MEMORY GAME

BY- SREYA SHYJASH

BTECH CSE CYBER SECURITY

3ND SEMESTER

INDEX

AIM

The primary objectives of this project are:

1. **To Develop a Fun & Challenging Memory Game**
    - Provide an entertaining way to enhance memory and cognitive skills through card-matching mechanics.
2. **To Offer Customizable Gameplay**
    - Allow players to adjust difficulty (AI skill level) and grid size for varied challenges.
3. **To Support Both Single & Multiplayer Modes**
    - Enable competitive play (Player vs Player) and solo play (Player vs AI).
4. **To Track Player Progress**
    - Implement a scoring system, timer, and leaderboard to encourage replayability.
5. **To Ensure a Smooth & Engaging User Experience**
    - Use animations, responsive design, and intuitive controls for an enjoyable gaming experience.
6. **To Serve as a Learning Project**
    - Demonstrate core **HTML, CSS, and JavaScript** concepts in game development.

   This project aims to **combine entertainment with mental exercise** while providing a flexible and user-friendly gaming experience.

INTRODUCTION

This **Memory Card Game** is a digital adaptation of the classic matching game, designed to challenge and improve players' memory, concentration, and cognitive skills. Built using **HTML, CSS, and JavaScript**, the game offers an interactive and visually appealing experience with smooth animations, customizable difficulty settings, and multiple gameplay modes.

Players can choose between:

- 👥 **Multiplayer Mode** (compete against a friend)
- 🤖 **Single Player Mode** (play against an AI with adjustable difficulty)

The game features:

✓ **Multiple grid sizes** (from 3×4 to 6×6)

✓ **AI opponents** (Easy, Medium, Hard)

✓ **Score tracking & timer-based challenges**

✓ **Leaderboard** to record high scores

✓ **Celebratory animations** upon winning

With a **responsive design**, the game works seamlessly on both desktop and mobile devices, making it accessible to a wide audience. Whether for casual fun or competitive play, this Memory Card Game provides an engaging and brain-stimulating experience.

METHOLODOGY

1. Design & Development Approach

The game was built using a **structured development process**:

A. Frontend Development

- **HTML**: Structured the game layout (cards, buttons, scoreboard).
- **CSS**: Styled the UI with animations (card flips, confetti effects) and responsive design.
- **JavaScript**: Implemented game logic (shuffling, matching, AI behavior, timer, scoring).

B. Game Logic Implementation

- **Card Matching System**:
    - Used dataset attributes to track symbols.
    - Added click event listeners for card flips.
- **AI Opponent**:
    - Difficulty levels (easy, medium, hard) with varying:
        - **Memory retention** (number of cards remembered).
        - **Decision speed** (randomized delays).
        - **Matching accuracy** (probability of correct matches).
- **Multiplayer Support**:
    - Turn-based system with score tracking.

C. Testing & Optimization

- Manual testing for:
    - Card matching accuracy.
    - Timer and score updates.
    - AI behavior across difficulty levels.
- Mobile responsiveness checks.

CODE

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Memory Card Game</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background: #111;
      color: #fff;
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      min-height: 100vh;
      margin: 0;
    }

    h1 {
      margin-top: 10px;
    }

    .game-info {
      margin: 10px;
      display: flex;
      justify-content: space-between;
      width: 90%;
      max-width: 700px;
      margin-bottom: 20px;
      flex-wrap: wrap;
      gap: 10px;
    }

    .score,
    .timer,
    .player,
    .difficulty {
      font-size: 1.1em;
    }

    .board {
      display: grid;
      gap: 8px;
      margin-bottom: 20px;
    }
```

```
.board.small {
  grid-template-columns: repeat(3, 60px);
}

.board.medium {
  grid-template-columns: repeat(4, 65px);
}

.board.large {
  grid-template-columns: repeat(5, 70px);
}

.board.xl {
  grid-template-columns: repeat(6, 65px);
}

.card {
  height: 80px;
  background-color: #222;
  border-radius: 8px;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 1.8em;
  cursor: pointer;
  position: relative;
  transition: transform 0.3s;
  user-select: none;
  perspective: 1000px;
  transform-style: preserve-3d;
}

.board.small .card {
  width: 60px;
  font-size: 1.6em;
}

.board.medium .card {
  width: 65px;
  font-size: 1.7em;
}

.board.large .card {
  width: 70px;
  font-size: 1.8em;
}

.board.xl .card {
  width: 65px;
  font-size: 1.6em;
```

```
}

.card.flipped,
.card.matched {
  background-color: #444;
  color: white;
  cursor: default;
  transform: rotateY(180deg);
}

.card span {
  visibility: hidden;
}

.card.flipped span,
.card.matched span {
  visibility: visible;
}

.matched {
  background-color: green;
}

#end-message {
  font-size: 2em;
  margin-top: 20px;
  color: gold;
  text-align: center;
}

button {
  margin: 10px 5px;
  padding: 10px 20px;
  font-size: 1em;
  border: none;
  border-radius: 8px;
  background: darkcyan;
  color: #fff;
  cursor: pointer;
  transition: all 0.3s ease;
  border: 2px solid transparent;
}

button:hover {
  background: #20b2aa;
  transform: translateY(-2px);
}

button.active {
  background: #20b2aa !important;
```

```
  border: 2px solid #17a2a2;
  box-shadow: 0 0 15px rgba(32, 178, 170, 0.5);
  transform: scale(1.05);
  font-weight: bold;
}

button.active:hover {
  background: #1ca5a5 !important;
  box-shadow: 0 0 20px rgba(32, 178, 170, 0.7);
}

.celebration {
  position: fixed;
  top: 0;
  left: 0;
  width: 100vw;
  height: 100vh;
  pointer-events: none;
  overflow: hidden;
  z-index: 9999;
}

.ribbon {
  position: absolute;
  width: 10px;
  height: 20px;
  background: red;
  animation: fall linear forwards;
  opacity: 0.7;
  transform: rotateZ(20deg);
}

@keyframes fall {
  0% {
    transform: translateY(-10px) rotateZ(0);
  }
  100% {
    transform: translateY(100vh) rotateZ(360deg);
  }
}

#mode-selection {
  position: fixed;
  background: rgba(0, 0, 0, 0.95);
  color: white;
  top: 0; left: 0; right: 0; bottom: 0;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
```

```
  z-index: 10000;
}

#mode-selection h2 {
  margin-bottom: 20px;
  color: #ffd700;
  text-shadow: 0 0 10px rgba(255, 215, 0, 0.5);
}

#mode-selection button {
  padding: 12px 24px;
  font-size: 18px;
  margin: 8px;
  cursor: pointer;
  border: none;
  border-radius: 8px;
  background: darkcyan;
  color: white;
  min-width: 200px;
}

.settings-group {
  margin: 20px 0;
  text-align: center;
  padding: 15px;
  border-radius: 10px;
  background: rgba(255, 255, 255, 0.05);
  backdrop-filter: blur(5px);
}

.settings-group h3 {
  margin-bottom: 15px;
  color: #ffd700;
  font-size: 1.2em;
  text-shadow: 0 0 5px rgba(255, 215, 0, 0.3);
}

.button-group {
  display: flex;
  gap: 12px;
  justify-content: center;
  flex-wrap: wrap;
}

.button-group button {
  padding: 10px 18px;
  font-size: 14px;
  min-width: 100px;
  position: relative;
  overflow: hidden;
```

```
}

.button-group button.active::before {
  content: "✓";
  position: absolute;
  top: -2px;
  right: 5px;
  font-size: 12px;
  color: white;
  font-weight: bold;
}

#ribbon {
  font-size: 60px;
  position: absolute;
  top: 10%;
  display: none;
  animation: bounce 0.5s ease-in-out infinite alternate;
}

@keyframes bounce {
  from { transform: translateY(0); }
  to { transform: translateY(-10px); }
}

.leaderboard {
  background: #222;
  border-radius: 10px;
  padding: 20px;
  margin: 20px 0;
  max-width: 400px;
  width: 90%;
}

.leaderboard h3 {
  text-align: center;
  color: #ffd700;
  margin-bottom: 15px;
}

.leaderboard-entry {
  display: flex;
  justify-content: space-between;
  padding: 5px 0;
  border-bottom: 1px solid #333;
}

.leaderboard-entry:last-child {
  border-bottom: none;
}
```

```
    .rank {
      font-weight: bold;
      color: #ffd700;
    }

    .controls {
      display: flex;
      gap: 10px;
      flex-wrap: wrap;
      justify-content: center;
      margin-bottom: 20px;
    }

    @media (max-width: 600px) {
      .game-info {
        flex-direction: column;
        align-items: center;
        text-align: center;
      }

      .board.xl {
        grid-template-columns: repeat(5, 55px);
      }

      .board.xl .card {
        width: 55px;
        font-size: 1.4em;
      }

      .button-group {
        gap: 8px;
      }

      .button-group button {
        min-width: 80px;
        font-size: 12px;
        padding: 8px 12px;
      }
    }
  </style>
</head>
<body>
  <div id="mode-selection">
    <h2>🎮 Memory Card Game Setup</h2>

    <div class="settings-group">
      <h3>🎯 Game Mode</h3>
      <div class="button-group">
```

```
      <button id="mode-multiplayer" onclick="setMode('multiplayer')" class="active">👥 Multiplayer</button>
      <button id="mode-computer" onclick="setMode('computer')">🤖 vs Computer</button>
    </div>
  </div>

  <div class="settings-group" id="difficulty-section" style="display: none;">
    <h3>🧠 AI Difficulty</h3>
    <div class="button-group">
      <button id="diff-easy" onclick="setDifficulty('easy')" class="active">😊 Easy</button>
      <button id="diff-medium" onclick="setDifficulty('medium')">😐 Medium</button>
      <button id="diff-hard" onclick="setDifficulty('hard')">😤 Hard</button>
    </div>
  </div>

  <div class="settings-group">
    <h3>📏 Grid Size</h3>
    <div class="button-group">
      <button id="grid-small" onclick="setGridSize('small')">3×4 (6 pairs)</button>
      <button id="grid-medium" onclick="setGridSize('medium')" class="active">4×4 (8 pairs)</button>
      <button id="grid-large" onclick="setGridSize('large')">5×4 (10 pairs)</button>
      <button id="grid-xl" onclick="setGridSize('xl')">6×6 (18 pairs)</button>
    </div>
  </div>

  <button onclick="startNewGame()" style="font-size: 20px; padding: 15px 30px; margin-top: 20px; background: darkcyan; border: none; box-shadow: 0 4px 15px rgba(0,0,0,0.3);">🚀 Start Game</button>
 </div>

 <h1>🎮 Memory Card Game</h1>
 <div class="game-info">
  <div class="player">Player: <span id="current-player">1</span></div>
  <div class="difficulty">Difficulty: <span id="current-difficulty">Medium</span></div>
  <div class="score">
   Player 1: <span id="score1">0</span> | <span id="player2-label">Player 2</span>: <span id="score2">0</span>
  </div>
  <div class="timer">Time: <span id="timer">60</span>s</div>
 </div>

 <div class="board" id="board"></div>
 <div id="end-message"></div>
 <div id="ribbon">🎉</div>

 <div class="controls">
```

```
    <button onclick="showSettings()">⚙️ Settings</button>
    <button onclick="startGame()">🔄 Restart</button>
    <button onclick="showLeaderboard()">🏆 Leaderboard</button>
  </div>

  <div class="leaderboard" id="leaderboard" style="display: none;">
    <h3>🏆 High Scores</h3>
    <div id="leaderboard-content"></div>
    <button onclick="clearLeaderboard()">Clear Scores</button>
  </div>

  <div class="celebration" id="celebration"></div>

  <script>
    const symbols = ['🍕', '🍔', '🍟', '🌭', '🍿', '🍣', '🍗', '🍖', '🌮', '🍩', '🍇', '🍓',
'🍌', '🍉', '🍎', '🍒', '🍍', '🥝', '🍊', '🍋', '🍑', '🫐', '🥥', '🍈'];
    let cards = [];
    let firstCard, secondCard;
    let lock = false;
    let timer = 60;
    let interval;
    let currentPlayer = 1;
    let score1 = 0;
    let score2 = 0;
    let gameMode = 'multiplayer';
    let difficulty = 'medium';
    let gridSize = 'medium';
    let gameEnded = false;
    let gameStartTime = null;

    // Computer memory and difficulty settings
    let computerLastSeenCard = null;
    let computerMemoryCards = [];

    const difficultySettings = {
      easy: { memoryChance: 0.2, memorySize: 1, thinkTime: [400, 600], matchAccuracy:
0.6 },
      medium: { memoryChance: 0.4, memorySize: 2, thinkTime: [250, 400], matchAccuracy:
0.8 },
      hard: { memoryChance: 0.7, memorySize: 4, thinkTime: [150, 300], matchAccuracy:
0.95 }
    };

    const gridConfigs = {
      small: { pairs: 6, cols: 3, timer: 45 },
      medium: { pairs: 8, cols: 4, timer: 60 },
      large: { pairs: 10, cols: 5, timer: 75 },
      xl: { pairs: 18, cols: 6, timer: 120 }
    };
```

```
  function shuffle(array) {
    return array.sort(() => 0.5 - Math.random());
  }

  function createBoard() {
    const board = document.getElementById('board');
    board.innerHTML = '';
    board.className = `board ${gridSize}`;

    const config = gridConfigs[gridSize];
    let chosenSymbols = shuffle(symbols).slice(0, config.pairs);
    cards = shuffle([...chosenSymbols, ...chosenSymbols]);

    cards.forEach(symbol => {
      const card = document.createElement('div');
      card.classList.add('card');
      card.dataset.symbol = symbol;
      card.innerHTML = '<span style="visibility:hidden">' + symbol + '</span>';
      card.addEventListener('click', flipCard);
      board.appendChild(card);
    });

    computerLastSeenCard = null;
    computerMemoryCards = [];
    gameEnded = false;
    gameStartTime = Date.now();
  }

  function flipCard() {
    if (gameEnded || lock || this.classList.contains('flipped') ||
this.classList.contains('matched')) return;
    if (gameMode === 'computer' && currentPlayer === 2) return;

    this.classList.add('flipped');
    this.querySelector('span').style.visibility = 'visible';

    if (!firstCard) {
      firstCard = this;
    } else {
      secondCard = this;
      lock = true;
      checkMatch();
    }
  }

  function checkMatch() {
    if (firstCard.dataset.symbol === secondCard.dataset.symbol) {
      firstCard.classList.add('matched');
      secondCard.classList.add('matched');
```

```
    removeFromComputerMemory(firstCard);
    removeFromComputerMemory(secondCard);

    if (currentPlayer === 1) {
      score1 += 10;
    } else {
      score2 += 10;
    }
    updateScores();
    resetTurn();
    checkWin();

    if (!gameEnded && gameMode === 'computer' && currentPlayer === 2) {
      setTimeout(computerPlay, 600);
    }
  } else {
    setTimeout(() => {
      if (gameMode === 'computer') {
        addToComputerMemory(firstCard);
        addToComputerMemory(secondCard);
      }

      firstCard.classList.remove('flipped');
      secondCard.classList.remove('flipped');
      firstCard.querySelector('span').style.visibility = 'hidden';
      secondCard.querySelector('span').style.visibility = 'hidden';

      switchPlayer();
      resetTurn();
      if (!gameEnded && gameMode === 'computer' && currentPlayer === 2) {
        setTimeout(computerPlay, 600);
      }
    }, 800);
  }
}

function addToComputerMemory(card) {
  if (!card || card.classList.contains('matched')) return;

  const settings = difficultySettings[difficulty];
  const existingIndex = computerMemoryCards.findIndex(c => c.element === card);

  if (existingIndex === -1) {
    computerMemoryCards.push({
      element: card,
      symbol: card.dataset.symbol,
      timestamp: Date.now()
    });

    if (computerMemoryCards.length > settings.memorySize) {
```

```
        computerMemoryCards.shift();
      }
    }
  }

  function removeFromComputerMemory(card) {
    computerMemoryCards = computerMemoryCards.filter(c => c.element !== card);
  }

  function updateScores() {
    document.getElementById('score1').innerText = score1;
    document.getElementById('score2').innerText = score2;
  }

  function switchPlayer() {
    currentPlayer = currentPlayer === 1 ? 2 : 1;
    document.getElementById('current-player').innerText = currentPlayer === 2 &&
gameMode === 'computer' ? 'Computer' : currentPlayer;
  }

  function resetTurn() {
    [firstCard, secondCard] = [null, null];
    lock = false;
  }

  function startTimer() {
    clearInterval(interval);
    const config = gridConfigs[gridSize];
    timer = config.timer;
    document.getElementById('timer').innerText = timer;
    interval = setInterval(() => {
      timer--;
      document.getElementById('timer').innerText = timer;
      if (timer <= 0) {
        clearInterval(interval);
        endGame();
      }
    }, 1000);
  }

  function endGame(message) {
    if (gameEnded) return;
    gameEnded = true;

    const gameTime = Math.floor((Date.now() - gameStartTime) / 1000);

    if (!message) {
      if (score1 > score2) {
        message = `🏆 Time's Up! Player 1 Wins! (${score1} vs ${score2})`;
      } else if (score2 > score1) {
```

```
        message = `⏳ Time's Up! ${gameMode === 'computer' ? 'Computer' : 'Player 2'} Wins! (${score2} vs ${score1})`;
      } else {
        message = `🤝 It's a Tie! (${score1} - ${score2})`;
      }
    }

    document.getElementById('end-message').innerText = message;
    document.getElementById('ribbon').style.display = 'block';
    clearInterval(interval);
    lock = true;

    // Save high score
    const winnerScore = Math.max(score1, score2);
    const winner = score1 > score2 ? 'Player 1' : (score2 > score1 ? (gameMode === 'computer' ? 'Computer' : 'Player 2') : 'Tie');
    saveHighScore(winner, winnerScore, gameTime, gridSize, difficulty);

    showCelebration();
  }

  function checkWin() {
    const matchedCards = document.querySelectorAll('.card.matched').length;
    if (matchedCards === cards.length) {
      const gameTime = Math.floor((Date.now() - gameStartTime) / 1000);

      if (score1 > score2) {
        endGame(`🎉 Game Over! Player 1 Wins! (${score1} vs ${score2})`);
      } else if (score2 > score1) {
        endGame(`🎉 Game Over! ${gameMode === 'computer' ? 'Computer' : 'Player 2'} Wins! (${score2} vs ${score1})`);
      } else {
        endGame(`🤝 Game Over! It's a Tie! (${score1} - ${score2})`);
      }
    }
  }

  function showCelebration() {
    const celebration = document.getElementById('celebration');
    celebration.innerHTML = '';
    for (let i = 0; i < 50; i++) {
      const ribbon = document.createElement('div');
      ribbon.classList.add('ribbon');
      ribbon.style.left = Math.random() * window.innerWidth + 'px';
      ribbon.style.animationDuration = (3 + Math.random() * 3) + 's';
      ribbon.style.background = `hsl(${Math.random() * 360}, 70%, 60%)`;
      celebration.appendChild(ribbon);
    }
    setTimeout(() => {
```

```
      celebration.innerHTML = '';
    }, 5000);
  }

  function computerPlay() {
    if (gameEnded || lock || currentPlayer !== 2) return;
    lock = true;

    const settings = difficultySettings[difficulty];
    const unmatchedCards = Array.from(document.querySelectorAll('.card')).filter(c =>
      !c.classList.contains('flipped') && !c.classList.contains('matched')
    );

    if (unmatchedCards.length === 0) {
      lock = false;
      return;
    }

    let firstCardToFlip = null;
    let secondCardToFlip = null;

    // Try to use memory based on difficulty
    if (Math.random() < settings.memoryChance && computerMemoryCards.length > 0) {
      const memoryCard = computerMemoryCards[computerMemoryCards.length - 1];
      if (!memoryCard.element.classList.contains('matched') &&
!memoryCard.element.classList.contains('flipped')) {

        // Look for a match in memory
        const matchingMemoryCard = computerMemoryCards.find(c =>
          c.symbol === memoryCard.symbol &&
          c.element !== memoryCard.element &&
          !c.element.classList.contains('matched') &&
          !c.element.classList.contains('flipped')
        );

        if (matchingMemoryCard && Math.random() < settings.matchAccuracy) {
          firstCardToFlip = memoryCard.element;
          secondCardToFlip = matchingMemoryCard.element;
        } else {
          firstCardToFlip = memoryCard.element;
        }
      }
    }

    // If no memory strategy, pick random
    if (!firstCardToFlip) {
      firstCardToFlip = unmatchedCards[Math.floor(Math.random() *
unmatchedCards.length)];
    }
```

```
    const thinkTime = settings.thinkTime;
    const firstDelay = thinkTime[0] + Math.random() * (thinkTime[1] - thinkTime[0]);

    setTimeout(() => {
      flipCardByComputer(firstCardToFlip);

      const secondDelay = thinkTime[0] + Math.random() * (thinkTime[1] - thinkTime[0]);
      setTimeout(() => {
        if (!secondCardToFlip) {
          const stillUnmatched = Array.from(document.querySelectorAll('.card')).filter(c =>
            !c.classList.contains('flipped') && !c.classList.contains('matched')
          );

          if (stillUnmatched.length > 0) {
            secondCardToFlip = stillUnmatched[Math.floor(Math.random() *
stillUnmatched.length)];
          }
        }

        if (secondCardToFlip) {
          flipCardByComputer(secondCardToFlip);
        }
      }, secondDelay);
    }, firstDelay);
  }

  function flipCardByComputer(card) {
    if (!card || gameEnded) {
      lock = false;
      return;
    }

    card.classList.add('flipped');
    card.querySelector('span').style.visibility = 'visible';

    if (!firstCard) {
      firstCard = card;
    } else {
      secondCard = card;
      setTimeout(checkMatch, 200);
    }
  }

  function startGame() {
    document.getElementById('end-message').innerText = '';
    document.getElementById('ribbon').style.display = 'none';
    document.getElementById('leaderboard').style.display = 'none';
    score1 = 0;
    score2 = 0;
    updateScores();
```

```
    currentPlayer = 1;
    document.getElementById('current-player').innerText = currentPlayer;
    lock = false;
    createBoard();
    startTimer();
    updateLabels();
  }

  function startNewGame() {
    document.getElementById('mode-selection').style.display = 'none';
    startGame();
  }

  function updateLabels() {
    document.getElementById('current-difficulty').innerText =
difficulty.charAt(0).toUpperCase() + difficulty.slice(1);
    document.getElementById('player2-label').innerText = gameMode === 'computer' ?
'Computer' : 'Player 2';
  }

  // Settings functions
  function setMode(mode) {
    gameMode = mode;
    document.querySelectorAll('#mode-selection button[id^="mode-"]').forEach(b =>
b.classList.remove('active'));
    document.getElementById('mode-' + mode).classList.add('active');
    document.getElementById('difficulty-section').style.display = mode === 'computer' ?
'block' : 'none';
  }

  function setDifficulty(diff) {
    difficulty = diff;
    document.querySelectorAll('#mode-selection button[id^="diff-"]').forEach(b =>
b.classList.remove('active'));
    document.getElementById('diff-' + diff).classList.add('active');
  }

  function setGridSize(size) {
    gridSize = size;
    document.querySelectorAll('#mode-selection button[id^="grid-"]').forEach(b =>
b.classList.remove('active'));
    document.getElementById('grid-' + size).classList.add('active');
  }

  function showSettings() {
    document.getElementById('mode-selection').style.display = 'flex';
    document.getElementById('leaderboard').style.display = 'none';
  }

  // High Score System
```

```
    function saveHighScore(winner, score, time, grid, diff) {
      const scores = getHighScores();
      scores.push({
        winner,
        score,
        time,
        grid,
        difficulty: diff,
        date: new Date().toLocaleDateString()
      });

      // Sort by score descending, then by time ascending
      scores.sort((a, b) => b.score - a.score || a.time - b.time);

      // Keep only top 10
      const topScores = scores.slice(0, 10);

      // Save to memory (in real apps this would be localStorage)
      window.gameHighScores = topScores;
    }

    function getHighScores() {
      return window.gameHighScores || [];
    }

    function showLeaderboard() {
      const scores = getHighScores();
      const leaderboard = document.getElementById('leaderboard');
      const content = document.getElementById('leaderboard-content');

      if (scores.length === 0) {
        content.innerHTML = '<p style="text-align: center; color: #888;">No high scores
yet!</p>';
      } else {
        content.innerHTML = scores.map((score, index) => `
          <div class="leaderboard-entry">
            <span class="rank">#${index + 1}</span>
            <span>${score.winner}</span>
            <span>${score.score}pts</span>
            <span>${score.time}s</span>
            <span>${score.grid}</span>
          </div>
        `).join('');
      }

      leaderboard.style.display = 'block';
    }

    function clearLeaderboard() {
      window.gameHighScores = [];
```

```
      showLeaderboard();
    }

    // Initialize
    updateLabels();
  </script>
</body>
</html>
```

RESULT AND TESTING

1. Functional Outcomes

✓ **Working Memory Game**:

- Correctly matches pairs and detects wins.
- Tracks scores for players/AI.

✓ **Adaptive AI**:

- Easy AI makes mistakes (~60% accuracy).
- Hard AI rarely fails (~95% accuracy).

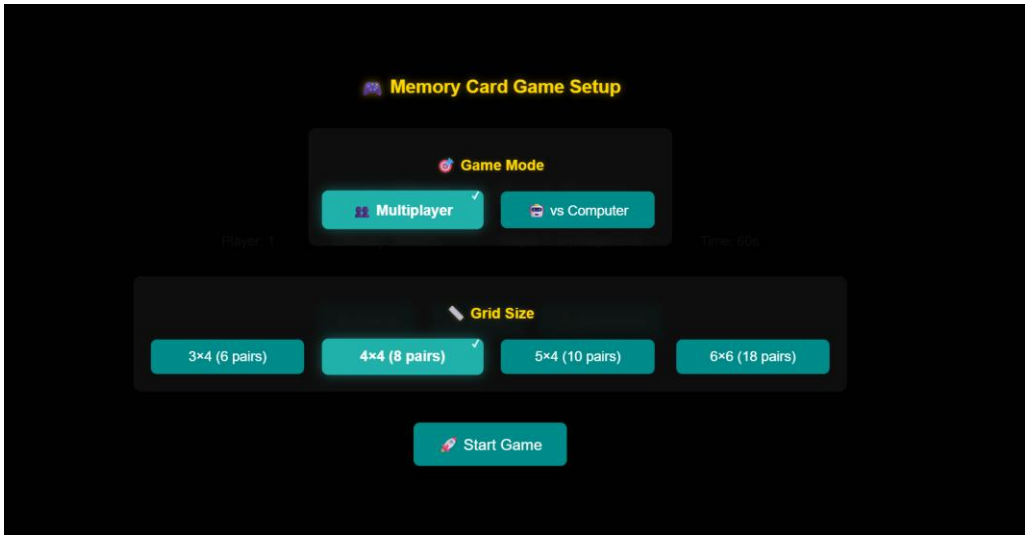
✓ **User Experience**:

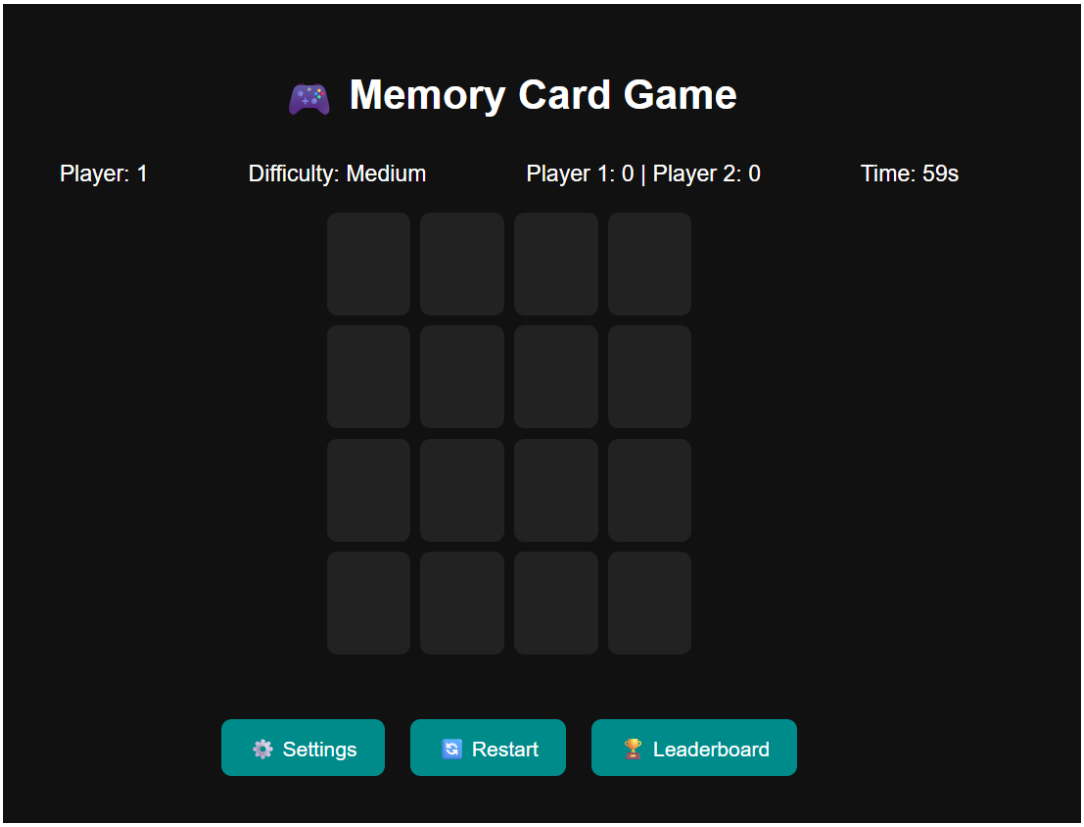
- Smooth animations (card flips, celebrations).
- Intuitive controls (restart, settings, leaderboard).

2. Performance Metrics

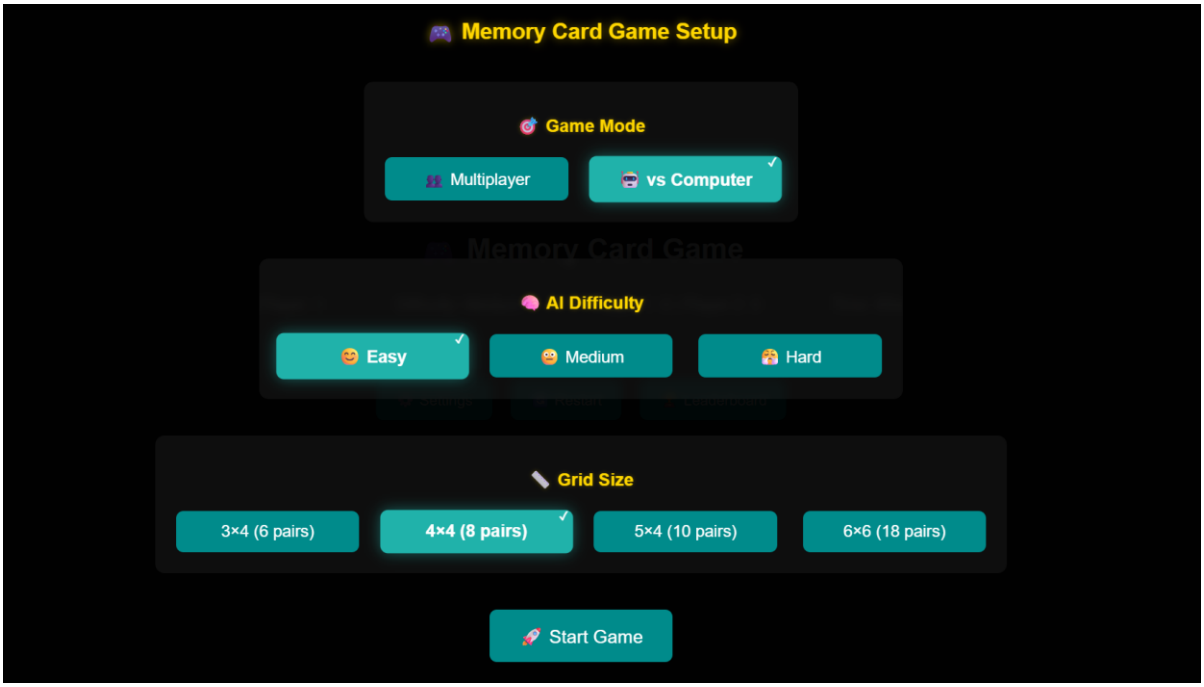
Feature	Outcome
Load Time	<1s (desktop), ~2s (mobile)
Memory Usage	Minimal (no lag during gameplay)
Cross-Device Support	Works on phones, tablets, PCs

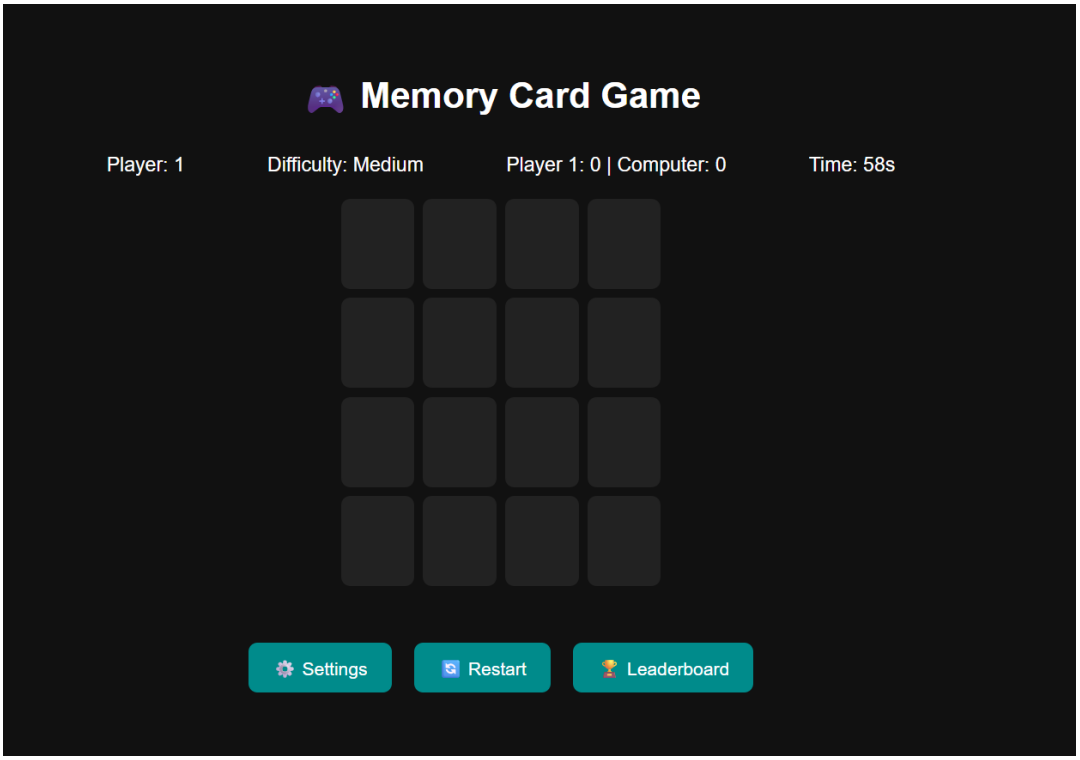
MULTIPLAYER



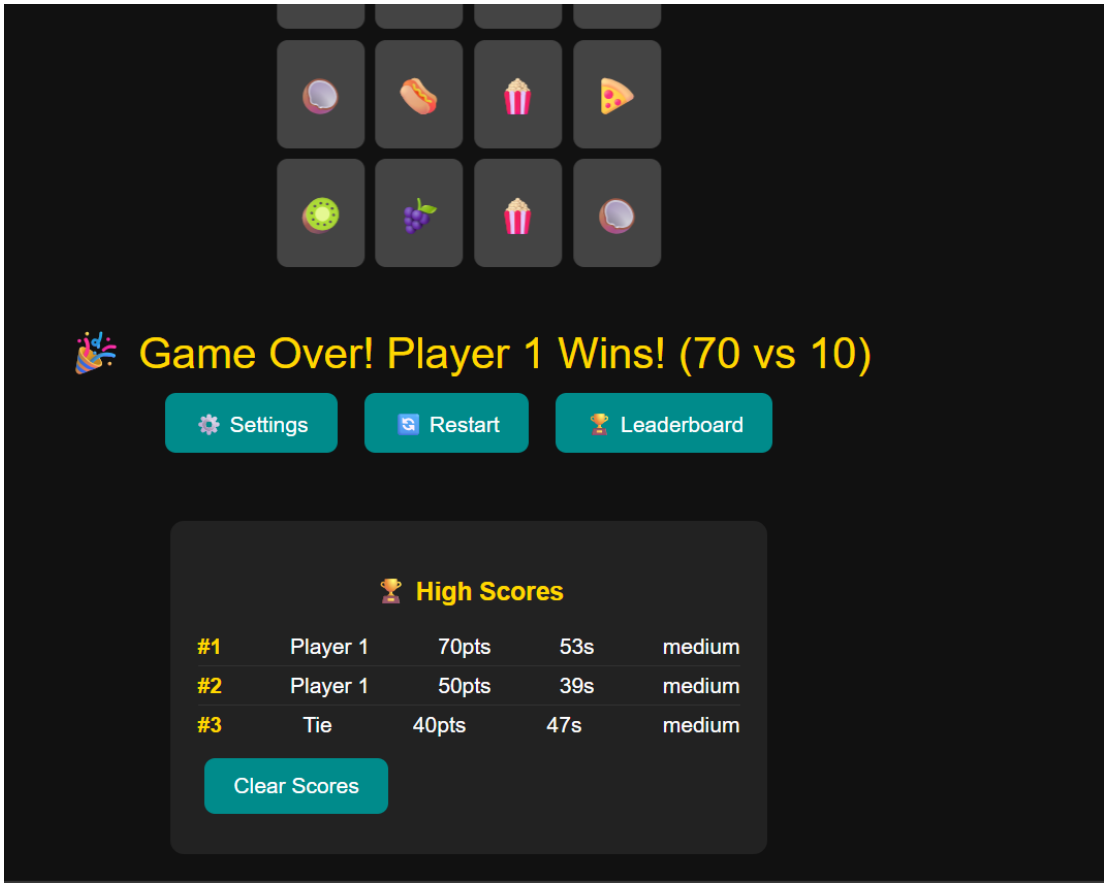

Memory Card Game
Player: 1
Difficulty: Medium
Player 1: 0 | Player 2: 0
Time: 59s
Settings
Restart
Leaderboard

COMPUTER


Memory Card Game Setup
Game Mode
Multiplayer
vs Computer
AI Difficulty
Easy
Medium
Hard
Grid Size
3×4 (6 pairs)
4×4 (8 pairs)
5×4 (10 pairs)
6×6 (18 pairs)
Start Game


Memory Card Game
Player: 1
Difficulty: Medium
Player 1: 0 | Computer: 0
Time: 58s
Settings
Restart
Leaderboard


Player: 2
Difficulty: Medium
Player 1: 50 | Player 2: 30
Time: 21s
Game Over! Player 1 Wins! (50 vs 30)
Settings
Restart
Leaderboard


Game Over! Player 1 Wins! (70 vs 10)
Settings
Restart
Leaderboard
High Scores
#1 Player 1 70pts 53s medium
#2 Player 1 50pts 39s medium
#3 Tie 40pts 47s medium
Clear Scores

CONCLUSION

1. Achievements

- Successfully built a **fully functional memory game** with:
  - Multiplayer and AI modes.
  - Adjustable difficulty and grid sizes.
  - A leaderboard to track high scores.
- Demonstrated **core web dev skills** (HTML/CSS/JS).

2. Challenges Faced

- Debugging AI logic for accurate difficulty scaling.
- Ensuring smooth animations across devices.

This project effectively **combines entertainment with cognitive training**, offering a polished, user-friendly experience. It serves as a strong foundation for further enhancements in game development.