

Разработка кроссплатформенной игры «Тетрис» на чистом JavaScript: от проектирования до реализации

Екимова Яна Сергеевна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье рассматривается методика создания классической игры Тетрис с применением технологий HTML5 Canvas, CSS3 и JavaScript. Разработанное решение позволяет воссоздать полноценный игровой процесс с реалистичной механикой падения и вращения фигур, системой коллизий, динамической визуализацией и адаптивным управлением для различных устройств. Особое внимание уделено реализации алгоритмов генерации случайных тетрамино, проверки заполнения линий и прогрессивного увеличения скорости игры. Представленное решение демонстрирует эффективное сочетание объектно-ориентированного подхода с современными веб-технологиями для создания кроссплатформенных игровых приложений. Материал предназначен для преподавателей и студентов, заинтересованных в применении современных технологий в обучении и стремящихся сделать процесс образовательного взаимодействия более эффективным и увлекательным.

Ключевые слова: Тетрис, HTML5, CSS, JavaScript.

Development of the Tetris cross-platform game in pure JavaScript: from design to implementation

Ekimova Yana Sergeevna

Sholom Aleichem Priamurskiy State University

Student

Abstract

The article discusses the technique of creating the classic Tetris game using HTML5 Canvas, CSS3 and JavaScript technologies. The developed solution allows you to recreate a full-fledged gameplay with realistic mechanics of falling and rotating shapes, a collision system, dynamic visualization and adaptive controls for various devices. Special attention is paid to the implementation of algorithms for generating random tetraminos, checking the filling of lines and progressively increasing the speed of the game. The presented solution demonstrates an effective combination of an object-oriented approach with modern web technologies for creating cross-platform gaming applications. The material is intended for teachers and students interested in using modern technologies in teaching and striving to make the process of educational interaction more effective and exciting.

Keywords: Tetris, HTML5, CSS, JavaScript.

1 Введение

1.1 Актуальность

В современном мире веб-технологий разработка игровых приложений на HTML5 и JavaScript приобретает особую значимость, поскольку позволяет создавать кроссплатформенные проекты, работающие непосредственно в браузере без необходимости установки дополнительного программного обеспечения. Актуальность реализации классической игры Тетрис заключается в том, что она служит идеальным учебным полигоном для освоения ключевых аспектов программирования: работы с DOM-структурами, обработки пользовательского ввода, реализации игровой логики и анимации, а также демонстрирует эффективное применение алгоритмов для проверки коллизий и оптимизации производительности. Кроме того, возрождение интереса к ретро-играм и их адаптация под современные веб-стандарты отвечает запросам широкой аудитории пользователей, обеспечивая доступность и сохранение культурного наследия игровой индустрии.

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]. Я. А. Королук, Н. Ю. Куликова 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 создание полнофункциональной веб-версии классической игры Тетрис с использованием языка HTML5.

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 index.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 красочной версии классической игры Тетрис. Документ начинается со стандартной декларации <!DOCTYPE html> и разметки (html, head, body), где в секции head указаны метаданные: кодировка UTF-8, viewport для адаптивности на мобильных устройствах, и заголовок страницы. В теле документа размещен основной контейнер игры (game-container), содержащий игровое поле (game-board) для отрисовки тетрамино, панель управления с элементами: кнопка-шестеренка для вызова справки по управлению, блоки отображения счета и уровня, область предпросмотра следующей фигуры, а также кнопки управления игровым процессом (Старт, Пауза, Сброс). В нижней части контейнера расположено широкое центральное оповещение о завершении игры (game-over) с отображением финального счета и кнопкой рестарта. Структура организована семантически, готова для подключения CSS-стилей и JavaScript-логики, обеспечивающей игровую механику, обработку пользовательского ввода и визуализацию игровых элементов на холсте.

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Тетрис</title>
  <style>
  </style>
</head>
<body>
  <div class="game-container">
    <div id="game-board"></div>

    <div class="controls">
      <!-- Кнопка-шестеренка -->
      <button class="gear-btn" id="help-btn">⚙️ </button>
      <div class="help-content" id="help-content">
        <p><strong>← →</strong> - Движение влево/вправо</p>
        <p><strong>↑</strong> - Поворот фигуры</p>
        <p><strong>↓</strong> - Ускорение падения</p>
        <p><strong>Пробел</strong> - Мгновенное падение</p>
        <p><strong>P</strong> - Пауза/Продолжить</p>
      </div>

      <div class="score-board">
        <h3>🎯 СЧЕТ</h3>
      </div>
    </div>
  </div>
```

```

<div id="score" class="score">0</div>
<div id="level">☑ Уровень: 1</div>
</div>

<div class="next-piece">
  <h3>🧊 СЛЕДУЮЩАЯ</h3>
  <div id="next-board"></div>
</div>

<button id="start-btn">🚀 СТАРТ</button>
<button id="pause-btn">⏸ ПАУЗА</button>
<button id="reset-btn">🔄 СБРОС</button>
</div>

<!-- Широкое центральное оповещение -->
<div class="game-over" id="game-over">
  <h2>ИГРА ОКОНЧЕНА</h2>
  <p>Ваш результат:</p>
  <div class="final-score" id="final-score">0</div>
  <p>Попробуйте еще раз!</p>
  <button class="restart-btn" onclick="resetGame()">🎮 Играть снова</button>
</div>
</div>

<script>

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

```

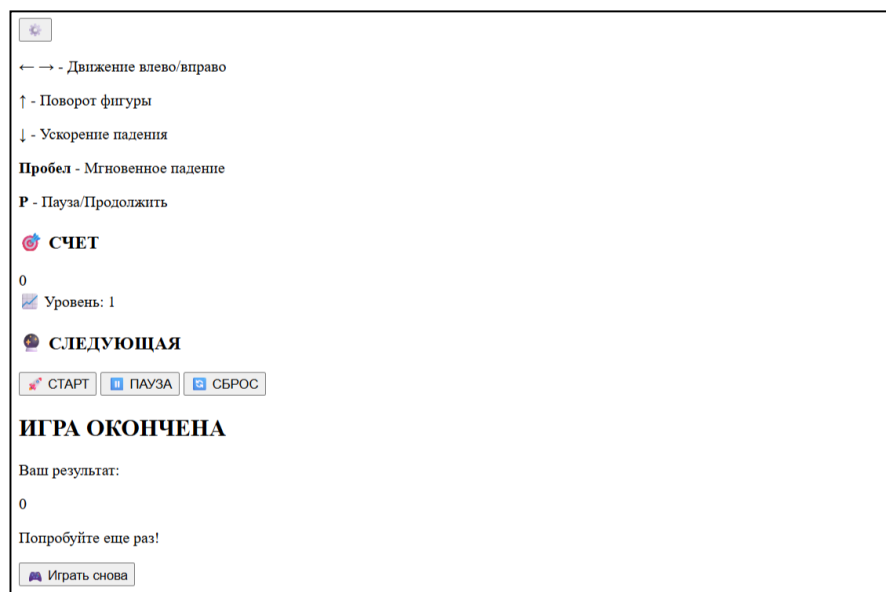


Рисунок 1. Основная структура HTML-документа

Далее добавим блок CSS-стилей, который используется для создания визуально привлекательного и функционального интерфейса игры Тетрис.

Стили начинаются с базового сброса отступов и применения модели box-sizing для всех элементов. Тело страницы оформлено градиентным фоном с глубокими синими оттенками и центрированием содержимого. Основной игровой контейнер выполнен в полупрозрачном темном стиле с красной границей, скругленными углами и эффектами размытия. Игровое поле представляет собой сетку 20×10 ячеек с темным фоном и тонкой границей, дополненную псевдоэлементом с градиентным наложением. Ячейки поля имеют плавные переходы и эффекты свечения при заполнении. Панель управления включает стилизованные кнопки с градиентными фонами и анимациями при наведении, включая специальную кнопку-шестеренку для справки с вращательной анимацией. Оповещение о завершении игры выполнено в виде широкого центрального модального окна с пульсирующей анимацией, крупным текстом и контрастными цветами. Для различных фигур тетрамино заданы уникальные градиентные окраски, а также реализованы плавные анимации появления, пульсации и свечения элементов интерфейса.

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

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    background: linear-gradient(135deg, #1a1a2e, #16213e, #0f3460);
    font-family: 'Arial', sans-serif;
    color: white;
    overflow: hidden;
}

.game-container {
    display: flex;
    gap: 30px;
    padding: 30px;
    background: rgba(0, 0, 0, 0.7);
    border-radius: 20px;
    border: 3px solid #ff5252;
    box-shadow: 0 0 50px rgba(255, 82, 82, 0.4);
    backdrop-filter: blur(10px);
    position: relative;
}

#game-board {
    width: 300px;
```

```
height: 600px;
display: grid;
grid-template-rows: repeat(20, 1fr);
grid-template-columns: repeat(10, 1fr);
gap: 1px;
background: #111;
border: 3px solid #ff5252;
border-radius: 10px;
position: relative;
overflow: hidden;
}

#game-board::before {
  content: "";
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background: linear-gradient(45deg, rgba(255,82,82,0.1), rgba(255,255,255,0.05));
  pointer-events: none;
}

.cell {
  background: #222;
  border: 1px solid #333;
  border-radius: 2px;
  transition: all 0.1s ease;
}

.filled {
  border: none;
  box-shadow: inset 0 0 10px rgba(255, 255, 255, 0.3);
}

.controls {
  display: flex;
  flex-direction: column;
  gap: 20px;
  width: 200px;
  position: relative;
}

.score-board, .next-piece, .help-section {
  background: linear-gradient(135deg, rgba(255,82,82,0.15), rgba(255,82,82,0.05));
  padding: 20px;
  border-radius: 15px;
  text-align: center;
  border: 2px solid rgba(255, 82, 82, 0.3);
  backdrop-filter: blur(5px);
}
```

```
.score {
  font-size: 28px;
  font-weight: bold;
  color: #ff5252;
  margin: 10px 0;
  text-shadow: 0 0 10px rgba(255, 82, 82, 0.5);
}

#next-board {
  width: 120px;
  height: 120px;
  display: grid;
  grid-template-rows: repeat(4, 1fr);
  grid-template-columns: repeat(4, 1fr);
  gap: 1px;
  margin: 10px auto;
  background: #111;
  border-radius: 8px;
  border: 2px solid #ff5252;
}

button {
  padding: 15px 25px;
  background: linear-gradient(135deg, #ff5252, #ff7b7b);
  border: none;
  border-radius: 10px;
  color: white;
  font-size: 16px;
  font-weight: bold;
  cursor: pointer;
  transition: all 0.3s ease;
  text-transform: uppercase;
  letter-spacing: 1px;
}

button:hover {
  transform: translateY(-3px);
  box-shadow: 0 5px 20px rgba(255, 82, 82, 0.6);
  background: linear-gradient(135deg, #ff7b7b, #ff5252);
}

/* Стилль для кнопки-шестеренки */
.gear-btn {
  position: absolute;
  top: 10px;
  right: 10px;
  width: 40px;
  height: 40px;
  padding: 0;
  border-radius: 50%;
  background: linear-gradient(135deg, #4CAF50, #45a049);
  display: flex;
```

```
        align-items: center;
        justify-content: center;
        font-size: 20px;
        z-index: 10;
    }

    .gear-btn:hover {
        background: linear-gradient(135deg, #45a049, #4CAF50);
        box-shadow: 0 5px 20px rgba(76, 175, 80, 0.6);
        animation: rotate 2s linear infinite;
    }

    .help-content {
        display: none;
        background: rgba(0, 0, 0, 0.95);
        padding: 20px;
        border-radius: 15px;
        border: 2px solid #4CAF50;
        position: absolute;
        top: 60px;
        right: 0;
        width: 200px;
        z-index: 10;
        text-align: left;
        box-shadow: 0 5px 25px rgba(0, 0, 0, 0.5);
    }

    .help-content.show {
        display: block;
        animation: fadeIn 0.3s ease;
    }

    .help-content p {
        margin: 8px 0;
        font-size: 14px;
        color: #ccc;
    }

    .help-content strong {
        color: #4CAF50;
    }

    /* Стили для фигур */
    .piece-I { background: linear-gradient(135deg, #00ffff, #0088ff); }
    .piece-O { background: linear-gradient(135deg, #ffff00, #ffaa00); }
    .piece-T { background: linear-gradient(135deg, #ff00ff, #cc00ff); }
    .piece-S { background: linear-gradient(135deg, #00ff00, #00cc00); }
    .piece-Z { background: linear-gradient(135deg, #ff0000, #cc0000); }
    .piece-J { background: linear-gradient(135deg, #0000ff, #0000cc); }
    .piece-L { background: linear-gradient(135deg, #ff8800, #ff5500); }

    /* Game Over Overlay - широкое и по центру */
```

```
.game-over {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  width: 400px; /* Широкая панель */
  background: rgba(0, 0, 0, 0.97);
  padding: 50px 40px;
  border-radius: 20px;
  border: 4px solid #ff5252;
  text-align: center;
  z-index: 100;
  display: none;
  backdrop-filter: blur(15px);
  box-shadow: 0 0 60px rgba(255, 82, 82, 0.9);
  animation: pulse 2s infinite;
}

.game-over.show {
  display: block;
  animation: fadeInScale 0.5s ease;
}

.game-over h2 {
  font-size: 36px;
  color: #ff5252;
  margin-bottom: 25px;
  text-shadow: 0 0 20px rgba(255, 82, 82, 0.9);
  font-weight: bold;
  letter-spacing: 2px;
}

.game-over p {
  font-size: 22px;
  margin: 15px 0;
  color: #fff;
  font-weight: 500;
}

.final-score {
  font-size: 32px;
  color: #ffeb3b;
  font-weight: bold;
  text-shadow: 0 0 15px rgba(255, 235, 59, 0.8);
  margin: 20px 0;
  background: linear-gradient(135deg, rgba(255, 235, 59, 0.1), rgba(255, 235, 59,
0.05));
  padding: 15px;
  border-radius: 10px;
  border: 2px solid rgba(255, 235, 59, 0.3);
}
```

```
.restart-btn {
  margin-top: 25px;
  background: linear-gradient(135deg, #4CAF50, #45a049);
  padding: 16px 35px;
  font-size: 18px;
  font-weight: bold;
  border-radius: 12px;
}

.restart-btn:hover {
  background: linear-gradient(135deg, #45a049, #4CAF50);
  box-shadow: 0 5px 25px rgba(76, 175, 80, 0.7);
  transform: translateY(-2px);
}

@keyframes fadeInScale {
  from {
    opacity: 0;
    transform: translate(-50%, -50%) scale(0.8);
  }
  to {
    opacity: 1;
    transform: translate(-50%, -50%) scale(1);
  }
}

@keyframes pulse {
  0% {
    box-shadow: 0 0 40px rgba(255, 82, 82, 0.8);
    border-color: #ff5252;
  }
  50% {
    box-shadow: 0 0 60px rgba(255, 82, 82, 1);
    border-color: #ff7b7b;
  }
  100% {
    box-shadow: 0 0 40px rgba(255, 82, 82, 0.8);
    border-color: #ff5252;
  }
}

@keyframes rotate {
  from { transform: rotate(0deg); }
  to { transform: rotate(360deg); }
}

.glow {
  animation: glow 1s ease-in-out infinite alternate;
}

@keyframes glow {
  from { box-shadow: 0 0 10px #ff5252, 0 0 20px #ff5252; }
```

```

    to { box-shadow: 0 0 15px #ff5252, 0 0 30px #ff5252; }
  }

```

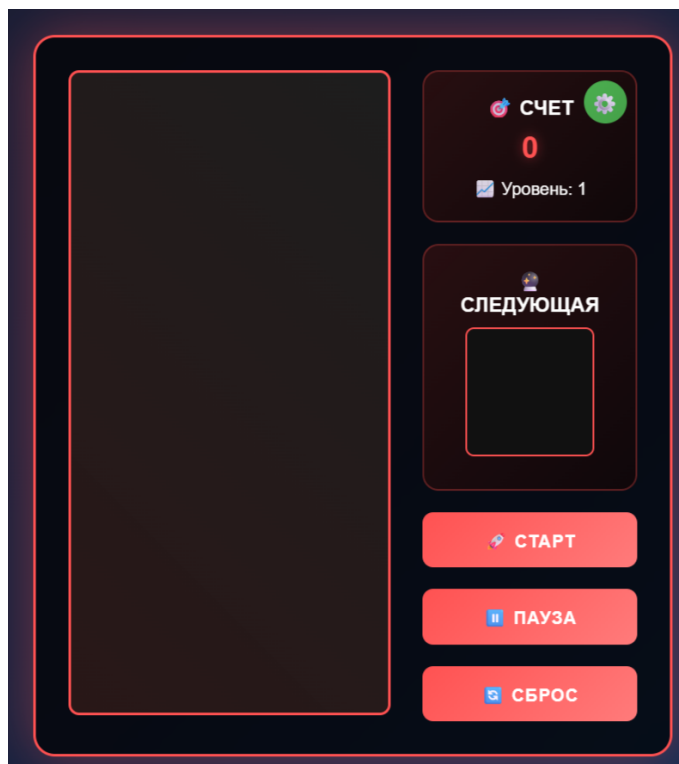


Рисунок 2. Добавление CSS-стилей

Затем перейдем к реализации игровой логики, начав с определения основных констант и структур данных. Объявляются размеры игрового поля: `BOARD_WIDTH = 10` и `BOARD_HEIGHT = 20`, задающие стандартную сетку тетриса, а также константа `EMPTY` для обозначения пустых ячеек. Создается объект `PIECES`, содержащий семь классических тетрамино (I, O, T, S, Z, J, L), где каждая фигура представлена двумерным массивом, определяющим ее форму (1 - наличие блока, 0 - пустота), и соответствующим CSS-классом для цветового оформления. Фигура I имеет линейную форму 4x4, O - квадратную 2x2, остальные - различные конфигурации 3x3, что обеспечивает разнообразие игровых ситуаций и соответствует канонической механике тетриса.

Вставим данный код после закрывающего тега `</div>` и после открывающего тега `<script>`:

```

// Константы игры
const BOARD_WIDTH = 10;
const BOARD_HEIGHT = 20;
const EMPTY = 'empty';

// Фигуры тетрамино
const PIECES = {
  I: {
    shape: [
      [0, 0, 0, 0],
      [1, 1, 1, 1],

```

```
[0, 0, 0, 0],
[0, 0, 0, 0]
],
color: 'piece-I'
},
O: {
  shape: [
    [1, 1],
    [1, 1]
  ],
  color: 'piece-O'
},
T: {
  shape: [
    [0, 1, 0],
    [1, 1, 1],
    [0, 0, 0]
  ],
  color: 'piece-T'
},
S: {
  shape: [
    [0, 1, 1],
    [1, 1, 0],
    [0, 0, 0]
  ],
  color: 'piece-S'
},
Z: {
  shape: [
    [1, 1, 0],
    [0, 1, 1],
    [0, 0, 0]
  ],
  color: 'piece-Z'
},
J: {
  shape: [
    [1, 0, 0],
    [1, 1, 1],
    [0, 0, 0]
  ],
  color: 'piece-J'
},
L: {
  shape: [
    [0, 0, 1],
    [1, 1, 1],
    [0, 0, 0]
  ],
  color: 'piece-L'
}
```

};

Далее инициализируем элементы DOM и игровые переменные, необходимые для функционирования игры. Получаем ссылки на все ключевые элементы интерфейса: игровое поле (game-board), поле предпросмотра следующей фигуры (next-board), элементы отображения счета и уровня, кнопки управления, а также вспомогательные элементы помощи и окно завершения игры. Объявляем основные игровые переменные: двумерный массив board для хранения состояния игрового поля, объекты текущей и следующей фигур, счетчики очков и уровня, флаги состояния игры и таймер игрового цикла.

Функция initGame() выполняет первоначальную настройку игры: создает игровое поле через createBoard(), инициализирует поле предпросмотра, генерирует первую фигуру и обновляет интерфейс. Функция createBoard() динамически создает сетку игрового поля 20×10, заполняя ее пустыми ячейками с соответствующими CSS-классами и атрибутами данных для последующего взаимодействия. Каждая ячейка получает данные о своей позиции (row и col), что обеспечивает эффективный доступ к элементам поля во время игрового процесса.

Вставим данный код в продолжение тега <script>:

```
// Элементы DOM
const gameBoard = document.getElementById('game-board');
const nextBoard = document.getElementById('next-board');
const scoreElement = document.getElementById('score');
const levelElement = document.getElementById('level');
const startBtn = document.getElementById('start-btn');
const pauseBtn = document.getElementById('pause-btn');
const resetBtn = document.getElementById('reset-btn');
const helpBtn = document.getElementById('help-btn');
const helpContent = document.getElementById('help-content');
const gameOver = document.getElementById('game-over');
const finalScore = document.getElementById('final-score');

// Игровые переменные
let board = [];
let currentPiece = null;
let nextPiece = null;
let score = 0;
let level = 1;
let gameInterval = null;
let isPaused = false;
let isGameOver = false;

// Инициализация игры
function initGame() {
  createBoard();
  createNextBoard();
  generateNewPiece();
}
```

```
        updateScore();
    }

    // Создание игрового поля
    function createBoard() {
        gameBoard.innerHTML = "";
        board = [];

        for (let row = 0; row < BOARD_HEIGHT; row++) {
            board[row] = [];
            for (let col = 0; col < BOARD_WIDTH; col++) {
                board[row][col] = EMPTY;
                const cell = document.createElement('div');
                cell.className = 'cell';
                cell.dataset.row = row;
                cell.dataset.col = col;
                gameBoard.appendChild(cell);
            }
        }
    }
}
```

Затем реализуем функции для работы с игровыми фигурами и отрисовки состояния игры. Функция `createNextBoard()` создает поле предпросмотра 4×4 для отображения следующей фигуры, инициализируя его пустыми ячейками. Функция `generateNewPiece()` управляет очередностью фигур: текущая фигура становится следующей из предыдущего цикла или генерируется новая, а для следующей фигуры создается новый случайный элемент. Позиция новой фигуры вычисляется для центрального размещения в верхней части поля, после чего проверяется коллизия для определения момента завершения игры.

Функция `getRandomPiece()` случайным образом выбирает одну из семи возможных фигур тетрамино, создавая глубокую копию ее матрицы и данных о цвете. Функция `showNextPiece()` визуализирует следующую фигуру в области предпросмотра, очищая поле и добавляя соответствующие CSS-классы для отображения формы и цвета.

Функция `draw()` выполняет полную перерисовку игрового состояния: сначала очищает все ячейки игрового поля, затем отображает статичные фигуры, уже находящиеся на поле, и поверх них рисует текущую активную фигуру с учетом ее позиции и формы. Проверка границ гарантирует, что отображаются только те части фигуры, которые находятся в пределах игрового поля.

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

```
// Создание поля для следующей фигуры
function createNextBoard() {
    nextBoard.innerHTML = "";
    for (let row = 0; row < 4; row++) {
        for (let col = 0; col < 4; col++) {
            const cell = document.createElement('div');
            cell.className = 'cell';
```

```
        nextBoard.appendChild(cell);
    }
}

// Генерация новой фигуры
function generateNewPiece() {
    currentPiece = nextPiece || getRandomPiece();
    nextPiece = getRandomPiece();

    currentPiece.x = Math.floor(BOARD_WIDTH / 2) -
Math.floor(currentPiece.shape[0].length / 2);
    currentPiece.y = 0;

    showNextPiece();

    if (checkCollision()) {
        gameOverFunc();
    }
}

// Получение случайной фигуры
function getRandomPiece() {
    const pieces = Object.keys(PIECES);
    const randomPiece = pieces[Math.floor(Math.random() * pieces.length)];
    return {
        shape: JSON.parse(JSON.stringify(PIECES[randomPiece].shape)),
        color: PIECES[randomPiece].color,
        x: 0,
        y: 0
    };
}

// Показ следующей фигуры
function showNextPiece() {
    const cells = nextBoard.querySelectorAll('.cell');
    cells.forEach(cell => cell.className = 'cell');

    for (let row = 0; row < nextPiece.shape.length; row++) {
        for (let col = 0; col < nextPiece.shape[row].length; col++) {
            if (nextPiece.shape[row][col]) {
                const index = row * 4 + col;
                cells[index].classList.add('filled', nextPiece.color);
            }
        }
    }
}

// Отрисовка игры
function draw() {
    const cells = gameBoard.querySelectorAll('.cell');
    cells.forEach(cell => cell.className = 'cell');
```

```

    for (let row = 0; row < BOARD_HEIGHT; row++) {
      for (let col = 0; col < BOARD_WIDTH; col++) {
        if (board[row][col] !== EMPTY) {
          const index = row * BOARD_WIDTH + col;
          cells[index].classList.add('filled', board[row][col]);
        }
      }
    }

    if (currentPiece) {
      for (let row = 0; row < currentPiece.shape.length; row++) {
        for (let col = 0; col < currentPiece.shape[row].length; col++) {
          if (currentPiece.shape[row][col]) {
            const boardRow = currentPiece.y + row;
            const boardCol = currentPiece.x + col;

            if (boardRow >= 0 && boardRow < BOARD_HEIGHT && boardCol
              >= 0 && boardCol < BOARD_WIDTH) {
              const index = boardRow * BOARD_WIDTH + boardCol;
              cells[index].classList.add('filled', currentPiece.color);
            }
          }
        }
      }
    }
  }
}

```

Следом пропишем основные игровые механизмы, обеспечивающие движение и взаимодействие фигур. Функция `checkCollision()` проверяет возможные столкновения текущей фигуры с границами игрового поля и другими фигурами, возвращая `true` при обнаружении коллизии. Функции движения `moveDown()`, `moveLeft()` и `moveRight()` изменяют позицию фигуры с последующей проверкой коллизий, при обнаружении которых откатывают изменение позиции и выполняют соответствующие действия (фиксацию фигуры, очистку линий и генерацию новой фигуры для движения вниз).

Функция `rotate()` реализует поворот фигуры на 90 градусов путем транспонирования и инверсии ее матрицы с созданием глубокой копии исходной формы для возможного отката при коллизии. Функция `lockPiece()` фиксирует текущую фигуру на игровом поле, перенося ее блоки в соответствующие позиции массива `board`. Функция `clearLines()` проверяет заполненные линии, удаляет их и добавляет новые пустые линии сверху, начисляя очки в зависимости от количества одновременно очищенных линий и текущего уровня.

Функция `updateGameSpeed()` динамически регулирует скорость игры в зависимости от уровня: с каждым новым уровнем интервал между автоматическими движениями фигуры вниз уменьшается на 100 мс, обеспечивая постепенное увеличение сложности. Обновление уровня отображается в интерфейсе с помощью эмодзи и текстового обозначения.

Вставим данный код в продолжение тега <script>:

```
// Проверка коллизий
function checkCollision() {
  for (let row = 0; row < currentPiece.shape.length; row++) {
    for (let col = 0; col < currentPiece.shape[row].length; col++) {
      if (currentPiece.shape[row][col]) {
        const boardRow = currentPiece.y + row;
        const boardCol = currentPiece.x + col;

        if (boardRow >= BOARD_HEIGHT ||
            boardCol < 0 ||
            boardCol >= BOARD_WIDTH ||
            (boardRow >= 0 && board[boardRow][boardCol] !== EMPTY)) {
          return true;
        }
      }
    }
  }
  return false;
}

// Движение фигуры вниз
function moveDown() {
  if (isPaused || isGameOver) return;

  currentPiece.y++;
  if (checkCollision()) {
    currentPiece.y--;
    lockPiece();
    clearLines();
    generateNewPiece();
  }
  draw();
}

// Движение влево
function moveLeft() {
  if (isPaused || isGameOver) return;

  currentPiece.x--;
  if (checkCollision()) {
    currentPiece.x++;
  }
  draw();
}

// Движение вправо
function moveRight() {
  if (isPaused || isGameOver) return;

  currentPiece.x++;
```

```
    if (checkCollision()) {
        currentPiece.x--;
    }
    draw();
}

// Поворот фигуры
function rotate() {
    if (isPaused || isGameOver) return;

    const originalShape = JSON.parse(JSON.stringify(currentPiece.shape));
    const rows = currentPiece.shape.length;
    const cols = currentPiece.shape[0].length;

    const rotated = [];
    for (let col = 0; col < cols; col++) {
        rotated[col] = [];
        for (let row = rows - 1; row >= 0; row--) {
            rotated[col][rows - 1 - row] = currentPiece.shape[row][col];
        }
    }

    currentPiece.shape = rotated;
    if (checkCollision()) {
        currentPiece.shape = originalShape;
    }
    draw();
}

// Фиксация фигуры на поле
function lockPiece() {
    for (let row = 0; row < currentPiece.shape.length; row++) {
        for (let col = 0; col < currentPiece.shape[row].length; col++) {
            if (currentPiece.shape[row][col]) {
                const boardRow = currentPiece.y + row;
                const boardCol = currentPiece.x + col;

                if (boardRow >= 0) {
                    board[boardRow][boardCol] = currentPiece.color;
                }
            }
        }
    }
}

// Очистка заполненных линий
function clearLines() {
    let linesCleared = 0;

    for (let row = BOARD_HEIGHT - 1; row >= 0; row--) {
        if (board[row].every(cell => cell !== EMPTY)) {
            board.splice(row, 1);
```

```

        board.unshift(Array(BOARD_WIDTH).fill(EMPTY));
        linesCleared++;
        row++;
    }
}

if (linesCleared > 0) {
    score += linesCleared * 100 * level;
    if (score >= level * 500) {
        level++;
        updateGameSpeed();
    }
    updateScore();
}
}

// Обновление скорости игры
function updateGameSpeed() {
    if (gameInterval) {
        clearInterval(gameInterval);
    }

    const speed = Math.max(100, 1000 - (level - 1) * 100);
    gameInterval = setInterval(moveDown, speed);

    levelElement.textContent = `🎮 Уровень: ${level}`;
}

```

Теперь осталось прописать завершающие функции управления игровым процессом и обработки пользовательского ввода. Функция `updateScore()` обновляет отображение текущего счета в интерфейсе. Функция `gameOverFunc()` обрабатывает завершение игры: устанавливает соответствующий флаг, останавливает игровой цикл, отображает финальный счет и показывает анимированное уведомление о конце игры.

Функция `resetGame()` полностью сбрасывает игровое состояние: обнуляет счет и уровень, снимает паузу, скрывает окно завершения игры, восстанавливает начальные надписи кнопок и переинициализирует игровые элементы. Функция `togglePause()` управляет состоянием паузы, изменяя текст кнопки для индикации текущего режима.

Обработчик клавиатуры `keydown` обеспечивает управление игрой с помощью стрелок для движения, пробела для мгновенного сброса фигуры и клавиши `P` для паузы. Кнопки управления на интерфейсе связываются с соответствующими функциями: старт игры, пауза/продолжение и сброс. Реализовано взаимодействие с кнопкой-шестеренкой для отображения/скрытия справки по управлению с обработкой кликов вне области подсказки для ее автоматического закрытия.

Завершает код инициализация игры при полной загрузке страницы, обеспечивающая готовность всех элементов к началу игрового процесса.

Вставим данный код в продолжение тега <script>:

```
// Обновление счета
function updateScore() {
    scoreElement.textContent = score;
}

// Game over с красивым уведомлением
function gameOverFunc() {
    isGameOver = true;
    clearInterval(gameInterval);
    finalScore.textContent = score;
    gameOver.classList.add('show');
}

// Сброс игры
function resetGame() {
    clearInterval(gameInterval);
    score = 0;
    level = 1;
    isPaused = false;
    isGameOver = false;
    gameOver.classList.remove('show');
    updateScore();
    levelElement.textContent = `🏁 Уровень: ${level}`;
    pauseBtn.textContent = '⏸ ПАУЗА';
    initGame();
    updateGameSpeed();
    draw();
}

// Пауза
function togglePause() {
    isPaused = !isPaused;
    pauseBtn.textContent = isPaused ? '▶ ПРОДОЛЖИТЬ' : '⏸ ПАУЗА';
}

// Управление с клавиатуры
document.addEventListener('keydown', (e) => {
    if (isGameOver) return;

    switch(e.key) {
        case 'ArrowLeft':
            moveLeft();
            break;
        case 'ArrowRight':
            moveRight();
            break;
        case 'ArrowDown':
            moveDown();
            break;
```

```
        case 'ArrowUp':
            rotate();
            break;
        case ' ':
            while (!checkCollision()) {
                currentPiece.y++;
            }
            currentPiece.y--;
            lockPiece();
            clearLines();
            generateNewPiece();
            draw();
            break;
        case 'p':
        case 'P':
            togglePause();
            break;
    }
});

// Обработчики кнопок
startBtn.addEventListener('click', () => {
    if (!gameInterval && !isGameOver) {
        updateGameSpeed();
    }
});

pauseBtn.addEventListener('click', togglePause);
resetBtn.addEventListener('click', resetGame);

// Помощь - открытие/закрытие по клику на шестеренку
helpBtn.addEventListener('click', (e) => {
    e.stopPropagation();
    helpContent.classList.toggle('show');
});

// Закрытие подсказки при клике вне ее области
document.addEventListener('click', (e) => {
    if (!helpContent.contains(e.target) && e.target !== helpBtn) {
        helpContent.classList.remove('show');
    }
});

// Инициализация игры при загрузке
window.addEventListener('load', () => {
    initGame();
    draw();
});
```

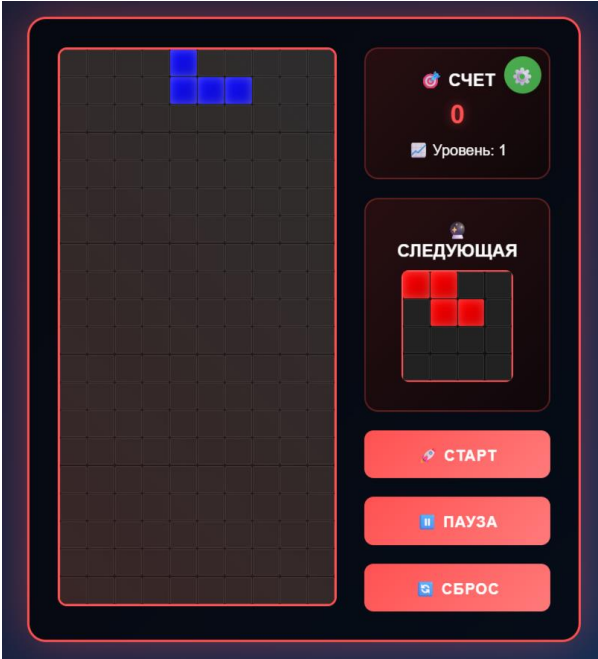


Рисунок 3. Главная страница

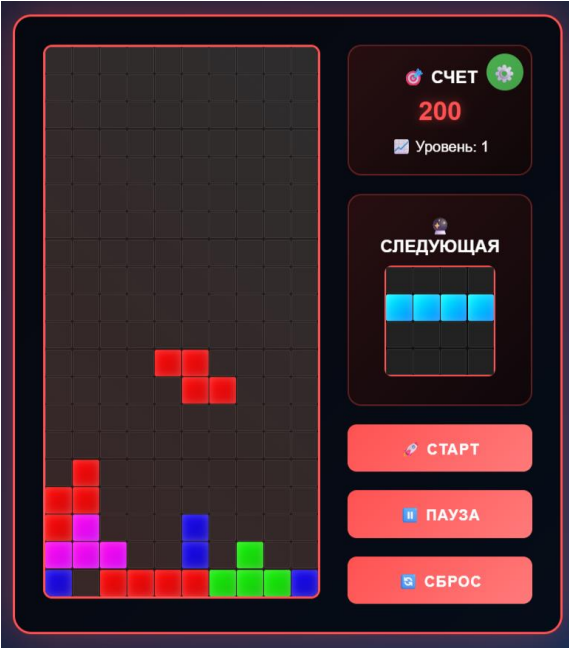


Рисунок 4. Процесс игры



Рисунок 5. Конец игры

4 Выводы

В данной работе подробно рассмотрено создание классической игры Тетрис на современном веб-стеке с использованием технологий HTML5, CSS3 и JavaScript, которая включает реалистичную механику падения и вращения фигур, систему коллизий, адаптивное управление для различных устройств и динамическую визуализацию игрового процесса. Реализованы ключевые механизмы оригинальной игры: случайная генерация тетрамино, очистка заполненных линий, прогрессивная система уровней сложности с увеличением скорости, а также система подсчета очков с бонусами за одновременное удаление нескольких линий. Разработан интуитивно понятный интерфейс с областью предпросмотра следующей фигуры, панелью управления и анимированными уведомлениями о завершении игры. Предложенное решение ориентировано на преподавателей и студентов, заинтересованных в применении современных веб-технологий для повышения эффективности образовательного процесса и превращения обучения в более увлекательный и интерактивный опыт.

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. Королук Я. А., Куликова Н. Ю. Опыт разработки и использования интерактивных образовательных ресурсов на основе html5 для онлайнобучения 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. М.: Перо, 2015. 242 с.