

## **Реализация логики игрового процесса в браузерной игре 2048: обработка движения и слияния плиток**

*Екимова Яна Сергеевна*

*Приамурский государственный университет имени Шолом-Алейхема*

*Студент*

### **Аннотация**

В статье рассматривается процесс разработки интерактивной браузерной игры 2048 с использованием современных веб-технологий: HTML5 для структуры игрового интерфейса, CSS3 — для стилизации и адаптации под различные устройства, а также JavaScript — для реализации логики перемещения и слияния плиток. Описаны ключевые аспекты создания игровой механики, включая обработку пользовательского ввода, генерацию случайных чисел, проверку условий завершения игры и отображение текущего счёта. Представленная методика может быть использована как основа для изучения принципов веб-разработки, а также как образовательный инструмент для студентов и преподавателей, заинтересованных в применении игровых технологий в учебном процессе. Реализация демонстрирует практическое применение клиентских технологий для создания простых, но функциональных и увлекательных веб-приложений.

**Ключевые слова:** веб-разработка, 2048, HTML5, CSS3, JavaScript.

### **Implementation of gameplay logic in the browser game 2048: handling movement and tile merging**

*Ekimova Yana Sergeevna*

*Sholom Aleichem Priamurskiy State University*

*Student*

### **Abstract**

The article discusses the process of developing an interactive browser game 2048 using modern web technologies: HTML5 for the structure of the game interface, CSS3 for styling and adapting to various devices, as well as JavaScript for implementing the logic of moving and merging tiles. The key aspects of creating game mechanics are described, including processing user input, generating random numbers, checking game completion conditions, and displaying the current score. The presented methodology can be used as a basis for studying the principles of web development, as well as as an educational tool for students and teachers interested in using gaming technologies in the educational process. The implementation demonstrates the practical application of client technologies to create simple, but functional and exciting web applications.

**Keywords:** web development, 2048, HTML5, CSS3, JavaScript.

## **1 Введение**

### **1.1 Актуальность**

Актуальность разработки игры 2048 на современных веб-технологиях обусловлена растущим интересом к созданию кроссплатформенных игровых приложений, не требующих установки и работающих в браузере любых устройств. Данный проект демонстрирует практическое применение базовых технологий веб-разработки (HTML, CSS, JavaScript) для создания полноценной логической игры, что особенно ценно для образовательного процесса при обучении программированию. Игра 2048, сохраняя свою популярность благодаря простым правилам и увлекательному геймплею, служит отличным примером для изучения ключевых аспектов frontend-разработки: работы с DOM, обработки событий, управления состоянием приложения и создания анимаций, что делает её идеальным учебным проектом для начинающих разработчиков.

### **1.2 Обзор исследований**

В статье С.В. Рогатина, В.А. Антошкина рассматривается методика разработки компьютерных игр по технологии HTML5 с помощью CASE-средства GameMakerStudio компании YoYoGamesLtd. Проводится тестирование производительности работы компьютерных игр в браузере в зависимости от сложности использованных спрайтов игровых объектов [1]. Разработали игровое приложение «league of football heads» с использованием html5, css3, javascript И.А. Фёдоров, Е.Ю. Кузьменкова[2]. Королук Я. А., Куликова Н. Ю. показали опыт разработки и использования интерактивных образовательных ресурсов на основе html5 для онлайн-обучения школьников информатике [3]. В работе рассмотрели возможности языка разметки гипертекста HTML5 для создания макета электронного учебника Д.М. Размологов, А.С. Панина, П.В. Захаров. Привели разработанный макет, а также часть таблицы стилей для дизайна макета [4]. И.С. Шегаев, А.Г.Полозов разработали новейший политологический глоссарий: основные категории, понятия, термины [5].

### **1.3 Цель исследования**

Целью исследования является реализация логики игрового процесса в браузерной игре 2048 с использованием HTML, CSS и JavaScript.

## **2 Материалы и методы**

Для создания приложения используется язык программирования HTML5. В качестве среды разработки используется текстовым редактор Notepad++, оптимизированным для языков программирования.

## **3 Результаты**

Открываем Notepad++ и создаём файл igra.html, для этого в меню выбираем файл → новый. Далее нажимаем сочетание клавиш "Ctrl+S" и в поле

"Имя файла" вводим название файла "index" с расширением .html, сохраняем данный файл на рабочем столе.

Создадим основную структуру HTML-документа для игры 2048. Объявляем DOCTYPE и открываем теги html с русской локализацией, в секции head задаем мета-теги для корректного отображения на мобильных устройствах и заголовок страницы, затем в body размещаем контейнер с шапкой (содержащей название игры и блоки счета), панелью управления (кнопки "Новая игра" и "Отменить"), игровым полем (сетка 4x4 для плиток), а также всплывающими окнами для сообщений о конце игры и победе (с соответствующими кнопками действий), внизу добавляем инструкции по управлению, а в конце документа оставляем место для JavaScript-кода, который будет реализовывать логику игры.

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>2048 - Идеальное позиционирование</title>
  <style>

  </style>
</head>
<body>
  <div class="container">
    <header>
      <h1>2048</h1>
      <div class="scores-container">
        <div class="score-box">
          <div class="score-title">Счет</div>
          <div class="score-value" id="score">0</div>
        </div>
        <div class="score-box">
          <div class="score-title">Рекорд</div>
          <div class="score-value" id="best-score">0</div>
        </div>
      </div>
    </header>

    <div class="controls">
      <button class="btn" id="new-game-btn">Новая игра</button>
      <button class="btn" id="undo-btn">Отменить</button>
    </div>

    <div class="game-container">
      <div class="grid-container">

      </div>
    </div>

    <div class="game-over" id="game-over">
      <div class="game-over-content">
        <div class="game-over-text">Игра окончена!</div>
        <div class="final-score">Ваш счет: <span id="final-score">0</span></div>
        <button class="btn" id="try-again-btn">Попробовать снова</button>
      </div>
    </div>
  </div>
</body>
```

```

    <div class="win-message" id="win-message">
      <div class="win-content">
        <div class="win-text">Победа!</div>
        <div class="win-message-text">Вы достигли 2048!</div>
        <button class="btn" id="continue-btn">Продолжить</button>
      </div>
    </div>

    <div class="instructions">
      Используйте стрелки на клавиатуре или свайпы для перемещения плиток
    </div>

</div>

<script>

</script>
</body>
</html>

```

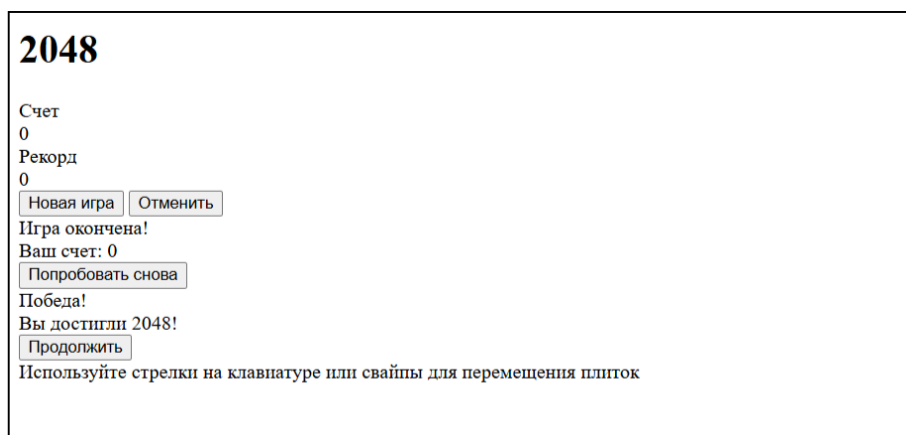


Рисунок 1. Основная структура HTML-документа

Добавим стили, которые задают визуальное оформление игры: сначала определяем CSS-переменные в `root` для основных цветов (основного, фоновых, текста и цветов плиток разного номинала), затем сбрасываем отступы и задаем `box-sizing` для всех элементов, устанавливаем шрифт и оформление `body` с центрированием контента. Для заголовка `h1` создаем градиентный текст, оформляем блоки счета с тенями и скругленными углами, стилизуем кнопки с эффектами при наведении. Игровое поле (`grid`) и ячейки (`cell`) получают фоновые цвета и отступы, а плитки (`tile`) позиционируются абсолютно с анимацией появления, разными цветами в зависимости от номинала и адаптивным размером текста. Добавлены стили для всплывающих окон окончания игры и победы с анимацией появления, эффектом размытия фона и конфетти-анимацией. В конце предусмотрены медиа-запросы для адаптации интерфейса на мобильных устройствах с уменьшением размеров текста и элементов.

Данный код пропишем на следующей строчке после тега `<title> 2048 </title>` в секции `<head>` HTML-документа.

```

:root {
  --primary-color: #6a5acd;

```

```
--secondary-color: #9370db;
--background-color: #f8f9fa;
--text-color: #333;
--grid-background: #bbada0;
--cell-background: rgba(238, 228, 218, 0.35);
--tile-2: #eee4da;
--tile-4: #ede0c8;
--tile-8: #f2b179;
--tile-16: #f59563;
--tile-32: #f67c5f;
--tile-64: #f65e3b;
--tile-128: #edcf72;
--tile-256: #edcc61;
--tile-512: #edc850;
--tile-1024: #edc53f;
--tile-2048: #edc22e;
}

* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

body {
  background-color: var(--background-color);
  color: var(--text-color);
  display: flex;
  flex-direction: column;
  align-items: center;
  min-height: 100vh;
  padding: 20px;
}

.container {
  width: 100%;
  max-width: 500px;
}

header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 20px;
  width: 100%;
}

h1 {
  font-size: 3rem;
  background: linear-gradient(135deg, var(--primary-color), var(--secondary-color));
  -webkit-background-clip: text;
  background-clip: text;
  color: transparent;
  text-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

.scores-container {
  display: flex;
  gap: 10px;
}

.score-box {
```

```
background: white;
padding: 10px 15px;
border-radius: 10px;
text-align: center;
box-shadow: 0 2px 8px rgba(0,0,0,0.1);
min-width: 80px;
}

.score-title {
  font-size: 0.8rem;
  color: #666;
  font-weight: bold;
  text-transform: uppercase;
}

.score-value {
  font-size: 1.5rem;
  font-weight: bold;
  color: var(--primary-color);
}

.controls {
  display: flex;
  gap: 10px;
  margin-bottom: 20px;
}

.btn {
  background: var(--primary-color);
  color: white;
  border: none;
  border-radius: 8px;
  padding: 12px 0;
  font-size: 1rem;
  font-weight: bold;
  cursor: pointer;
  flex: 1;
  transition: all 0.2s ease;
  box-shadow: 0 2px 5px rgba(0,0,0,0.1);
}

.btn:hover {
  transform: translateY(-2px);
  box-shadow: 0 4px 8px rgba(0,0,0,0.15);
}

.btn:active {
  transform: translateY(0);
  box-shadow: 0 2px 3px rgba(0,0,0,0.1);
}

.game-container {
  background: white;
  border-radius: 12px;
  padding: 15px;
  width: 100%;
  box-shadow: 0 5px 15px rgba(0,0,0,0.1);
  position: relative;
  overflow: hidden;
}

.grid-container {
  position: relative;
```

```
width: 100%;
padding-bottom: 100%; /* Квадратный контейнер */
}

.grid {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-template-rows: repeat(4, 1fr);
  gap: 15px;
  background: var(--grid-background);
  border-radius: 8px;
  padding: 15px;
}

.cell {
  background: var(--cell-background);
  border-radius: 8px;
}

.tile {
  position: absolute;
  display: flex;
  justify-content: center;
  align-items: center;
  font-size: 2rem;
  font-weight: bold;
  border-radius: 8px;
  z-index: 10;
  transition: all 0.15s ease-out;
  animation: appear 0.2s;
  box-shadow: 0 2px 5px rgba(0,0,0,0.1);
}

@keyframes appear {
  0% {
    opacity: 0;
    transform: scale(0.5);
  }
  100% {
    opacity: 1;
    transform: scale(1);
  }
}

.tile-2 { background: var(--tile-2); color: #776e65; }
.tile-4 { background: var(--tile-4); color: #776e65; }
.tile-8 { background: var(--tile-8); color: white; }
.tile-16 { background: var(--tile-16); color: white; }
.tile-32 { background: var(--tile-32); color: white; }
.tile-64 { background: var(--tile-64); color: white; }
.tile-128 { background: var(--tile-128); color: white; font-size: 1.8rem; }
.tile-256 { background: var(--tile-256); color: white; font-size: 1.8rem; }
.tile-512 { background: var(--tile-512); color: white; font-size: 1.8rem; }
.tile-1024 { background: var(--tile-1024); color: white; font-size: 1.5rem; }
.tile-2048 { background: var(--tile-2048); color: white; font-size: 1.5rem; }
.tile-super { background: #3c3a32; color: white; font-size: 1.3rem; }

.game-over {
```

```
position: absolute;
top: 0;
left: 0;
width: 100%;
height: 100%;
background: rgba(255, 255, 255, 0.95);
display: flex;
flex-direction: column;
align-items: center;
justify-content: center;
border-radius: 12px;
z-index: 100;
opacity: 0;
pointer-events: none;
transition: all 0.4s ease;
transform: scale(0.8);
backdrop-filter: blur(2px);
}

.game-over.show {
  opacity: 1;
  pointer-events: all;
  transform: scale(1);
}

.win-message {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: rgba(255, 255, 255, 0.95);
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  border-radius: 12px;
  z-index: 101;
  opacity: 0;
  pointer-events: none;
  transition: all 0.4s ease;
  transform: scale(0.8);
  backdrop-filter: blur(2px);
}

.win-message.show {
  opacity: 1;
  pointer-events: all;
  transform: scale(1);
}

.game-over-content, .win-content {
  text-align: center;
  animation: fadeInUp 0.5s ease-out;
}

@keyframes fadeInUp {
  0% {
    opacity: 0;
    transform: translateY(20px);
  }
  100% {
    opacity: 1;
  }
}
```

```
        transform: translateY(0);
    }
}

.game-over-text, .win-text {
    font-size: 2.5rem;
    font-weight: bold;
    margin-bottom: 15px;
    color: var(--primary-color);
    text-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

.final-score, .win-message-text {
    font-size: 1.2rem;
    margin-bottom: 25px;
    color: #666;
}

.final-score span {
    font-weight: bold;
    color: var(--primary-color);
}

.confetti {
    position: absolute;
    width: 10px;
    height: 10px;
    background-color: #f00;
    opacity: 0;
    z-index: 99;
}

.instructions {
    margin-top: 20px;
    font-size: 0.9rem;
    color: #666;
    text-align: center;
}

@media (max-width: 600px) {
    h1 {
        font-size: 2rem;
    }

    .score-box {
        padding: 8px 12px;
        min-width: 70px;
    }

    .score-title {
        font-size: 0.7rem;
    }

    .score-value {
        font-size: 1.2rem;
    }

    .tile {
        font-size: 1.5rem;
    }

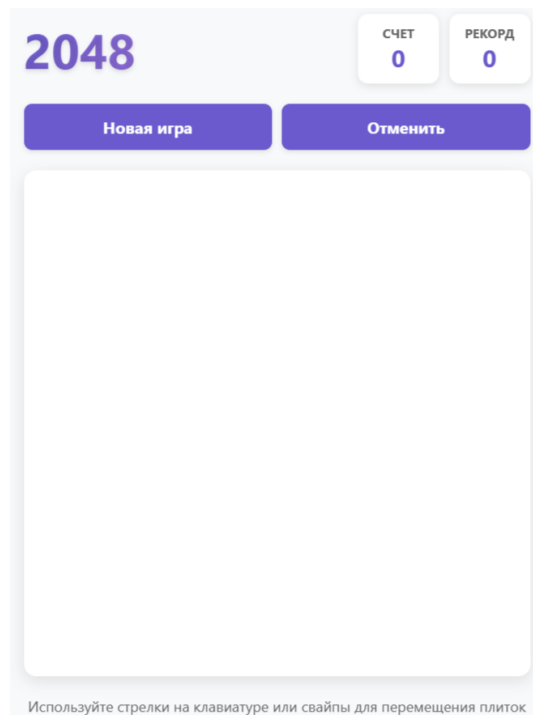
    .tile-128, .tile-256, .tile-512 {
        font-size: 1.3rem;
    }
}
```

```
}

.tile-1024, .tile-2048 {
  font-size: 1.1rem;
}

.game-over-text, .win-text {
  font-size: 2rem;
}

.final-score, .win-message-text {
  font-size: 1rem;
}
}
```



## Рисунок 2. Добавление CSS-стилей

Затем добавим игровое поле, состоящее из 16 ячеек (cell), организованных в сетку 4×4 с помощью элемента grid, где каждая ячейка представляет собой пустой контейнер с закругленными углами и полупрозрачным фоном, который будет динамически заполняться плитками (tile) в процессе игры через JavaScript, при этом сама сетка имеет фоновый цвет и внутренние отступы для визуального разделения ячеек.

Вставьте данный код после открывающего тега `<div class="grid-container">`.

[illegible]

```

<div class="cell"></div>
<div class="cell"></div>
<div class="cell"></div>
<div class="cell"></div>
<div class="cell"></div>
<div class="cell"></div>
<div class="cell"></div>
<div class="cell"></div>

```

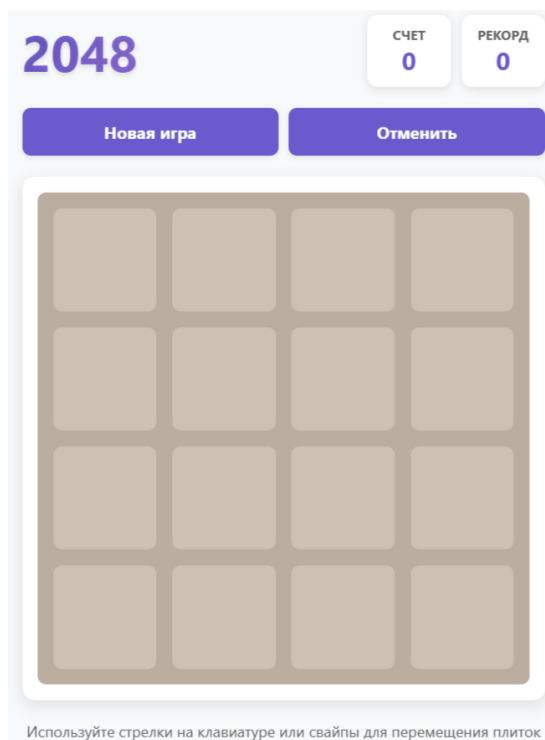


Рисунок 3. Создание числовых кнопок

Далее займёмся JavaScript-кодом, который реализует логику игры: после загрузки DOM получаем ссылки на все необходимые элементы интерфейса, инициализируем переменные для хранения состояния игры (игровое поле, счёт, рекорд, флаги окончания игры и победы), создаём функцию `initGame()` для подготовки игрового поля (очистка, добавление двух начальных плиток), функции `clearBoard()` и `renderBoard()` для управления отображением плиток, `addRandomTile()` для генерации новых плиток в случайных пустых ячейках, `saveState()` и `undo()` для реализации отмены хода, а также `move()` с обработкой направления движения, где для каждого направления (в данном случае 'up') мы обрабатываем столбцы, объединяем одинаковые плитки через `processLine()` и обновляем состояние доски, сохраняя флаг `moved` для отслеживания изменений. Код также включает обновление счёта, сохранение рекорда в `localStorage` и визуальное отображение всех изменений на игровом поле.

Вставьте данный код после открывающего тега `<script>`.

```

document.addEventListener('DOMContentLoaded', () => {
  const gridContainer = document.querySelector('.grid-container');
  const grid = document.querySelector('.grid');
  const scoreDisplay = document.getElementById('score');
  const bestScoreDisplay = document.getElementById('best-score');
  const newGameBtn = document.getElementById('new-game-btn');

```

```
const undoBtn = document.getElementById('undo-btn');
const tryAgainBtn = document.getElementById('try-again-btn');
const continueBtn = document.getElementById('continue-btn');
const gameOverScreen = document.getElementById('game-over');
const winMessageScreen = document.getElementById('win-message');
const finalScoreDisplay = document.getElementById('final-score');

let board = [];
let score = 0;
let bestScore = localStorage.getItem('bestScore2048') || 0;
let isGameOver = false;
let hasWon = false;
let previousStates = [];
let cellSize = 0;
let gapSize = 15;

bestScoreDisplay.textContent = bestScore;

function initGame() {
  board = [
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0]
  ];

  score = 0;
  isGameOver = false;
  hasWon = false;
  previousStates = [];

  updateScore();
  clearBoard();

  // Calculate cell size based on container
  const containerSize = gridContainer.offsetWidth;
  cellSize = (containerSize - gapSize * 5) / 4;

  // Add initial tiles
  addRandomTile();
  addRandomTile();

  renderBoard();
  gameOverScreen.classList.remove('show');
  winMessageScreen.classList.remove('show');
}

function clearBoard() {
  const tiles = document.querySelectorAll('.tile');
  tiles.forEach(tile => tile.remove());
}

function updateScore() {
  scoreDisplay.textContent = score;
  if (score > bestScore) {
    bestScore = score;
    bestScoreDisplay.textContent = bestScore;
    localStorage.setItem('bestScore2048', bestScore);
  }
}

function addRandomTile() {
  const emptyCells = [];
```

```
    for (let i = 0; i < 4; i++) {
      for (let j = 0; j < 4; j++) {
        if (board[i][j] === 0) {
          emptyCells.push({i, j});
        }
      }
    }
  }

  if (emptyCells.length > 0) {
    const randomCell = emptyCells[Math.floor(Math.random() * emptyCells.length)];
    board[randomCell.i][randomCell.j] = Math.random() < 0.9 ? 2 : 4;
    return true;
  }
  return false;
}

function renderBoard() {
  clearBoard();

  for (let i = 0; i < 4; i++) {
    for (let j = 0; j < 4; j++) {
      if (board[i][j] !== 0) {
        createTile(i, j, board[i][j]);
      }
    }
  }
}

function createTile(row, col, value) {
  const tile = document.createElement('div');
  tile.className = `tile tile-${value}`;
  if (value > 2048) tile.classList.add('tile-super');
  tile.textContent = value;

  // Calculate exact position
  const x = col * (cellSize + gapSize) + gapSize;
  const y = row * (cellSize + gapSize) + gapSize;

  tile.style.width = `${cellSize}px`;
  tile.style.height = `${cellSize}px`;
  tile.style.left = `${x}px`;
  tile.style.top = `${y}px`;

  grid.appendChild(tile);
}

function saveState() {
  previousStates.push({
    board: JSON.parse(JSON.stringify(board)),
    score: score
  });

  if (previousStates.length > 5) {
    previousStates.shift();
  }
}

function undo() {
  if (previousStates.length > 0) {
    const state = previousStates.pop();
    board = state.board;
    score = state.score;
  }
}
```

```

        updateScore();
        renderBoard();
        isGameOver = false;
        gameOverScreen.classList.remove('show');
    }
}

function move(direction) {
    if (isGameOver) return;

    saveState();

    let moved = false;
    const newBoard = JSON.parse(JSON.stringify(board));
    let scoreIncrease = 0;

    if (direction === 'up') {
        for (let j = 0; j < 4; j++) {
            const column = [board[0][j], board[1][j], board[2][j], board[3][j]];
            const { newLine, score } = processLine(column);
            scoreIncrease += score;

            for (let i = 0; i < 4; i++) {
                newBoard[i][j] = newLine[i];
                if (newBoard[i][j] !== board[i][j]) {
                    moved = true;
                }
            }
        }
    }
}

```

Теперь займемся реализацией обработки остальных направлений движения: для движения вниз (down) переворачиваем столбцы перед обработкой и возвращаем обратный порядок после объединения плиток, для движения влево (left) обрабатываем каждую строку напрямую, а для движения вправо (right) предварительно переворачиваем строки, обрабатываем их и затем восстанавливаем исходный порядок, при этом во всех случаях проверяем, изменилось ли состояние доски (переменная moved), увеличиваем счет при слиянии плиток через scoreIncrease, и сохраняем новое состояние в массив newBoard, что позволяет корректно обрабатывать все возможные ходы игрока с единой логикой обработки линий через функцию processLine().

Вставьте данный код в продолжение тега <script>:

```

else if (direction === 'down') {
    for (let j = 0; j < 4; j++) {
        const column = [board[3][j], board[2][j], board[1][j], board[0][j]];
        const { newLine, score } = processLine(column);
        scoreIncrease += score;

        for (let i = 0; i < 4; i++) {
            newBoard[i][j] = newLine[3 - i];
            if (newBoard[i][j] !== board[i][j]) {
                moved = true;
            }
        }
    }
}
else if (direction === 'left') {

```

```

    for (let i = 0; i < 4; i++) {
      const row = [...board[i]];
      const { newLine, score } = processLine(row);
      scoreIncrease += score;

      for (let j = 0; j < 4; j++) {
        newBoard[i][j] = newLine[j];
        if (newBoard[i][j] !== board[i][j]) {
          moved = true;
        }
      }
    }
  }
}
else if (direction === 'right') {
  for (let i = 0; i < 4; i++) {
    const row = [...board[i]].reverse();
    const { newLine, score } = processLine(row);
    scoreIncrease += score;

    for (let j = 0; j < 4; j++) {
      newBoard[i][j] = newLine[3 - j];
      if (newBoard[i][j] !== board[i][j]) {
        moved = true;
      }
    }
  }
}
}

```

Затем реализуем завершающую часть обработки хода: если движение было зарегистрировано (`moved`), обновляем игровое поле и счёт, проверяем достижение победы (наличие плитки 2048) с показом соответствующего сообщения через `showWinMessage()`, добавляем новую случайную плитку и перерисовываем поле, после чего проверяем условие завершения игры через `isGameOverCheck()`, либо, если ход не привёл к изменениям, удаляем последнее сохранённое состояние из истории, что обеспечивает плавный игровой процесс с контролем победных и проигрышных условий.

Вставьте данный код в продолжение тега `<script>`:

```

if (moved) {
  board = newBoard;
  score += scoreIncrease;
  updateScore();

  // Проверка на победу (2048)
  if (!hasWon) {
    for (let i = 0; i < 4; i++) {
      for (let j = 0; j < 4; j++) {
        if (board[i][j] === 2048) {
          hasWon = true;
          showWinMessage();
        }
      }
    }
  }
}

addRandomTile();
renderBoard();

```

```

        if (isGameOverCheck()) {
            gameOver();
        }
    } else {
        previousStates.pop();
    }
}

```

Завершим реализацию игры основными функциями: `processLine()` обрабатывает линии плиток, объединяя одинаковые значения и подсчитывая очки, `isGameOverCheck()` определяет окончание игры при заполнении поля без возможных ходов, `gameOver()` и `showWinMessage()` выводят соответствующие сообщения с анимацией конфетти. Функция `createConfetti()` создает праздничный эффект из разноцветных элементов с рандомной анимацией. Обработчики событий реагируют на клавиатуру (стрелки) и сенсорные жесты (свайпы), а кнопки управления запускают новую игру, отмену хода или продолжение после победы. Игра инициализируется вызовом `initGame()` при загрузке страницы, обеспечивая полный игровой цикл с интерактивным взаимодействием.

Вставьте данный код в продолжение тега `<script>`:

```

function processLine(line) {
    let filtered = line.filter(cell => cell !== 0);
    let newLine = [];
    let score = 0;
    let i = 0;

    while (i < filtered.length) {
        if (i < filtered.length - 1 && filtered[i] === filtered[i + 1]) {
            newLine.push(filtered[i] * 2);
            score += filtered[i] * 2;
            i += 2;
        } else {
            newLine.push(filtered[i]);
            i += 1;
        }
    }

    while (newLine.length < 4) {
        newLine.push(0);
    }

    return { newLine, score };
}

function isGameOverCheck() {
    for (let i = 0; i < 4; i++) {
        for (let j = 0; j < 4; j++) {
            if (board[i][j] === 0) {
                return false;
            }
        }
    }
}

for (let i = 0; i < 4; i++) {
    for (let j = 0; j < 4; j++) {
        if (j < 3 && board[i][j] === board[i][j + 1]) {

```

```

        return false;
    }
    if (i < 3 && board[i][j] === board[i + 1][j]) {
        return false;
    }
}
}

return true;
}

function gameOver() {
    isGameOver = true;
    finalScoreDisplay.textContent = score;
    gameOverScreen.classList.add('show');
    createConfetti();
}

function showWinMessage() {
    winMessageScreen.classList.add('show');
    createConfetti();
}

function createConfetti() {
    const colors = ['#ff0000', '#00ff00', '#0000ff', '#ffff00', '#ff00ff', '#00ffff'];
    const container = document.querySelector('.game-container');

    for (let i = 0; i < 100; i++) {
        const confetti = document.createElement('div');
        confetti.className = 'confetti';
        confetti.style.backgroundColor = colors[Math.floor(Math.random() * colors.length)];
        confetti.style.left = Math.random() * 100 + '%';
        confetti.style.top = -10 + 'px';
        confetti.style.width = Math.random() * 10 + 5 + 'px';
        confetti.style.height = Math.random() * 10 + 5 + 'px';
        confetti.style.borderRadius = Math.random() > 0.5 ? '50%' : '0';
        container.appendChild(confetti);

        const animationDuration = Math.random() * 3 + 2;

        const keyframes = [
            {
                transform: `translate(${Math.random() * 200 - 100}px, 0) rotate(0deg)`,
                opacity: 0
            },
            {
                transform: `translate(${Math.random() * 200 - 100}px, ${Math.random() * 100 + 100}px)
rotate(${Math.random() * 360}deg)`,
                opacity: 1
            },
            {
                transform: `translate(${Math.random() * 200 - 100}px, ${Math.random() * 100 + 300}px)
rotate(${Math.random() * 720}deg)`,
                opacity: 0
            }
        ];

        const options = {
            duration: animationDuration * 1000,
            easing: 'cubic-bezier(0.1, 0.8, 0.2, 1)',
            fill: 'forwards'
        };

```

```
        confetti.animate(keyframes, options);

        setTimeout(() => {
            confetti.remove();
        }, animationDuration * 1000);
    }
}

// Event listeners
document.addEventListener('keydown', (e) => {
    if (e.key === 'ArrowUp') {
        move('up');
    } else if (e.key === 'ArrowDown') {
        move('down');
    } else if (e.key === 'ArrowLeft') {
        move('left');
    } else if (e.key === 'ArrowRight') {
        move('right');
    }
});

newGameBtn.addEventListener('click', initGame);
undoBtn.addEventListener('click', undo);
tryAgainBtn.addEventListener('click', initGame);
continueBtn.addEventListener('click', () => {
    winMessageScreen.classList.remove('show');
});

// Touch events
let touchStartX = 0;
let touchStartY = 0;

document.addEventListener('touchstart', (e) => {
    touchStartX = e.touches[0].clientX;
    touchStartY = e.touches[0].clientY;
});

document.addEventListener('touchend', (e) => {
    const touchEndX = e.changedTouches[0].clientX;
    const touchEndY = e.changedTouches[0].clientY;

    const dx = touchEndX - touchStartX;
    const dy = touchEndY - touchStartY;

    if (Math.abs(dx) > Math.abs(dy)) {
        if (dx > 50) {
            move('right');
        } else if (dx < -50) {
            move('left');
        }
    } else {
        if (dy > 50) {
            move('down');
        } else if (dy < -50) {
            move('up');
        }
    }
});

// Initialize
initGame();
});
```

Теперь игра полностью завершена и готова к использованию: реализована логика обработки движений во всех направлениях с объединением плиток и подсчётом очков, добавлена система сохранения рекордов через localStorage, реализованы экраны победы (при достижении 2048) и поражения (при заполнении поля) с анимацией конфетти, предусмотрены функции отмены хода и новой игры, а также адаптивное управление как с клавиатуры (стрелки), так и для сенсорных устройств (свайпы). Игра автоматически инициализируется при загрузке страницы, имеет визуально приятный интерфейс с анимациями и полностью соответствует классическим правилам 2048, позволяя игрокам наслаждаться процессом как на компьютерах, так и на мобильных устройствах.

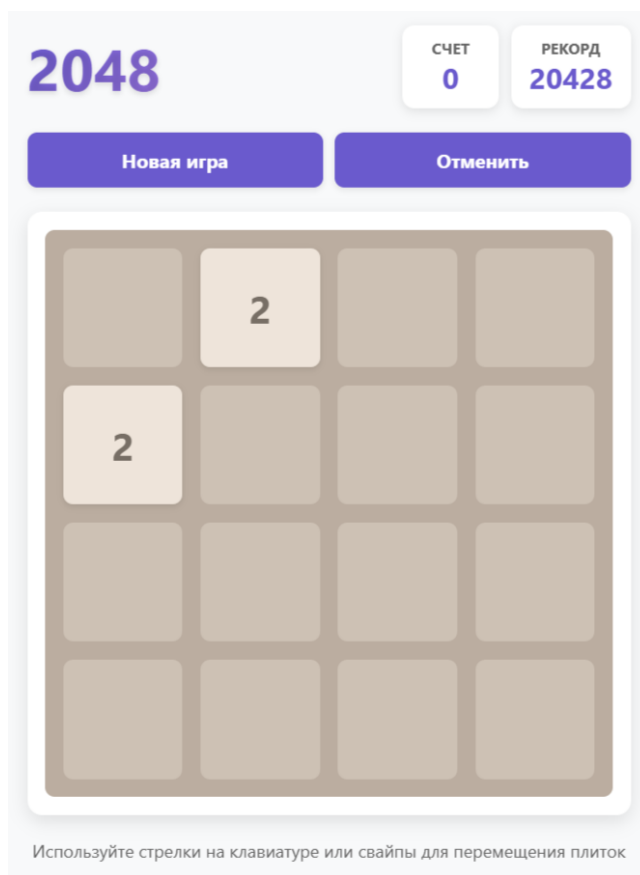


Рисунок 4. Игра

При достижении плитки номиналом 2048 игра не завершается, а отмечает это событие как победу: появляется праздничное сообщение "Победа!", после чего игрок может продолжить игру, чтобы набрать ещё больше очков или попытаться достичь более высоких значений плиток. Это классическое поведение игры 2048, где основная цель - получить плитку 2048, но можно играть дальше до полного заполнения поля. Текущая реализация корректно обрабатывает этот сценарий - при первом появлении плитки 2048 активируется победный экран, но игровой процесс не блокируется, сохраняя все возможности для дальнейшего развития игры.

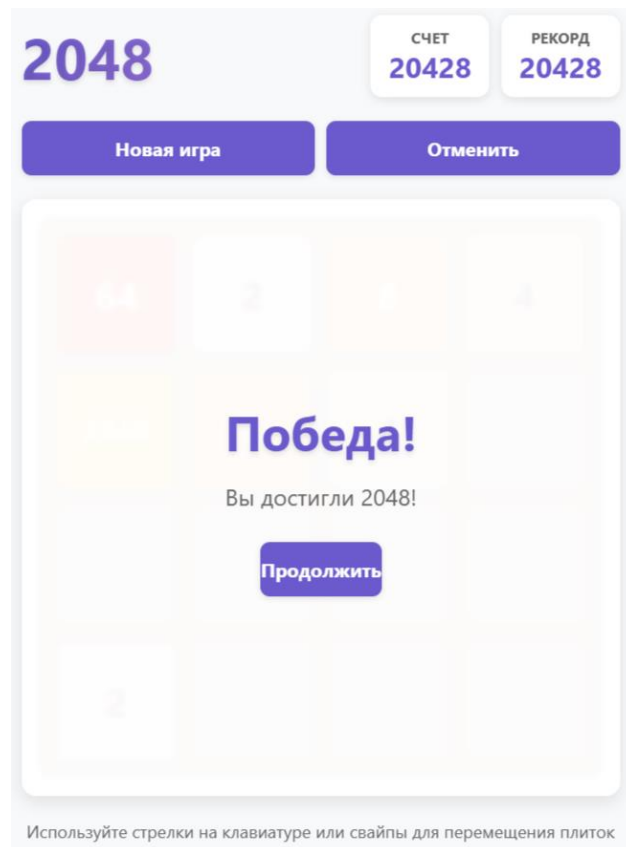


Рисунок 5. Завершение игры

Игра завершается, когда на поле не остается свободных клеток и невозможно сделать ни одного хода для объединения плиток. В этом случае срабатывает функция `isGameOverCheck()`, которая проверяет два условия: отсутствие нулевых значений (пустых клеток) на игровом поле и невозможность слияния соседних плиток (по горизонтали и вертикали). Если оба условия выполняются, вызывается функция `gameOver()`, которая отображает экран окончания игры с итоговым счетом и анимацией конфетти. Игроку предлагается начать новую игру кнопкой "Попробовать снова". Это классическое условие завершения 2048, обеспечивающее честный и предсказуемый финал игры.

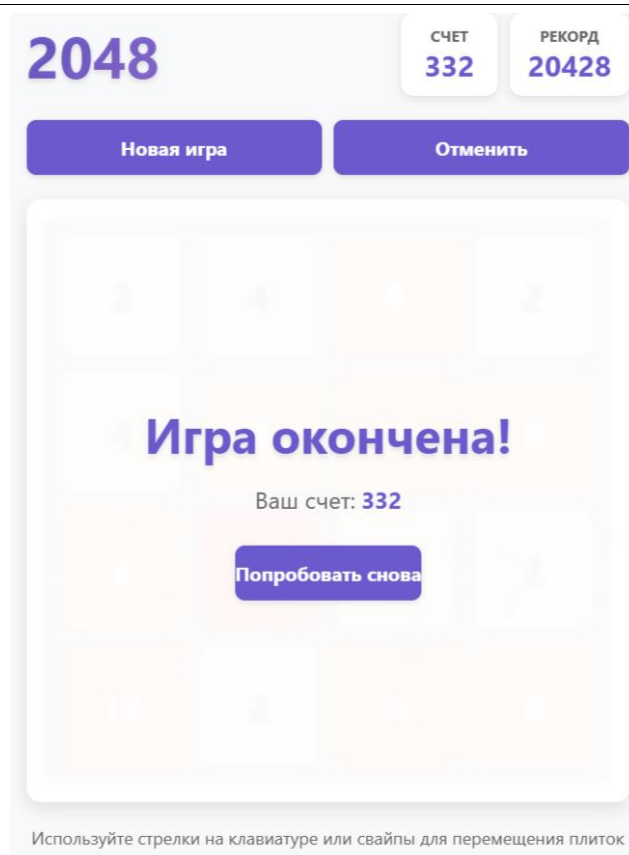


Рисунок 6. Завершение игры

#### 4 Выводы

В данной работе реализована классическая браузерная игра 2048, демонстрирующая эффективность современных веб-технологий для создания интерактивных логических игр. Проект сочетает в себе развлекательную и образовательную ценность: игровой процесс развивает стратегическое мышление, умение анализировать и прогнозировать последствия ходов, а также тренирует навыки быстрого принятия решений. Реализованные механизмы делают игру увлекательной и мотивирующей к совершенствованию. Особую практическую значимость проект представляет для начинающих разработчиков, так как наглядно иллюстрирует ключевые аспекты frontend-разработки: работу с DOM, обработку событий, управление состоянием приложения и создание анимаций. Предложенное решение может быть использовано в образовательных целях для изучения основ веб-программирования, а также как основа для создания более сложных игровых проектов.

#### Библиографический список

1. Рогатин С. В., Антошкин В. А. Разработка компьютерных игр по технологии html5 //Информатика и прикладная математика. 2015. №. 21. С. 12-16.
2. Фёдоров И. А., Кузьменкова Е. Ю. Разработка игрового приложения «league of football heads» с использованием html5, css3, javascript //Новые

- математические методы и компьютерные технологии в проектировании, производстве и научных исследованиях. 2018. С. 311.
3. Королюк Я. А., Куликова Н. Ю. Опыт разработки и использования интерактивных образовательных ресурсов на основе html5 для онлайн-обучения школьников информатике //XXV Всероссийская студенческая научно-практическая конференция Нижневартковского государственного университета. 2023. С. 48-55.
  4. Размологов Д. М., Панина А. С., Захаров П. В. Использование html5 для разработки электронного учебника //Современные тенденции развития науки и технологий. 2016. №. 4-4. С. 92-94.
  5. Шегаев И. С., Полозов А. Г. Новейший политологический глоссарий: основные категории, понятия, термины. Общество с ограниченной ответственностью Издательство Перо, 2015. 242 с.