mb-3">Themes</h3><div id="themeSelection" class="flex justify-center space-x-2"></div></div><button id="settingsBackButton" class="font-pixel bg-yellow-500 hover:bg-yellow-600 text-gray-900 font-bold py-2 px-6 rounded-lg">Back</button></div></div>

    <div id="gameOverModal" class="modal hidden"><div class="modal-content rounded-lg p-8 text-center shadow-2xl"><h2 id="gameOverTitle" class="font-pixel text-3xl text-red-500 mb-4">GAME OVER</h2><p class="text-lg mb-4">Final Score: <span id="finalScore" class="font-bold text-red-400">0</span></p><div id="newHighScoreContainer" class="hidden"><p class="text-yellow-300 font-pixel mb-2">New High Score!</p><input type="text" id="playerNameInput" placeholder="Enter Name" maxlength="10" class="font-pixel text-center uppercase bg-gray-700 border-2 border-yellow-500 rounded p-2 w-full max-w-xs"></div><button id="restartButton" class="font-pixel mt-4 bg-yellow-500 hover:bg-yellow-600 text-gray-900 font-bold py-2 px-6 rounded-lg">Play Again</button></div></div>

    <div id="touchControlsContainer" class="fixed bottom-4 right-4 grid grid-cols-3 gap-2 w-48 hidden"><div></div><button id="touchUp">▲</button><div></div><button id="touchLeft">◀</button><div></div><button id="touchRight">▶</button><div></div><button id="touchDown">▼</button><div></div></div>

    <script type="module">

        // --- FIREBASE IMPORTS ---

        import { initializeApp } from "https://www.gstatic.com/firebasejs/10.7.1/firebase-app.js";

        import { getAuth, signInAnonymously, signInWithCustomToken } from "https://www.gstatic.com/firebasejs/10.7.1/firebase-auth.js";

        import { getFirestore, collection, addDoc, query, orderBy, limit, getDocs, setLogLevel } from "https://www.gstatic.com/firebasejs/10.7.1/firebase-firestore.js";

        // --- SETUP ---

        const canvas = document.getElementById('gameCanvas'); const ctx = canvas.getContext('2d');

        const scoreElement = document.getElementById('score'), finalScoreElement = document.getElementById('finalScore'), timerDisplay = document.getElementById('timerDisplay'), comboDisplay = document.getElementById('comboDisplay');

        const gameContainer = document.getElementById('gameContainer'), homeScreen = document.getElementById('homeScreen'), gameOverModal = document.getElementById('gameOverModal'), pauseScreen = document.getElementById('pauseScreen'), settingsScreen = document.getElementById('settingsScreen'), gameModeScreen = document.getElementById('gameModeScreen');

        const playButton = document.getElementById('playButton'), restartButton = document.getElementById('restartButton'), homeButtons = document.getElementById('homeButtons'), settingsButton = document.getElementById('settingsButton'), settingsBackButton = document.getElementById('settingsBackButton'), soundToggle = document.getElementById('soundToggle'), modeBackButton = document.getElementById('modeBackButton'), themeSelection = document.getElementById('themeSelection');

        const leaderboardList = document.getElementById('leaderboardList'), newHighScoreContainer = document.getElementById('newHighScoreContainer'), playerNameInput = document.getElementById('playerNameInput'), leaderboardTitle = document.getElementById('leaderboardTitle');

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

        const playPauseButton = document.getElementById('playPauseButton'), pauseIcon = document.getElementById('pauseIcon'), playIcon = document.getElementById('playIcon');

        // --- FIREBASE SETUP ---

        let db = null;

        try {

            const firebaseConfigStr = 'eyJhcGlLZXkiOiJBSXphU3lBOFhkMVVhUmhWQTNoQl9OMnFBU2tWejVNM2oteFhKdzQiLCJhdXRoRG9tYWluIjoic25ha2UtZ2FtZS1sZWFkZXJib2FyZC00MTQ1Zi5maXJlYmFzZWFwcC5jb20iLCJwcm9qZWN0SWQiOiJzbmFrZS1nYW1lLWxlYWRlcmJvYXJkLTQxNDVmIiwic3RvcmFnZUJ1Y2tldCI6InNuYWtlLWdhbWUtbGVhZGVyYm9hcmQtNDE0NWYuZmlyZWJhc2VzdG9yYWdlLmFwcCIsIm1lc3NhZ2luZ1NlbmRlcklkIjoiNjU3ODYwNzA5NjA5IiwiYXBwSWQiOiIxOjY1NzgwNzE5NjA5OndlYjozMWI0ZTFiYjIwYjVkZWJjN2ZmNDc4IiwibWVhc3VyZW1lbnRJZCI6IkctUkdTSFJWMUdYQyJ9';

            const firebaseConfig = JSON.parse(atob(firebaseConfigStr));

            const app = initializeApp(firebaseConfig);

            db = getFirestore(app);

            const auth = getAuth();

            await signInAnonymously(auth);

            console.log("Firebase initialized and user signed in anonymously.");

        } catch (e) {

            console.error("Firebase initialization failed:", e);

            db = null;

        }

        // --- AUDIO & MUSIC ---

        const audioCtx = new (window.AudioContext || window.webkitAudioContext)(); let musicNode = null;

        function playSound(type) { if (!audioCtx || !gameSettings.sound) return; const osc = audioCtx.createOscillator(), gain = audioCtx.createGain(); osc.connect(gain); gain.connect(audioCtx.destination); if (type === 'eat') { osc.type = 'square'; osc.frequency.setValueAtTime(440, audioCtx.currentTime); gain.gain.setValueAtTime(0.1, audioCtx.currentTime); gain.gain.exponentialRampToValueAtTime(0.00001, audioCtx.currentTime + 0.1); } if (type === 'powerup') { osc.type = 'sine'; osc.frequency.setValueAtTime(660, audioCtx.currentTime); gain.gain.setValueAtTime(0.1, audioCtx.currentTime); gain.gain.exponentialRampToValueAtTime(0.00001, audioCtx.currentTime + 0.2); } if (type === 'gameover') { osc.type = 'sawtooth'; osc.frequency.setValueAtTime(164, audioCtx.currentTime); osc.frequency.exponentialRampToValueAtTime(110, audioCtx.currentTime + 0.5); gain.gain.setValueAtTime(0.2, audioCtx.currentTime); gain.gain.exponentialRampToValueAtTime(0.00001, audioCtx.currentTime + 0.5); } if (type === 'click') { osc.type = 'triangle'; osc.frequency.setValueAtTime(880, audioCtx.currentTime); gain.gain.setValueAtTime(0.05, audioCtx.currentTime); gain.gain.exponentialRampToValueAtTime(0.00001, audioCtx.currentTime + 0.1); } osc.start(audioCtx.currentTime); osc.stop(audioCtx.currentTime + 0.5); }

        function startMusic() { if (!audioCtx || !gameSettings.sound || musicNode) return; musicNode = { osc: audioCtx.createOscillator(), gain: audioCtx.createGain(), interval: null }; musicNode.osc.connect(musicNode.gain); musicNode.gain.connect(audioCtx.destination); musicNode.osc.type = 'sine'; musicNode.gain.gain.value = 0.05; const notes = [130, 164, 196, 164]; let noteIndex = 0; musicNode.interval = setInterval(() => { musicNode.osc.frequency.setValueAtTime(notes[noteIndex % notes.length], audioCtx.currentTime); noteIndex++; }, 500); musicNode.osc.start(); }

        function stopMusic() { if (musicNode) { musicNode.osc.stop(); clearInterval(musicNode.interval); musicNode = null; } }

        // --- CONFIG ---

        const TILE\_SIZE = 32, TILE\_COUNT\_X = 20, TILE\_COUNT\_Y = 15;

        canvas.width = TILE\_SIZE \* TILE\_COUNT\_X; canvas.height = TILE\_SIZE \* TILE\_COUNT\_Y;

        const SPEED\_INCREMENT = 5;

        const SETTINGS\_KEY = 'snakeGameSettings', PLAYER\_NAME\_KEY = 'snakePlayerName', UNLOCKED\_THEMES\_KEY = 'snakeUnlockedThemes', TOTAL\_SCORE\_KEY = 'snakeTotalScore';

        const DIFFICULTY\_SPEEDS = { easy: 200, medium: 150, hard: 100 };

        // --- GAME STATE ---

        let snake, foods, enemies, direction, score, gameOver, logicInterval, currentSpeed, particles, highScores, isPaused = false, gameMode = 'classic', maze, timer, timerInterval, applesEaten, combo, comboTimer;

        let gameSettings = { difficulty: 'medium', sound: true, theme: 'default' };

        const assets = {};

        // --- THEME DEFINITIONS ---

        const THEMES = {

            'default': { name: 'Starry Night', unlockScore: 0, assets: { background: createStarryBackground, wall: createWall, apple: createApple, goldenApple: createGoldenApple, slowPotion: createSlowPotion, shrinkPotion: createShrinkPotion, snakeHead: createSnakeHead, snakeBody: createSnakeBody, snakeTail: createSnakeTail, enemy: createEnemy } },

            'tron': { name: 'Tron', unlockScore: 100, assets: { background: createTronBackground, wall: createTronWall, apple: createTronApple, goldenApple: createTronGoldenApple, slowPotion: createTronSlowPotion, shrinkPotion: createTronShrinkPotion, snakeHead: createTronSnakeHead, snakeBody: createTronSnakeBody, snakeTail: createTronSnakeTail, enemy: createTronEnemy } },

            'jungle': { name: 'Jungle', unlockScore: 250, assets: { background: createJungleBackground, wall: createJungleWall, apple: createJungleApple, goldenApple: createJungleGoldenApple, slowPotion: createJungleSlowPotion, shrinkPotion: createJungleShrinkPotion, snakeHead: createJungleSnakeHead, snakeBody: createJungleSnakeBody, snakeTail: createJungleSnakeTail, enemy: createJungleEnemy } }

        };

        const DIRECTIONS = { UP: { x: 0, y: -1 }, DOWN: { x: 0, y: 1 }, LEFT: { x: -1, y: 0 }, RIGHT: { x: 1, y: 0 }, STOP: { x: 0, y: 0 } };

        // --- ASSET & THEME LOADING ---

        function createSvgImage(svgString, callback) { const img = new Image(); img.onload = () => callback(img); img.src = `data:image/svg+xml;base64,${btoa(svgString)}`; }

        function loadThemeAssets() {

            const themeAssets = THEMES[gameSettings.theme].assets;

            const totalAssets = Object.keys(themeAssets).length;

            let assetsLoaded = 0;

            homeButtons.classList.add('hidden');

            loadingText.classList.remove('hidden');

            for (const key in themeAssets) {

                createSvgImage(themeAssets[key](), (img) => {

                    assets[key] = img;

                    assetsLoaded++;

                    if (assetsLoaded === totalAssets) {

                        loadingText.classList.add('hidden');

                        homeButtons.classList.remove('hidden');

                    }

                });

            }

        }

        function createStarryBackground() { return `<svg width="64" height="64" viewBox="0 0 64 64" xmlns="http://www.w3.org/2000/svg"><defs><radialGradient id="grad" cx="50%" cy="50%" r="50%"><stop offset="0%" stop-color="#2a2a4a"/><stop offset="100%" stop-color="#0c0a18"/></radialGradient><style>.star{animation:twinkle 2s ease-in-out infinite alternate;}@keyframes twinkle{0%{opacity:0.5;}100%{opacity:1;}}</style></defs><rect width="64" height="64" fill="url(#grad)"/><circle class="star" cx="10" cy="15" r="1" fill="white"/><circle class="star" cx="50" cy="20" r="1.2" fill="white" style="animation-delay:0.5s;"/><circle class="star" cx="30" cy="50" r="0.8" fill="white" style="animation-delay:1s;"/><circle class="star" cx="60" cy="55" r="1" fill="white" style="animation-delay:1.5s;"/></svg>`; }

        function createWall() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><rect width="32" height="32" fill="#4A5568"/><rect x="2" y="2" width="28" height="28" fill="#718096"/><path d="M0 0 H16 V4 H4 V16 H0Z M32 0 H16 V4 H28 V16 H32Z M0 32 H16 V28 H4 V16 H0Z M32 32 H16 V28 H28 V16 H32Z" fill="#2D3748"/></svg>`; }

        function createApple() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><circle cx="16" cy="16" r="12" fill="#DC2626"/><path d="M16 4 A 8 8 0 0 1 20 6" stroke="#166534" stroke-width="3" fill="none"/><circle cx="19" cy="11" r="2" fill="rgba(255,255,255,0.5)"/></svg>`; }

        function createGoldenApple() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><circle cx="16" cy="16" r="12" fill="#FBBF24"/><path d="M16 4 A 8 8 0 0 1 20 6" stroke="#F59E0B" stroke-width="3" fill="none"/><circle cx="19" cy="11" r="3" fill="white" fill-opacity="0.7"/></svg>`; }

        function createSlowPotion() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><path d="M16 2 L28 12 L24 30 L8 30 L4 12 Z" fill="#3B82F6"/><circle cx="16" cy="18" r="8" fill="#60A5FA"/></svg>`; }

        function createShrinkPotion() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><path d="M16 2 L28 12 L24 30 L8 30 L4 12 Z" fill="#8B5CF6"/><circle cx="16" cy="18" r="8" fill="#A78BFA"/></svg>`; }

        function createSnakeHead() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><rect x="1" y="1" width="30" height="30" rx="15" fill="#FFD700"/><circle cx="22" cy="10" r="4" fill="#111827" transform="rotate(90 16 16)"/><circle cx="22" cy="22" r="4" fill="#111827" transform="rotate(90 16 16)"/></svg>`; }

        function createSnakeBody() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><rect x="1" y="1" width="30" height="30" rx="8" fill="#DAA520"/><rect x="6" y="6" width="20" height="20" rx="4" fill="#FFD700" /></svg>`; }

        function createSnakeTail() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><path d="M16 1 L31 16 L16 31 L1 16 Z" fill="#DAA520"/></svg>`; }

        function createEnemy() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><rect width="32" height="32" fill="#4B0082"/><path d="M8 8 L24 24 M24 8 L8 24" stroke="#FF00FF" stroke-width="4"/></svg>`; }

        function createTronBackground() { return `<svg width="64" height="64" viewBox="0 0 64 64" xmlns="http://www.w3.org/2000/svg"><defs><filter id="glow"><feGaussianBlur stdDeviation="1.5" result="coloredBlur"/><feMerge><feMergeNode in="coloredBlur"/><feMergeNode in="SourceGraphic"/></feMerge></filter><style>.grid-line{animation:pulse 3s linear infinite;}@keyframes pulse{0%{stroke-opacity:0.3;}50%{stroke-opacity:0.8;}100%{stroke-opacity:0.3;}}</style></defs><rect width="64" height="64" fill="#000"/><path class="grid-line" d="M0 0 H64 M0 8 H64 M0 16 H64 M0 24 H64 M0 32 H64 M0 40 H64 M0 48 H64 M0 56 H64 M0 0 V64 M8 0 V64 M16 0 V64 M24 0 V64 M32 0 V64 M40 0 V64 M48 0 V64 M56 0 V64" stroke="#00FFFF" stroke-width="1" filter="url(#glow)"/></svg>`; }

        function createTronWall() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><rect width="32" height="32" fill="#00FFFF"/><rect x="2" y="2" width="28" height="28" fill="#000"/></svg>`; }

        function createTronApple() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><circle cx="16" cy="16" r="14" stroke="#FF00FF" stroke-width="3" fill="none"/></svg>`; }

        function createTronGoldenApple() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><circle cx="16" cy="16" r="14" stroke="#FFFF00" stroke-width="3" fill="none"/></svg>`; }

        function createTronSlowPotion() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><rect x="4" y="4" width="24" height="24" stroke="#0000FF" stroke-width="3" fill="none"/></svg>`; }

        function createTronShrinkPotion() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><path d="M16 2 L30 16 L16 30 L2 16 Z" stroke="#FF00FF" stroke-width="3" fill="none"/></svg>`; }

        function createTronSnakeHead() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><rect x="1" y="1" width="30" height="30" rx="15" fill="#00FFFF" transform="rotate(90 16 16)"/></svg>`; }

        function createTronSnakeBody() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><rect x="1" y="1" width="30" height="30" rx="8" fill="#00AAAA"/></svg>`; }

        function createTronSnakeTail() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><path d="M16 1 L31 16 L16 31 L1 16 Z" fill="#00AAAA"/></svg>`; }

        function createTronEnemy() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><rect width="32" height="32" fill="#FF0000"/></svg>`; }

        function createJungleBackground() { return `<svg width="64" height="64" viewBox="0 0 64 64" xmlns="http://www.w3.org/2000/svg"><rect width="64" height="64" fill="#004d00"/><path d="M0 64 C 10 50, 20 50, 32 64 S 54 50, 64 64" stroke="#008000" stroke-width="4" fill="none" opacity="0.4"/><path d="M-5 5 C 10 20, 20 10, 32 20 S 54 10, 64 20" stroke="#32CD32" stroke-width="3" fill="none" opacity="0.5"/><path d="M0 0 C 10 14, 20 14, 32 0 S 54 14, 64 0" stroke="#008000" stroke-width="4" fill="none" opacity="0.4"/></svg>`;}

        function createJungleWall() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><rect width="32" height="32" fill="#8B4513"/><rect x="4" y="0" width="8" height="32" fill="#A0522D"/><rect x="20" y="0" width="8" height="32" fill="#A0522D"/></svg>`; }

        function createJungleApple() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><circle cx="16" cy="16" r="12" fill="#FFD700"/><path d="M10 10 C 16 4, 22 4, 22 10" fill="#FF8C00"/></svg>`; }

        function createJungleGoldenApple() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><circle cx="16" cy="16" r="12" fill="#FF4500"/><circle cx="16" cy="16" r="8" fill="#FF6347"/><circle cx="16" cy="16" r="4" fill="#FF7F50"/></svg>`; }

        function createJungleSlowPotion() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><circle cx="16" cy="16" r="12" fill="#4682B4"/><path d="M16 8 C 12 12, 12 20, 16 24 C 20 20, 20 12, 16 8" fill="#87CEEB"/></svg>`; }

        function createJungleShrinkPotion() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><circle cx="16" cy="16" r="12" fill="#9370DB"/><circle cx="16" cy="16" r="8" fill="#BA55D3"/></svg>`; }

        function createJungleSnakeHead() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><rect x="1" y="1" width="30" height="30" rx="15" fill="#228B22" transform="rotate(90 16 16)"/><circle cx="22" cy="10" r="4" fill="#FFFF00" /><circle cx="22" cy="22" r="4" fill="#FFFF00" /></svg>`; }

        function createJungleSnakeBody() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><rect x="1" y="1" width="30" height="30" rx="8" fill="#32CD32"/><rect x="6" y="6" width="20" height="20" rx="4" fill="#ADFF2F" /></svg>`; }

        function createJungleSnakeTail() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><path d="M16 1 L31 16 L16 31 L1 16 Z" fill="#32CD32"/></svg>`; }

        function createJungleEnemy() { return `<svg width="32" height="32" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><path d="M16 2 L2 16 L16 30 L30 16 Z" fill="#696969"/><circle cx="10" cy="16" r="3" fill="red"/><circle cx="22" cy="16" r="3" fill="red"/></svg>`; }

        // --- MAZE & LEVEL ---

        const MAZES = [

            [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0],[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0],[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0],[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0],[0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],

            [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0],[0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]

        ];

        function loadMaze() { maze = MAZES[Math.floor(Math.random() \* MAZES.length)]; }

        // --- SETTINGS & LEADERBOARD ---

        function loadSettings() { const saved = localStorage.getItem(SETTINGS\_KEY); if (saved) gameSettings = JSON.parse(saved); updateSettingsUI(); }

        function saveSettings() { localStorage.setItem(SETTINGS\_KEY, JSON.stringify(gameSettings)); }

        function updateSettingsUI() { soundToggle.textContent = `Sound: ${gameSettings.sound ? 'ON' : 'OFF'}`; document.querySelectorAll('.difficulty-btn').forEach(btn => btn.classList.toggle('btn-active', btn.dataset.difficulty === gameSettings.difficulty)); updateThemeSelection(); }

        function getLeaderboardKey() { return `leaderboard\_${gameMode}`; }

        async function loadHighScores() {

            if (!db) {

                leaderboardList.innerHTML = '<li class="text-red-500">Online scores unavailable</li>';

                return [];

            }

            const q = query(collection(db, getLeaderboardKey()), orderBy("score", "desc"), limit(5));

            const querySnapshot = await getDocs(q);

            const scores = [];

            querySnapshot.forEach((doc) => scores.push(doc.data()));

            return scores;

        }

        async function addHighScore(name, score) {

            let totalScore = parseInt(localStorage.getItem(TOTAL\_SCORE\_KEY) || '0');

            totalScore += score;

            localStorage.setItem(TOTAL\_SCORE\_KEY, totalScore);

            if (!db) return;

            try {

                await addDoc(collection(db, getLeaderboardKey()), { name, score, createdAt: new Date() });

            } catch (e) {

                console.error("Error adding document: ", e);

            }

        }

        async function displayLeaderboard() {

            leaderboardTitle.textContent = `${gameMode.charAt(0).toUpperCase() + gameMode.slice(1)} Scores`;

            leaderboardList.innerHTML = '<li class="text-gray-500">Loading...</li>';

            highScores = await loadHighScores();

            leaderboardList.innerHTML = '';

            if (highScores.length === 0) { leaderboardList.innerHTML = '<li class="text-gray-500">Play to set a score!</li>'; return; }

            highScores.forEach((entry, i) => { const li = document.createElement('li'); li.innerHTML = `<span class="text-yellow-500">${i + 1}.</span> ${entry.name} - <span class="font-bold">${entry.score}</span>`; leaderboardList.appendChild(li); });

        }

        async function checkHighScore() {

            finalScoreElement.textContent = score;

            const savedName = localStorage.getItem(PLAYER\_NAME\_KEY);

            if (savedName) playerNameInput.value = savedName;

            const currentHighScores = await loadHighScores();

            const lowestScore = currentHighScores.length < 5 ? 0 : currentHighScores[4].score;

            if (score > 0 && score > lowestScore) {

                newHighScoreContainer.classList.remove('hidden');

                playerNameInput.focus();

                restartButton.textContent = "Save & Restart";

            } else {

                newHighScoreContainer.classList.add('hidden');

                restartButton.textContent = "Play Again";

            }

        }

        function updateThemeSelection() {

            const unlocked = JSON.parse(localStorage.getItem(UNLOCKED\_THEMES\_KEY) || '["default"]');

            const totalScore = parseInt(localStorage.getItem(TOTAL\_SCORE\_KEY) || '0');

            themeSelection.innerHTML = '';

            for(const key in THEMES) {

                const theme = THEMES[key];

                const isUnlocked = unlocked.includes(key) || totalScore >= theme.unlockScore;

                if (isUnlocked && !unlocked.includes(key)) unlocked.push(key);

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

                btn.textContent = theme.name;

                btn.dataset.theme = key;

                btn.className = `theme-btn font-pixel p-2 rounded-lg flex-1 ${isUnlocked ? 'bg-gray-600 hover:bg-gray-700' : 'locked bg-gray-800'}`;

                if (gameSettings.theme === key) btn.classList.add('btn-active');

                if (!isUnlocked) btn.title = `Unlock at ${theme.unlockScore} total score`;

                else btn.addEventListener('click', (e) => { gameSettings.theme = e.target.dataset.theme; saveSettings(); updateSettingsUI(); playSound('click'); loadThemeAssets(); });

                themeSelection.appendChild(btn);

            }

            localStorage.setItem(UNLOCKED\_THEMES\_KEY, JSON.stringify(unlocked));

        }

        // --- GAME LOGIC ---

        function initializeGame() { snake = [{ x: 10, y: 7, visualPos: {x: 10 \* TILE\_SIZE, y: 7 \* TILE\_SIZE} }]; direction = DIRECTIONS.STOP; score = 0; gameOver = false; currentSpeed = DIFFICULTY\_SPEEDS[gameSettings.difficulty]; particles = []; foods = []; enemies = []; applesEaten = 0; combo = 0; if (comboTimer) clearTimeout(comboTimer); comboDisplay.classList.add('hidden'); scoreElement.textContent = '0'; gameOverModal.classList.add('hidden'); isPaused = false; pauseScreen.classList.add('hidden'); if (gameMode === 'maze') { loadMaze(); } if (gameMode === 'timeAttack') { timer = 90; timerDisplay.textContent = `TIME: ${timer}`; timerInterval = setInterval(() => { timer--; timerDisplay.textContent = `TIME: ${timer}`; if (timer <= 0) { gameOver = true; playSound('gameover'); stopMusic(); document.getElementById('gameOverTitle').textContent = "TIME'S UP!"; } }, 1000); } else { timerDisplay.textContent = ''; document.getElementById('gameOverTitle').textContent = "GAME OVER"; } placeFood(); if (gameMode !== 'timeAttack' && gameMode !== 'maze') spawnEnemy(); startGameLoop(); startMusic(); requestAnimationFrame(gameLoop); }

        function startGameLoop() { if (logicInterval) clearInterval(logicInterval); logicInterval = setInterval(update, currentSpeed); }

        function gameLoop() { if (gameOver) { if (logicInterval) clearInterval(logicInterval); if(timerInterval) clearInterval(timerInterval); showGameOverScreen(); return; } if (!isPaused) { draw(); } requestAnimationFrame(gameLoop); }

        function update() {

            if (isPaused || gameOver || direction === DIRECTIONS.STOP) return;

            enemies.forEach(enemy => { enemy.x += enemy.vx; if (enemy.x >= TILE\_COUNT\_X || enemy.x < 0) enemy.vx \*= -1; });

            const head = { ...snake[0] };

            let wrapped = false;

            head.x += direction.x;

            head.y += direction.y;

            if (head.x >= TILE\_COUNT\_X) { head.x = 0; wrapped = true; }

            if (head.x < 0) { head.x = TILE\_COUNT\_X - 1; wrapped = true; }

            if (head.y >= TILE\_COUNT\_Y) { head.y = 0; wrapped = true; }

            if (head.y < 0) { head.y = TILE\_COUNT\_Y - 1; wrapped = true; }

            if (gameMode === 'maze' && maze[head.y][head.x] === 1) {

                gameOver = true; playSound('gameover'); stopMusic(); return;

            }

            for (let i = 1; i < snake.length; i++) { if (head.x === snake[i].x && head.y === snake[i].y) { gameOver = true; playSound('gameover'); stopMusic(); return; } }

            enemies.forEach(enemy => { if (head.x === Math.floor(enemy.x) && head.y === enemy.y) { gameOver = true; playSound('gameover'); stopMusic(); return; } });

            const visualPos = wrapped ? { x: head.x \* TILE\_SIZE, y: head.y \* TILE\_SIZE } : { ...snake[0].visualPos };

            snake.unshift({ ...head, visualPos });

            let ateFood = false;

            foods.forEach((food, index) => { if (head.x === food.x && head.y === food.y) { handleFoodConsumption(food.type); foods.splice(index, 1); ateFood = true; } });

            if (!ateFood) snake.pop();

            if (foods.length === 0) placeFood();

        }

        function handleFoodConsumption(type) {

            createParticles(snake[0].x, snake[0].y);

            let shouldGrow = true;

            let scoreBonus = 1;

            if (comboTimer) clearTimeout(comboTimer);

            combo++;

            comboTimer = setTimeout(() => { combo = 0; comboDisplay.classList.add('hidden'); }, 3000);

            if (combo > 1) { scoreBonus = combo; comboDisplay.textContent = `COMBO x${combo}`; comboDisplay.classList.remove('hidden'); }

            switch (type) {

                case 'apple': score += (1 \* scoreBonus); applesEaten++; if (gameSettings.difficulty === 'easy' && applesEaten % 2 !== 0) shouldGrow = false; playSound('eat'); break;

                case 'goldenApple': score += (5 \* scoreBonus); playSound('powerup'); break;

                case 'slowPotion': currentSpeed = Math.min(DIFFICULTY\_SPEEDS.easy, currentSpeed + 30); startGameLoop(); playSound('powerup'); setTimeout(() => { currentSpeed = Math.max(DIFFICULTY\_SPEEDS.hard, currentSpeed - 30); startGameLoop(); }, 5000); shouldGrow = false; break;

                case 'shrinkPotion': if (snake.length > 3) { snake.pop(); snake.pop(); } playSound('powerup'); shouldGrow = false; break;

            }

            if (!shouldGrow) snake.pop();

            scoreElement.textContent = score;

            const speedUpInterval = gameSettings.difficulty === 'hard' ? 3 : 5;

            if (score > 0 && score % speedUpInterval === 0 && gameMode !== 'timeAttack') { currentSpeed = Math.max(50, currentSpeed - SPEED\_INCREMENT); startGameLoop(); }

        }

        function draw() {

            ctx.clearRect(0,0,canvas.width, canvas.height);

            ctx.drawImage(assets.background, 0, 0, canvas.width, canvas.height);

            if (gameMode === 'maze') { maze.forEach((row, y) => row.forEach((tile, x) => { if (tile === 1) ctx.drawImage(assets.wall, x \* TILE\_SIZE, y \* TILE\_SIZE, TILE\_SIZE, TILE\_SIZE); })); }

            foods.forEach(food => ctx.drawImage(assets[food.type], food.x \* TILE\_SIZE, food.y \* TILE\_SIZE, TILE\_SIZE, TILE\_SIZE));

            snake.forEach((seg, i) => {

                const targetX = seg.x \* TILE\_SIZE;

                const targetY = seg.y \* TILE\_SIZE;

                seg.visualPos.x += (targetX - seg.visualPos.x) \* 0.5;

                seg.visualPos.y += (targetY - seg.visualPos.y) \* 0.5;

                ctx.save();

                ctx.translate(seg.visualPos.x + TILE\_SIZE / 2, seg.visualPos.y + TILE\_SIZE / 2);

                let angle = 0;

                if (i === 0) {

                    if (direction.x === 1) angle = Math.PI / 2; else if (direction.x === -1) angle = -Math.PI / 2;

                    else if (direction.y === 1) angle = Math.PI; else if (direction.y === -1) angle = 0;

                } else {

                    const nextSeg = snake[i - 1];

                    if (nextSeg.x > seg.x) angle = Math.PI / 2; else if (nextSeg.x < seg.x) angle = -Math.PI / 2;

                    else if (nextSeg.y > seg.y) angle = Math.PI; else if (nextSeg.y < seg.y) angle = 0;

                }

                ctx.rotate(angle);

                let img = assets.snakeBody;

                if (i === 0) img = assets.snakeHead;

                if (i === snake.length - 1 && snake.length > 1) img = assets.snakeTail;

                ctx.drawImage(img, -TILE\_SIZE / 2, -TILE\_SIZE / 2, TILE\_SIZE, TILE\_SIZE);

                ctx.restore();

            });

            enemies.forEach(enemy => ctx.drawImage(assets.enemy, enemy.x \* TILE\_SIZE, enemy.y \* TILE\_SIZE, TILE\_SIZE, TILE\_SIZE));

            particles.forEach(p => { ctx.fillStyle = `rgba(220, 38, 38, ${p.alpha})`; ctx.fillRect(p.x, p.y, p.size, p.size); p.x += p.vx; p.y += p.vy; p.alpha -= 0.04; });

            particles = particles.filter(p => p.alpha > 0);

        }

        function placeFood() {

            let foodTypes;

            switch(gameSettings.difficulty) {

                case 'easy': foodTypes = ['apple', 'apple', 'apple', 'slowPotion', 'shrinkPotion', 'goldenApple']; break;

                case 'hard': foodTypes = ['apple', 'apple', 'goldenApple']; break;

                default: foodTypes = ['apple', 'apple', 'apple', 'goldenApple', 'slowPotion', 'shrinkPotion']; break;

            }

            const type = foodTypes[Math.floor(Math.random() \* foodTypes.length)]; let foodPos; while (true) { foodPos = { x: Math.floor(Math.random() \* TILE\_COUNT\_X), y: Math.floor(Math.random() \* TILE\_COUNT\_Y) }; let onWall = (gameMode === 'maze' && maze[foodPos.y][foodPos.x] === 1); if (!onWall && !snake.some(seg => seg.x === foodPos.x && seg.y === foodPos.y) && !foods.some(f => f.x === foodPos.x && f.y === foodPos.y)) break; }

            foods.push({ ...foodPos, type });

        }

        function spawnEnemy() {

            if (enemies.length > 3) return;

            let enemyPos;

            while (true) {

                enemyPos = { x: Math.floor(Math.random() \* TILE\_COUNT\_X), y: Math.floor(Math.random() \* TILE\_COUNT\_Y) };

                if (!snake.some(seg => seg.x === enemyPos.x && seg.y === enemyPos.y)) break;

            }

            const enemy = { ...enemyPos, vx: 0.05 \* (Math.random() > 0.5 ? 1 : -1) };

            enemies.push(enemy);

        }

        function createParticles(x, y) { for (let i = 0; i < 15; i++) particles.push({ x: x \* TILE\_SIZE + TILE\_SIZE/2, y: y \* TILE\_SIZE + TILE\_SIZE/2, vx: (Math.random() - 0.5) \* 4, vy: (Math.random() - 0.5) \* 4, size: Math.random() \* 3 + 1, alpha: 1 }); }

        function showGameOverScreen() { gameOverModal.classList.remove('hidden'); checkHighScore(); }

        function togglePause() { if (gameOver) return; isPaused = !isPaused; pauseScreen.classList.toggle('hidden'); playIcon.classList.toggle('hidden'); pauseIcon.classList.toggle('hidden'); if (isPaused) { stopMusic(); if(timerInterval) clearInterval(timerInterval); } else { startMusic(); if(gameMode === 'timeAttack' && timer > 0) timerInterval = setInterval(() => { timer--; timerDisplay.textContent = `TIME: ${timer}`; if (timer <= 0) { gameOver = true; playSound('gameover'); stopMusic(); document.getElementById('gameOverTitle').textContent = "TIME'S UP!"; } }, 1000); } }

        // --- EVENT HANDLERS ---

        function handleKeyDown(e) {

            const key = e.key.toLowerCase();

            if (key === 'p') {

                togglePause();

                return;

            }

            if (isPaused) return;

            if (direction === DIRECTIONS.STOP) {

                switch (key) {

                    case 'arrowup': case 'w': direction = DIRECTIONS.UP; break;

                    case 'arrowdown': case 's': direction = DIRECTIONS.DOWN; break;

                    case 'arrowleft': case 'a': direction = DIRECTIONS.LEFT; break;

                    case 'arrowright': case 'd': direction = DIRECTIONS.RIGHT; break;

                }

            } else {

                switch (key) {

                    case 'arrowup': case 'w': if (direction !== DIRECTIONS.DOWN) direction = DIRECTIONS.UP; break;

                    case 'arrowdown': case 's': if (direction !== DIRECTIONS.UP) direction = DIRECTIONS.DOWN; break;

                    case 'arrowleft': case 'a': if (direction !== DIRECTIONS.RIGHT) direction = DIRECTIONS.LEFT; break;

                    case 'arrowright': case 'd': if (direction !== DIRECTIONS.LEFT) direction = DIRECTIONS.RIGHT; break;

                }

            }

        }

        async function handleRestart() { playSound('click'); const currentHighScores = await loadHighScores(); const lowestScore = currentHighScores.length < 5 ? 0 : currentHighScores[4].score; if (score > 0 && score >= lowestScore) { let name = playerNameInput.value.trim().toUpperCase() || 'PLAYER'; localStorage.setItem(PLAYER\_NAME\_KEY, name); await addHighScore(name, score); } await displayLeaderboard(); homeScreen.classList.remove('hidden'); gameOverModal.classList.add('hidden'); gameContainer.classList.add('hidden'); touchControlsContainer.classList.add('hidden'); stopMusic(); }

        function setupTouchControls() { document.getElementById('touchUp').addEventListener('click', () => { if (direction !== DIRECTIONS.DOWN) direction = DIRECTIONS.UP; }); document.getElementById('touchDown').addEventListener('click', () => { if (direction !== DIRECTIONS.UP) direction = DIRECTIONS.DOWN; }); document.getElementById('touchLeft').addEventListener('click', () => { if (direction !== DIRECTIONS.RIGHT) direction = DIRECTIONS.LEFT; }); document.getElementById('touchRight').addEventListener('click', () => { if (direction !== DIRECTIONS.LEFT) direction = DIRECTIONS.RIGHT; }); }

        // --- INITIALIZE ---

        document.addEventListener('keydown', handleKeyDown);

        playButton.addEventListener('click', () => { playSound('click'); homeScreen.classList.add('hidden'); gameModeScreen.classList.remove('hidden'); });

        document.querySelectorAll('.game-mode-btn').forEach(btn => btn.addEventListener('click', (e) => { gameMode = e.target.dataset.mode; playSound('click'); displayLeaderboard(); gameModeScreen.classList.add('hidden'); gameContainer.classList.remove('hidden'); if ('ontouchstart' in window) touchControlsContainer.classList.remove('hidden'); initializeGame(); }));

        modeBackButton.addEventListener('click', () => { playSound('click'); gameModeScreen.classList.add('hidden'); homeScreen.classList.remove('hidden'); });

        restartButton.addEventListener('click', handleRestart);

        settingsButton.addEventListener('click', () => { playSound('click'); settingsScreen.classList.remove('hidden'); });

        settingsBackButton.addEventListener('click', () => { playSound('click'); settingsScreen.classList.add('hidden'); });

        soundToggle.addEventListener('click', () => { gameSettings.sound = !gameSettings.sound; saveSettings(); updateSettingsUI(); playSound('click'); });

        document.querySelectorAll('.difficulty-btn').forEach(btn => btn.addEventListener('click', (e) => { gameSettings.difficulty = e.target.dataset.difficulty; saveSettings(); updateSettingsUI(); playSound('click'); }));

        playPauseButton.addEventListener('click', togglePause);

        loadSettings();

        loadThemeAssets();

        displayLeaderboard();

        setupTouchControls();

    </script>

</body>

</html>