

Реализация игровой механики и визуализации классического Pacman на JavaScript и Canvas

Екимова Яна Сергеевна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье рассматривается методика создания классической игры Pacman с применением технологий HTML5 Canvas, CSS3 и JavaScript. Разработанное решение позволяет воссоздать полноценный игровой процесс с реалистичной физикой движения, системой коллизий, динамической визуализацией и адаптивным управлением для различных устройств. 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: Pacman, HTML5, CSS, JavaScript.

Implementation of classic Pacman game mechanics and visualization in JavaScript and Canvas

Ekimova Yana Sergeevna

Sholom-Aleichem Priamurskiy State University

Student

Abstract

The article discusses the technique of creating a classic Pacman game using HTML5 Canvas, CSS3 and JavaScript technologies. The developed solution allows you to recreate a full-fledged gameplay with realistic motion physics, a collision system, dynamic visualization and adaptive controls for various devices. 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: Pacman, HTML5, CSS, JavaScript.

1 Введение

1.1 Актуальность

В современном мире веб-технологий разработка игровых приложений на HTML5 Canvas приобретает особую значимость, поскольку позволяет создавать кроссплатформенные проекты, работающие непосредственно в браузере без дополнительных плагинов. Актуальность реализации классической игры Pacman заключается в том, что она служит идеальным учебным полигоном для освоения ключевых аспектов программирования:

работы с графикой, анимацией, коллизиями, обработкой пользовательского ввода и управлением состоянием приложения, одновременно отвечая на растущий спрос на ретро-гейминг и интерактивный образовательный контент.

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 Распан 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 интерактивной веб-версии классической игры Распан. Документ начинается со стандартной декларации <!DOCTYPE html> и разметки (html, head, body), где в секции head указаны ключевые метаданные: кодировка UTF-8 для корректного отображения текста, viewport для адаптации под мобильные устройства, а также заголовок страницы, отражающий особенности игры. В теле документа размещена комплексная структура, включающая основной контейнер игры (game-wrapper), который содержит интерфейс управления с отображением счета (изначально скрытого), индикатора жизней в виде сердечек и

переключателя скорости игры, а также центральный игровой контейнер с холстом для отрисовки графики.

Игровой контейнер (game-container) представляет собой многослойную композицию: элемент canvas для основной графики, поверх которого размещены экраны состояний — стартовый с приветствием и кнопкой начала, экран завершения игры с итоговым счетом, а также специализированные элементы для мобильных устройств — панель управления с кнопками-стрелками и подсказка для пользователя.

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Pacman - Счёт появляется после старта</title>
  <style>

</style>
</head>
<body>
  <div id="game-wrapper">
    <!-- Счёт появится ТОЛЬКО после старта -->
    <div id="ui">
      <div id="score-container"><span class="pacman-icon">◎</span>Счёт: <span
id="score">0</span></div>
      <div id="lives-container">Жизни: <span id="lives"><span
class="life">♥</span><span class="life">♥</span><span
class="life">♥</span></div>
      <div id="speed-controls">
        <div class="speed-btn active" id="slow-btn">Медл</div>
        <div class="speed-btn" id="normal-btn">Норм</div>
      </div>
    </div>

    <div id="game-container">
      <canvas id="game-canvas" width="600" height="620"></canvas>

      <div id="start-screen" class="screen">
        <h1>PACMAN</h1>
        <p>Управляйте Пакманом, собирайте точки и избегайте призраков!</p>
        <button class="btn" id="start-btn">Начать игру</button>
        <div id="controls-info">
          Управление: ← → ↑ ↓ для движения<br>
          Цель: Соберите все точки на уровне<br>
          Синие точки дают временную возможность съесть призраков!<br>
          <span style="color: #ff0;">Скорость игры уменьшена для удобства</span>
        </div>
      </div>

      <div id="game-over" class="screen">
```

```

<h1>ИГРА ОКОНЧЕНА</h1>
<p>Ваш результат: <span id="final-score">0</span> очков</p>
<button class="btn" id="restart-btn">Играть снова</button>
</div>

<div id="mobile-controls">
  <div id="direction-pad">
    <div class="control-btn" id="up-btn">↑</div>
    <div class="control-btn" id="left-btn">←</div>
    <div class="control-btn" id="right-btn">→</div>
    <div class="control-btn" id="down-btn">↓</div>
  </div>
</div>

<div id="hint">Используйте кнопки управления внизу экрана</div>
</div>

<script>

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

```



Рисунок 1. Основная структура HTML-документа

Далее добавим блок CSS-стилей, который используется для визуального оформления и поведения элементов веб-приложения (рис. 2). Эти стили формируют современный, аркадный интерфейс в стиле классических игр, с акцентом на тематику Распан, сочетая плавные анимации, глубину, адаптивность и интерактивность.

Данный код пропишем на следующей строчке после тега <title> Распан </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, #000, #1a1a40);
  font-family: 'Arial', sans-serif;
  color: white;
  overflow: hidden;
}
```

```
/* Общий контейнер для UI и игры */
```

```
#game-wrapper {
  display: flex;
  flex-direction: column;
  align-items: center;
  gap: 10px;
}
```

```
/* Панель счёта — скрыта до начала игры */
```

```
#ui {
  font-size: 20px;
  text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.7);
  background: rgba(0, 0, 0, 0.8);
  padding: 12px 20px;
  border-radius: 10px;
  border: 2px solid #ff0;
  display: flex;
  gap: 30px;
  justify-content: center;
  align-items: center;
  width: 600px;
  box-shadow: 0 4px 10px rgba(255, 255, 0, 0.3);
  color: white;
  display: none; /*  Скрыто по умолчанию */
}
```

```
#score-container, #lives-container {
  display: flex;
  align-items: center;
}
```

```
#score {
  color: #ff0;
  font-weight: bold;
}
```

```
#lives {
  color: #ff5252;
  font-weight: bold;
  display: flex;
```

```
        align-items: center;
    }

    .life {
        color: #ffeb3b;
        margin-left: 5px;
        font-size: 24px;
    }

    /* Игровой контейнер */
    #game-container {
        position: relative;
        width: 600px;
        height: 620px;
        border: 4px solid #ff0;
        border-radius: 12px;
        overflow: hidden;
        box-shadow: 0 0 25px rgba(255, 255, 0, 0.6);
        background: #000;
    }

    #game-canvas {
        background: #000;
        display: block;
    }

    .screen {
        position: absolute;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        display: flex;
        flex-direction: column;
        justify-content: center;
        align-items: center;
        background: rgba(0, 0, 0, 0.9);
        z-index: 10;
        text-align: center;
    }

    #game-over {
        display: none;
    }

    h1 {
        font-size: 60px;
        color: #ff0;
        margin-bottom: 20px;
        text-shadow: 0 0 15px rgba(255, 255, 0, 0.8);
        font-family: 'Arial Black', sans-serif;
    }
```

```
p {
  font-size: 20px;
  margin-bottom: 25px;
  max-width: 80%;
  line-height: 1.5;
  color: #fff;
}

.btn {
  padding: 15px 40px;
  font-size: 20px;
  background: linear-gradient(to bottom, #ff0, #ff9800);
  border: none;
  border-radius: 50px;
  color: #000;
  cursor: pointer;
  font-weight: bold;
  text-transform: uppercase;
  letter-spacing: 2px;
  transition: all 0.3s;
  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
  margin: 10px;
}

.btn:hover {
  transform: translateY(-3px);
  box-shadow: 0 8px 20px rgba(0, 0, 0, 0.4);
  background: linear-gradient(to bottom, #ffea00, #ffa000);
}

.btn:active {
  transform: translateY(1px);
}

#controls-info {
  margin-top: 30px;
  font-size: 18px;
  text-align: center;
  background: rgba(255, 255, 0, 0.1);
  padding: 15px;
  border-radius: 10px;
  width: 80%;
  border: 1px solid #ff0;
}

#final-score {
  color: #ff0;
  font-size: 36px;
  margin: 20px 0;
}
```

```
#mobile-controls {
  position: absolute;
  bottom: 20px;
  width: 100%;
  display: none;
  justify-content: center;
  padding: 0 30px;
  z-index: 5;
}

#direction-pad {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(3, 1fr);
  gap: 5px;
}

.control-btn {
  width: 60px;
  height: 60px;
  background: rgba(255, 255, 0, 0.2);
  border-radius: 10px;
  display: flex;
  justify-content: center;
  align-items: center;
  font-size: 24px;
  color: white;
  user-select: none;
  border: 2px solid rgba(255, 255, 0, 0.5);
}

#up-btn {
  grid-column: 2;
  grid-row: 1;
}

#left-btn {
  grid-column: 1;
  grid-row: 2;
}

#right-btn {
  grid-column: 3;
  grid-row: 2;
}

#down-btn {
  grid-column: 2;
  grid-row: 3;
}

#hint {
```

```
    position: absolute;
    bottom: 120px;
    left: 50%;
    transform: translateX(-50%);
    background: rgba(255, 215, 0, 0.2);
    padding: 10px 15px;
    border-radius: 10px;
    font-size: 16px;
    display: none;
}

#speed-controls {
    display: flex;
    gap: 10px;
    margin-top: 10px;
    justify-content: center;
}

.speed-btn {
    padding: 8px 15px;
    background: rgba(255, 255, 0, 0.2);
    border: 1px solid #ff0;
    border-radius: 5px;
    color: white;
    cursor: pointer;
    font-size: 14px;
}

.speed-btn.active {
    background: #ff0;
    color: black;
}

@media (max-width: 650px) {
    #game-container, #ui {
        width: 95vw;
    }

    #game-container {
        height: 600px;
    }

    #mobile-controls {
        display: flex;
    }

    #hint {
        display: block;
    }
}

.pacman-icon {
```

```
font-size: 24px;
color: #ff0;
margin-right: 5px;
}
```



Рисунок 2. Добавление CSS-стилей

Далее напишем код инициализации игровых элементов и переменных, который получает ссылки на все ключевые DOM-элементы: игровой холст (canvas) и его контекст для отрисовки, элементы интерфейса для отображения счета и жизней, экраны начала и завершения игры, кнопки управления, а также мобильные контролы. Затем объявляются основные игровые переменные: состояние игры (gameRunning), счет (score), количество жизней (lives), скорость игры (gameSpeed), общее количество точек и собранных точек. Завершает этот блок определение констант: размера ячейки игрового поля (CELL_SIZE) и расчет размеров сетки на основе ширины и высоты холста (GRID_WIDTH, GRID_HEIGHT), которые будут использоваться для построения игрового лабиринта и позиционирования объектов.

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

```
// Элементы
const canvas = document.getElementById('game-canvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const livesElement = document.getElementById('lives');
const startScreen = document.getElementById('start-screen');
const gameOverScreen = document.getElementById('game-over');
const startBtn = document.getElementById('start-btn');
const restartBtn = document.getElementById('restart-btn');
const finalScoreElement = document.getElementById('final-score');
const slowBtn = document.getElementById('slow-btn');
```

```

const normalBtn = document.getElementById('normal-btn');
const ui = document.getElementById('ui'); // Ссылка на UI

// Мобильные кнопки
const upBtn = document.getElementById('up-btn');
const leftBtn = document.getElementById('left-btn');
const rightBtn = document.getElementById('right-btn');
const downBtn = document.getElementById('down-btn');

// Игровые переменные
let gameRunning = false;
let score = 0;
let lives = 3;
let gameSpeed = 0.7;
let totalDots = 0;
let dotsCollected = 0;

// Константы
const CELL_SIZE = 20;
const GRID_WIDTH = Math.floor(canvas.width / CELL_SIZE);
const GRID_HEIGHT = Math.floor(canvas.height / CELL_SIZE);

```

Теперь напомним структуру игрового лабиринта, представленную в виде двумерного массива `maze`, где каждый элемент соответствует определенному типу клетки: 0 обозначает стену, через которую нельзя пройти, 1 представляет собой обычную точку для сбора, 2 специальные точки (энерджайзеры), дающие временную возможность съесть призраков, а 3 обозначает специальные проходимые области, часто используемые как туннели или зона появления призраков. Данная конфигурация точно воспроизводит классическую схему лабиринта из оригинальной игры *Rastan*, включая симметричное расположение точек, энерджайзеров в углах, центральную зону призраков и проходимые туннели по бокам, которые соединяют правую и левую части игрового поля.

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

```

// Лабиринт
const maze = [
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
  [0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0],
  [0, 2, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 2, 0],
  [0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0],
  [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
  [0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0],
  [0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0],
  [0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0],
  [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 3, 0, 0, 3, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 3, 0, 0, 3, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 1, 0, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 1, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 1, 0, 0, 0, 0],

```

```

[0, 0, 0, 0, 0, 0, 1, 0, 0, 3, 1, 1, 1, 1, 1, 1, 1, 0, 3, 0, 0, 1, 0, 0, 0, 0],
[0, 3, 3, 3, 3, 3, 1, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 0, 3, 3, 3, 1, 3, 3, 3, 0],
[0, 0, 0, 0, 0, 0, 1, 0, 0, 3, 1, 1, 1, 1, 1, 1, 1, 0, 3, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 1, 0, 0, 0, 0],
[0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
[0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0],
[0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0],
[0, 2, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 3, 3, 1, 1, 1, 1, 1, 1, 1, 0, 0, 2, 0],
[0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0],
[0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0],
[0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
[0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
];

```

Затем пропишем логику управления игровым состоянием и поведением персонажей. Создаем глубокую копию оригинального лабиринта для последующих сбросов и функции `resetMaze()` и `countDots()` для восстановления точек и подсчета очков. Реализуем класс `Рacman` с анимацией движения, открыванием рта, обработкой столкновений со стенами и коллекцией точек, включая режим "силы" при съезжении эннерджайзеров. Класс `Ghost` реализует поведение призраков со случайным изменением направления, проверкой возможности движения, визуализацией и обработкой столкновений с Пакманом - при обычном режиме игра заканчивается, а в режиме силы призраки возвращаются на стартовую позицию. Оба класса содержат методы обновления скорости, отрисовки и взаимодействия с игровой механикой, обеспечивая полноценный игровой процесс.

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

```

const originalMaze = JSON.parse(JSON.stringify(maze));

function resetMaze() {
  for (let y = 0; y < maze.length; y++) {
    for (let x = 0; x < maze[y].length; x++) {
      if (originalMaze[y][x] === 1 || originalMaze[y][x] === 2) {
        maze[y][x] = originalMaze[y][x];
      }
    }
  }
}

function countDots() {
  totalDots = 0;
  for (let y = 0; y < maze.length; y++) {
    for (let x = 0; x < maze[y].length; x++) {

```

```
        if (maze[y][x] === 1 || maze[y][x] === 2) {
            totalDots++;
        }
    }
}
dotsCollected = 0;
}

class Pacman {
    constructor() {
        this.x = 13 * CELL_SIZE + CELL_SIZE / 2;
        this.y = 23 * CELL_SIZE + CELL_SIZE / 2;
        this.radius = CELL_SIZE / 2 - 2;
        this.speed = 2 * gameSpeed;
        this.direction = null;
        this.nextDirection = null;
        this.mouthOpen = 0;
        this.mouthChange = 0.05;
        this.maxMouthOpen = 0.7;
        this.color = '#ff0';
        this.powerMode = false;
        this.powerTimer = 0;
    }

    update() {
        this.mouthOpen += this.mouthChange;
        if (this.mouthOpen > this.maxMouthOpen || this.mouthOpen < 0) {
            this.mouthChange *= -1;
        }

        if (this.powerMode) {
            this.powerTimer--;
            if (this.powerTimer <= 0) {
                this.powerMode = false;
                this.color = '#ff0';
            }
        }

        if (this.nextDirection && this.canMove(this.nextDirection)) {
            this.direction = this.nextDirection;
            this.nextDirection = null;
        }

        if (this.direction && this.canMove(this.direction)) {
            switch (this.direction) {
                case 'left':
                    this.x -= this.speed;
                    break;
                case 'right':
                    this.x += this.speed;
                    break;
                case 'up':
```

```

        this.y -= this.speed;
        break;
    case 'down':
        this.y += this.speed;
        break;
    }
}

if (this.x < 0) this.x = canvas.width;
if (this.x > canvas.width) this.x = 0;

this.collectDots();
this.draw();
}

draw() {
    ctx.save();
    ctx.translate(this.x, this.y);

    if (this.direction === 'left') ctx.rotate(Math.PI);
    else if (this.direction === 'up') ctx.rotate(-Math.PI / 2);
    else if (this.direction === 'down') ctx.rotate(Math.PI / 2);

    ctx.fillStyle = this.color;
    ctx.beginPath();
    ctx.arc(0, 0, this.radius, this.mouthOpen * Math.PI, (2 - this.mouthOpen) *
Math.PI);

    ctx.lineTo(0, 0);
    ctx.fill();

    ctx.restore();
}

canMove(direction) {
    let nextX = this.x;
    let nextY = this.y;

    switch (direction) {
        case 'left':
            nextX -= this.speed;
            break;
        case 'right':
            nextX += this.speed;
            break;
        case 'up':
            nextY -= this.speed;
            break;
        case 'down':
            nextY += this.speed;
            break;
    }
}

```

```
const gridX = Math.floor(nextX / CELL_SIZE);
const gridY = Math.floor(nextY / CELL_SIZE);

if (gridX < 0 || gridX >= GRID_WIDTH || gridY < 0 || gridY >= GRID_HEIGHT)
{
    return true;
}

return maze[gridY][gridX] !== 0;
}

collectDots() {
    const gridX = Math.floor(this.x / CELL_SIZE);
    const gridY = Math.floor(this.y / CELL_SIZE);

    if (gridX >= 0 && gridX < GRID_WIDTH && gridY >= 0 && gridY <
GRID_HEIGHT) {
        if (maze[gridY][gridX] === 1) {
            maze[gridY][gridX] = 3;
            score += 10;
            dotsCollected++;
            updateUI();
        } else if (maze[gridY][gridX] === 2) {
            maze[gridY][gridX] = 3;
            score += 50;
            dotsCollected++;
            this.activatePowerMode();
            updateUI();
        }

        if (dotsCollected >= totalDots) {
            winGame();
        }
    }
}

activatePowerMode() {
    this.powerMode = true;
    this.powerTimer = 200;
    this.color = '#0ff';
}

setDirection(dir) {
    this.nextDirection = dir;
}

updateSpeed() {
    this.speed = 2 * gameSpeed;
}
}

class Ghost {
```

```
constructor(color, x, y) {
  this.x = x;
  this.y = y;
  this.color = color;
  this.radius = CELL_SIZE / 2 - 2;
  this.speed = 1.5 * gameSpeed;
  this.direction = 'left';
  this.scared = false;
  this.directionChangeTimer = 0;
}

update() {
  this.directionChangeTimer++;
  if (this.directionChangeTimer > 60) {
    this.directionChangeTimer = 0;
    if (Math.random() < 0.1) {
      const directions = ['left', 'right', 'up', 'down'];
      this.direction = directions[Math.floor(Math.random() * directions.length)];
    }
  }
}

if (this.canMove(this.direction)) {
  switch (this.direction) {
    case 'left':
      this.x -= this.speed;
      break;
    case 'right':
      this.x += this.speed;
      break;
    case 'up':
      this.y -= this.speed;
      break;
    case 'down':
      this.y += this.speed;
      break;
  }
} else {
  const directions = ['left', 'right', 'up', 'down'];
  this.direction = directions[Math.floor(Math.random() * directions.length)];
}

if (this.x < 0) this.x = canvas.width;
if (this.x > canvas.width) this.x = 0;

this.draw();
}

draw() {
  ctx.fillStyle = this.scared ? '#00f' : this.color;
  ctx.beginPath();
  ctx.arc(this.x, this.y, this.radius, Math.PI, 0, false);
  ctx.lineTo(this.x + this.radius, this.y + this.radius);
}
```

```

    for (let i = 0; i < 3; i++) {
        ctx.lineTo(this.x + this.radius - (i * this.radius / 1.5), this.y + this.radius / 2);
        ctx.lineTo(this.x + this.radius - ((i + 1) * this.radius / 1.5), this.y + this.radius);
    }

    ctx.lineTo(this.x - this.radius, this.y + this.radius);
    ctx.closePath();
    ctx.fill();

    ctx.fillStyle = '#fff';
    ctx.beginPath();
    ctx.arc(this.x - this.radius / 3, this.y - this.radius / 4, this.radius / 3, 0, Math.PI *
2);
    ctx.arc(this.x + this.radius / 3, this.y - this.radius / 4, this.radius / 3, 0, Math.PI *
2);
    ctx.fill();

    ctx.fillStyle = '#000';
    let offsetX = 0;
    let offsetY = 0;

    if (this.direction === 'left') offsetX = -2;
    else if (this.direction === 'right') offsetX = 2;
    else if (this.direction === 'up') offsetY = -2;
    else if (this.direction === 'down') offsetY = 2;

    ctx.beginPath();
    ctx.arc(this.x - this.radius / 3 + offsetX, this.y - this.radius / 4 + offsetY,
this.radius / 6, 0, Math.PI * 2);
    ctx.arc(this.x + this.radius / 3 + offsetX, this.y - this.radius / 4 + offsetY,
this.radius / 6, 0, Math.PI * 2);
    ctx.fill();
}

canMove(direction) {
    let nextX = this.x;
    let nextY = this.y;

    switch (direction) {
        case 'left':
            nextX -= this.speed;
            break;
        case 'right':
            nextX += this.speed;
            break;
        case 'up':
            nextY -= this.speed;
            break;
        case 'down':
            nextY += this.speed;
            break;
    }
}

```

```

    }

    const gridX = Math.floor(nextX / CELL_SIZE);
    const gridY = Math.floor(nextY / CELL_SIZE);

    if (gridX < 0 || gridX >= GRID_WIDTH || gridY < 0 || gridY >= GRID_HEIGHT)
    {
        return true;
    }

    return maze[gridY][gridX] !== 0;
}

checkCollision(pacman) {
    const dx = this.x - pacman.x;
    const dy = this.y - pacman.y;
    const distance = Math.sqrt(dx * dx + dy * dy);

    if (distance < this.radius + pacman.radius) {
        if (pacman.powerMode) {
            this.x = 13 * CELL_SIZE + CELL_SIZE / 2;
            this.y = 11 * CELL_SIZE + CELL_SIZE / 2;
            score += 200;
            updateUI();
            return false;
        } else {
            return true;
        }
    }
    return false;
}

setScared(isScared) {
    this.scared = isScared;
}

updateSpeed() {
    this.speed = 1.5 * gameSpeed;
}
}

```

Теперь осталось реализовать основную игровую механику и управление. Создаем функцию `initGame()` для инициализации персонажей - Пакмана и четырех призраков разных цветов, сбрасываем счетчики и восстанавливаем лабиринт. Функция `drawMaze()` отрисовывает игровое поле: синие стены, белые точки и энерджайзеры. `checkCollisions()` обрабатывает столкновения с призраками - уменьшает жизни или возвращает призраков на базу в режиме силы. Основной игровой цикл `gameLoop()` очищает холст, обновляет и отрисовывает все объекты, проверяет столкновения. Реализовано управление с клавиатуры (стрелки) и сенсорное управление для мобильных устройств,

переключение скорости игры, обработчики кнопок старта и рестарта, которые скрывают/показывают соответствующие экраны и запускают игровой процесс.

Вставим данный код в продолжение тега <script>:

```

let pacman;
let ghosts = [];

function initGame() {
  pacman = new Pacman();
  ghosts = [
    new Ghost('#f00', 13 * CELL_SIZE + CELL_SIZE / 2, 11 * CELL_SIZE +
CELL_SIZE / 2),
    new Ghost('#0ff', 12 * CELL_SIZE + CELL_SIZE / 2, 11 * CELL_SIZE +
CELL_SIZE / 2),
    new Ghost('#f0f', 14 * CELL_SIZE + CELL_SIZE / 2, 11 * CELL_SIZE +
CELL_SIZE / 2),
    new Ghost('#ff8800', 13 * CELL_SIZE + CELL_SIZE / 2, 10 * CELL_SIZE +
CELL_SIZE / 2)
  ];

  score = 0;
  lives = 3;
  dotsCollected = 0;

  resetMaze();
  countDots();
  updateUI();
}

function drawMaze() {
  for (let y = 0; y < maze.length; y++) {
    for (let x = 0; x < maze[y].length; x++) {
      const cell = maze[y][x];
      if (cell === 0) {
        ctx.fillStyle = '#2244aa';
        ctx.fillRect(x * CELL_SIZE, y * CELL_SIZE, CELL_SIZE, CELL_SIZE);
      } else if (cell === 1) {
        ctx.fillStyle = 'fff';
        ctx.beginPath();
        ctx.arc(x * CELL_SIZE + CELL_SIZE / 2, y * CELL_SIZE + CELL_SIZE
/ 2, 2, 0, Math.PI * 2);
        ctx.fill();
      } else if (cell === 2) {
        ctx.fillStyle = 'fff';
        ctx.beginPath();
        ctx.arc(x * CELL_SIZE + CELL_SIZE / 2, y * CELL_SIZE + CELL_SIZE
/ 2, 5, 0, Math.PI * 2);
        ctx.fill();
      }
    }
  }
}

```

```

    }
  }

function checkCollisions() {
  ghosts.forEach(ghost => {
    if (ghost.checkCollision(pacman)) {
      lives--;
      updateUI();

      if (lives <= 0) {
        endGame();
      } else {
        pacman.x = 13 * CELL_SIZE + CELL_SIZE / 2;
        pacman.y = 23 * CELL_SIZE + CELL_SIZE / 2;
        pacman.direction = null;

        ghosts.forEach(g => {
          g.x = 13 * CELL_SIZE + CELL_SIZE / 2;
          g.y = 11 * CELL_SIZE + CELL_SIZE / 2;
        });
      }
    }
  });

  ghosts.forEach(ghost => {
    ghost.setScared(pacman.powerMode);
  });
}

function updateUI() {
  scoreElement.textContent = score;
  const livesElements = livesElement.querySelectorAll('.life');
  livesElements.forEach((life, index) => {
    life.style.display = index < lives ? 'inline' : 'none';
  });
}

function endGame() {
  gameRunning = false;
  finalScoreElement.textContent = score;
  gameOverScreen.querySelector('h1').textContent = 'ИГРА ОКОНЧЕНА';
  gameOverScreen.querySelector('p').innerHTML = `Ваш результат: <span
style="color:#ff0">${score}</span> очков`;
  gameOverScreen.style.display = 'flex';
}

function winGame() {
  gameRunning = false;
  finalScoreElement.textContent = score;
  gameOverScreen.querySelector('h1').textContent = 'ПОБЕДА!';
  gameOverScreen.querySelector('p').innerHTML = `Вы собрали все
точки!<br>Ваш результат: <span style="color:#ff0">${score}</span> очков`;

```

```
    gameOverScreen.style.display = 'flex';
  }

  function updateGameSpeed() {
    if (pacman) pacman.updateSpeed();
    ghosts.forEach(ghost => ghost.updateSpeed());
  }

  function gameLoop() {
    if (!gameRunning) return;

    ctx.clearRect(0, 0, canvas.width, canvas.height);
    ctx.fillStyle = '#000';
    ctx.fillRect(0, 0, canvas.width, canvas.height);

    drawMaze();
    pacman.update();
    ghosts.forEach(ghost => ghost.update());
    checkCollisions();

    requestAnimationFrame(gameLoop);
  }

  // Управление клавиатурой
  document.addEventListener('keydown', (e) => {
    if (!gameRunning) return;
    switch(e.key) {
      case 'ArrowLeft':
        pacman.setDirection('left');
        break;
      case 'ArrowRight':
        pacman.setDirection('right');
        break;
      case 'ArrowUp':
        pacman.setDirection('up');
        break;
      case 'ArrowDown':
        pacman.setDirection('down');
        break;
    }
  });

  // Мобильное управление
  upBtn.addEventListener('mousedown', () => pacman.setDirection('up'));
  upBtn.addEventListener('touchstart', () => pacman.setDirection('up'));
  leftBtn.addEventListener('mousedown', () => pacman.setDirection('left'));
  leftBtn.addEventListener('touchstart', () => pacman.setDirection('left'));
  rightBtn.addEventListener('mousedown', () => pacman.setDirection('right'));
  rightBtn.addEventListener('touchstart', () => pacman.setDirection('right'));
  downBtn.addEventListener('mousedown', () => pacman.setDirection('down'));
  downBtn.addEventListener('touchstart', () => pacman.setDirection('down'));
```

```
// Скорость
slowBtn.addEventListener('click', () => {
  gameSpeed = 0.7;
  slowBtn.classList.add('active');
  normalBtn.classList.remove('active');
  updateGameSpeed();
});

normalBtn.addEventListener('click', () => {
  gameSpeed = 1.0;
  normalBtn.classList.add('active');
  slowBtn.classList.remove('active');
  updateGameSpeed();
});

// 🎮 КНОПКА СТАРТА — показывает счёт!
startBtn.addEventListener('click', () => {
  startScreen.style.display = 'none';
  ui.style.display = 'flex'; // Показываем счёт
  gameRunning = true;
  initGame();
  gameLoop();
});

restartBtn.addEventListener('click', () => {
  gameOverScreen.style.display = 'none';
  gameRunning = true;
  initGame();
  gameLoop();
});
```



Рисунок 3. Главная страница



Рисунок 4. Процесс игры

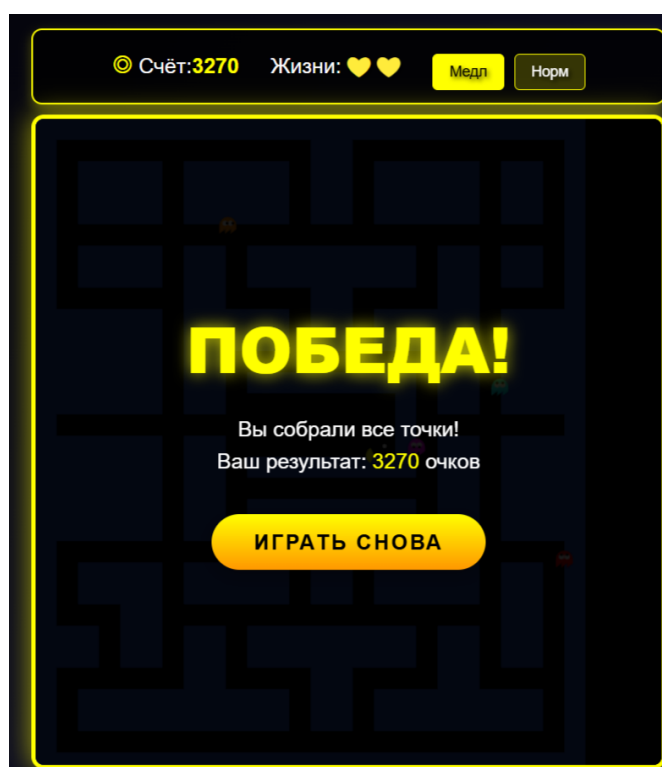


Рисунок 5. Конец игры

Если призрак касается Пакмана, то теряется жизнь.

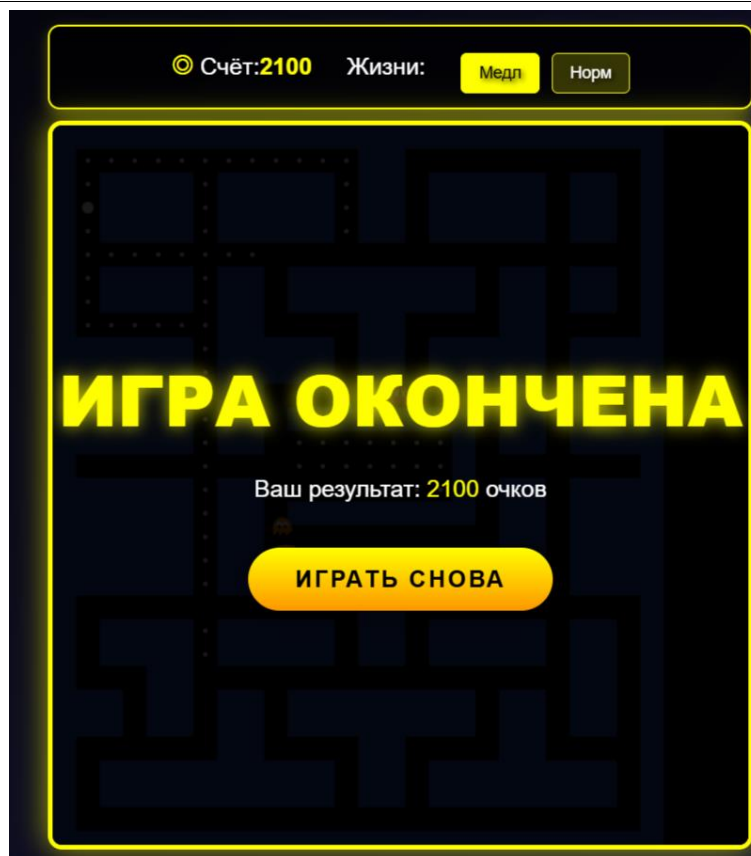


Рисунок 6. Конец игры

4 Выводы

В данной работе подробно рассмотрено создание классической игры Растап на современном веб-стеке с использованием языка HTML5, которая включает реалистичную физику движения персонажей, систему коллизий, адаптивное управление для различных устройств и динамическую визуализацию игрового процесса. Реализованы ключевые механизмы оригинальной игры: сбор точек и энерджайзеров, взаимодействие с призраками, система жизней и счета, а также специальные режимы игры с возможностью настройки скорости. 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.