```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1.0, user-scalable=no">
    <title>Neon Tetris</title>
    <style>
        :root {
            --bg-color: #0f172a;
            --grid-color: #1e293b;
            --text-color: #f8fafc;
            --accent-color: #38bdf8;
            --danger-color: #ef4444;
        }

        * {
            box-sizing: border-box;
            user-select: none;
            -webkit-tap-highlight-color: transparent;
        }

        body {
            background-color: var(--bg-color);
            color: var(--text-color);
            font-family: 'Segoe UI', system-ui, sans-serif;
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: center;
            min-height: 100vh;
            margin: 0;
            overflow: hidden;
            touch-action: none;
        }

        .game-wrapper {
            display: flex;
            flex-direction: column;
            align-items: center;
            gap: 15px;
            max-width: 100vw;
            padding: 10px;
        }

        .main-layout {
            display: flex;
            gap: 20px;
```

```
    background: #1e293b;
    padding: 15px;
    border-radius: 16px;
    border: 2px solid #334155;
    box-shadow: 0 25px 50px -12px rgba(0, 0, 0, 0.5);
}

canvas {
    border: 3px solid var(--accent-color);
    background-color: #000;
    border-radius: 4px;
    display: block;
    box-shadow: 0 0 15px rgba(56, 189, 248, 0.2);
}

.side-panel {
    display: flex;
    flex-direction: column;
    justify-content: space-between;
    width: 120px;
}

.stats-group {
    display: flex;
    flex-direction: column;
    gap: 10px;
}

.stat-card {
    background: #0f172a;
    padding: 10px;
    border-radius: 8px;
    text-align: center;
    border: 1px solid #475569;
}

.stat-label {
    font-size: 0.7rem;
    text-transform: uppercase;
    color: #94a3b8;
    letter-spacing: 0.05em;
}

.stat-value {
    font-size: 1.4rem;
    font-weight: 800;
    color: var(--accent-color);
}
```

```
        /* Controller Buttons */
        .controls-grid {
            display: grid;
            grid-template-columns: repeat(3, 70px);
            grid-template-rows: repeat(2, 70px);
            gap: 10px;
            margin-top: 10px;
        }

        .control-btn {
            background: #334155;
            border: none;
            border-radius: 12px;
            color: white;
            font-size: 1.5rem;
            display: flex;
            align-items: center;
            justify-content: center;
            cursor: pointer;
            box-shadow: 0 4px 0 #1e293b;
            transition: transform 0.05s, box-shadow 0.05s;
        }

        .control-btn:active {
            transform: translateY(2px);
            box-shadow: 0 2px 0 #1e293b;
            background: #475569;
        }

        .control-btn.wide { grid-column: span 3; height: 50px; font-size: 1rem; text-transform:
uppercase; font-weight: bold; }
        .btn-drop { background: #0ea5e9; }
        .btn-rotate { background: #8b5cf6; }

        .system-btns {
            display: flex;
            gap: 10px;
            width: 100%;
            margin-top: 10px;
        }

        .sys-btn {
            flex: 1;
            padding: 12px;
            border: none;
            border-radius: 8px;
            font-weight: bold;
```

```
            cursor: pointer;
            background: #475569;
            color: white;
        }

        /* Overlay */
        #overlay {
            position: fixed;
            inset: 0;
            background: rgba(15, 23, 42, 0.95);
            display: none;
            flex-direction: column;
            align-items: center;
            justify-content: center;
            z-index: 100;
            text-align: center;
        }

        #overlay h1 { font-size: 3.5rem; color: var(--danger-color); margin: 0; }

        @media (max-width: 500px) {
            .main-layout { flex-direction: column; align-items: center; }
            .side-panel { width: 100%; flex-direction: row; gap: 10px; }
            .side-panel .stat-card { flex: 1; }
            canvas { width: 240px; height: 480px; }
        }
    </style>
</head>
<body>

    <div class="game-wrapper">
        <div class="main-layout">
            <canvas id="game"></canvas>

            <div class="side-panel">
                <div class="stats-group">
                    <div class="stat-card">
                        <div class="stat-label">Score</div>
                        <div id="score" class="stat-value">0</div>
                    </div>
                    <div class="stat-card">
                        <div class="stat-label">Level</div>
                        <div id="level" class="stat-value">1</div>
                    </div>
                </div>

                <div class="system-btns">
                    <button class="sys-btn" id="pauseBtn">Pause</button>
```

```
            </div>
        </div>
    </div>

    <!-- Touch Controls -->
    <div class="controls-grid">
        <button class="control-btn btn-rotate" id="rotateBtn">⟳</button>
        <button class="control-btn" id="upBtn">↑</button>
        <button class="control-btn btn-drop" id="hardDropBtn">↓</button>

        <button class="control-btn" id="leftBtn">←</button>
        <button class="control-btn" id="downBtn">↓</button>
        <button class="control-btn" id="rightBtn">→</button>
    </div>
</div>

<div id="overlay">
    <h1>GAME OVER</h1>
    <p id="finalScore" style="font-size: 1.5rem; margin: 20px 0;">Score: 0</p>
    <button class="sys-btn" style="padding: 15px 40px; background: var(--accent-color);
color: #000;" onclick="resetGame()">PLAY AGAIN</button>
</div>

<script>
    const canvas = document.getElementById('game');
    const ctx = canvas.getContext('2d');
    const scoreElement = document.getElementById('score');
    const levelElement = document.getElementById('level');
    const overlay = document.getElementById('overlay');

    const COLS = 10;
    const ROWS = 20;
    const SQ = 30; // Base size, adjusted by canvas attributes

    canvas.width = COLS * SQ;
    canvas.height = ROWS * SQ;

    const PIECES = [
        [[], [[1,1,1,1]], [], []], // I
        [[2,0,0], [2,2,2], [0,0,0]], // J
        [[0,0,3], [3,3,3], [0,0,0]], // L
        [[4,4], [4,4]], // O
        [[0,5,5], [5,5,0], [0,0,0]], // S
        [[0,6,0], [6,6,6], [0,0,0]], // T
        [[7,7,0], [0,7,7], [0,0,0]]  // Z
    ];
```

```
    const COLORS = [null, '#00f0f0', '#0000f0', '#f0a000', '#f0f000', '#00f000', '#a000f0',
'#f00000'];

    let board = Array.from({length: ROWS}, () => Array(COLS).fill(0));
    let score = 0;
    let level = 1;
    let gameOver = false;
    let paused = false;
    let dropCounter = 0;
    let dropInterval = 1000;
    let lastTime = 0;

    const player = {
        pos: {x: 0, y: 0},
        matrix: null
    };

    function resetPlayer() {
        const id = Math.floor(Math.random() * PIECES.length);
        player.matrix = PIECES[id];
        player.pos.y = 0;
        player.pos.x = Math.floor(COLS/2) - Math.floor(player.matrix[0].length/2);

        if (collide()) {
            gameOver = true;
            document.getElementById('finalScore').innerText = `Score: ${score}`;
            overlay.style.display = 'flex';
        }
    }

    function collide() {
        for (let y = 0; y < player.matrix.length; ++y) {
            for (let x = 0; x < player.matrix[y].length; ++x) {
                if (player.matrix[y][x] !== 0 &&
                    (board[y + player.pos.y] && board[y + player.pos.y][x + player.pos.x]) !== 0) {
                     return true;
                }
            }
        }
        return false;
    }

    function merge() {
        player.matrix.forEach((row, y) => {
            row.forEach((value, x) => {
                if (value !== 0) board[y + player.pos.y][x + player.pos.x] = value;
            });
        });
```

```
}

function rotate(matrix) {
    const m = matrix.map((_, i) => matrix.map(row => row[i]).reverse());
    return m;
}

function playerRotate() {
    const pos = player.pos.x;
    let offset = 1;
    const oldMatrix = player.matrix;
    player.matrix = rotate(player.matrix);
    while (collide()) {
        player.pos.x += offset;
        offset = -(offset + (offset > 0 ? 1 : -1));
        if (offset > player.matrix[0].length) {
            player.matrix = oldMatrix;
            player.pos.x = pos;
            return;
        }
    }
}

function clearLines() {
    let rowCount = 0;
    for (let y = ROWS - 1; y >= 0; y--) {
        if (board[y].every(value => value !== 0)) {
            board.splice(y, 1);
            board.unshift(Array(COLS).fill(0));
            rowCount++;
            y++;
        }
    }
    if (rowCount > 0) {
        score += [0, 10, 30, 50, 100][rowCount] * level;
        scoreElement.innerText = score;
        if (score >= level * 100) {
            level++;
            levelElement.innerText = level;
            dropInterval = Math.max(100, 1000 - (level * 80));
        }
    }
}

function drop() {
    player.pos.y++;
    if (collide()) {
        player.pos.y--;
```

```
        merge();
        clearLines();
        resetPlayer();
    }
    dropCounter = 0;
}

function hardDrop() {
    while(!collide()) { player.pos.y++; }
    player.pos.y--;
    merge();
    clearLines();
    resetPlayer();
    dropCounter = 0;
}

function move(dir) {
    player.pos.x += dir;
    if (collide()) player.pos.x -= dir;
}

function draw() {
    ctx.fillStyle = '#000';
    ctx.fillRect(0, 0, canvas.width, canvas.height);

    // Board
    board.forEach((row, y) => {
        row.forEach((value, x) => {
            if (value) drawSquare(x, y, COLORS[value]);
        });
    });

    // Player
    player.matrix.forEach((row, y) => {
        row.forEach((value, x) => {
            if (value) drawSquare(player.pos.x + x, player.pos.y + y, COLORS[value]);
        });
    });
}

function drawSquare(x, y, color) {
    ctx.fillStyle = color;
    ctx.fillRect(x * SQ, y * SQ, SQ, SQ);
    ctx.strokeStyle = 'rgba(255,255,255,0.1)';
    ctx.strokeRect(x * SQ, y * SQ, SQ, SQ);
}

function update(time = 0) {
```

```
            if (!paused && !gameOver) {
                const deltaTime = time - lastTime;
                lastTime = time;
                dropCounter += deltaTime;
                if (dropCounter > dropInterval) drop();
                draw();
            }
            requestAnimationFrame(update);
        }

        function resetGame() {
            board = Array.from({length: ROWS}, () => Array(COLS).fill(0));
            score = 0;
            level = 1;
            gameOver = false;
            scoreElement.innerText = '0';
            levelElement.innerText = '1';
            overlay.style.display = 'none';
            resetPlayer();
        }

        // Input Handling
        window.addEventListener('keydown', e => {
            if (gameOver || paused) return;
            if (e.key === 'ArrowLeft') move(-1);
            if (e.key === 'ArrowRight') move(1);
            if (e.key === 'ArrowDown') drop();
            if (e.key === 'ArrowUp') playerRotate();
            if (e.key === ' ') hardDrop();
        });

        // Button Listeners
        document.getElementById('leftBtn').onclick = () => move(-1);
        document.getElementById('rightBtn').onclick = () => move(1);
        document.getElementById('downBtn').onclick = () => drop();
        document.getElementById('upBtn').onclick = () => drop(); // Logic for fast drop
        document.getElementById('rotateBtn').onclick = () => playerRotate();
        document.getElementById('hardDropBtn').onclick = () => hardDrop();
        document.getElementById('pauseBtn').onclick = (e) => {
            paused = !paused;
            e.target.innerText = paused ? 'Resume' : 'Pause';
        };

        resetPlayer();
        update();
    </script>
</body>
</html>
```