```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Neon House Casino – £100 Balance + Vault + Settings</title>
  <style>
    body {
      margin: 0;
      font-family: Arial, sans-serif;
      background: radial-gradient(circle at top, #222 0, #000 60%);
      color: #fff;
    }

    header {
      background: #111;
      padding: 10px 20px;
      display: flex;
      justify-content: space-between;
      align-items: center;
      border-bottom: 2px solid #0ff;
    }

    header h1 {
      margin: 0;
      font-size: 22px;
    }

    #balanceDisplay {
      font-size: 16px;
    }

    #layout {
      display: flex;
      height: calc(100vh - 54px);
    }

    #sidebar {
      width: 230px;
      background: #111;
      border-right: 2px solid #0ff;
      padding: 15px;
      box-sizing: border-box;
      overflow-y: auto;
    }
```

```
#sidebar h2 {
  font-size: 16px;
  margin-top: 0;
  margin-bottom: 10px;
}

.nav-btn {
  width: 100%;
  padding: 8px 10px;
  margin-bottom: 6px;
  background: #222;
  border: 1px solid #444;
  color: #fff;
  cursor: pointer;
  text-align: left;
  font-size: 14px;
}

.nav-btn.active {
  border-color: #0ff;
  background: #033;
}

#main {
  flex: 1;
  padding: 20px;
  box-sizing: border-box;
  overflow-y: auto;
}

h2 {
  margin-top: 0;
}

.game-card-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(180px, 1fr));
  gap: 15px;
}

.game-card {
  background: #111;
  border: 2px solid #333;
  border-radius: 8px;
  padding: 15px;
  cursor: pointer;
  transition: transform 0.15s, border-color 0.15s;
  text-align: center;
}

.game-card:hover {
  transform: translateY(-3px);
  border-color: #0ff;
}

.game-title {
  font-size: 18px;
  margin-bottom: 8px;
}

.game-desc {
  font-size: 13px;
  color: #ccc;
}

.game-panel {
  display: none;
  max-width: 700px;
}

.game-panel.active {
  display: block;
}

.field-row {
  margin-bottom: 10px;
}

input[type="number"] {
  padding: 4px 6px;
  width: 90px;
}

select {
  padding: 4px 6px;
}
```

```
button.action {
  padding: 8px 14px;
  margin-top: 5px;
  cursor: pointer;
  border: none;
  background: #0f0;
  color: #000;
  font-weight: bold;
}

.result {
  margin-top: 10px;
  min-height: 22px;
  font-size: 14px;
}

.inline-label {
  display: inline-block;
  width: 160px;
}

.mines-grid {
  display: grid;
  grid-template-columns: repeat(5, 40px);
  grid-auto-rows: 40px;
  gap: 5px;
  margin-top: 10px;
}

.mine-cell {
  background: #222;
  border: 1px solid #444;
  cursor: pointer;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 18px;
  user-select: none;
}

.mine-cell.revealed-safe {
  background: #064;
}
```

```
.mine-cell.revealed-mine {
  background: #600;
}

.tag {
  font-size: 11px;
  background: #222;
  border-radius: 4px;
  padding: 2px 5px;
  border: 1px solid #555;
  display: inline-block;
  margin-top: 4px;
  color: #ccc;
}

.limbo-multiplier {
  font-size: 26px;
  margin: 8px 0;
}

/* Slots */
.slots-reels {
  display: flex;
  gap: 8px;
  margin: 10px 0;
  justify-content: center;
}

.slot-reel {
  width: 60px;
  height: 60px;
  background: #111;
  border: 2px solid #444;
  border-radius: 6px;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 30px;
}

/* Video Poker */
.vp-cards {
  display: flex;
  gap: 10px;
  margin: 15px 0;
}

.vp-card {
  width: 70px;
  height: 100px;
  border-radius: 8px;
  border: 2px solid #fff;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background: #022;
  cursor: pointer;
  user-select: none;
}

.vp-card.held {
  border-color: #ff0;
  box-shadow: 0 0 10px #ff0;
}

.vp-rank {
  font-size: 20px;
  margin-bottom: 4px;
}

.vp-suit {
  font-size: 18px;
}

/* Plinko */
.plinko-board {
  margin: 15px 0;
}

/* Hi-Lo */
.card-display {
  width: 80px;
  height: 110px;
  border-radius: 8px;
  border: 2px solid #fff;
  display: flex;
  flex-direction: column;
```

```css
  justify-content: center;
  align-items: center;
  background: #022;
  margin: 10px 0;
}

/* Keno */
.keno-grid {
  display: grid;
  grid-template-columns: repeat(8, 40px);
  grid-auto-rows: 40px;
  gap: 5px;
  margin: 10px 0;
}

.keno-cell {
  background: #222;
  border: 1px solid #444;
  cursor: pointer;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 14px;
  user-select: none;
}

.keno-cell.selected {
  background: #064;
}

.keno-cell.hit {
  background: #0a0;
}

.keno-cell.missed {
  background: #600;
}

/* Settings */
#settings .stat-row {
  margin: 4px 0;
}

#house-edge {
      width: 200px;
      vertical-align: middle;
    }
  </style>
</head>
<body>
<header>
  <h1>Neon House Casino</h1>
  <div id="balanceDisplay">Balance: £100.00 | Vault: £0.00</div>
</header>

<div id="layout">
  <aside id="sidebar">
    <h2>Games</h2>
    <button class="nav-btn active" data-view="home">🏠 Home</button>
    <button class="nav-btn" data-view="vault">🏦 Vault</button>
    <button class="nav-btn" data-view="settings">⚙️ Settings</button>
    <button class="nav-btn" data-view="coinflip">🪙 Coin Flip</button>
    <button class="nav-btn" data-view="dice">🎲 Dice</button>
    <button class="nav-btn" data-view="mines">💣 Mines</button>
    <button class="nav-btn" data-view="limbo">📈 Limbo</button>
    <button class="nav-btn" data-view="roulette">🎡 Roulette</button>
    <button class="nav-btn" data-view="slots">🎰 Slots</button>
    <button class="nav-btn" data-view="videopoker">🃏 Video Poker</button>
    <button class="nav-btn" data-view="plinko">🧱 Plinko</button>
    <button class="nav-btn" data-view="hilo">🟦 Hi-Lo</button>
    <button class="nav-btn" data-view="keno">🧨 Keno</button>
  </aside>

  <main id="main">
    <!-- HOME VIEW -->
    <section id="home" class="game-panel active">
      <h2>Casino Lobby</h2>
      <p>Select a game to play. All games share the same balance in British pounds (£).</p>
      <div class="game-card-grid">
        <div class="game-card" data-open="vault">
          <div class="game-title">🏦 Vault</div>
          <div class="game-desc">Store money safely. Deposit and withdraw anytime.</div>
        </div>
        <div class="game-card" data-open="settings">
          <div class="game-title">⚙️ Settings</div>
          <div class="game-desc">Reset progress, view stats, tune house edge.</div>
        </div>
        <div class="game-card" data-open="coinflip">
      <div class="game-title">🪙 Coin Flip</div>
      <div class="game-desc">Bet on Heads or Tails. Double or nothing.</div>
    </div>
    <div class="game-card" data-open="dice">
      <div class="game-title">🎲 Dice</div>
      <div class="game-desc">Roll 2 dice. Win if total is 8 or more.</div>
    </div>
    <div class="game-card" data-open="mines">
      <div class="game-title">💣 Mines</div>
      <div class="game-desc">Click safe tiles, avoid mines, cash out early.</div>
    </div>
    <div class="game-card" data-open="limbo">
      <div class="game-title">📈 Limbo</div>
      <div class="game-desc">Pick a multiplier. Survive the crash to win.</div>
    </div>
    <div class="game-card" data-open="roulette">
      <div class="game-title">🎡 Roulette</div>
      <div class="game-desc">Bet on Red or Black. European wheel (0–36).</div>
    </div>
    <div class="game-card" data-open="slots">
      <div class="game-title">🎰 Slots</div>
      <div class="game-desc">3-reel slot. Line up symbols to win.</div>
    </div>
    <div class="game-card" data-open="videopoker">
      <div class="game-title">🃏 Video Poker</div>
      <div class="game-desc">Jacks or Better. Hold and draw.</div>
    </div>
    <div class="game-card" data-open="plinko">
      <div class="game-title">🧱 Plinko</div>
      <div class="game-desc">Drop a ball, hit a multiplier.</div>
    </div>
    <div class="game-card" data-open="hilo">
      <div class="game-title">🟦 Hi-Lo</div>
      <div class="game-desc">Guess if next card is higher or lower.</div>
    </div>
    <div class="game-card" data-open="keno">
      <div class="game-title">🧨 Keno</div>
      <div class="game-desc">Pick numbers, hope they hit.</div>
    </div>
  </div>
</section>

<!-- VAULT -->
<section id="vault" class="game-panel">
```

```
      <h2>🏦 Vault</h2>
      <p>Move money between your playable balance and your vault. Vault funds are safe from
bets and persist after refresh.</p>
      <div class="field-row">
        <span class="inline-label">Current Balance:</span>
        <span id="vault-balanceDisplay">£0.00</span>
      </div>
      <div class="field-row">
        <span class="inline-label">Current Vault:</span>
        <span id="vault-vaultDisplay">£0.00</span>
      </div>
      <hr>
      <div class="field-row">
        <span class="inline-label">Deposit to Vault (£):</span>
        <input type="number" id="vault-deposit" value="10" min="1">
        <button class="action" id="vault-depositBtn">Deposit</button>
      </div>
      <div class="field-row">
        <span class="inline-label">Withdraw from Vault (£):</span>
        <input type="number" id="vault-withdraw" value="10" min="1">
        <button class="action" id="vault-withdrawBtn">Withdraw</button>
      </div>
      <div class="result" id="vault-result"></div>
    </section>

    <!-- SETTINGS -->
    <section id="settings" class="game-panel">
      <h2>⚙️ Settings & Stats</h2>

      <h3>House Edge</h3>
      <p>House edge reduces payouts on all wins. 0% = fair, higher values = more rigged in the
casino’s favour.</p>
      <div class="field-row">
        <span class="inline-label">House Edge:</span>
        <input type="range" id="house-edge" min="0" max="20" step="0.5">
        <span id="house-edge-label">0.0%</span>
      </div>

      <hr>

      <h3>Session Stats</h3>
      <div class="stat-row">Total Bet: £<span id="stat-totalBet">0.00</span></div>
      <div class="stat-row">Total Won (profit): £<span id="stat-totalWon">0.00</span></div>
      <div class="stat-row">Total Lost: £<span id="stat-totalLost">0.00</span></div>
```

```
  <div class="stat-row">Net Profit: £<span id="stat-net">0.00</span></div>
  <div class="stat-row">ROI: <span id="stat-roi">0.00</span>%</div>

  <hr>

  <h3>Progress</h3>
  <p>This clears your balance, vault, and stats, and resets everything to defaults.</p>
  <button class="action" id="reset-progress">Reset All Progress</button>
  <div class="result" id="settings-result"></div>
</section>

<!-- COIN FLIP -->
<section id="coinflip" class="game-panel">
  <h2>🪙 Coin Flip</h2>
  <p>Bet on Heads or Tails. If you win, you get 2x your bet (reduced by house edge).</p>
  <div class="field-row">
    <span class="inline-label">Bet (£):</span>
    <input type="number" id="cf-bet" value="1" min="1">
  </div>
  <div class="field-row">
    <span class="inline-label">Choice:</span>
    <select id="cf-choice">
      <option value="Heads">Heads</option>
      <option value="Tails">Tails</option>
    </select>
  </div>
  <button class="action" id="cf-play">Flip Coin</button>
  <div class="result" id="cf-result"></div>
</section>

<!-- DICE -->
<section id="dice" class="game-panel">
  <h2>🎲 Dice</h2>
  <p>Two dice roll. You win if total is 8 or higher. Payout: 2x bet (reduced by house edge).</p>
  <div class="field-row">
    <span class="inline-label">Bet (£):</span>
    <input type="number" id="d-bet" value="1" min="1">
  </div>
  <button class="action" id="d-play">Roll Dice</button>
  <div class="result" id="d-result"></div>
</section>

<!-- MINES -->
<section id="mines" class="game-panel">
```

```
    <h2>💣 Mines</h2>
    <p>Set your bet and number of mines. Click tiles to reveal safe spots. Cash out before you
hit a mine.</p>
    <div class="field-row">
      <span class="inline-label">Bet (£):</span>
      <input type="number" id="m-bet" value="1" min="1">
    </div>
    <div class="field-row">
      <span class="inline-label">Mines:</span>
      <input type="number" id="m-mines" value="3" min="1" max="20">
    </div>
    <button class="action" id="m-start">Start Round</button>
    <button class="action" id="m-cashout" disabled>Cash Out</button>
    <div class="tag" id="m-statusTag">No active round.</div>
    <div class="mines-grid" id="m-grid"></div>
    <div class="result" id="m-result"></div>
  </section>

  <!-- LIMBO -->
  <section id="limbo" class="game-panel">
    <h2>📈 Limbo</h2>
    <p>Pick a target multiplier. If the crash happens after your target, you win bet × multiplier
(reduced by house edge).</p>
    <div class="field-row">
      <span class="inline-label">Bet (£):</span>
      <input type="number" id="l-bet" value="1" min="1">
    </div>
    <div class="field-row">
      <span class="inline-label">Target x:</span>
      <input type="number" id="l-target" value="2.00" min="1.01" step="0.01">
    </div>
    <button class="action" id="l-start">Start Round</button>
    <div class="limbo-multiplier" id="l-mult">1.00x</div>
    <div class="result" id="l-result"></div>
  </section>

  <!-- ROULETTE -->
  <section id="roulette" class="game-panel">
    <h2>🎡 Roulette</h2>
    <p>European roulette (single 0). Bet on Red or Black. Payout: 2x bet (reduced by house
edge).</p>
    <div class="field-row">
      <span class="inline-label">Bet (£):</span>
      <input type="number" id="r-bet" value="1" min="1">
    </div>
    <div class="field-row">
      <span class="inline-label">Colour:</span>
      <select id="r-choice">
        <option value="Red">Red</option>
        <option value="Black">Black</option>
      </select>
    </div>
    <button class="action" id="r-spin">Spin</button>
    <div class="result" id="r-result"></div>
  </section>

  <!-- SLOTS -->
  <section id="slots" class="game-panel">
    <h2>🎰 Slots</h2>
    <p>3-reel slot. Three of a kind = 5x bet. Any two matching = 2x bet (payouts reduced by
house edge).</p>
    <div class="field-row">
      <span class="inline-label">Bet (£):</span>
      <input type="number" id="s-bet" value="1" min="1">
    </div>
    <button class="action" id="s-spin">Spin</button>
    <div class="slots-reels">
      <div class="slot-reel" id="s-r1">🍒</div>
      <div class="slot-reel" id="s-r2">🍒</div>
      <div class="slot-reel" id="s-r3">🍒</div>
    </div>
    <div class="result" id="s-result"></div>
  </section>

  <!-- VIDEO POKER -->
  <section id="videopoker" class="game-panel">
    <h2>🃏 Video Poker – Jacks or Better</h2>
    <p>Deal, click cards to hold, then draw. Payouts: pair of Jacks or better up to royal flush
(reduced by house edge).</p>
    <div class="field-row">
      <span class="inline-label">Bet (£):</span>
      <input type="number" id="vp-bet" value="1" min="1">
    </div>
    <button class="action" id="vp-dealdraw">Deal</button>
    <div class="vp-cards" id="vp-cards"></div>
    <div class="result" id="vp-result"></div>
  </section>
  <!-- PLINKO -->
  <section id="plinko" class="game-panel">
    <h2>🧱 Plinko</h2>
    <p>Drop a ball. It will land on a random multiplier. Payout = bet × multiplier (reduced by
house edge).</p>
    <div class="field-row">
      <span class="inline-label">Bet (£):</span>
      <input type="number" id="p-bet" value="1" min="1">
    </div>
    <button class="action" id="p-drop">Drop Ball</button>
    <div class="plinko-board">
      <p>Multipliers (sample): [0.5x, 1x, 1.5x, 2x, 3x, 5x]</p>
    </div>
    <div class="result" id="p-result"></div>
  </section>

  <!-- HI-LO -->
  <section id="hilo" class="game-panel">
    <h2>🟦 Hi-Lo</h2>
    <p>Guess if the next card will be higher or lower than the current card. Payout: 2x bet if
correct (reduced by house edge).</p>
    <div class="field-row">
      <span class="inline-label">Bet (£):</span>
      <input type="number" id="hl-bet" value="1" min="1">
    </div>
    <button class="action" id="hl-start">Start Round</button>
    <div class="card-display" id="hl-currentCard">
      <div>?</div>
    </div>
    <button class="action" id="hl-higher" disabled>Higher</button>
    <button class="action" id="hl-lower" disabled>Lower</button>
    <div class="result" id="hl-result"></div>
  </section>

  <!-- KENO -->
  <section id="keno" class="game-panel">
    <h2>🧨 Keno</h2>
    <p>Pick up to 10 numbers (1–40). 20 numbers are drawn. The more hits, the higher the
payout (reduced by house edge).</p>
    <div class="field-row">
      <span class="inline-label">Bet (£):</span>
      <input type="number" id="k-bet" value="1" min="1">
    </div>
    <div class="keno-grid" id="k-grid"></div>
```

```
      <button class="action" id="k-play">Play Keno</button>
      <div class="result" id="k-result"></div>
    </section>
  </main>
</div>

<script>
  // ====== GLOBAL BALANCE + VAULT + STATS + HOUSE EDGE (PERSISTENT) ======
  let balance = parseFloat(localStorage.getItem('casino_balance'));
  let vaultBalance = parseFloat(localStorage.getItem('casino_vault'));

  let totalBet = parseFloat(localStorage.getItem('casino_totalBet'));
  let totalWon = parseFloat(localStorage.getItem('casino_totalWon'));   // net profit from winning
rounds
  let totalLost = parseFloat(localStorage.getItem('casino_totalLost')); // net losses

  let houseEdgePercent = parseFloat(localStorage.getItem('casino_houseEdge'));

  if (isNaN(balance)) balance = 100.0;
  if (isNaN(vaultBalance)) vaultBalance = 0.0;
  if (isNaN(totalBet)) totalBet = 0.0;
  if (isNaN(totalWon)) totalWon = 0.0;
  if (isNaN(totalLost)) totalLost = 0.0;
  if (isNaN(houseEdgePercent)) houseEdgePercent = 5.0; // default 5%

  let houseEdge = houseEdgePercent / 100.0;

  const balanceDisplay = document.getElementById('balanceDisplay');
  const vaultBalanceDisplay = document.getElementById('vault-balanceDisplay');
  const vaultVaultDisplay = document.getElementById('vault-vaultDisplay');
  const vaultResult = document.getElementById('vault-result');

  const edgeSlider = document.getElementById('house-edge');
  const edgeLabel = document.getElementById('house-edge-label');

  const statTotalBet = document.getElementById('stat-totalBet');
  const statTotalWon = document.getElementById('stat-totalWon');
  const statTotalLost = document.getElementById('stat-totalLost');
  const statNet = document.getElementById('stat-net');
  const statROI = document.getElementById('stat-roi');
  const settingsResult = document.getElementById('settings-result');
  const resetBtn = document.getElementById('reset-progress');

  function saveState() {
```

```
  localStorage.setItem('casino_balance', balance);
  localStorage.setItem('casino_vault', vaultBalance);
  localStorage.setItem('casino_totalBet', totalBet);
  localStorage.setItem('casino_totalWon', totalWon);
  localStorage.setItem('casino_totalLost', totalLost);
  localStorage.setItem('casino_houseEdge', houseEdgePercent);
}

function updateBalanceDisplays() {
  balanceDisplay.textContent =
    'Balance: £' + balance.toFixed(2) + ' | Vault: £' + vaultBalance.toFixed(2);
  if (vaultBalanceDisplay && vaultVaultDisplay) {
    vaultBalanceDisplay.textContent = '£' + balance.toFixed(2);
    vaultVaultDisplay.textContent = '£' + vaultBalance.toFixed(2);
  }
}

function updateStatsUI() {
  statTotalBet.textContent = totalBet.toFixed(2);
  statTotalWon.textContent = totalWon.toFixed(2);
  statTotalLost.textContent = totalLost.toFixed(2);
  const net = totalWon - totalLost;
  statNet.textContent = net.toFixed(2);
  const roi = totalBet > 0 ? (net / totalBet) * 100 : 0;
  statROI.textContent = roi.toFixed(2);
}

edgeSlider.value = houseEdgePercent.toString();
edgeLabel.textContent = houseEdgePercent.toFixed(1) + '%';

edgeSlider.addEventListener('input', () => {
  houseEdgePercent = parseFloat(edgeSlider.value);
  if (isNaN(houseEdgePercent)) houseEdgePercent = 0;
  houseEdge = houseEdgePercent / 100.0;
  edgeLabel.textContent = houseEdgePercent.toFixed(1) + '%';
  saveState();
});

function applyHouseEdge(payout) {
  return payout * (1.0 - houseEdge);
}

function recordBetOutcome(bet, payout) {
  totalBet += bet;
  const net = payout - bet;
  if (net > 0) {
    totalWon += net;
  } else if (net < 0) {
    totalLost += -net;
  }
  saveState();
  updateStatsUI();
}

function hardResetAll() {
  localStorage.clear();
  balance = 100.0;
  vaultBalance = 0.0;
  totalBet = 0.0;
  totalWon = 0.0;
  totalLost = 0.0;
  houseEdgePercent = 5.0;
  houseEdge = houseEdgePercent / 100.0;

  edgeSlider.value = houseEdgePercent.toString();
  edgeLabel.textContent = houseEdgePercent.toFixed(1) + '%';

  saveState();
  updateBalanceDisplays();
  updateStatsUI();
  settingsResult.textContent = 'All progress reset. Balance and vault restored to defaults.';
}

resetBtn.addEventListener('click', () => {
  hardResetAll();
});

updateBalanceDisplays();
updateStatsUI();
saveState();

// ====== NAVIGATION ======
const navButtons = document.querySelectorAll('.nav-btn');
const panels = document.querySelectorAll('.game-panel');

function showPanel(name) {
  panels.forEach(p => {
    p.classList.toggle('active', p.id === name);
  });
  navButtons.forEach(btn => {
    btn.classList.toggle('active', btn.dataset.view === name);
  });
  if (name === 'vault') {
    updateBalanceDisplays();
    vaultResult.textContent = '';
  }
  if (name === 'settings') {
    updateStatsUI();
    settingsResult.textContent = '';
  }
}

navButtons.forEach(btn => {
  btn.addEventListener('click', () => {
    showPanel(btn.dataset.view);
  });
});

document.querySelectorAll('.game-card').forEach(card => {
  card.addEventListener('click', () => {
    const target = card.dataset.open;
    showPanel(target);
  });
});

// ====== VAULT LOGIC ======
const vaultDepositInput = document.getElementById('vault-deposit');
const vaultWithdrawInput = document.getElementById('vault-withdraw');
const vaultDepositBtn = document.getElementById('vault-depositBtn');
const vaultWithdrawBtn = document.getElementById('vault-withdrawBtn');

vaultDepositBtn.addEventListener('click', () => {
  const amount = parseFloat(vaultDepositInput.value);
  if (isNaN(amount) || amount <= 0) {
    vaultResult.textContent = 'Enter a valid deposit amount.';
    return;
  }
  if (amount > balance) {
    vaultResult.textContent = 'Cannot deposit more than your current balance.';
    return;
  }
  balance -= amount;
  vaultBalance += amount;
  saveState();
  updateBalanceDisplays();
  vaultResult.textContent = `Deposited £${amount.toFixed(2)} into the vault.`;
});

vaultWithdrawBtn.addEventListener('click', () => {
  const amount = parseFloat(vaultWithdrawInput.value);
  if (isNaN(amount) || amount <= 0) {
    vaultResult.textContent = 'Enter a valid withdraw amount.';
    return;
  }
  if (amount > vaultBalance) {
    vaultResult.textContent = 'Cannot withdraw more than your vault balance.';
    return;
  }
  vaultBalance -= amount;
  balance += amount;
  saveState();
  updateBalanceDisplays();
  vaultResult.textContent = `Withdrew £${amount.toFixed(2)} from the vault.`;
});

// ====== COIN FLIP ======
const cfBet = document.getElementById('cf-bet');
const cfChoice = document.getElementById('cf-choice');
const cfPlay = document.getElementById('cf-play');
const cfResult = document.getElementById('cf-result');

cfPlay.addEventListener('click', () => {
  const bet = parseFloat(cfBet.value);
  if (isNaN(bet) || bet <= 0) {
    cfResult.textContent = 'Enter a valid bet.';
    return;
  }
  if (bet > balance) {
    cfResult.textContent = 'Bet exceeds balance.';
    return;
  }
  const choice = cfChoice.value;
  balance -= bet;

  const outcome = Math.random() < 0.5 ? 'Heads' : 'Tails';
  let msg = `Coin landed on ${outcome}. `;
  let payout = 0;
  if (outcome === choice) {
    payout = applyHouseEdge(bet * 2);
    balance += payout;
    msg += `You win £${payout.toFixed(2)}.`;
  } else {
    msg += `You lose £${bet.toFixed(2)}.`;
  }
  recordBetOutcome(bet, payout);
  saveState();
  updateBalanceDisplays();
  cfResult.textContent = msg;
});

// ====== DICE ======
const dBet = document.getElementById('d-bet');
const dPlay = document.getElementById('d-play');
const dResult = document.getElementById('d-result');

dPlay.addEventListener('click', () => {
  const bet = parseFloat(dBet.value);
  if (isNaN(bet) || bet <= 0) {
    dResult.textContent = 'Enter a valid bet.';
    return;
  }
  if (bet > balance) {
    dResult.textContent = 'Bet exceeds balance.';
    return;
  }
  balance -= bet;
  const d1 = Math.floor(Math.random() * 6) + 1;
  const d2 = Math.floor(Math.random() * 6) + 1;
  const total = d1 + d2;
  let msg = `You rolled ${d1} + ${d2} = ${total}. `;
  let payout = 0;
  if (total >= 8) {
    payout = applyHouseEdge(bet * 2);
    balance += payout;
    msg += `You win £${payout.toFixed(2)}.`;
  } else {
    msg += `You lose £${bet.toFixed(2)}.`;
  }
  recordBetOutcome(bet, payout);
  saveState();
  updateBalanceDisplays();
  dResult.textContent = msg;
});

// ====== MINES ======
const mBet = document.getElementById('m-bet');
const mMines = document.getElementById('m-mines');
const mStart = document.getElementById('m-start');
const mCashout = document.getElementById('m-cashout');
const mGrid = document.getElementById('m-grid');
const mResult = document.getElementById('m-result');
const mStatusTag = document.getElementById('m-statusTag');

let minesActive = false;
let minesBet = 0;
let minesCount = 0;
let minesArray = [];
let minesRevealedSafe = 0;
const totalCells = 25;

function setupMinesGrid() {
  mGrid.innerHTML = '';
  for (let i = 0; i < totalCells; i++) {
    const cell = document.createElement('div');
    cell.className = 'mine-cell';
    cell.dataset.index = i;
    cell.addEventListener('click', onMinesCellClick);
    mGrid.appendChild(cell);
  }
}

function startMinesRound() {
  const bet = parseFloat(mBet.value);
  const count = parseInt(mMines.value, 10);
  if (isNaN(bet) || bet <= 0) {
    mResult.textContent = 'Enter a valid bet.';
    return;
  }
  if (bet > balance) {
    mResult.textContent = 'Bet exceeds balance.';
    return;
  }
  if (isNaN(count) || count < 1 || count >= totalCells) {
    mResult.textContent = 'Invalid number of mines.';
    return;
  }
  balance -= bet;
  saveState();
  updateBalanceDisplays();

  minesBet = bet;
  minesCount = count;
  minesRevealedSafe = 0;
  minesActive = true;
  mCashout.disabled = false;
  mResult.textContent = '';
  mStatusTag.textContent = 'Round active: click tiles or cash out.';

  minesArray = new Array(totalCells).fill(false);
  let placed = 0;
  while (placed < minesCount) {
    const idx = Math.floor(Math.random() * totalCells);
    if (!minesArray[idx]) {
      minesArray[idx] = true;
      placed++;
    }
  }
  setupMinesGrid();
}

function endMinesRound(loss) {
  minesActive = false;
  mCashout.disabled = true;
  const cells = mGrid.querySelectorAll('.mine-cell');
  cells.forEach((cell, idx) => {
    cell.removeEventListener('click', onMinesCellClick);
    if (minesArray[idx]) {
      cell.classList.add('revealed-mine');
      cell.textContent = '💣';
    }
  });
  mStatusTag.textContent = loss ? 'You hit a mine. Round over.' : 'Round ended.';
}

function onMinesCellClick(e) {
  if (!minesActive) return;
  const cell = e.currentTarget;
  const idx = parseInt(cell.dataset.index, 10);
  if (cell.classList.contains('revealed-safe') ||
      cell.classList.contains('revealed-mine')) {
    return;
  }
  if (minesArray[idx]) {
    cell.classList.add('revealed-mine');
    cell.textContent = '💣';
    mResult.textContent = `Boom! You lost £${minesBet.toFixed(2)}.`;
    recordBetOutcome(minesBet, 0);
    endMinesRound(true);
    saveState();
    updateBalanceDisplays();
    return;
  } else {
    cell.classList.add('revealed-safe');
    minesRevealedSafe++;
    const safeCells = totalCells - minesCount;
    const multiplier = 1 + (minesRevealedSafe * (minesCount / safeCells));
    const potential = applyHouseEdge(minesBet * multiplier);
    mResult.textContent =
      `Safe! Potential cashout: £${potential.toFixed(2)} (x${multiplier.toFixed(2)} before edge).`;
  }
}

mStart.addEventListener('click', startMinesRound);

mCashout.addEventListener('click', () => {
  if (!minesActive) return;
  const safeCells = totalCells - minesCount;
  const multiplier = 1 + (minesRevealedSafe * (minesCount / safeCells));
  let payout = minesBet * multiplier;
  payout = applyHouseEdge(payout);
  balance += payout;
  recordBetOutcome(minesBet, payout);
  saveState();
  updateBalanceDisplays();
  mResult.textContent =
    `You cashed out for £${payout.toFixed(2)} (x${multiplier.toFixed(2)} before edge).`;
  endMinesRound(false);
});

setupMinesGrid();

// ====== LIMBO ======
const lBet = document.getElementById('l-bet');
const lTarget = document.getElementById('l-target');
const lStart = document.getElementById('l-start');
const lMult = document.getElementById('l-mult');
const lResult = document.getElementById('l-result');

let limboRunning = false;
let limboInterval = null;

function fairCrashPoint() {
  const r = Math.random();
  return 1 / (1 - r * 0.99);
}

lStart.addEventListener('click', () => {
  if (limboRunning) return;
  const bet = parseFloat(lBet.value);
  let target = parseFloat(lTarget.value);
  if (isNaN(bet) || bet <= 0) {
    lResult.textContent = 'Enter a valid bet.';
    return;
  }
  if (bet > balance) {
    lResult.textContent = 'Bet exceeds balance.';
    return;
  }
  if (isNaN(target) || target <= 1.0) {
    lResult.textContent = 'Target must be greater than 1.0x.';
    return;
  }
  balance -= bet;
  saveState();
  updateBalanceDisplays();
  lResult.textContent = '';
  limboRunning = true;
  lMult.textContent = '1.00x';

  const crash = fairCrashPoint();
  let current = 1.0;

  limboInterval = setInterval(() => {
    current += 0.05;
    lMult.textContent = current.toFixed(2) + 'x';
    if (current >= crash) {
        clearInterval(limboInterval);
        limboRunning = false;
        lMult.textContent = crash.toFixed(2) + 'x';
        let payout = 0;
        if (target <= crash) {
          payout = bet * target;
          payout = applyHouseEdge(payout);
          balance += payout;
          lResult.textContent =
            `Crash at ${crash.toFixed(2)}x. You WIN £${payout.toFixed(2)} (x${target.toFixed(2)}
before edge).`;
        } else {
          lResult.textContent =
            `Crash at ${crash.toFixed(2)}x before your target (x${target.toFixed(2)}). You lose
£${bet.toFixed(2)}.`;
        }
        recordBetOutcome(bet, payout);
        saveState();
        updateBalanceDisplays();
      }
    }, 60);
  });

  // ====== ROULETTE ======
  const rBet = document.getElementById('r-bet');
  const rChoice = document.getElementById('r-choice');
  const rSpin = document.getElementById('r-spin');
  const rResult = document.getElementById('r-result');

  const redNumbers = new Set([1,3,5,7,9,12,14,16,18,19,21,23,25,27,30,32,34,36]);
  const blackNumbers = new Set([2,4,6,8,10,11,13,15,17,20,22,24,26,28,29,31,33,35]);

  rSpin.addEventListener('click', () => {
    const bet = parseFloat(rBet.value);
    if (isNaN(bet) || bet <= 0) {
      rResult.textContent = 'Enter a valid bet.';
      return;
    }
    if (bet > balance) {
      rResult.textContent = 'Bet exceeds balance.';
      return;
    }
    const choice = rChoice.value;
    balance -= bet;
  const number = Math.floor(Math.random() * 37);
  let colour = 'Green';
  if (redNumbers.has(number)) colour = 'Red';
  else if (blackNumbers.has(number)) colour = 'Black';

  let msg = `Result: ${number} (${colour}). `;
  let payout = 0;
  if (colour === choice) {
    payout = applyHouseEdge(bet * 2);
    balance += payout;
    msg += `You win £${payout.toFixed(2)}.`;
  } else {
    msg += `You lose £${bet.toFixed(2)}.`;
  }
  recordBetOutcome(bet, payout);
  saveState();
  updateBalanceDisplays();
  rResult.textContent = msg;
});

// ====== SLOTS ======
const sBet = document.getElementById('s-bet');
const sSpin = document.getElementById('s-spin');
const sR1 = document.getElementById('s-r1');
const sR2 = document.getElementById('s-r2');
const sR3 = document.getElementById('s-r3');
const sResult = document.getElementById('s-result');

const slotSymbols = ['🍒','🍋','🔔','⭐','7'];

sSpin.addEventListener('click', () => {
  const bet = parseFloat(sBet.value);
  if (isNaN(bet) || bet <= 0) {
    sResult.textContent = 'Enter a valid bet.';
    return;
  }
  if (bet > balance) {
    sResult.textContent = 'Bet exceeds balance.';
    return;
  }
  balance -= bet;

  const reels = [
    slotSymbols[Math.floor(Math.random()*slotSymbols.length)],
    slotSymbols[Math.floor(Math.random()*slotSymbols.length)],
    slotSymbols[Math.floor(Math.random()*slotSymbols.length)]
  ];

  sR1.textContent = reels[0];
  sR2.textContent = reels[1];
  sR3.textContent = reels[2];

  let msg = `Result: ${reels.join(' | ')}. `;
  let payout = 0;
  if (reels[0] === reels[1] && reels[1] === reels[2]) {
    payout = applyHouseEdge(bet * 5);
    msg += `Three of a kind! You win £${payout.toFixed(2)}.`;
  } else if (reels[0] === reels[1] || reels[1] === reels[2] || reels[0] === reels[2]) {
    payout = applyHouseEdge(bet * 2);
    msg += `Two of a kind! You win £${payout.toFixed(2)}.`;
  } else {
    msg += `No win. You lose £${bet.toFixed(2)}.`;
  }

  balance += payout;
  recordBetOutcome(bet, payout);
  saveState();
  updateBalanceDisplays();
  sResult.textContent = msg;
});

// ====== VIDEO POKER ======
const vpBet = document.getElementById('vp-bet');
const vpDealDrawBtn = document.getElementById('vp-dealdraw');
const vpCardsContainer = document.getElementById('vp-cards');
const vpResult = document.getElementById('vp-result');

let vpDeck = [];
let vpHand = [];
let vpHeld = [false, false, false, false, false];
let vpIsDealPhase = true;

function createDeck() {
  const suits = ['♠','♥','♦','♣'];
  const ranks = ['2','3','4','5','6','7','8','9','10','J','Q','K','A'];
  const deck = [];
  for (const s of suits) {
    for (const r of ranks) {
      deck.push({rank: r, suit: s});
    }
  }
  for (let i = deck.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random()*(i+1));
    [deck[i], deck[j]] = [deck[j], deck[i]];
  }
  return deck;
}

function renderVpHand() {
  vpCardsContainer.innerHTML = '';
  for (let i = 0; i < 5; i++) {
    const cardDiv = document.createElement('div');
    cardDiv.className = 'vp-card';
    if (vpHeld[i]) cardDiv.classList.add('held');

    if (vpHand[i]) {
      const rankDiv = document.createElement('div');
      rankDiv.className = 'vp-rank';
      rankDiv.textContent = vpHand[i].rank;

      const suitDiv = document.createElement('div');
      suitDiv.className = 'vp-suit';
      suitDiv.textContent = vpHand[i].suit;

      if (vpHand[i].suit === '♥' || vpHand[i].suit === '♦') {
        rankDiv.style.color = 'red';
        suitDiv.style.color = 'red';
      }

      cardDiv.appendChild(rankDiv);
      cardDiv.appendChild(suitDiv);
    } else {
      cardDiv.textContent = '';
    }

    cardDiv.addEventListener('click', () => {
      if (!vpHand[i] || vpIsDealPhase) return;
      vpHeld[i] = !vpHeld[i];
      renderVpHand();
    });
    vpCardsContainer.appendChild(cardDiv);
  }
}

function evaluateVpHand(cards) {
  const ranksOrder = ['2','3','4','5','6','7','8','9','10','J','Q','K','A'];
  const rankCount = {};
  const suitCount = {};
  const idxs = [];

  for (const c of cards) {
    rankCount[c.rank] = (rankCount[c.rank] || 0) + 1;
    suitCount[c.suit] = (suitCount[c.suit] || 0) + 1;
    idxs.push(ranksOrder.indexOf(c.rank));
  }

  idxs.sort((a,b)=>a-b);
  const isFlush = Object.keys(suitCount).length === 1;

  let isStraight = false;
  if (idxs[4] - idxs[0] === 4 && new Set(idxs).size === 5) {
    isStraight = true;
  }
  if (JSON.stringify(idxs) === JSON.stringify([0,1,2,3,12])) {
    isStraight = true;
  }

  const counts = Object.values(rankCount).sort((a,b)=>b-a);
  const hasFour = counts[0] === 4;
  const hasThree = counts[0] === 3;
  const pairs = Object.values(rankCount).filter(v=>v===2).length;

  const isRoyal = isStraight && isFlush &&
    ['10','J','Q','K','A'].every(r => rankCount[r] === 1);

  if (isRoyal) return 'Royal Flush';
  if (isStraight && isFlush) return 'Straight Flush';
  if (hasFour) return 'Four of a Kind';
  if (hasThree && pairs === 1) return 'Full House';
  if (isFlush) return 'Flush';
  if (isStraight) return 'Straight';
  if (hasThree) return 'Three of a Kind';
  if (pairs === 2) return 'Two Pair';
  if (pairs === 1) {
    for (const r of ['J','Q','K','A']) {
      if (rankCount[r] === 2) return 'Jacks or Better';
    }
  }
  return null;
}

function vpPayout(handName, bet) {
  if (!handName) return 0;
  const table = {
    'Royal Flush': 250,
    'Straight Flush': 50,
    'Four of a Kind': 25,
    'Full House': 9,
    'Flush': 6,
    'Straight': 4,
    'Three of a Kind': 3,
    'Two Pair': 2,
    'Jacks or Better': 1
  };
  return bet * (table[handName] || 0);
}

vpDealDrawBtn.addEventListener('click', () => {
  const bet = parseFloat(vpBet.value);
  if (isNaN(bet) || bet <= 0) {
    vpResult.textContent = 'Enter a valid bet.';
    return;
  }
  if (vpIsDealPhase) {
    if (bet > balance) {
      vpResult.textContent = 'Bet exceeds balance.';
      return;
    }
    balance -= bet;
    saveState();
    updateBalanceDisplays();
    vpDeck = createDeck();
    vpHand = [];
    vpHeld = [false,false,false,false,false];
    for (let i=0;i<5;i++) vpHand.push(vpDeck.pop());
    vpIsDealPhase = false;
    vpDealDrawBtn.textContent = 'Draw';
    vpResult.textContent = 'Click cards to hold, then click Draw.';
    renderVpHand();
  } else {
    for (let i=0;i<5;i++) {
      if (!vpHeld[i]) vpHand[i] = vpDeck.pop();
    }
    vpIsDealPhase = true;
    vpDealDrawBtn.textContent = 'Deal';
    const handName = evaluateVpHand(vpHand);
    let payout = vpPayout(handName, bet);
    payout = applyHouseEdge(payout);
    balance += payout;
    recordBetOutcome(bet, payout);
    saveState();
    updateBalanceDisplays();
    if (handName) {
      vpResult.textContent = `${handName} – You win £${payout.toFixed(2)}.`;
    } else {
      vpResult.textContent = `No winning hand – You lose £${bet.toFixed(2)}.`;
    }
    renderVpHand();
  }
});

renderVpHand();

// ====== PLINKO ======
const pBet = document.getElementById('p-bet');
const pDrop = document.getElementById('p-drop');
const pResult = document.getElementById('p-result');
const plinkoMultipliers = [0.5, 1, 1.5, 2, 3, 5];

pDrop.addEventListener('click', () => {
  const bet = parseFloat(pBet.value);
  if (isNaN(bet) || bet <= 0) {
    pResult.textContent = 'Enter a valid bet.';
    return;
  }
  if (bet > balance) {
    pResult.textContent = 'Bet exceeds balance.';
    return;
  }
  balance -= bet;
  const mult = plinkoMultipliers[Math.floor(Math.random()*plinkoMultipliers.length)];
  let payout = bet * mult;
  payout = applyHouseEdge(payout);
  balance += payout;
  recordBetOutcome(bet, payout);
  saveState();
  updateBalanceDisplays();
  pResult.textContent = `Ball landed on x${mult.toFixed(2)}. You win £${payout.toFixed(2)}.`;
});

// ====== HI-LO ======
const hlBet = document.getElementById('hl-bet');
const hlStart = document.getElementById('hl-start');
const hlHigher = document.getElementById('hl-higher');
const hlLower = document.getElementById('hl-lower');
const hlCurrentCardDiv = document.getElementById('hl-currentCard');
const hlResult = document.getElementById('hl-result');

const hlRanks = ['2','3','4','5','6','7','8','9','10','J','Q','K','A'];
let hlCurrentRankIndex = null;
let hlRoundActive = false;
let hlRoundBet = 0;

function renderHlCard(rank) {
  hlCurrentCardDiv.innerHTML = '';
  const rDiv = document.createElement('div');
  rDiv.textContent = rank || '?';
  rDiv.style.fontSize = '24px';
  hlCurrentCardDiv.appendChild(rDiv);
}

hlStart.addEventListener('click', () => {
  const bet = parseFloat(hlBet.value);
  if (isNaN(bet) || bet <= 0) {
    hlResult.textContent = 'Enter a valid bet.';
    return;
  }
  if (bet > balance) {
    hlResult.textContent = 'Bet exceeds balance.';
    return;
  }
  balance -= bet;
  saveState();
  updateBalanceDisplays();
  hlRoundBet = bet;
  hlRoundActive = true;
  hlHigher.disabled = false;
  hlLower.disabled = false;
  hlResult.textContent = '';
  hlCurrentRankIndex = Math.floor(Math.random()*hlRanks.length);
  renderHlCard(hlRanks[hlCurrentRankIndex]);
});

function finishHl(guessHigher) {
  if (!hlRoundActive) return;
  const newIndex = Math.floor(Math.random()*hlRanks.length);
  const oldRank = hlRanks[hlCurrentRankIndex];
  const newRank = hlRanks[newIndex];
  let msg = `Current: ${oldRank}, Next: ${newRank}. `;
  let payout = 0;

  if (newIndex === hlCurrentRankIndex) {
    msg += 'Same rank – push (no win, no loss).';
    payout = hlRoundBet;
    balance += payout;
  } else {
    const correct = (guessHigher && newIndex > hlCurrentRankIndex) ||
                    (!guessHigher && newIndex < hlCurrentRankIndex);
    if (correct) {
      payout = applyHouseEdge(hlRoundBet * 2);
      balance += payout;
      msg += `You guessed correctly! You win £${payout.toFixed(2)}.`;
    } else {
      payout = 0;
      msg += `Wrong guess. You lose £${hlRoundBet.toFixed(2)}.`;
    }
  }

  hlRoundActive = false;
  hlHigher.disabled = true;
  hlLower.disabled = true;
  renderHlCard(newRank);
  hlResult.textContent = msg;
  recordBetOutcome(hlRoundBet, payout);
  saveState();
  updateBalanceDisplays();
}
```

```
hlHigher.addEventListener('click', () => finishHl(true));
hlLower.addEventListener('click', () => finishHl(false));

// ====== KENO ======
const kBet = document.getElementById('k-bet');
const kGrid = document.getElementById('k-grid');
const kPlay = document.getElementById('k-play');
const kResult = document.getElementById('k-result');

const kMaxPick = 10;
let kSelected = new Set();

function renderKenoGrid() {
  kGrid.innerHTML = '';
  for (let i = 1; i <= 40; i++) {
    const cell = document.createElement('div');
    cell.className = 'keno-cell';
    cell.textContent = i;
    cell.dataset.num = i;
    if (kSelected.has(i)) cell.classList.add('selected');
    cell.addEventListener('click', () => {
      const n = parseInt(cell.dataset.num,10);
      if (kSelected.has(n)) {
        kSelected.delete(n);
      } else {
        if (kSelected.size >= kMaxPick) return;
        kSelected.add(n);
      }
      renderKenoGrid();
    });
    kGrid.appendChild(cell);
  }
}

renderKenoGrid();

const kPayoutTable = {
  0: 0,
  1: 0,
  2: 1,
  3: 2,
  4: 5,
  5: 10,
  6: 20,
  7: 40,
  8: 80,
  9: 150,
  10: 300
};

kPlay.addEventListener('click', () => {
  const bet = parseFloat(kBet.value);
  if (isNaN(bet) || bet <= 0) {
    kResult.textContent = 'Enter a valid bet.';
    return;
  }
  if (bet > balance) {
    kResult.textContent = 'Bet exceeds balance.';
    return;
  }
  if (kSelected.size === 0) {
    kResult.textContent = 'Select at least 1 number.';
    return;
  }

  balance -= bet;
  saveState();
  updateBalanceDisplays();

  const drawn = new Set();
  while (drawn.size < 20) {
    drawn.add(Math.floor(Math.random()*40)+1);
  }

  const cells = kGrid.querySelectorAll('.keno-cell');
  cells.forEach(cell => {
    const n = parseInt(cell.dataset.num,10);
    cell.classList.remove('hit','missed');
    if (drawn.has(n) && kSelected.has(n)) {
      cell.classList.add('hit');
    } else if (drawn.has(n)) {
      cell.classList.add('missed');
    }
  });

  let hits = 0;
  kSelected.forEach(n => {
    if (drawn.has(n)) hits++;
    });

    const multiplier = kPayoutTable[hits] || 0;
    let payout = bet * multiplier;
    payout = applyHouseEdge(payout);
    balance += payout;
    recordBetOutcome(bet, payout);
    saveState();
    updateBalanceDisplays();
    kResult.textContent =
      `You picked ${kSelected.size} numbers, hit ${hits}. Multiplier x${multiplier} (before edge).
You win £${payout.toFixed(2)}.`;
  });
</script>
</body>
</html>
```