

Разработка адаптивного платформера на HTML5 Canvas с динамической генерацией контента и мультиплатформенным управлением

Екимова Яна Сергеевна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье представлен поэтапный процесс разработки кроссплатформенной веб-игры в жанре платформера с использованием современных веб-технологий (HTML5, CSS3, JavaScript). Рассмотрены ключевые аспекты реализации: динамическая генерация игрового контента, адаптивная физика персонажа, система управления для различных устройств и алгоритмы обработки столкновений. Особое внимание уделено механизмам прокрутки игрового мира, реализации трех уровней сложности и сохранению игрового прогресса. 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: HTML5 Canvas, адаптивная игра, платформер, динамическая генерация, веб-разработка, кроссплатформенное управление, игровая физика, уровни сложности, CSS, JavaScript.

Development of an adaptive HTML5 Canvas platformer with dynamic content generation and multiplatform management

Ekimova Yana Sergeevna

Sholom Aleichem Priamurskiy State University

Student

Abstract

The article presents a step-by-step process of developing a cross-platform web game in the platformer genre using modern web technologies (HTML5, CSS3, JavaScript). The key aspects of the implementation are considered: dynamic generation of game content, adaptive character physics, a control system for various devices, and collision processing algorithms. Special attention is paid to the mechanisms of scrolling the game world, the implementation of three levels of difficulty and the preservation of game progress. 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: HTML5 Canvas, adaptive game, platformer, dynamic generation, web development, cross-platform control, game physics, difficulty levels, CSS, JavaScript.

1 Введение

1.1 Актуальность

Актуальность исследования обусловлена стремительным развитием веб-технологий и растущим спросом на кросс-платформенные игровые решения, не требующие установки дополнительного программного обеспечения. Разработка HTML5-игр с использованием Canvas API представляет особый интерес, так как сочетает доступность веб-приложений с богатыми возможностями графического рендеринга, что особенно востребовано в условиях увеличения мобильного трафика и необходимости адаптации контента для различных устройств. Предлагаемое решение актуально для образовательной сферы как наглядный пример реализации игровой механики, для индустрии развлечений в качестве готового шаблона браузерной игры, а также для исследований в области пользовательского опыта, демонстрируя эффективные подходы к созданию интуитивного управления и динамической балансировки сложности. Особую значимость работа приобретает в контексте популярности Progressive Web Apps, требующих оптимизированной производительности при работе с графикой.

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. Королюк, 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 разработка адаптивного платформера на HTML5 Canvas с динамической генерацией контента и мультиплатформенным управлением 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 веб-игры в жанре платформера с удобными прыжками, которая включает подключение метатегов для корректного отображения на мобильных устройствах и задания области просмотра, заголовков страницы, пустой блок стилей для последующего оформления, а также разметку интерфейса с тремя экранами — начальным, игровым и экраном окончания игры. Интерфейс содержит элементы отображения очков, кнопки управления (пауза, перезапуск), виртуальные кнопки для управления персонажем на мобильных устройствах, выбор уровня сложности и информационные блоки. Игровое поле представлено через элемент `<canvas>`, что позволяет реализовать динамическую 2D-графику, а также предусмотрены элементы обратной связи — счёт, рекорд и кнопка повторной игры после проигрыша.

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

</style>
</head>
<body>
  <div id="score">Очки: 0</div>
  <div id="gameUI">
    <button id="pauseButton" class="ui-btn">Пауза</button>
    <button id="restartButton" class="ui-btn">Заново</button>
  </div>
  <canvas id="gameCanvas"></canvas>

  <div id="controls">
    <div id="leftBtn" class="control-btn">←</div>
    <div id="rightBtn" class="control-btn">→</div>
  </div>

  <div id="startScreen">
    <h1>Платформер с удобными прыжками</h1>
    <div id="instructions">
      <p>Прыгай по платформам и собирай монеты!</p>
      <p>Используй кнопки внизу или клавиши ← →</p>
      <p>Не падай вниз!</p>
    </div>
  </div>
```

```

<div id="levelSelect">
  <button class="level-btn active" data-level="easy">Легкий</button>
  <button class="level-btn" data-level="medium">Средний</button>
  <button class="level-btn" data-level="hard">Сложный</button>
</div>
<button id="startButton">Начать игру</button>
</div>

<div id="gameOverScreen">
  <h1>Игра окончена!</h1>
  <div id="finalScore">Ваш счет: 0</div>
  <div id="highScore">Рекорд: 0</div>
  <button id="restartGameButton" class="ui-btn" style="margin-top: 20px; font-size: 20px; padding:
15px 30px;">Играть снова</button>
</div>

<script>

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

```

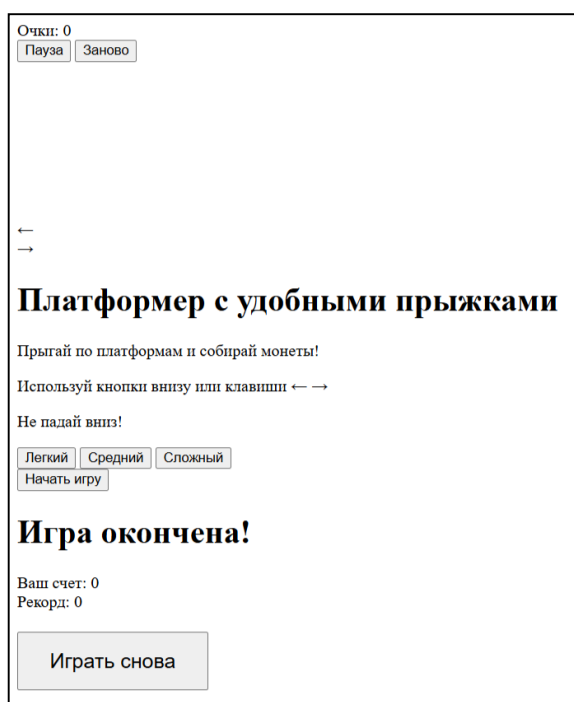


Рисунок 1. Основная структура HTML-документа

Далее добавим блок CSS-стилей, который используется для оформления интерфейса платформера с удобными прыжками и обеспечивает визуальную целостность и функциональность игрового приложения. Стили задают базовые настройки страницы: убирают отступы, скрывают прокрутку с помощью `overflow: hidden`, устанавливают фон с градиентом и шрифт для текста. Игровое поле настраивается через элемент `#gameCanvas` с фиксированным отображением и фоном небесно-голубого цвета. Элементы управления (`#controls`) размещаются в нижней части экрана по центру, включая виртуальные кнопки с закруглённой формой и белыми символами для управления персонажем. Блок счёта (`#score`) выводится в левом верхнем углу

с чётким стилем и тенью для лучшей читаемости. Панель пользовательского интерфейса (#gameUI) содержит кнопки «Пауза» и «Заново», расположенные в правом верхнем углу. Экраны начала игры и окончания уровня (#startScreen, #gameOverScreen) позиционируются поверх холста, имеют затемнённый фон и содержат стилизованные кнопки выбора уровня сложности и стартовой кнопки. Все элементы оформлены с учётом мобильной адаптивности и удобства взаимодействия, что делает игру визуально привлекательной и интуитивно понятной для пользователя.

Данный код пропишем на следующей строчке после тега <title>Платформер с удобными прыжками</title> в секции <head> HTML-документа.

```
body {
  margin: 0;
  overflow: hidden;
  background: linear-gradient(to bottom, #87CEEB, #E0F7FA);
  font-family: Arial, sans-serif;
  touch-action: manipulation;
}
#gameCanvas {
  display: block;
  background-color: #87CEEB;
}
#score {
  position: absolute;
  top: 10px;
  left: 10px;
  font-size: 24px;
  color: white;
  text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
}
#controls {
  position: absolute;
  bottom: 20px;
  width: 100%;
  display: flex;
  justify-content: center;
  gap: 20px;
}
.control-btn {
  width: 80px;
  height: 80px;
  background-color: rgba(255, 255, 255, 0.3);
  border-radius: 50%;
  display: flex;
  justify-content: center;
  align-items: center;
  font-size: 24px;
  color: white;
  border: 2px solid white;
  user-select: none;
  -webkit-tap-highlight-color: transparent;
}
#gameUI {
  position: absolute;
  top: 10px;
  right: 10px;
  display: flex;
```

```
        gap: 10px;
        flex-direction: column;
        align-items: flex-end;
    }
    .ui-btn {
        padding: 8px 15px;
        background-color: rgba(0, 0, 0, 0.5);
        color: white;
        border: none;
        border-radius: 5px;
        font-size: 16px;
        cursor: pointer;
        min-width: 100px;
        text-align: center;
    }
    #startScreen {
        position: absolute;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        background-color: rgba(0, 0, 0, 0.7);
        display: flex;
        flex-direction: column;
        justify-content: center;
        align-items: center;
        color: white;
        z-index: 10;
    }
    #levelSelect {
        margin: 20px 0;
        display: flex;
        gap: 10px;
    }
    .level-btn {
        padding: 10px 15px;
        background-color: #2196F3;
        color: white;
        border: none;
        border-radius: 5px;
        cursor: pointer;
    }
    .level-btn.active {
        background-color: #0b7dda;
        box-shadow: 0 0 10px rgba(255, 255, 255, 0.5);
    }
    #startButton {
        padding: 15px 30px;
        font-size: 20px;
        background-color: #4CAF50;
        color: white;
        border: none;
        border-radius: 10px;
        cursor: pointer;
        margin-top: 20px;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.3);
    }
    #instructions {
        text-align: center;
        max-width: 300px;
        margin-bottom: 20px;
    }
    #gameOverScreen {
```

```

position: absolute;
top: 0;
left: 0;
width: 100%;
height: 100%;
background-color: rgba(0, 0, 0, 0.7);
display: none;
flex-direction: column;
justify-content: center;
align-items: center;
color: white;
z-index: 10;
}

```

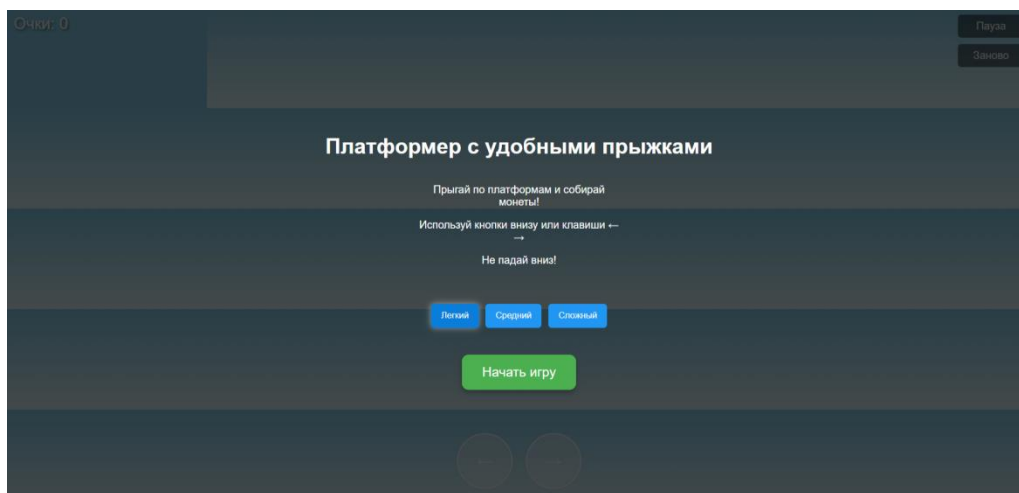


Рисунок 2. Добавление CSS-стилей

Теперь напомним код, который реализует подключение к ключевым элементам интерфейса платформера и инициализирует переменные для управления игровыми объектами и состоянием игры: объявляются константы для получения доступа к canvas-элементу и его контексту отрисовки, счётчику очков, кнопкам управления (начала игры, паузы, перезапуска, движения влево/вправо), а также к экранам приветствия и окончания игры с элементами отображения результата и рекорда. Этот блок обеспечивает взаимодействие между HTML-интерфейсом и JavaScript-логикой, формируя основу для динамического обновления состояния игры и реакции на действия пользователя.

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

```

const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const startScreen = document.getElementById('startScreen');
const startButton = document.getElementById('startButton');
const pauseButton = document.getElementById('pauseButton');
const restartButton = document.getElementById('restartButton');
const leftBtn = document.getElementById('leftBtn');
const rightBtn = document.getElementById('rightBtn');
const gameOverScreen = document.getElementById('gameOverScreen');
const finalScoreElement = document.getElementById('finalScore');

```

```
const highScoreElement = document.getElementById('highScore');  
const restartGameButton = document.getElementById('restartGameButton');  
const levelButtons = document.querySelectorAll('.level-btn');
```

Затем напишем код, который инициализирует основные игровые переменные и настраивает размеры холста под текущее окно браузера: функция `resizeCanvas()` устанавливает ширину и высоту элемента `<canvas>` равными размерам окна, обеспечивая адаптивность интерфейса; объявляются переменные для хранения счёта, рекорда, состояния игры (активна/приостановлена), а также создаются массивы для хранения платформ, монет и пружин. Объект `player` описывает параметры персонажа — его начальные координаты, размеры, скорость передвижения, силу прыжка, гравитацию и другие свойства, необходимые для физики движения и отрисовки на экране. Этот блок формирует основу игровой логики и подготовки к рендерингу сцены.

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

```
function resizeCanvas() {  
    canvas.width = window.innerWidth;  
    canvas.height = window.innerHeight;  
}  
resizeCanvas();  
  
// Игровые переменные  
let score = 0;  
let highScore = localStorage.getItem('highScore') || 0;  
let gameRunning = false;  
let gamePaused = false;  
let platforms = [];  
let coins = [];  
let springs = [];  
let currentLevel = 'easy';  
let player = {  
    x: canvas.width / 2 - 15,  
    y: canvas.height * 0.7 - 30,  
    width: 30,  
    height: 30,  
    velocityX: 0,  
    velocityY: 0,  
    speed: 6,  
    jumpForce: -16,  
    gravity: 0.4,  
    color: '#FF5722',  
    isMovingLeft: false,  
    isMovingRight: false,  
    canJump: true  
};
```

Теперь займемся настройкой игровых уровней и созданием классов игровых объектов: в объекте `levelSettings` определены параметры трех уровней сложности (`easy`, `medium`, `hard`), включая расстояние между платформами, их ширину, частоту появления монет и специальных платформ; класс `Platform` реализует различные типы платформ (обычные, пружинящие и разрушаемые) с уникальной отрисовкой для каждого типа; класс `Coin` отвечает за создание и

обработку столкновений с монетами, включая их визуализацию золотыми кругами; класс Spring реализует пружины, которые придают игроку дополнительный импульс, с проверкой столкновений и состоянием использования. Все классы содержат методы draw() для отрисовки и checkCollision() для обработки взаимодействий с игроком, используя Canvas API для визуализации элементов игры.

Вставьте данный код в продолжение тега <script>:

```
// Настройки уровней
const levelSettings = {
  easy: {
    platformGap: 80,
    platformWidth: 120,
    coinFrequency: 0.7,
    springFrequency: 0.3,
    bouncyFrequency: 0.1,
    breakableFrequency: 0.05,
    playerStartY: canvas.height * 0.7
  },
  medium: {
    platformGap: 100,
    platformWidth: 100,
    coinFrequency: 0.5,
    springFrequency: 0.2,
    bouncyFrequency: 0.2,
    breakableFrequency: 0.1,
    playerStartY: canvas.height * 0.7
  },
  hard: {
    platformGap: 120,
    platformWidth: 80,
    coinFrequency: 0.3,
    springFrequency: 0.1,
    bouncyFrequency: 0.3,
    breakableFrequency: 0.2,
    playerStartY: canvas.height * 0.7
  }
};

// Класс платформы
class Platform {
  constructor(x, y, width, type = 'normal') {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = 10;
    this.type = type;
    this.color = type === 'bouncy' ? '#9C27B0' :
      type === 'breakable' ? '#795548' : '#4CAF50';
  }

  draw() {
    ctx.fillStyle = this.color;
    ctx.fillRect(this.x, this.y, this.width, this.height);

    if (this.type === 'normal') {
      ctx.fillStyle = '#388E3C';
      for (let i = 0; i < this.width; i += 5) {
        if (i % 10 === 0) ctx.fillRect(this.x + i, this.y, 5, 3);
      }
    }
  }
}
```

```
    }
  } else if (this.type === 'bouncy') {
    ctx.fillStyle = '#7B1FA2';
    ctx.beginPath();
    ctx.moveTo(this.x, this.y + this.height);
    ctx.lineTo(this.x + this.width, this.y + this.height);
    ctx.lineTo(this.x + this.width - 5, this.y);
    ctx.lineTo(this.x + 5, this.y);
    ctx.closePath();
    ctx.fill();
  } else if (this.type === 'breakable') {
    ctx.fillStyle = '#5D4037';
    ctx.beginPath();
    ctx.moveTo(this.x, this.y);
    for (let i = 0; i < this.width; i += 10) {
      ctx.lineTo(this.x + i, this.y + (Math.random() * 4 - 2));
    }
    ctx.lineTo(this.x + this.width, this.y);
    ctx.lineTo(this.x + this.width, this.y + this.height);
    ctx.lineTo(this.x, this.y + this.height);
    ctx.closePath();
    ctx.fill();
  }
}
}

// Класс монеты
class Coin {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.radius = 10;
    this.collected = false;
  }

  draw() {
    if (this.collected) return;
    ctx.fillStyle = '#FFD700';
    ctx.beginPath();
    ctx.arc(this.x, this.y, this.radius, 0, Math.PI * 2);
    ctx.fill();
  }

  checkCollision() {
    if (this.collected) return false;
    const dx = player.x + player.width/2 - this.x;
    const dy = player.y + player.height/2 - this.y;
    return Math.sqrt(dx*dx + dy*dy) < this.radius + player.width/2;
  }
}

// Класс пружины
class Spring {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.width = 15;
    this.height = 10;
    this.used = false;
  }

  draw() {
    if (this.used) return;
```

```

        ctx.fillStyle = '#FF5722';
        ctx.fillRect(this.x, this.y, this.width, this.height);
    }

    checkCollision() {
        if (this.used) return false;
        return player.x + player.width > this.x &&
            player.x < this.x + this.width &&
            player.y + player.height > this.y &&
            player.y < this.y + this.height;
    }
}

```

Далее реализуем ключевые игровые механизмы: функция `createInitialPlatforms()` генерирует стартовые платформы, создавая широкую начальную платформу под игроком и последующие 14 платформ со случайным горизонтальным смещением, определяя их тип (обычная/пружинящая/разрушаемая) на основе вероятностей из `levelSettings`, а также размещая пружины и монеты согласно заданным частотам; `drawPlayer()` отрисовывает игрока в виде прямоугольника с анимированными глазами, где зрачки сдвигаются в зависимости от направления движения; `checkPlatformCollision()` обрабатывает взаимодействие игрока с платформами, реализуя разные эффекты для каждого типа (усиленный прыжок с пружинящих, разрушение хрупких платформ); `checkCoinCollisions()` и `checkSpringCollisions()` отвечают за сбор монет (с увеличением счета) и активацию пружин (с мощным прыжком) соответственно, обновляя игровое состояние и интерфейс.

```

// Создание начальных платформ
function createInitialPlatforms() {
    platforms = [];
    const settings = levelSettings[currentLevel];

    // Первая платформа под игроком (широкая)
    platforms.push(new Platform(
        canvas.width/2 - settings.platformWidth * 0.75,
        settings.playerStartY,
        settings.platformWidth * 1.5
    ));

    // Генерация последующих платформ
    for (let i = 1; i < 15; i++) {
        const prevPlatform = platforms[i-1];
        const x = Math.max(0, Math.min(
            canvas.width - settings.platformWidth,
            prevPlatform.x + (Math.random() * 200 - 100)
        ));
        const y = prevPlatform.y - settings.platformGap;

        let type = 'normal';
        const rand = Math.random();
        if (rand < settings.breakableFrequency) type = 'breakable';
        else if (rand < settings.breakableFrequency + settings.bouncyFrequency) type = 'bouncy';

        platforms.push(new Platform(x, y, settings.platformWidth, type));
    }
}

```

```
// Добавление пружин и монет
if (Math.random() < settings.springFrequency && type === 'normal') {
  springs.push(new Spring(x + settings.platformWidth/2 - 7.5, y - 15));
}
if (Math.random() < settings.coinFrequency) {
  coins.push(new Coin(x + Math.random() * settings.platformWidth, y - 50));
}
}

// Отрисовка игрока
function drawPlayer() {
  ctx.fillStyle = player.color;
  ctx.fillRect(player.x, player.y, player.width, player.height);

  // Глаза
  ctx.fillStyle = 'white';
  ctx.fillRect(player.x + 5, player.y + 5, 8, 8);
  ctx.fillRect(player.x + player.width - 13, player.y + 5, 8, 8);

  // Зрачки
  ctx.fillStyle = 'black';
  const lookDir = player.velocityX > 0 ? 2 : (player.velocityX < 0 ? -2 : 0);
  ctx.fillRect(player.x + 7 + lookDir, player.y + 7, 4, 4);
  ctx.fillRect(player.x + player.width - 11 + lookDir, player.y + 7, 4, 4);
}

// Проверка столкновений с платформами
function checkPlatformCollision() {
  player.canJump = false;
  for (const platform of platforms) {
    if (player.y + player.height >= platform.y &&
        player.y + player.height <= platform.y + platform.height &&
        player.x + player.width > platform.x &&
        player.x < platform.x + platform.width &&
        player.velocityY > 0) {

      player.canJump = true;
      if (platform.type === 'bouncy') player.velocityY = player.jumpForce * 1.8;
      else if (platform.type === 'breakable') {
        platforms.splice(platforms.indexOf(platform), 1);
        player.velocityY = player.jumpForce;
      } else {
        player.velocityY = player.jumpForce;
      }
    }
  }
}

// Проверка столкновений с монетами
function checkCoinCollisions() {
  for (let i = coins.length - 1; i >= 0; i--) {
    if (coins[i].checkCollision()) {
      coins.splice(i, 1);
      score += 10;
      scoreElement.textContent = `Очки: ${score}`;
    }
  }
}

// Проверка столкновений с пружинами
function checkSpringCollisions() {
```

```

    for (const spring of springs) {
      if (spring.checkCollision()) {
        spring.used = true;
        player.velocityY = player.jumpForce * 2;
      }
    }
  }
}

```

Затем реализуем основной игровой цикл в функции `update()`: проверяем состояние игры (`running/paused`), обрабатываем движение игрока с учетом гравитации и границ экрана, проверяем столкновения с платформами (вызывая `checkPlatformCollision()`), монетами (`checkCoinCollisions()`) и пружинами (`checkSpringCollisions()`). При достижении верхней половины экрана включаем механизм прокрутки мира - смещаем все объекты вниз на разницу между позицией игрока и центром экрана, одновременно увеличивая счет. Удаляем объекты, выпавшие за нижнюю границу, и генерируем новые платформы с прикрепленными пружинами и монетами по мере необходимости (когда количество платформ становится меньше 15), используя параметры текущего уровня сложности для определения частоты появления и характеристик объектов. Все изменения позиций и состояния объектов немедленно отражаются на игровом поле через последующую отрисовку.

```

// Обновление игрового состояния
function update() {
  if (!gameRunning || gamePaused) return;

  // Движение игрока
  player.velocityX = player.isMovingLeft ? -player.speed :
    player.isMovingRight ? player.speed :
    player.velocityX * 0.8;

  player.velocityY += player.gravity;
  player.x += player.velocityX;
  player.y += player.velocityY;

  // Границы экрана
  if (player.x < 0) player.x = 0;
  if (player.x + player.width > canvas.width) player.x = canvas.width - player.width;
  if (player.y > canvas.height) gameOver();

  // Проверка столкновений
  checkPlatformCollision();
  checkCoinCollisions();
  checkSpringCollisions();

  // Прокрутка мира
  if (player.y < canvas.height / 2) {
    const diff = canvas.height / 2 - player.y;
    player.y = canvas.height / 2;

    [platforms, coins, springs].forEach(arr => {
      arr.forEach(obj => obj.y += diff);
    });

    score += Math.floor(diff);
    scoreElement.textContent = `Очки: ${score}`;
  }
}

```

```

    }

    // Удаление вышедших за пределы объектов
    platforms = platforms.filter(p => p.y < canvas.height + 50);
    coins = coins.filter(c => c.y < canvas.height + 50);
    springs = springs.filter(s => s.y < canvas.height + 50);

    // Добавление новых платформ
    if (platforms.length < 15) {
        const settings = levelSettings[currentLevel];
        const lastPlatform = platforms[platforms.length - 1];
        const x = Math.max(0, Math.min(
            canvas.width - settings.platformWidth,
            lastPlatform.x + (Math.random() * 200 - 100)
        ));
        const y = lastPlatform.y - settings.platformGap;

        let type = 'normal';
        const rand = Math.random();
        if (rand < settings.breakableFrequency) type = 'breakable';
        else if (rand < settings.breakableFrequency + settings.bouncyFrequency) type = 'bouncy';

        platforms.push(new Platform(x, y, settings.platformWidth, type));

        if (Math.random() < settings.springFrequency && type === 'normal') {
            springs.push(new Spring(x + settings.platformWidth/2 - 7.5, y - 15));
        }
        if (Math.random() < settings.coinFrequency) {
            coins.push(new Coin(x + Math.random() * settings.platformWidth, y - 50));
        }
    }
}

```

Перейдем к визуализации и управлению игрой: функция `draw()` очищает холст, рисует голубой фон с анимированными облаками (движущимися с плавными колебаниями), отображает все игровые объекты (платформы, монеты, пружины) и игрока; игровой цикл `gameLoop()` связывает обновление состояния (`update()`) и отрисовку (`draw()`), обеспечивая плавную анимацию через `requestAnimationFrame`. Функции `gameOver()`, `startGame()` и `restartGame()` управляют жизненным циклом игры - сохраняют рекорды, показывают соответствующие экраны и сбрасывают игровое состояние. Управление реализовано через кнопки интерфейса (выбор уровня, пауза, рестарт), сенсорные элементы (лево/право) и клавиатуру (стрелки, пробел для прыжка, "P" для паузы), с обработкой как `mouse/touch`, так и `keyboard` событий. Дополнительные обработчики обеспечивают адаптацию к изменению размера окна и прыжок по клику/тапу на холст. Все элементы управления синхронизированы с игровым состоянием через флаги `isMovingLeft/isMovingRight` и `canJump`.

```

// Отрисовка игры
function draw() {
    // Очистка экрана
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    // Фон
    ctx.fillStyle = '#87CEEB';
}

```

```
ctx.fillRect(0, 0, canvas.width, canvas.height);

// Облака
ctx.fillStyle = 'rgba(255, 255, 255, 0.7)';
for (let i = 0; i < 5; i++) {
  const cloudX = (i * 300) % (canvas.width + 200) - 100;
  const cloudY = 50 + Math.sin(Date.now() / 2000 + i) * 20;
  ctx.beginPath();
  ctx.arc(cloudX, cloudY, 30, 0, Math.PI * 2);
  ctx.arc(cloudX + 25, cloudY - 10, 25, 0, Math.PI * 2);
  ctx.arc(cloudX + 50, cloudY, 20, 0, Math.PI * 2);
  ctx.arc(cloudX + 25, cloudY + 10, 25, 0, Math.PI * 2);
  ctx.fill();
}

// Отрисовка объектов
platforms.forEach(p => p.draw());
coins.forEach(c => c.draw());
springs.forEach(s => s.draw());
drawPlayer();
}

// Игровой цикл
function gameLoop() {
  update();
  draw();
  if (gameRunning && !gamePaused) requestAnimationFrame(gameLoop);
}

// Окончание игры
function gameOver() {
  gameRunning = false;
  if (score > highScore) {
    highScore = score;
    localStorage.setItem('highScore', highScore);
  }
  finalScoreElement.textContent = `Ваш счет: ${score}`;
  highScoreElement.textContent = `Рекорд: ${highScore}`;
  gameOverScreen.style.display = 'flex';
}

// Начало игры
function startGame() {
  startScreen.style.display = 'none';
  restartGame();
}

// Перезапуск игры
function restartGame() {
  gameOverScreen.style.display = 'none';
  gamePaused = false;
  pauseButton.textContent = "Пауза";

  const settings = levelSettings[currentLevel];
  score = 0;
  scoreElement.textContent = `Очки: ${score}`;
  player = {
    x: canvas.width/2 - 15,
    y: settings.playerStartY - 30,
    width: 30,
    height: 30,
    velocityX: 0,
    velocityY: 0,
```

```
        speed: 6,
        jumpForce: -16,
        gravity: 0.4,
        color: '#FF5722',
        isMovingLeft: false,
        isMovingRight: false,
        canJump: true
    };

    createInitialPlatforms();
    coins = [];
    springs = [];

    gameRunning = true;
    gameLoop();
}

// Пауза игры
function togglePause() {
    gamePaused = !gamePaused;
    pauseButton.textContent = gamePaused ? "Продолжить" : "Пауза";
    if (!gamePaused && gameRunning) gameLoop();
}

// Обработчики событий

// Выбор уровня сложности
levelButtons.forEach(button => {
    button.addEventListener('click', () => {
        levelButtons.forEach(btn => btn.classList.remove('active'));
        button.classList.add('active');
        currentLevel = button.dataset.level;
    });
});

// Кнопки управления
startButton.addEventListener('click', startGame);
pauseButton.addEventListener('click', togglePause);
restartButton.addEventListener('click', restartGame);
restartGameButton.addEventListener('click', restartGame);

// Управление с кнопок на экране
leftBtn.addEventListener('mousedown', () => player.isMovingLeft = true);
leftBtn.addEventListener('mouseup', () => player.isMovingLeft = false);
leftBtn.addEventListener('mouseleave', () => player.isMovingLeft = false);
leftBtn.addEventListener('touchstart', () => player.isMovingLeft = true);
leftBtn.addEventListener('touchend', () => player.isMovingLeft = false);

rightBtn.addEventListener('mousedown', () => player.isMovingRight = true);
rightBtn.addEventListener('mouseup', () => player.isMovingRight = false);
rightBtn.addEventListener('mouseleave', () => player.isMovingRight = false);
rightBtn.addEventListener('touchstart', () => player.isMovingRight = true);
rightBtn.addEventListener('touchend', () => player.isMovingRight = false);

// Управление с клавиатуры
window.addEventListener('keydown', (e) => {
    if (e.key === 'ArrowLeft') player.isMovingLeft = true;
    else if (e.key === 'ArrowRight') player.isMovingRight = true;
    else if (e.key === ' ' && player.canJump) {
        player.velocityY = player.jumpForce;
        player.canJump = false;
    }
    else if (e.key === 'p') togglePause();
});
```

```
});

window.addEventListener('keyup', (e) => {
  if (e.key === 'ArrowLeft') player.isMovingLeft = false;
  else if (e.key === 'ArrowRight') player.isMovingRight = false;
});

// Прыжок по клику/тапу
canvas.addEventListener('click', () => {
  if (player.canJump) {
    player.velocityY = player.jumpForce;
    player.canJump = false;
  }
});

canvas.addEventListener('touchstart', (e) => {
  e.preventDefault();
  if (player.canJump) {
    player.velocityY = player.jumpForce;
    player.canJump = false;
  }
}, { passive: false });

// Изменение размера окна
window.addEventListener('resize', () => {
  resizeCanvas();
  if (gameRunning) draw();
});
```

Теперь игра полностью завершена и готова к запуску - можем начать увлекательное прохождение, выбирая один из трех уровней сложности (легкий, средний или сложный), каждый из которых предлагает уникальный игровой опыт с разной частотой появления платформ, монет и специальных объектов. Управление реализовано как для клавиатуры (стрелки и пробел), так и для мобильных устройств (сенсорные кнопки), с возможностью прыгать по клику, собирать монеты для набора очков, использовать пружины для мощных прыжков и избегать опасных платформ, а система автоматической прокрутки мира и генерации новых платформ обеспечивает плавный и динамичный геймплей с сохранением рекордов и удобными элементами управления (пауза, рестарт).

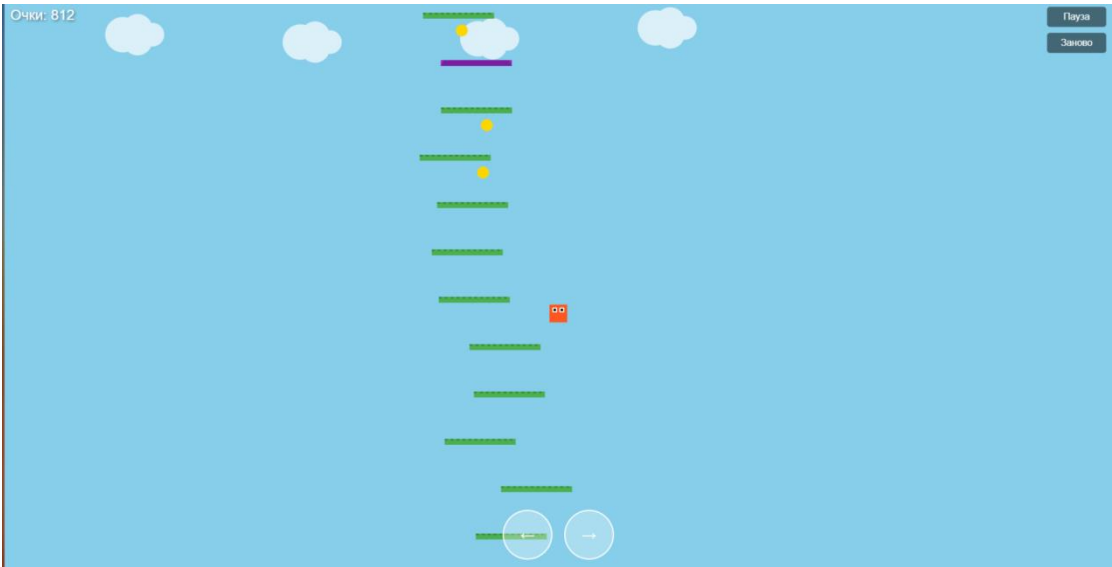


Рисунок 3. Игра



Рисунок 4. Игра

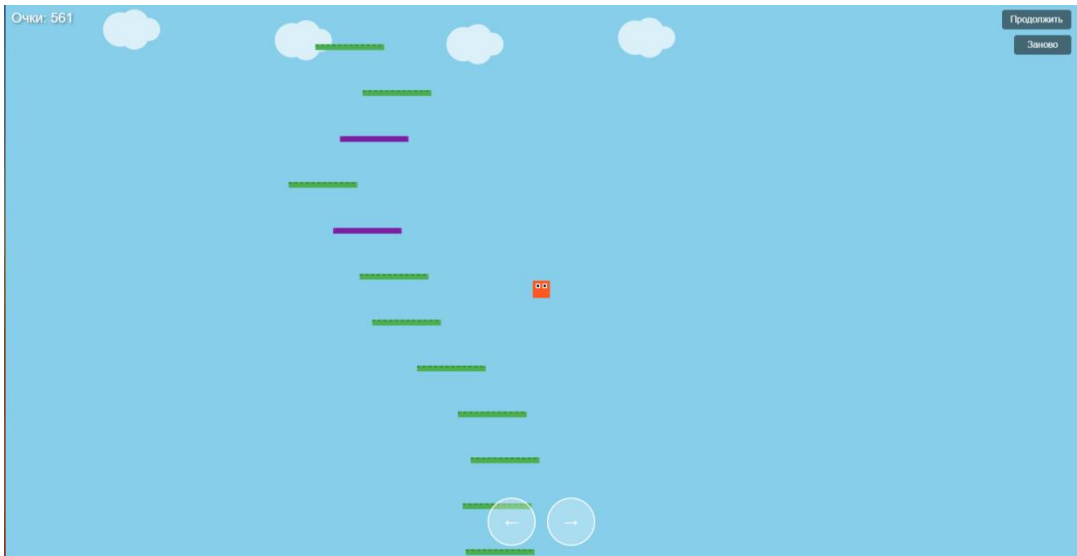


Рисунок 5. Пауза

4 Выводы

В данной работе подробно рассмотрена методика создания и разработки адаптивного платформера на HTML5 Canvas с динамической генерацией контента и мультиплатформенным управлением. 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. 311-311.
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.

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Платформер с удобными прыжками</title>
  <style>
    body {
      margin: 0;
      overflow: hidden;
      background: linear-gradient(to bottom, #87CEEB, #E0F7FA);
      font-family: Arial, sans-serif;
      touch-action: manipulation;
    }
    #gameCanvas {
      display: block;
      background-color: #87CEEB;
    }
    #score {
      position: absolute;
      top: 10px;
      left: 10px;
      font-size: 24px;
      color: white;
      text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
    }
    #controls {
      position: absolute;
      bottom: 20px;
      width: 100%;
      display: flex;
      justify-content: center;
      gap: 20px;
    }
    .control-btn {
      width: 80px;
      height: 80px;
      background-color: rgba(255, 255, 255, 0.3);
      border-radius: 50%;
      display: flex;
      justify-content: center;
      align-items: center;
      font-size: 24px;
      color: white;
      border: 2px solid white;
      user-select: none;
      -webkit-tap-highlight-color: transparent;
    }
    #gameUI {
      position: absolute;
      top: 10px;
      right: 10px;
      display: flex;
      gap: 10px;
      flex-direction: column;
      align-items: flex-end;
    }
    .ui-btn {
      padding: 8px 15px;
      background-color: rgba(0, 0, 0, 0.5);
      color: white;
```

```
border: none;
border-radius: 5px;
font-size: 16px;
cursor: pointer;
min-width: 100px;
text-align: center;
}
#startScreen {
position: absolute;
top: 0;
left: 0;
width: 100%;
height: 100%;
background-color: rgba(0, 0, 0, 0.7);
display: flex;
flex-direction: column;
justify-content: center;
align-items: center;
color: white;
z-index: 10;
}
#levelSelect {
margin: 20px 0;
display: flex;
gap: 10px;
}
.level-btn {
padding: 10px 15px;
background-color: #2196F3;
color: white;
border: none;
border-radius: 5px;
cursor: pointer;
}
.level-btn.active {
background-color: #0b7dda;
box-shadow: 0 0 10px rgba(255, 255, 255, 0.5);
}
#startButton {
padding: 15px 30px;
font-size: 20px;
background-color: #4CAF50;
color: white;
border: none;
border-radius: 10px;
cursor: pointer;
margin-top: 20px;
box-shadow: 0 4px 8px rgba(0, 0, 0, 0.3);
}
#instructions {
text-align: center;
max-width: 300px;
margin-bottom: 20px;
}
#gameOverScreen {
position: absolute;
top: 0;
left: 0;
width: 100%;
height: 100%;
background-color: rgba(0, 0, 0, 0.7);
display: none;
flex-direction: column;
```

```

        justify-content: center;
        align-items: center;
        color: white;
        z-index: 10;
    }
</style>
</head>
<body>
    <div id="score">Очки: 0</div>
    <div id="gameUI">
        <button id="pauseButton" class="ui-btn">Пауза</button>
        <button id="restartButton" class="ui-btn">Заново</button>
    </div>
    <canvas id="gameCanvas"></canvas>

    <div id="controls">
        <div id="leftBtn" class="control-btn">←</div>
        <div id="rightBtn" class="control-btn">→</div>
    </div>

    <div id="startScreen">
        <h1>Платформер с удобными прыжками</h1>
        <div id="instructions">
            <p>Прыгай по платформам и собирай монеты!</p>
            <p>Используй кнопки внизу или клавиши ← →</p>
            <p>Не падай вниз!</p>
        </div>
        <div id="levelSelect">
            <button class="level-btn active" data-level="easy">Легкий</button>
            <button class="level-btn" data-level="medium">Средний</button>
            <button class="level-btn" data-level="hard">Сложный</button>
        </div>
        <button id="startButton">Начать игру</button>
    </div>

    <div id="gameOverScreen">
        <h1>Игра окончена!</h1>
        <div id="finalScore">Ваш счет: 0</div>
        <div id="highScore">Рекорд: 0</div>
        <button id="restartGameButton" class="ui-btn" style="margin-top: 20px; font-size: 20px; padding: 15px 30px;">Играть снова</button>
    </div>

    <script>
        // Инициализация игры
        const canvas = document.getElementById('gameCanvas');
        const ctx = canvas.getContext('2d');
        const scoreElement = document.getElementById('score');
        const startScreen = document.getElementById('startScreen');
        const startButton = document.getElementById('startButton');
        const pauseButton = document.getElementById('pauseButton');
        const restartButton = document.getElementById('restartButton');
        const leftBtn = document.getElementById('leftBtn');
        const rightBtn = document.getElementById('rightBtn');
        const gameOverScreen = document.getElementById('gameOverScreen');
        const finalScoreElement = document.getElementById('finalScore');
        const highScoreElement = document.getElementById('highScore');
        const restartGameButton = document.getElementById('restartGameButton');
        const levelButtons = document.querySelectorAll('.level-btn');

        // Установка размеров холста
        function resizeCanvas() {
            canvas.width = window.innerWidth;

```

```
        canvas.height = window.innerHeight;
    }
    resizeCanvas();

    // Игровые переменные
    let score = 0;
    let highScore = localStorage.getItem('highScore') || 0;
    let gameRunning = false;
    let gamePaused = false;
    let platforms = [];
    let coins = [];
    let springs = [];
    let currentLevel = 'easy';
    let player = {
        x: canvas.width / 2 - 15,
        y: canvas.height * 0.7 - 30,
        width: 30,
        height: 30,
        velocityX: 0,
        velocityY: 0,
        speed: 6,
        jumpForce: -16,
        gravity: 0.4,
        color: '#FF5722',
        isMovingLeft: false,
        isMovingRight: false,
        canJump: true
    };
    // Настройки уровней
    const levelSettings = {
        easy: {
            platformGap: 80,
            platformWidth: 120,
            coinFrequency: 0.7,
            springFrequency: 0.3,
            bouncyFrequency: 0.1,
            breakableFrequency: 0.05,
            playerStartY: canvas.height * 0.7
        },
        medium: {
            platformGap: 100,
            platformWidth: 100,
            coinFrequency: 0.5,
            springFrequency: 0.2,
            bouncyFrequency: 0.2,
            breakableFrequency: 0.1,
            playerStartY: canvas.height * 0.7
        },
        hard: {
            platformGap: 120,
            platformWidth: 80,
            coinFrequency: 0.3,
            springFrequency: 0.1,
            bouncyFrequency: 0.3,
            breakableFrequency: 0.2,
            playerStartY: canvas.height * 0.7
        }
    };

    // Класс платформы
    class Platform {
        constructor(x, y, width, type = 'normal') {
            this.x = x;
```

```

        this.y = y;
        this.width = width;
        this.height = 10;
        this.type = type;
        this.color = type === 'bouncy' ? '#9C27B0' :
            type === 'breakable' ? '#795548' : '#4CAF50';
    }

    draw() {
        ctx.fillStyle = this.color;
        ctx.fillRect(this.x, this.y, this.width, this.height);

        if (this.type === 'normal') {
            ctx.fillStyle = '#388E3C';
            for (let i = 0; i < this.width; i += 5) {
                if (i % 10 === 0) ctx.fillRect(this.x + i, this.y, 5, 3);
            }
        } else if (this.type === 'bouncy') {
            ctx.fillStyle = '#7B1FA2';
            ctx.beginPath();
            ctx.moveTo(this.x, this.y + this.height);
            ctx.lineTo(this.x + this.width, this.y + this.height);
            ctx.lineTo(this.x + this.width - 5, this.y);
            ctx.lineTo(this.x + 5, this.y);
            ctx.closePath();
            ctx.fill();
        } else if (this.type === 'breakable') {
            ctx.fillStyle = '#5D4037';
            ctx.beginPath();
            ctx.moveTo(this.x, this.y);
            for (let i = 0; i < this.width; i += 10) {
                ctx.lineTo(this.x + i, this.y + (Math.random() * 4 - 2));
            }
            ctx.lineTo(this.x + this.width, this.y);
            ctx.lineTo(this.x + this.width, this.y + this.height);
            ctx.lineTo(this.x, this.y + this.height);
            ctx.closePath();
            ctx.fill();
        }
    }
}

// Класс монеты
class Coin {
    constructor(x, y) {
        this.x = x;
        this.y = y;
        this.radius = 10;
        this.collected = false;
    }

    draw() {
        if (this.collected) return;
        ctx.fillStyle = '#FFD700';
        ctx.beginPath();
        ctx.arc(this.x, this.y, this.radius, 0, Math.PI * 2);
        ctx.fill();
    }

    checkCollision() {
        if (this.collected) return false;
        const dx = player.x + player.width/2 - this.x;
        const dy = player.y + player.height/2 - this.y;

```

```
        return Math.sqrt(dx*dx + dy*dy) < this.radius + player.width/2;
    }
}

// Класс пружины
class Spring {
    constructor(x, y) {
        this.x = x;
        this.y = y;
        this.width = 15;
        this.height = 10;
        this.used = false;
    }

    draw() {
        if (this.used) return;
        ctx.fillStyle = '#FF5722';
        ctx.fillRect(this.x, this.y, this.width, this.height);
    }

    checkCollision() {
        if (this.used) return false;
        return player.x + player.width > this.x &&
            player.x < this.x + this.width &&
            player.y + player.height > this.y &&
            player.y < this.y + this.height;
    }
}

// Создание начальных платформ
function createInitialPlatforms() {
    platforms = [];
    const settings = levelSettings[currentLevel];

    // Первая платформа под игроком (широкая)
    platforms.push(new Platform(
        canvas.width/2 - settings.platformWidth * 0.75,
        settings.playerStartY,
        settings.platformWidth * 1.5
    ));

    // Генерация последующих платформ
    for (let i = 1; i < 15; i++) {
        const prevPlatform = platforms[i-1];
        const x = Math.max(0, Math.min(
            canvas.width - settings.platformWidth,
            prevPlatform.x + (Math.random() * 200 - 100)
        ));
        const y = prevPlatform.y - settings.platformGap;

        let type = 'normal';
        const rand = Math.random();
        if (rand < settings.breakableFrequency) type = 'breakable';
        else if (rand < settings.breakableFrequency + settings.bouncyFrequency) type = 'bouncy';

        platforms.push(new Platform(x, y, settings.platformWidth, type));

        // Добавление пружин и монет
        if (Math.random() < settings.springFrequency && type === 'normal') {
            springs.push(new Spring(x + settings.platformWidth/2 - 7.5, y - 15));
        }
        if (Math.random() < settings.coinFrequency) {
            coins.push(new Coin(x + Math.random() * settings.platformWidth, y - 50));
        }
    }
}
```

```

    }
  }

  // Отрисовка игрока
  function drawPlayer() {
    ctx.fillStyle = player.color;
    ctx.fillRect(player.x, player.y, player.width, player.height);

    // Глаза
    ctx.fillStyle = 'white';
    ctx.fillRect(player.x + 5, player.y + 5, 8, 8);
    ctx.fillRect(player.x + player.width - 13, player.y + 5, 8, 8);

    // Зрачки
    ctx.fillStyle = 'black';
    const lookDir = player.velocityX > 0 ? 2 : (player.velocityX < 0 ? -2 : 0);
    ctx.fillRect(player.x + 7 + lookDir, player.y + 7, 4, 4);
    ctx.fillRect(player.x + player.width - 11 + lookDir, player.y + 7, 4, 4);
  }

  // Проверка столкновений с платформами
  function checkPlatformCollision() {
    player.canJump = false;
    for (const platform of platforms) {
      if (player.y + player.height >= platform.y &&
        player.y + player.height <= platform.y + platform.height &&
        player.x + player.width > platform.x &&
        player.x < platform.x + platform.width &&
        player.velocityY > 0) {

        player.canJump = true;
        if (platform.type === 'bouncy') player.velocityY = player.jumpForce * 1.8;
        else if (platform.type === 'breakable') {
          platforms.splice(platforms.indexOf(platform), 1);
          player.velocityY = player.jumpForce;
        } else {
          player.velocityY = player.jumpForce;
        }
      }
    }
  }

  // Проверка столкновений с монетами
  function checkCoinCollisions() {
    for (let i = coins.length - 1; i >= 0; i--) {
      if (coins[i].checkCollision()) {
        coins.splice(i, 1);
        score += 10;
        scoreElement.textContent = `Очки: ${score}`;
      }
    }
  }

  // Проверка столкновений с пружинами
  function checkSpringCollisions() {
    for (const spring of springs) {
      if (spring.checkCollision()) {
        spring.used = true;
        player.velocityY = player.jumpForce * 2;
      }
    }
  }

  // Обновление игрового состояния

```

```
function update() {
  if (!gameRunning || gamePaused) return;

  // Движение игрока
  player.velocityX = player.isMovingLeft ? -player.speed :
    player.isMovingRight ? player.speed :
    player.velocityX * 0.8;

  player.velocityY += player.gravity;
  player.x += player.velocityX;
  player.y += player.velocityY;

  // Границы экрана
  if (player.x < 0) player.x = 0;
  if (player.x + player.width > canvas.width) player.x = canvas.width - player.width;
  if (player.y > canvas.height) gameOver();

  // Проверка столкновений
  checkPlatformCollision();
  checkCoinCollisions();
  checkSpringCollisions();

  // Прокрутка мира
  if (player.y < canvas.height / 2) {
    const diff = canvas.height / 2 - player.y;
    player.y = canvas.height / 2;

    [platforms, coins, springs].forEach(arr => {
      arr.forEach(obj => obj.y += diff);
    });

    score += Math.floor(diff);
    scoreElement.textContent = `Очки: ${score}`;
  }

  // Удаление вышедших за пределы объектов
  platforms = platforms.filter(p => p.y < canvas.height + 50);
  coins = coins.filter(c => c.y < canvas.height + 50);
  springs = springs.filter(s => s.y < canvas.height + 50);

  // Добавление новых платформ
  if (platforms.length < 15) {
    const settings = levelSettings[currentLevel];
    const lastPlatform = platforms[platforms.length - 1];
    const x = Math.max(0, Math.min(
      canvas.width - settings.platformWidth,
      lastPlatform.x + (Math.random() * 200 - 100)
    ));
    const y = lastPlatform.y - settings.platformGap;

    let type = 'normal';
    const rand = Math.random();
    if (rand < settings.breakableFrequency) type = 'breakable';
    else if (rand < settings.breakableFrequency + settings.bouncyFrequency) type = 'bouncy';

    platforms.push(new Platform(x, y, settings.platformWidth, type));

    if (Math.random() < settings.springFrequency && type === 'normal') {
      springs.push(new Spring(x + settings.platformWidth/2 - 7.5, y - 15));
    }
    if (Math.random() < settings.coinFrequency) {
      coins.push(new Coin(x + Math.random() * settings.platformWidth, y - 50));
    }
  }
}
```

```
}  
}  
// Отрисовка игры  
function draw() {  
    // Очистка экрана  
    ctx.clearRect(0, 0, canvas.width, canvas.height);  
  
    // Фон  
    ctx.fillStyle = '#87CEEB';  
    ctx.fillRect(0, 0, canvas.width, canvas.height);  
  
    // Облака  
    ctx.fillStyle = 'rgba(255, 255, 255, 0.7)';  
    for (let i = 0; i < 5; i++) {  
        const cloudX = (i * 300) % (canvas.width + 200) - 100;  
        const cloudY = 50 + Math.sin(Date.now() / 2000 + i) * 20;  
        ctx.beginPath();  
        ctx.arc(cloudX, cloudY, 30, 0, Math.PI * 2);  
        ctx.arc(cloudX + 25, cloudY - 10, 25, 0, Math.PI * 2);  
        ctx.arc(cloudX + 50, cloudY, 20, 0, Math.PI * 2);  
        ctx.arc(cloudX + 25, cloudY + 10, 25, 0, Math.PI * 2);  
        ctx.fill();  
    }  
  
    // Отрисовка объектов  
    platforms.forEach(p => p.draw());  
    coins.forEach(c => c.draw());  
    springs.forEach(s => s.draw());  
    drawPlayer();  
}  
  
// Игровой цикл  
function gameLoop() {  
    update();  
    draw();  
    if (gameRunning && !gamePaused) requestAnimationFrame(gameLoop);  
}  
  
// Окончание игры  
function gameOver() {  
    gameRunning = false;  
    if (score > highScore) {  
        highScore = score;  
        localStorage.setItem('highScore', highScore);  
    }  
    finalScoreElement.textContent = `Ваш счет: ${score}`;  
    highScoreElement.textContent = `Рекорд: ${highScore}`;  
    gameOverScreen.style.display = 'flex';  
}  
  
// Начало игры  
function startGame() {  
    startScreen.style.display = 'none';  
    restartGame();  
}  
  
// Перезапуск игры  
function restartGame() {  
    gameOverScreen.style.display = 'none';  
    gamePaused = false;  
    pauseButton.textContent = "Пауза";  
  
    const settings = levelSettings[currentLevel];
```

```
score = 0;
scoreElement.textContent = `Очки: ${score}`;
player = {
  x: canvas.width/2 - 15,
  y: settings.playerStartY - 30,
  width: 30,
  height: 30,
  velocityX: 0,
  velocityY: 0,
  speed: 6,
  jumpForce: -16,
  gravity: 0.4,
  color: '#FF5722',
  isMovingLeft: false,
  isMovingRight: false,
  canJump: true
};

createInitialPlatforms();
coins = [];
springs = [];

gameRunning = true;
gameLoop();
}

// Пауза игры
function togglePause() {
  gamePaused = !gamePaused;
  pauseButton.textContent = gamePaused ? "Продолжить" : "Пауза";
  if (!gamePaused && gameRunning) gameLoop();
}

// Обработчики событий

// Выбор уровня сложности
levelButtons.forEach(button => {
  button.addEventListener('click', () => {
    levelButtons.forEach(btn => btn.classList.remove('active'));
    button.classList.add('active');
    currentLevel = button.dataset.level;
  });
});

// Кнопки управления
startButton.addEventListener('click', startGame);
pauseButton.addEventListener('click', togglePause);
restartButton.addEventListener('click', restartGame);
restartGameButton.addEventListener('click', restartGame);

// Управление с кнопок на экране
leftBtn.addEventListener('mousedown', () => player.isMovingLeft = true);
leftBtn.addEventListener('mouseup', () => player.isMovingLeft = false);
leftBtn.addEventListener('mouseleave', () => player.isMovingLeft = false);
leftBtn.addEventListener('touchstart', () => player.isMovingLeft = true);
leftBtn.addEventListener('touchend', () => player.isMovingLeft = false);

rightBtn.addEventListener('mousedown', () => player.isMovingRight = true);
rightBtn.addEventListener('mouseup', () => player.isMovingRight = false);
rightBtn.addEventListener('mouseleave', () => player.isMovingRight = false);
rightBtn.addEventListener('touchstart', () => player.isMovingRight = true);
rightBtn.addEventListener('touchend', () => player.isMovingRight = false);
```

```
// Управление с клавиатуры
window.addEventListener('keydown', (e) => {
  if (e.key === 'ArrowLeft') player.isMovingLeft = true;
  else if (e.key === 'ArrowRight') player.isMovingRight = true;
  else if (e.key === ' ' && player.canJump) {
    player.velocityY = player.jumpForce;
    player.canJump = false;
  }
  else if (e.key === 'p') togglePause();
});

window.addEventListener('keyup', (e) => {
  if (e.key === 'ArrowLeft') player.isMovingLeft = false;
  else if (e.key === 'ArrowRight') player.isMovingRight = false;
});

// Прыжок по клику/тапу
canvas.addEventListener('click', () => {
  if (player.canJump) {
    player.velocityY = player.jumpForce;
    player.canJump = false;
  }
});

canvas.addEventListener('touchstart', (e) => {
  e.preventDefault();
  if (player.canJump) {
    player.velocityY = player.jumpForce;
    player.canJump = false;
  }
}, { passive: false });

// Изменение размера окна
window.addEventListener('resize', () => {
  resizeCanvas();
  if (gameRunning) draw();
});
</script>
</body>
</html>
```