Babbar education.index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Babbar Education Snake & Ladder</title>
    <!-- Load Tailwind CSS -->
    <script src="https://cdn.tailwindcss.com"></script>
    <!-- Load Lucide Icons -->
    <script type="module">
        import { createIcons, Zap, User, AlertTriangle, GraduationCap, Trophy, RotateCcw } from
'https://unpkg.com/lucide@latest';
        window.lucide = { Zap, User, AlertTriangle, GraduationCap, Trophy, RotateCcw };
        window.addEventListener('load', () => createIcons());
    </script>
    <style>
        /* Custom CSS for tokens and dice animation */
        body { font-family: 'Inter', sans-serif; }
        #game-board {
            /* 100% width of the max-w-md container */
            width: 100%;
            padding-bottom: 100%; /* Aspect Ratio Hack: 1:1 */
            position: relative;
        }

        .token {
            position: absolute;
            width: 24px;
            height: 24px;
            border-radius: 9999px;
            border: 2px solid white;
            box-shadow: 0 4px 6px rgba(0, 0, 0, 0.2);
            display: flex;
            align-items: center;
            justify-content: center;
            transition: all 0.7s cubic-bezier(0.68, -0.55, 0.27, 1.55); /* Bounce/spring effect */
            z-index: 20;
        }

        @keyframes dice-roll {
            0% { transform: rotate(0deg); }
            100% { transform: rotate(360deg); }
        }

        .rolling {
            animation: dice-roll 0.05s infinite linear;
        }
    </style>
</head>
<body class="min-h-screen bg-slate-900 flex flex-col items-center justify-center p-4 text-slate-100">
```

```
    <!-- Header -->
    <header class="mb-4 text-center">
        <h1 class="text-3xl sm:text-4xl font-extrabold text-transparent bg-clip-text bg-gradient-to-r
from-orange-400 to-amber-300">
            BABBAR EDUCATION
        </h1>
        <p class="text-slate-400 text-sm font-medium tracking-wide mt-1">SNAKE & LADDER
CHALLENGE</p>
    </header>

    <!-- Main Game Container -->
    <div class="flex flex-col lg:flex-row gap-6 items-center w-full max-w-5xl">

        <!-- Game Board Area -->
        <div id="board-container" class="relative w-full max-w-md aspect-square bg-white rounded-lg
shadow-2xl overflow-hidden border-4 border-slate-700">
            <div id="game-board" class="absolute inset-0 grid grid-cols-10 grid-rows-10">
                <!-- Board Cells rendered here by JS -->
            </div>

            <!-- SVG Overlay for Snakes and Ladders -->
            <svg id="connectors-svg" class="absolute top-0 left-0 w-full h-full pointer-events-none z-10
opacity-80"></svg>

            <!-- Player Tokens rendered here by JS -->
        </div>

        <!-- Controls Panel -->
        <div class="flex-1 w-full max-w-md flex flex-col gap-4">

            <!-- Status Card -->
            <div class="bg-slate-800 p-6 rounded-xl border border-slate-700 shadow-lg">
                <h2 class="text-xl font-bold text-white mb-4 flex items-center gap-2">
                    <i data-lucide="zap" class="text-yellow-400"></i> Game Status
                </h2>

                <div class="bg-slate-900/50 p-3 rounded-lg border border-slate-700 mb-4 h-16 flex items-
center justify-center text-center">
                    <p id="game-log" class="text-sm text-slate-300">Welcome to Babbar Education Game!
Player 1's turn.</p>
                </div>

                <!-- Players Info -->
                <div class="grid grid-cols-2 gap-3 mb-6">
                    <div id="p1-info" class="p-3 rounded-lg border-2 border-blue-500 bg-blue-500/10
transition-colors">
                        <div class="flex items-center gap-2 mb-1">
                            <div class="w-3 h-3 rounded-full bg-blue-500"></div>
                            <span class="font-bold text-blue-100">Player 1</span>
                        </div>
                        <div id="p1-pos" class="text-2xl font-mono text-white">1</div>
                    </div>

                    <div id="p2-info" class="p-3 rounded-lg border-2 border-transparent bg-slate-700/50
transition-colors">
                        <div class="flex items-center gap-2 mb-1">
                            <div class="w-3 h-3 rounded-full bg-orange-500"></div>
                            <span class="font-bold text-orange-100">Player 2</span>
```

```
                </div>
                <div id="p2-pos" class="text-2xl font-mono text-white">1</div>
              </div>
            </div>

            <!-- Dice Control -->
            <div class="flex flex-col items-center justify-center py-4 bg-slate-700/30 rounded-xl border
border-dashed border-slate-600">
              <div class="mb-4">
                <div id="dice-display" class="w-16 h-16 bg-white rounded-xl flex items-center justify-
center shadow-inner text-4xl font-bold text-slate-800">
                  1
                </div>
              </div>

              <button
                id="roll-dice-btn"
                class="w-full max-w-[200px] py-3 rounded-lg font-bold text-lg shadow-lg transition-all
bg-blue-600 hover:bg-blue-500 shadow-blue-900/50 hover:-translate-y-1 active:translate-y-0"
              >
                Roll Dice (P1)
              </button>
            </div>
          </div>

          <!-- Legend -->
          <div class="bg-slate-800 p-4 rounded-xl border border-slate-700 text-sm">
            <h3 class="text-slate-400 font-bold mb-2 uppercase text-xs tracking-wider">Legend</h3>
            <div class="flex justify-around">
              <div class="flex items-center gap-2">
                <div class="w-4 h-1 bg-red-500 rounded"></div>
                <span>Snake (Down)</span>
              </div>
              <div class="flex items-center gap-2">
                <div class="w-4 h-1 bg-green-500 border-t border-b border-transparent border-dashed">
</div>
                <span>Ladder (Up)</span>
              </div>
            </div>
          </div>
        </div>

      </div>

      <div class="mt-8 text-slate-500 text-xs">
        © 2025 Babbar Education. All Rights Reserved.
      </div>

      <!-- Winner Modal Overlay (Hidden by default) -->
      <div id="winner-overlay" class="hidden fixed inset-0 z-50 bg-black/80 flex flex-col items-center
justify-center animate-in fade-in">
        <i id="winner-trophy" data-lucide="trophy" class="text-yellow-400 mb-4 animate-bounce"
style="width: 64px; height: 64px;"></i>
        <h2 id="winner-text" class="text-3xl font-bold text-white mb-2"></h2>
        <p class="text-slate-300 mb-6">Congratulations on completing the course!</p>
        <button
          id="reset-game-btn-modal"
          class="px-6 py-3 bg-gradient-to-r from-orange-500 to-red-500 text-white rounded-full font-bold
shadow-lg hover:scale-105 transition-transform flex items-center gap-2"
```

```
        >
          <i data-lucide="rotate-ccw" style="width: 18px; height: 18px;"></i> Play Again
        </button>
    </div>

    <script>
        document.addEventListener('DOMContentLoaded', () => {
            // --- DOM Elements ---
            const boardContainer = document.getElementById('board-container');
            const boardEl = document.getElementById('game-board');
            const svgEl = document.getElementById('connectors-svg');
            const diceDisplay = document.getElementById('dice-display');
            const rollDiceBtn = document.getElementById('roll-dice-btn');
            const gameLogEl = document.getElementById('game-log');
            const p1InfoEl = document.getElementById('p1-info');
            const p2InfoEl = document.getElementById('p2-info');
            const p1PosEl = document.getElementById('p1-pos');
            const p2PosEl = document.getElementById('p2-pos');
            const winnerOverlay = document.getElementById('winner-overlay');
            const winnerText = document.getElementById('winner-text');
            const resetGameBtnModal = document.getElementById('reset-game-btn-modal');

            // --- Game Constants & Data ---
            const SNAKES = { 16: 6, 46: 25, 49: 11, 62: 19, 64: 60, 74: 53, 89: 68, 92: 88, 95: 75, 99: 80 };
            const LADDERS = { 2: 38, 7: 14, 8: 31, 15: 26, 21: 42, 28: 84, 36: 44, 51: 67, 71: 91, 78: 98, 87:
94 };

            // --- Game State ---
            let positions = { p1: 1, p2: 1 };
            let currentPlayer = 'p1';
            let diceValue = 1;
            let isRolling = false;
            let winner = null;

            // --- Utility Functions ---

            // Helper to get x,y coordinates (0-100%) for a board number (1-100)
            const getCoords = (number) => {
                if (number < 1) number = 1;
                if (number > 100) number = 100;

                const zeroIndex = number - 1;
                const rowFromBottom = Math.floor(zeroIndex / 10);
                const colIndex = zeroIndex % 10;

                // Zig-zag logic: Even rows go Left->Right (0, 2, 4...), Odd rows go Right->Left (1, 3, 5...)
                const isEvenRow = rowFromBottom % 2 === 0;

                const actualCol = isEvenRow ? colIndex : 9 - colIndex;
                const visualRow = 9 - rowFromBottom; // 0 is top in CSS, so invert

                return {
                    x: actualCol * 10 + 5, // Center of cell (+5%)
                    y: visualRow * 10 + 5
                };
            };

            const getPlayerName = (p) => p === 'p1' ? 'Player 1' : 'Player 2';
            const getPlayerColor = (p) => p === 'p1' ? '#3b82f6' : '#f97316';
```

```
        const getPlayerOffset = (p) => p === 'p1' ? -3 : 3;

        // --- UI Rendering Functions ---

        const renderCells = () => {
            let html = '';
            // Render row by row, from top (90-99) to bottom (0-9)
            for (let row = 9; row >= 0; row--) {
                const isEvenRow = row % 2 === 0;
                for (let col = 0; col < 10; col++) {
                    let number;
                    if (isEvenRow) {
                        // Left to right (e.g., Row 9: 91, 92...)
                        number = row * 10 + 1 + col;
                    } else {
                        // Right to left (e.g., Row 8: 90, 89...)
                        number = row * 10 + 10 - col;
                    }

                    const isDark = (row + col) % 2 === 1;
                    let cellClasses = `
                        relative w-full h-full flex items-center justify-center border-[0.5px] border-white/10
text-[9px] sm:text-xs font-bold
                        ${isDark ? 'bg-indigo-50/50 text-indigo-900' : 'bg-white/80 text-indigo-900'}
                        ${number === 100 ? 'bg-yellow-200 text-yellow-800' : ''}
                        ${number === 1 ? 'bg-green-200 text-green-800' : ''}
                    `;

                    // Add content (number and icons)
                    let content = `
                        <span class="absolute top-0.5 left-1 opacity-50">${number}</span>
                    `;

                    if (SNAKES[number]) {
                         // Lucide icon implementation for HTML
                        content += `<i data-lucide="alert-triangle" class="text-red-500 opacity-80"
style="width: 10px; height: 10px;"></i>`;
                    }
                    if (Object.values(SNAKES).includes(number)) {
                        content += `<div class="text-red-800 opacity-50 text-[8px] absolute bottom-0 right-
1">Tail</div>`;
                    }
                    if (LADDERS[number]) {
                        content += `<i data-lucide="graduation-cap" class="text-green-600 opacity-80"
style="width: 12px; height: 12px;"></i>`;
                    }
                    if (Object.values(LADDERS).includes(number)) {
                        content += `<div class="text-green-800 opacity-50 text-[8px] absolute bottom-0
right-1">Top</div>`;
                    }
                    if (number === 100) {
                        content += `<i data-lucide="trophy" class="text-yellow-600" style="width: 16px;
height: 16px;"></i>`;
                    }

                    html += `<div class="${cellClasses}">${content}</div>`;
                }
            }
            boardEl.innerHTML = html;
```

```
            // Re-initialize Lucide icons after DOM update
            window.lucide && window.lucide.createIcons();
        };

        const renderTokens = () => {
            // Ensure tokens exist, or create them
            let p1Token = document.getElementById('p1-token');
            let p2Token = document.getElementById('p2-token');

            if (!p1Token) {
                p1Token = document.createElement('div');
                p1Token.id = 'p1-token';
                p1Token.className = 'token bg-blue-500';
                p1Token.innerHTML = '<i data-lucide="user" class="text-white" style="width: 12px; height:
12px;"></i>';
                boardContainer.appendChild(p1Token);
            }
            if (!p2Token) {
                p2Token = document.createElement('div');
                p2Token.id = 'p2-token';
                p2Token.className = 'token bg-orange-500';
                p2Token.innerHTML = '<i data-lucide="user" class="text-white" style="width: 12px; height:
12px;"></i>';
                boardContainer.appendChild(p2Token);
            }

            // Update positions
            const p1Coords = getCoords(positions.p1);
            const p2Coords = getCoords(positions.p2);

            const p1Offset = getPlayerOffset('p1');
            const p2Offset = getPlayerOffset('p2');

            // Apply CSS transforms for smooth movement
            p1Token.style.left = `calc(${p1Coords.x}% - 12px + ${p1Offset}px)`;
            p1Token.style.top = `calc(${p1Coords.y}% - 12px + ${p1Offset}px)`;

            p2Token.style.left = `calc(${p2Coords.x}% - 12px + ${p2Offset}px)`;
            p2Token.style.top = `calc(${p2Coords.y}% - 12px + ${p2Offset}px)`;

            // Re-initialize Lucide icons after DOM update
            window.lucide && window.lucide.createIcons();
        };

        const renderConnectors = () => {
            let svgContent = '';

            // SVG Definitions for markers/arrows (optional, but good practice)
            svgContent += `<defs>
              <marker id="arrowhead-snake" markerWidth="6" markerHeight="4" refX="5" refY="2"
orient="auto"><polygon points="0 0, 6 2, 0 4" fill="#ef4444" /></marker>
                <marker id="arrowhead-ladder" markerWidth="6" markerHeight="4" refX="5" refY="2"
orient="auto"><polygon points="0 0, 6 2, 0 4" fill="#22c55e" /></marker>
            </defs>`;

            // Snakes (Red)
            Object.entries(SNAKES).forEach(([start, end]) => {
                const startCoords = getCoords(parseInt(start));
                const endCoords = getCoords(parseInt(end));
```

```
            svgContent += `
                <line
                    x1="${startCoords.x}%" y1="${startCoords.y}%"
                    x2="${endCoords.x}%" y2="${endCoords.y}%"
                    stroke="#ef4444"
                    stroke-width="4"
                    stroke-linecap="round"
                    class="drop-shadow-sm opacity-70"
                />
            `;
        });

        // Ladders (Green)
        Object.entries(LADDERS).forEach(([start, end]) => {
            const startCoords = getCoords(parseInt(start));
            const endCoords = getCoords(parseInt(end));
            svgContent += `
                <line
                    x1="${startCoords.x}%" y1="${startCoords.y}%"
                    x2="${endCoords.x}%" y2="${endCoords.y}%"
                    stroke="#22c55e"
                    stroke-width="5"
                    stroke-dasharray="4,2"
                    stroke-linecap="round"
                    class="drop-shadow-sm opacity-70"
                />
            `;
        });

        svgEl.innerHTML = svgContent;
    };

    const updateUI = () => {
        // Update Log
        gameLogEl.textContent = winner
            ? `🎉 ${getPlayerName(winner)} Wins!`
            : `${getPlayerName(currentPlayer)}'s turn. Roll the dice!`;

        // Update Positions Display
        p1PosEl.textContent = positions.p1;
        p2PosEl.textContent = positions.p2;

        // Update Dice Display
        diceDisplay.textContent = diceValue;
        diceDisplay.classList.toggle('rolling', isRolling);

        // Update Turn Indicators
        if (currentPlayer === 'p1') {
            p1InfoEl.className = p1InfoEl.className.replace('border-transparent bg-slate-700/50',
'border-blue-500 bg-blue-500/10');
            p2InfoEl.className = p2InfoEl.className.replace('border-orange-500 bg-orange-
500/10', 'border-transparent bg-slate-700/50');
            rollDiceBtn.className = rollDiceBtn.className.replace(/bg-orange-600.*|shadow-
orange-900.*|bg-blue-600.*|shadow-blue-900.*/g, 'bg-blue-600 hover:bg-blue-500 shadow-blue-
900/50');
            rollDiceBtn.textContent = isRolling ? 'Rolling...' : 'Roll Dice (P1)';
        } else {
            p1InfoEl.className = p1InfoEl.className.replace('border-blue-500 bg-blue-500/10',
'border-transparent bg-slate-700/50');
```

```
            p2InfoEl.className = p2InfoEl.className.replace('border-transparent bg-slate-700/50',
'border-orange-500 bg-orange-500/10');
            rollDiceBtn.className = rollDiceBtn.className.replace(/bg-blue-600.*|shadow-blue-
900.*|bg-orange-600.*|shadow-orange-900.*/g, 'bg-orange-600 hover:bg-orange-500 shadow-
orange-900/50');
            rollDiceBtn.textContent = isRolling ? 'Rolling...' : 'Roll Dice (P2)';
        }

        // Disable button if rolling or winner
        rollDiceBtn.disabled = !!winner || isRolling;
        rollDiceBtn.classList.toggle('disabled:opacity-50', !!winner || isRolling);

        // Render tokens on the board
        renderTokens();

        // Show winner overlay if game over
        if (winner) {
            winnerText.textContent = `${getPlayerName(winner)} Wins!`;
            winnerOverlay.classList.remove('hidden');
        } else {
            winnerOverlay.classList.add('hidden');
        }
    };

    // --- Game Logic ---

    const handleMove = (roll) => {
        isRolling = false;

        setTimeout(() => {
            let currentPos = positions[currentPlayer];
            let nextPos = currentPos + roll;
            let logMsg = `${getPlayerName(currentPlayer)} rolled a ${roll}.`;

            // Bounce back rule
            if (nextPos > 100) {
                const diff = nextPos - 100;
                nextPos = 100 - diff;
            }

            // 1. Move to initial dice roll position
            positions[currentPlayer] = nextPos;
            updateUI(); // Immediate UI update after the roll

            // Check for win immediately if 100
            if (nextPos === 100) {
                winner = currentPlayer;
                updateUI();
                return;
            }

            // 2. Check for Snakes or Ladders after a brief pause
            setTimeout(() => {
                let finalPos = nextPos;

                if (SNAKES[nextPos]) {
                    finalPos = SNAKES[nextPos];
                    logMsg += " Oh no! Bitten by a snake! 🐍";
                } else if (LADDERS[nextPos]) {
```

```
            finalPos = LADDERS[nextPos];
            logMsg += " Yay! Climbed a ladder! 🎓";
        }

        if (finalPos !== nextPos) {
            positions[currentPlayer] = finalPos;
            // Update tokens again for the slide movement
            renderTokens();
        }

        gameLogEl.textContent = logMsg;

        // Check win again after slide
        if (finalPos === 100) {
            winner = currentPlayer;
        } else {
            // Switch turn
            currentPlayer = currentPlayer === 'p1' ? 'p2' : 'p1';
        }

        updateUI();

    }, 700); // Delay for player to process the move before sliding
  }, 400); // Delay for dice roll animation to finish
};

const rollDice = () => {
  if (winner || isRolling) return;

  isRolling = true;
  let rolls = 10;
  let count = 0;

  const interval = setInterval(() => {
    diceValue = Math.ceil(Math.random() * 6);
    diceDisplay.textContent = diceValue;
    count++;
    if (count >= rolls) {
      clearInterval(interval);
      const finalValue = Math.ceil(Math.random() * 6);
      diceValue = finalValue;
      handleMove(finalValue);
    }
  }, 50);

  updateUI();
};

const resetGame = () => {
  positions = { p1: 1, p2: 1 };
  currentPlayer = 'p1';
  winner = null;
  diceValue = 1;
  gameLogEl.textContent = "New Game Started. Player 1's turn.";
  updateUI();
};

// --- Event Listeners ---
rollDiceBtn.addEventListener('click', rollDice);
```

```
        resetGameBtnModal.addEventListener('click', resetGame);

        // --- Initialization ---
        const initGame = () => {
            renderCells();
            renderConnectors();
            resetGame(); // Initializes positions and UI
        };

        initGame();
    });
  </script>
</body>
</html>
```