ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. Н.П. ОГАРЁВА»

Факультет математики и информационных технологий
Кафедра фундаментальной информатики

УТВЕРЖДАЮ

Зав. кафедрой
канд. физ.-мат. наук, доц.
_____ А.Г. Смольянов
(подпись)
«10» 06 2019 г.

БАКАЛАВРСКАЯ РАБОТА

ПРИМЕНЕНИЕ ТЕХНОЛОГИЙ HTML5 ДЛЯ РАЗРАБОТКИ КОМПЬЮТЕРНЫХ ИГР

Автор бакалаврской работы 10.06.19 А. А. Пархачев
подпись, дата

Обозначение бакалаврской работы БР-02069964-02.03.02-11-19

Направление 02.03.02 Фундаментальная информатика и информационные технологии

Руководитель работы
канд. физ.-мат. наук 10.06.19 А. В. Попов
подпись, дата

Нормоконтролер
канд. техн. наук, доц. 10.06.19 С. В. Гарина
подпись, дата

Рецензент
канд. физ.-мат. наук, доц. 10.06.19 Л. А. Сухарев
подпись, дата

Саранск

2019

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. Н.П. ОГАРЁВА»

Факультет математики и информационных технологий
Кафедра фундаментальной информатики

УТВЕРЖДАЮ

Зав. кафедрой
канд. физ.-мат. наук, доц.
_____ А.Г. Смольянов
(подпись)
«12» 12 2018 г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

(в форме бакалаврской работы)

Студент Пархачев Андрей Александрович

1 Тема «Применение технологий HTML5 для разработки компьютерных игр»

Утверждена приказом № 10282-С от 12.12.2018

2 Срок предоставления работы к защите 13.06.19

3 Исходные данные для научного исследования литература по вопросам разработки игровых веб-приложений с использованием компьютерной графики

4 Содержание бакалаврской работы

4.1 Современный стандарт HTML5

4.2 Возможности для работы с графикой в HTML5

4.3 Разработка компьютерной игры средствами HTML5

5 Приложение код программы

Руководитель работы канд. физ.-мат. наук — 12.12.18 — А. В. Попов
подпись, дата

Задание принял к исполнению — 12.12.18 — А. А. Пархачев
подпись, дата

РЕФЕРАТ

Бакалаврская работа содержит 54 листа, 10 изображений, 5 таблиц, 7 использованных источников, 2 приложения.

HTML5, КОМПЬЮТЕРНАЯ ГРАФИКА, ИЗОБРАЖЕНИЕ, ТЕХНОЛОГИЯ, ТРЕХМЕРНЫЙ ОБЪЕКТ, JAVASCRIPT, BABYLON.JS, ИГРА, WEBGL, ИНТЕРНЕТ.

В качестве объекта исследования выбраны HTML5 технологии, предназначенные для создания веб-графики и игровых веб-приложений, предметом исследования являются аспекты различных веб-технологий, связанных с разработкой таких приложений.

Цель работы – изучить технологии, предназначенные для реализации браузерной версии игры «Тетрис». Для этого были поставлены следующие задачи:

– изучить историю создания, основные понятия и область применения технологии HTML5;

– рассмотреть возможности работы с веб-графикой посредством применения технологий HTML5;

– спроектировать и реализовать игровое веб-приложения с использованием веб-графики.

Область применения – сеть Интернет

В ходе исследования были изучены элементы компьютерной графики и технологии необходимые для реализации игрового приложения. Затем была реализована веб-браузерная игра «Тетрис».

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время глобальная сеть Интернет стала неотъемлемой частью жизни современного человека. К числу важнейших технологий Интернета можно отнести HTML (англ. Hyper Text Markup Language) – язык разметки гипертекста. Именно этот язык в прошлом заложил фундамент, на котором была построена Всемирная паутина. Современная версия HTML5 представляет, помимо прочего, единый стандартный способ описания встраиваемого в веб-страницы аудиовизуального контента.

Целью данной бакалаврской работы является изучение графических возможностей HTML5 и разработка с помощью этих возможностей браузерной компьютерной игры. В соответствии с целью были сформулированы следующие задачи:

- ознакомиться с историей развития языка HTML;
- изучить основные особенности и области применения HTML5;
- детально рассмотреть возможности для работы с графикой посредством HTML5;
- применить технологии HTML5 для реализации веб-браузерной версии игры «Тетрис».

Бакалаврская работа состоит из введения, трех глав и заключения.

В первой главе рассмотрены история развития и возможности языка HTML, новые элементы последнего стандарта данного языка, а также области его применения.

Во второй главе рассмотрены основные виды компьютерной графики, а также принципы её работы. Помимо этого, были рассмотрены веб-технологии, которые позволяют размещать графику на веб-страницах и работать с ней.

В третьей главе с учетом рассмотренных технологий, было спроектировано и разработано игровое веб-приложение «Тетрис».

1 Современный стандарт HTML5

Язык гипертекстовой разметки HTML (англ. HyperText Markup Language) является стандартизированным языком разметки документов, размещаемых в Интернете, этот язык интерпретируется браузерами при просмотре сайтов. В результате текстовое, графическое и мультимедийное содержимое сайтов отображется на экранах телефонов, планшетов и компьютеров.

1.1 История развития стандартов языка HTML

Язык HTML был разработан британским ученым Тимоти Бёрнсом-Ли в 1986 году и изначально предназначался для структурного форматирования научных документов, без каких либо элементов описания цветовых схем, параметров шрифта и т.п.

Через пять лет, в 1991 году данный язык претерпел существенные доработки своих технических спецификаций, что привело к использованию HTML для разметки и передачи гипертекста по Всемирной паутине.

Позже в 1994 году была создана организация W3C (англ. World Wide Web Consortium) – Консорциум Всемирной паутины, целью которой было проведение консультаций для лидеров компьютерной индустрии. Крупнейшие корпорации и компании в мире договаривались в W3C о взаимном обеспечении совместимости своих продуктов, а также внедрении новых технологических стандартов. Таким образом главным успехом консорциума стала стандартизация языка гипертекстовой разметки HTML в 1996 году и как следствие выпуск стандарта HTML 2.0, всё это позволило в свое время избежать множественного создания собственных стандартов различными компаниями и производителями ПО, тем самым сделав веб-страницы такими, какими мы их знаем сейчас.

Вскоре после выпуска версии 2.0 вышли версии 3.2, а затем и 4.0, и наконец, в 1999г – версия HTML 4.01, которая в последующем использовалась веб-разработчиками на протяжении более 10 лет. При этом следует отметить,

что каждая из выпущенных версий стандарта привносила что-то новое в веб-разработку, что способствовало быстрому развитию Интернета.

После выхода последней версии вектор веб-стандартизации сместился в сторону XML и XHTML, а HTML отошел на второй план. Однако значительная часть веб-контента базировалось именно на HTML, поэтому в 2004 году отдельно от консорциума W3C была создана рабочая группа WHATWG (Working Hypertext Application Technology Working Group), которая стала заниматься дальнейшем развитием HTML. После того, как W3C не достигла результатов по проекту XHTML2, организации решили объединиться и совместными усилиями реализовать спецификацию «WebAplications 1.0», часто неофициально называемую HTML5.

И вот 28 октября 2014 года HTML5 был официально рекомендован консорциумом W3C к использованию.

Ниже представлена таблица 1 с краткой информацией об изменениях и нововведениях в официально рекомендованных стандартах HTML.

Таблица 1 – Описание нововведений и изменений стандартов HTML

Версии	**Дата публикации**	**Нововведения и изменения**
HTML 2.0	24.05.1995	Возможность поиска, по ключевым словам; формы для передачи данных с компьютера на сервер
HTML 3.0, HTML 3.2	28.03.1995 14.01.1997	Поддержка разметки математических формул; вставка рисунков, обтекаемых текстом; поддержка gif формата; поддержка CSS
HTML 4.0, HTML 4.01	18.12.1997 24.12.1999	Исключение устаревших тегов; поддержка фреймов, скриптов

HTML 5	28.10.2014	Введение более строго синтаксиса; поддержка мультимедиа-технологий; добавление новых структурных элементов; исключение части устаревших тегов

1.2 Области применения HTML5

Основное назначение языка HTML состоит в создании документов для размещения их в форме веб-страниц в Интернете. Группа таких страниц, соединённых ссылками между собой называется, сайтом. Однако помимо этого HTML может быть использован для создания документов, содержащих ссылки, но не размещенных в Интернете:

- Учебные пособия
- Файлы справки
- Мультимедийные справочники и библиотеки

Выход нового стандарта HTML5 не только оживил и упростил веб-разработку, но и заметно расширил свою область применения.

Наиболее заметной стала возможность создание приложений для смартфонов на базе HTML5. Помимо этого некоторые технологии HTML5 стали применять для работы с компьютерной графикой, а также в разработке браузерных игр с различной степенью сложности. В дополнение заметно облегчилась разработка кросс-браузерных веб-приложений и верстка веб-страниц.

1.3 Особенности нового стандарта

Выход HTML5 вдохнул новую жизнь в веб-разработку, объединив в себе все удобства предыдущей версии, а также привнеся множество новых возможностей и перспектив. Хотя он и построен на основе предыдущего стандарта, необходимо отметить что HTML5 является не просто языком

разметки гипертекста, а новой открытой платформой для проектирования и создания веб-приложений, использующих расширенные возможности:

- Новые элементы пользовательского интерфейса.
- Поддержка двумерной и трехмерной графики.
- Технология воспроизведения видео/аудио потоков.
- Технология WebStorage.
- Элементы семантической разметки.

Новые элементы пользовательского интерфейса контролируют и обеспечивают ввод числовых значений, даты, времени, адреса электронной почты, цвета и т.п. Тем самым намного облегчается разработка дружественного интерфейса.

Поддержка графики реализуется через элемент <canvas>, который позволяет выделить пространство и использовать его для составления графических изображений практически любой сложности, в том числе и анимированных.

Технология воспроизведения аудио и видео потоков позволяет размещать на странице аудио и видео файлы с помощью специально предусмотренных тегов <audio> и <video>. При это все это происходит без использования сторонних плагинов или расширений типа Adobe Flash, это позволяет браузерам избавиться от дополнительных программ и как следствие снизить риск заражения компьютера вирусами при использовании их устаревших версий. Так, например, известно, что iOS полностью несовместима с Flash, а операционная система Android совместима лишь частично. В случае с HTML5 данная проблема легко решается и веб-мастеру уже не нужно размещать несколько версий медиа контента для «полноценных» компьютеров и мобильных устройств.

Технология Web Storage представляет программный интерфейс, посредством которого данные страниц могут быть сохранены в JavaScript объекты, использующие локальное или сессионное хранилища. Используя различные типы хранилища, данные могут храниться в течении одной

пользовательской сессии или же неограниченное время. Данный подход призван устранить недостатки использования cookie-файлов, одним из которых является постоянная передача файлов на сервер и обратно.

Элементы семантической разметки представляют собой новые теги определяющие блоки различных частей веб-страницы (рисунок 1.1).

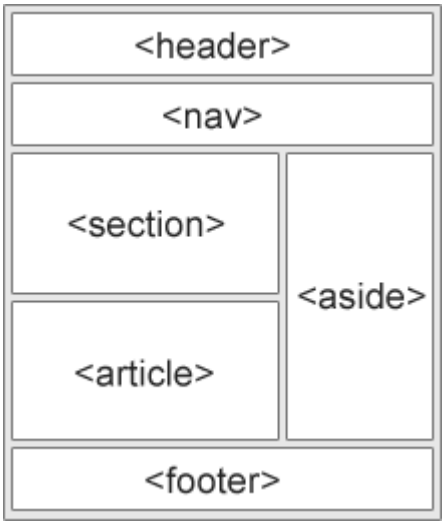

Рисунок 1.1 – Части веб-страницы

Данные теги добавляют разметке смысловой оттенок, делают код более удобочитаемым и простым [1]. Согласно спецификации HTML5 каждый элемент принадлежит к определенной категории, каждая из которых группирует элементы со схожими характеристиками. Выделяют следующие общие категории:

- Мета-содержимое.
- Потоковое содержимое.
- Секционное содержимое.
- Заголовочное содержимое.
- Текстовое содержимое.
- Встроенное содержимое.
- Интерактивное содержимое.

Ниже представлена таблица 2 с описанием семантических элементов и их назначением:

Таблица 2 – Описание семантических элементов и их назначения

Название элемента	**Назначение элемента**
<article>	Определение независимого, самодостаточного контента
<aside>	Определение контента, находящегося в стороне от контента, внутри которого он расположен
<details>	Определение дополнительной информации, которую пользователь может открывать или закрывать
<figcaption>	Определение подписи(пояснения) к тегу <figure>
<figure>	Группировка изображения и пояснения к нему
<footer>	Определение "подвала" документа или раздела
<header>	Определение заголовочного блока или "шапки" документа, или раздела
<main>	Определение основного контента документа
<mark>	Определение маркированного или подсвеченного текста
<nav>	Определение набора ссылок навигации
<section>	Определение раздела в документе
<summary>	Определение видимого заголовка элемента <details>
<time>	Определение даты и времени

2 Возможности для работы с графикой в HTML5

Компьютерная графика (также цифровая или машинная графика) – область деятельности, в которой компьютеры совместно со специальным программным обеспечением используются в качестве инструмента для создания и редактирования изображений, а также оцифровки визуальной информации, полученной из реального мира.

Существует несколько видов компьютерной графики:

- векторная
- растровая
- фрактальная

Каждый из видов графики имеет собственный способ формирования изображений на экране монитора, а также обладает своими преимуществами и недостатками.

Отдельным предметом считается трехмерная графика (3D графика), которая изучает приёмы и методы построения объемных объектов в виртуальном пространстве и сочетает в себе векторный и растровый способы формирования изображений.

2.1 Поддержка векторной графики

Векторная графика – способ описания объектов и изображений, с помощью использования математических формул и геометрических объектов, как правило называемых примитивами.

Линию можно назвать базовым элементом в векторной графике, и поскольку линия математически описывается как единый объект, отсюда следует вывод, что объем данных для отображения объекта средствами векторной графики существенно меньше, чем в той же растровой графике.

В векторной графике линия имеет различные свойства:

- форма (прямая или кривая),
- толщина,
- цвет,

- начертание (сплошная или пунктирная).

Замкнутые линии могут приобрести свойство заполнения. Пространство, охватываемое ими может быть заполнено определенным цветом или же другими объектами (например, текстурами).

Все прочие объекты векторной графики состоят из линий. Например, прямоугольник состоит из четырёх связанных линий, а из шести связанных прямоугольников можно составить куб.

Как уже было замечено векторная графика оперирует примитивами, каждый из которых имеет собственные представления и параметры необходимые для их построения. Ниже представлены некоторые объекты векторной графики:

- точка,
- прямая линия,
- отрезок прямой,
- кривая второго порядка,
- кривая третьего порядка,
- кривые Безье.

Данные примитивы имеют различные способы представления в объектно-ориентированной графике [5].

Например, для изображения точки на плоскости используются два числа (x, y), которые определяют положение точки относительно начала координат.

Для изображения более сложных объектов, используются более комплексные понятия. Например, для построения прямой линии в известной системе координат используется уравнение вида: $y=kx+b$, где k и b – параметры, которые необходимо указать для её задания. Для кривых второго и третьего порядка уравнения имеют более сложный вид:

$A_1x^2 + A_2y^2 + 2\,A_3x + 2\,A_4y + A_5 = 0,$

$x^3 + A_1y^3 + A_2x^2y + A_3xy^2 + A_4x^2 + A_5y^2 + A_6xy + A_7x + A_8y + A_9 = 0.$

Теперь можно выделить достоинства и недостатки векторной графики.

К **достоинствам** векторной графики можно отнести: малый размер изображения, ввиду того что объём данных, который используется для описания объектов не зависит от реальной величины объекта. К тому же параметры объектов векторной графики хранятся в памяти компьютера, что позволяет их легко изменить (переместить, масштабировать, вращать, заполнить) без потери качества изображения.

Тем не менее у векторной графики есть и **недостатки**, самый очевидный из них – это невозможность получения изображения фотографического качества. Поэтому не каждая графическая сцена может быть изображена в векторном виде, так как порой требуется описать большое количество примитивов с высокой сложностью.

Для размещения векторной графики на веб-страницах необходимо использовать специальный язык разметки масштабируемой векторной графики SVG (англ. Scalable Vector Graphics).

SVG является подмножеством расширяемого языка разметки XML и служит для описания двумерной векторной или смешанной векторно-растровой графики. Также SVG поддерживает статическую и анимированную графику [9].

В SVG существует возможность работы с тремя типами объектов:

- изображения,
- текст,
- векторные графические формы.

В HTML5 существует возможность включить SVG непосредственно в HTML-код страницы с помощью элемента <svg>, тем самым исключается необходимость отправки запросов браузером на сервер.

Рассмотрим пример построения круга, квадрата и треугольника средствами SVG:

```
<!DOCTYPE>
<html>
<body>
```

```
<svg width="500" height="200" >
<circle cx="75" cy="75" r="50" stroke="black" stroke-width="3" fill="yellow" />
<rect x="175" y="25" width="100" height="100" stroke="black" stroke-width="2" fill="blue"/>
<polygon points="320,125 370,25 420,125" stroke="black" stroke-width="1" fill="#00ff00"/>
</svg>
</body>
</html>
```

Данный код должен быть помещен в файл с расширением html, результат работы отображается на экране с помощью любого браузера (рисунок 2.1).

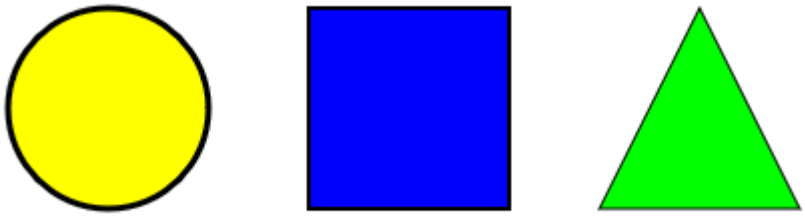

Рисунок 2.1 – Векторная графика средствами SVG

Базовым элементом в программе является элемент `<svg>` с атрибутами `width` и `height`, которые задают ширину и высоту окна для вывода векторной графики. Между открывающим и закрывающим тегом `<svg>` располагаются фигуры, которые необходимо отобразить на веб-странице. В данном случае это круг, квадрат и треугольник. Каждому из них соответствует свой тег: `<circle>`, `<rect>`, `<polygon>`. Все эти теги имеют собственные атрибуты, задающие размер, положение, рамку фигуры и многое другое.

Стоит отметить, что атрибуты могут быть как обязательными, так и необязательными (презентационными). Рассмотрим элементы и их атрибуты, отвечающие за создание фигур, более детально.

Элемент `<circle>` определяет круг и имеет следующие обязательные атрибуты:

- `cx` –координата x центра круга
- `cy` - координата y центра круга
- `r` – радиус круга

Элемент `<rect>` определяет прямоугольник, его обязательные атрибуты:

- `x` – координата x верхнего левого угла
- `y` – координата y верхнего левого угла
- `width` – ширина прямоугольника
- `height` – высота прямоугольника

Элемент `<polygon>` в контексте данного примера определяет треугольник, однако на самом деле этот элемент определяет графическую фигуру, у которой по меньшей мере есть три стороны. Таким образом, фактически он может быть использован для рисования различного рода многоугольников.

Элемент `<polygon>` имеет один обязательный атрибут `points`. Данный атрибут определяет координаты x и y каждого угла многоугольника.

2.2 Поддержка растровой графики

Растровая графика – способ вывода изображений, основанный на работе с сеткой пикселей на мониторах или других отображающих устройствах. Пиксель является наименьшим логическим элементом двумерного цифрового изображения в растровой графике и представляет собой неделимый объект прямоугольной, реже круглой формы, который характеризуется определенным цветом.

Основными характеристиками и понятиями растровых изображений являются:

- размер изображения,

- глубина цвета,
- цветовое пространство.

Размер изображения выражается в виде количества пикселей по ширине и высоте (например, 640x480px, 1024x768px, 3200x2048px и т. д.), помимо этого размер изображения можно выразить и суммарным количеством пикселей. Так, изображение размером 3200x2048px состоит из 6553600 точек или примерно 6,5 мегапикселей.

Глубина цвета – это термин компьютерной графики, означающий объем памяти, который используется для хранения и представления цвета при кодировании одного пикселя растрового изображения. Чаще всего глубина цвета выражается единицей бит на пиксель (англ. Bits per pixel, **bpp**).

Цветовое пространство – это модель представления цвета. Данная модель строится таким образом, чтобы каждый цвет был представлен точкой, и обладал собственными координатами [5].

Растровая графика может быть получена с помощью фотоаппаратов, сканеров, а также создана с помощью растрового редактора или даже конвертирована из векторного редактора.

К преимуществам вывода графики данным способом можно отнести:

- возможность создать изображение практически любой сложности;
- распространённость;
- высокая скорость обработки изображений, не требующих масштабирования.

Недостатками являются:

- большой размер файлов у изображения;
- отсутствие возможности увеличения изображения;
- отсутствие возможности разбиения изображения на отдельные части и их редактирование.

До появления HTML5 работа по созданию растровой графики могла быть выполнена только с помощью сторонних библиотек или расширений типа Flash или ActiveX.

На текущий момент HTML5 предоставляет возможность для работы с графикой путем добавления элемента <canvas> на страницу и взаимодействия с ним. Данный элемент предназначен для динамического генерирования двумерных фигур и растровых изображений с помощью скриптов (чаще всего написанных на JavaScript). По сути <canvas> является контейнером или растровым холстом, используемым для рисования графических объектов.

Для работы с элементом <canvas>, как и для работы с элементом <svg>, требуется создать HTML файл и поместить в него код:

```
<!DOCTYPE>
<html>
<body>
<canvas id="testCanvas" width="200" height="200" >
</canvas>
</body>
</html>

<script>
var canva = document.getElementById("testCanvas");
var context = canva.getContext("2d");

context.fillStyle = "#ff0";
context.fillRect(20, 20, 100, 100);
</script>
```

Результатом работы данного скрипта будет вывод на экран красного квадрата (рисунок 2.1).

Рисунок 2.1 – Растровая графика средствами HTML5

Теперь подробно разберем данный пример.

```
<canvas id="testCanvas" width="200" height="200">
```

О назначении тега `<canvas>` было уже сказано ранее, однако в данном примере он имеет важные атрибуты, которые необходимо рассмотреть:

- `id` – это уникальный идентификатор, который позволяет найти в документе холст `<canvas>` для последующей работы с ним.
- Атрибуты `width` и `height` задают ширину и высоту холста соответственно.

После того как элемент `<canvas>` создан, можно приступить к работе с ним. Для этого мы, используя язык сценариев JavaScript, создаем переменную canva и присваиваем ей объект `<canvas>` с помощью метода `getElementById("testCanvas")`.

Далее необходимо создать рисующий объект, для этого используем метод `getContext("2d")`, который позволяет создать нужный объект со свойствами и методами для рисования:

```
var context = canva.getContext("2d");
```

Наконец можно приступить к рисованию:

```
context.fillStyle = "#ff0";
context.fillRect(20, 20, 100, 100);
```

Используя свойство `fillStyle`, можно установить цвет заливки графического объекта в любой цвет, а вызвав метод `fillRect(x,y,width,height)` можно нарисовать прямоугольник,

начиная с координат x и y, определенной ширины и высоты, которые задаются с помощью параметров `width` и `height`. Необходимо также учесть, что отсчет координат в области для рисования начинается с нуля как для оси OX, так и для OY и находится в левом верхнем углу.

2.3 Поддержка трехмерной графики

Трехмерная графика – набор приёмов и инструментов, которые позволяют создать изображение или видео, путём моделирования объёмных объектов в трехмерном пространстве.

Для моделирования трехмерных объектов необходимо знать базовые понятия 3D графики:

- **Моделирование** – это процесс создания трехмерной математической модели объекта.
- **Наложение текстур** – это установка поверхностям модели, определенных растровых изображений, называемых текстурами.
- **Настройка освещения** позволяет правильно выставить свет, который падает на объекты, тем самым придавая им большую реалистичноть.
- **Выбор точки наблюдения** помогает взглянуть на объекты с различной стороны.
- **Рендеринг и визуализация** являются заключительным этапом создания трехмерной модели, необходимым для более точной детализации и настройки.

Для работы с компьютерной графикой существуют графические стандарты, однако не каждый из них подходит для работы с графикой в сети Интернет, а уж тем более с 3D графикой. Несмотря на то, что в HTML5 присутствует элемент canvas, обеспечить работу с 3D графикой только с его помощью невозможно.

По этой причине была создана технология WebGL (англ. Web-based Graphics Library), позволяющая отображать и совершать манипуляции с 3D

графикой на веб-страницах. WebGL исполняется как элемент HTML5 и по этой причине является частью объектной модели документа браузера, что в свою очередь позволяет использовать JavaScript или любой другой язык программирования, который поддерживает работу с DOM [3].

Эта технология примечательна тем, что она не требует установки дополнительного программного обеспечения, так как большинство современных браузеров поддерживают WebGL по умолчанию. Тем не менее использование «чистого» WebGL для создания 3D графики является трудоёмким и сложным процессом.

По этой причине для упрощения разработки графических приложений были созданы различные библиотеки и фреймворки. Наиболее популярными из них являются: WebGLU, GLGE, C3DL, Processing.js, Three.js и Babylon.js.

В качестве примера в данной работе будет использован JavaScript-фреймворк Babylon.js. Этот фреймворк базируется на WebGL API и имеет ряд преимуществ: небольшой объём занимаемой памяти, отсутствие необходимости в использовании сторонних дополнений, а также поддержка различными браузерами.

Данный фреймворк поддерживает следующие функции:

- Готовые 3D объекты
- Физический движок
- Поддержка сглаживания
- Анимационный движок
- Звуковой движок
- Система частиц
- Пошаговая загрузка сцены
- Блики
- Панель отладки
- Различные источники освещения
- Несколько видов камеры
- Карта высот

- Поддержка экспорта моделей в 3ds Max, Unity3D, Blender

Babylon.js состоит из большого числа компонентов, функционирующих вместе, которые при необходимости можно расширить. Поэтому данный фреймворк можно применять не только для создания простых 2D или 3D объектов, а также для более сложных вещей, будь то обработка звука, технические расчеты или даже исследования в области машинного зрения.

Теперь перейдем к реализации 3D изображения средствами Babylon.js. В качестве примера создадим 3D куб (полный текст сценария приведен в листинге 1 Приложения А).

Данный фреймворк работает непосредственно с HTML5-элементом canvas, поэтому процесс создания, размещения и получения элемента canvas можно опустить, так как он практически не отличается от аналогичного процесса при работе с растровой графикой.

Единственным изменением в структуре HTML-разметки является строка внутри тега <head>, которая отвечает за подключение фреймворка Babylon.js:

```
<script src=”/scripts/babylon.js”></scripts>
```

Теперь рассмотрим скрипт, написанный с помощью данного фреймворка. Для начала работы необходимо запустить движок Babylon.js и создать поле в котором будет размещаться наш куб, делается это с помощью методов класса BABYLON:

```
var babylonEngine = new BABYLON.Engine(canva);
var babylonScene = new BABYLON.Scene(babylonEngine);
```

Далее необходимо создать и настроить камеру таким образом, чтобы она имела возможность вращаться и была направлена на область, где будет располагаться наш куб:

```
var babylonCamera= new BABYLON.ArcRotateCamera("cam
era", 0, Math.PI/2,10,new BABYLON.Vector3(0,0,0),babylo
nScene);
```

Первый аргумент метода `ArcRotateCamera` задает название данной камеры. Второй и третий аргументы задают углы отклонения камеры, таким образом, чтобы камера могла двигаться только в верхней полусфере. Третий и четвертый аргументы задают радиус вращения и начальное расстояние относительно позиции камеры. Предпоследний аргумент задаёт позицию камеры. Наконец, последний аргумент прикрепляет камеру к текущей сцене.

Теперь, когда сцена и камера созданы, можно попытаться разместить наш куб (рисунок 2.2). Для этого используем следующий код:

```
Var cube = BABYLON.Mesh.CreateBox("cube",3, babylon Scene);
babylonEngine.runRenderLoop(function(){
    babylonScene.render();
});
```

Метод `CreateBox` создает куб с именем «cube», размером равным 3, который принадлежит текущей сцене. В свою очередь метод `runRenderLoop()` запускает цикл, который перерисовывает сцену 60 раз в секунду, тем самым обеспечивая возможность отображения состояния куба при его вращении.

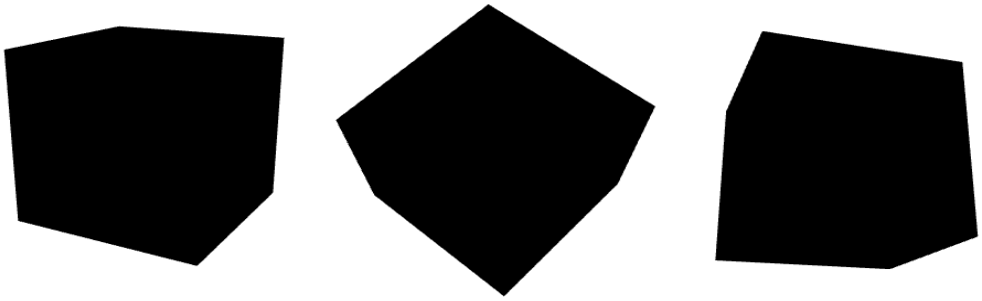

Рисунок 2.2 – Объёмный куб без источника света

Как можно заметить, куб имеет полностью черный цвет, что значительно осложняет его восприятие в качестве 3D объекта.

Для того, чтобы можно было различать его грани, на сцену необходимо добавить источник света:

```
Var babylonLight= new BABYLON.HemisphericLight("hLight", new BABYLON.Vector3(0,1,2),babylonScene);
```

В данном случаем метод `HemisphericLight` создает атмосферный свет, позиция которого, как и для камеры, задана трехмерным вектором. В результате мы получили 3D куб, который можно рассмотреть с различных сторон (рисунок 2.3).

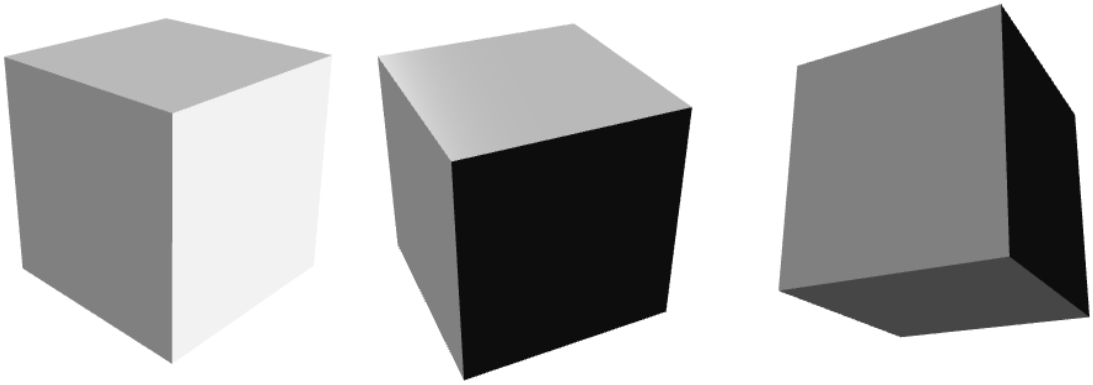

Рисунок 2.3 – Объёмный куб с источником света

2.4 Анимация изображений

Анимация – это компьютерная технология, воспроизводящая последовательности картинок, что в свою очередь создает впечатление движущегося изображения.

Нарисовать одну картинку может казаться довольно трудной задачей. Поэтому не удивительно, что рисование нескольких десятков изображений каждую секунду выглядит еще более затруднительным процессом.

Основная трудность при работе с анимацией состоит в необходимости обновлять содержимое холста достаточно быстро, чтобы оно выглядело натурально движущимся или изменяющимся. Оптимальная скорость смены изображения для человека равна 24 кадра в секунду. На самом деле в одном кадре может присутствовать несколько изображений (слоев) или всего одно, которое в каждом последующем кадре будет сменяться на небольшую величину.

Таким образом можно сделать вывод, что создание анимации является действительно трудоемким занятием, так как в большинстве случаев необходимо перерисовывать каждый кадр заново. Для упрощения создания двухмерной и трехмерной анимации используют специальное оборудование и программное обеспечение. Ниже представлен список наиболее известных программ для работы с анимацией:

- Microsoft GIF Animation
- Adobe Animate
- Synfig
- Pivot Animator
- Adobe Flash
- Blender
- Autodesk 3ds Max

В веб разработке анимация чаще всего используется для акцентирования внимания на изменении содержимого веб-страницы. Например, для выделения действий пользователя (подсветить кнопку при на ведении на неё мышкой, или придать ей красивый эффект сдвига и т.д.).

Анимация на веб страницах может быть создана несколькими способами, в том числе с помощью элемента HMTL5 canvas.

Сделать анимацию на холсте достаточно просто. Основная идея состоит в том, чтобы установить таймер, который постоянно будет вызывать элемент несколько десятков раз в секунду. При каждом таком вызове будет происходить обновление содержимого всего холста. И в случае если код написан правильно, постоянно сменяющиеся кадры сольются в плавную, реалистичную анимацию [8].

Язык JavaScript предоставляет несколько способов для управления повторяющимся обновлением содержимого холста. Наиболее подходящее решение этой задачи – использование функций `setTimeout()` или `setInterval()`. Эффект от использования данных функций практически идентичен. Разница этих функций состоит в том, что функция

`setTimeout()` дает указания браузеру подождать несколько миллисекунд, и после этого исполнить фрагмент кода, в то время как функция `setInterval()` выполняет фрагмент кода через определенный интервал времени, вызывается один раз и выполняется до тех пор, пока браузером не будет исполнена функция `clearInterval()`.

Используя возможности HTML5 создадим простую анимацию движения фигуры (рисунок 2.4).

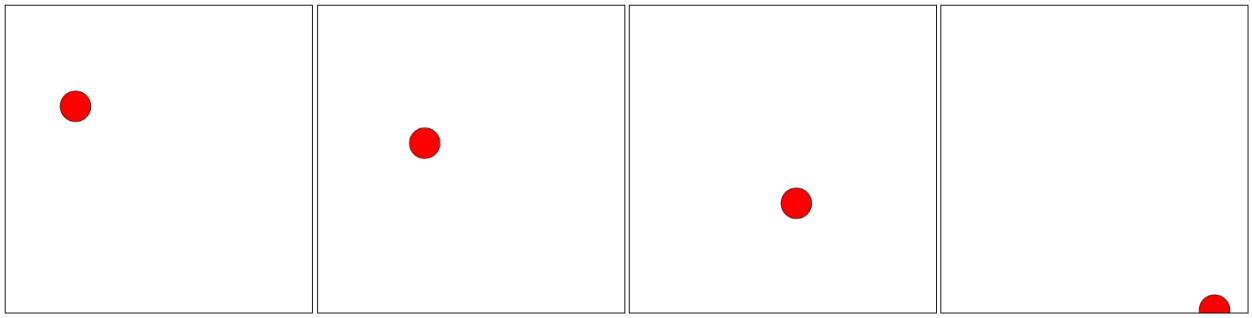

Рисунок 2.4 – Анимация движения круга

Как и в предыдущих примерах, нам понадобится создать HTML файл и поместить в него код (см. листинг 2 Приложение А).

Основными элементами данного кода являются функции `setTimeout()` и `draw()`. При загрузке страницы вызывается функция `setTimeout("draw()",20)` с двумя параметрами. Первый параметр указывает на то какую функцию следует вызывать, а второй аргумент отвечает за то, с какой задержкой будет происходить ее вызов.

При вызове функции `draw()` в первую очередь происходит очистка всей области для рисования. Сделано это для того, чтобы состояние фигуры на предыдущей итерации не отображалось на экране.

После этого с помощью метода `arc(circlePos_x, circlePos_y,circleRadius,0,2*Math.PI)` создаётся круг с центром и радиусом, которые изначально заданы следующими параметрами:

```
var circlePos_x = 100;
var circlePos_y = 150;
var circleRadius = 25;
```

Далее мы увеличиваем координаты x и y нашей фигуры на единицу и снова с помощью функции `setTimeout()` вызываем функцию `draw()` с задержкой в 20 миллисекунд. Данный процесс повторяется до тех пор, пока круг не выйдет за границы холста. Таким образом, вызывая функцию `setTimeout()` с различной задержкой, можно ускорить или замедлить анимацию.

3 Разработка компьютерной игры средствами HTML5

В качестве практического примера, иллюстрирующего использование технологий HTML5, была выбрана задача написания классической компьютерной игры «Тетрис», разработанной советским программистом Алексеем Пажитновым.

3.1 Проектирование приложения

Игра «Тетрис» представляет собой головоломку, построенную на фигурах, называемых «тетрамино». Данные фигуры состоят из четырех квадратов, представляющих собой разновидность полимино (плоские геометрические фигуры, образованные путём соединения нескольких одноклеточных квадратов по их сторонам).

Правила игры таковы: случайные фигуры «тетрамино» падают в стакан шириной в 10 и высотой 20 клеток. Во время полета фигуры, её можно передвигать, ускорять, а также вращать на 90°. Фигура падает до тех пор, пока не столкнется с другой фигурой или не окажется на дне стакана.

Если фигуры заполнят ряд в 10 клеток, то ряд исчезнет, а фигуры, стоящие выше и не имеющие опоры, упадут на одну клетку вниз. Сложность состоит в том, что каждый раз, как только фигуры складываются в ряд, темп игры постепенно увеличивается, что осложняет выбор правильной траектории фигуры. Игра заканчивается, как только новая фигура не сможет поместиться в стакан.

Помимо правил игры и её описания необходимо рассмотреть входные и выходные данные программы.

Входные данные: клавиши «W», «S», «A», «D», «E», нажимаемые в процессе игры в режиме реального времени. Каждая клавиша выполняет следующие действия:

- W – поворот фигуры на 90°
- S – ускорение фигуры
- A – движение фигуры влево

- D – движение фигуры вправо
- E – мгновенное падение фигуры вниз

Выходные данные: Информация, выводимая на экран в графическом режиме: рисунок стакана (игрового поля), падающие и лежащие на дне фигуры, а также поле, отображающее очки игрока и кнопку «Start».

Для корректного проектирования игры нужно рассмотреть дополнительные задачи:

- Реализация основного игрового цикла
- Способ представления игрового поля и фигур
- Движение фигур и обработка столкновений
- Отображение игрового процесса

Рассмотрим данные пункты подробнее:

Основной игровой цикл представляет собой центральный блок, на основании которого строится игра. В данном приложении игровой цикл в общем виде состоит из следующих пунктов:

- Обработка действий пользователя, а именно взаимодействие и управление игровым процессом (перемещение, поворот фигур).
- Обновление состояния игры в соответствии с изменениями, внесенными пользователем, или без их учета.
- Отображение игрового процесса на экране.

Игровое поле представлено в программе в виде двумерного массива. Так как игровое поле имеет размеры 20 в высоту и 10 в ширину клеток, то и массив, которым оно задано имеет соответствующий размер. Таким образом данный двумерный массив состоит из клеток (или ячеек). Наличие в ячейке числа, отличного от нуля означает, что она занята фрагментом фигуры и пуста в противном случае [10].

В данной игре будет реализовано 7 типов фигур, представленных на рисунке 3.1. Каждая фигура имеет свою форму и по аналогии с игровым полем описана в виде двумерного массива чисел.

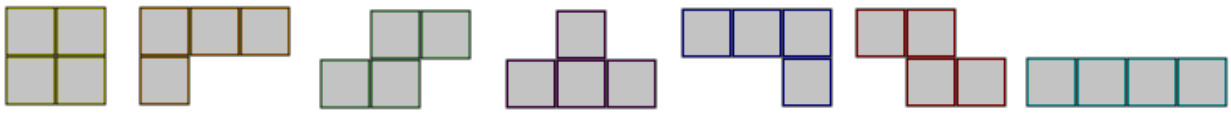

Рисунок 3.1 – Разновидности фигур

Движение фигур, как и проверка на их столкновение с препятствием, осуществляется довольно просто. Для этого необходимо знать направление смещения фигуры, её координаты, а также границы игрового поля. Перед смещением фигуры в заданном направлении нужно проверить координаты ячеек, на которые фигура должна будет сместиться. Если они свободны, то можно переместить или повернуть фигуру в заданном направлении. Умея обрабатывать столкновения, можно не допустить выход фигур за пределы игрового поля. Кроме того, проверка столкновения перемещаемой фигуры с другими фигурами позволит определить, когда необходимо будет окончить игру.

Отображение игрового процесса осуществляться с помощью элемента HTML5 canvas. Так как игра идет в реальном времени, изображение поля должно обновляться со скоростью не ниже скорости обновления состояния самой игры. В противном случае могут произойти разрывы в изображении, что приведет к некорректному отображению игрового процесса.

3.2 Реализация игры «Тетрис»

Для реализации игрового приложения «Тетрис» были выбраны следующие технологии: HTML5, CSS, JavaScript.

Рабочий проект состоит из нескольких файлов представленных в таблице 3.

Таблица 3 – Файловая структура проекта

Название файла	**Назначение файла**
index.html	Отвечает за разметку страницы с игровым приложением, а также за подключение файлов скриптов написанных на JavaScript и стилевых оформлений страницы, код которых написан с помощью технологии CSS
style.css	Описание оформления внешнего вида страницы
game_main.js	Описание основных переменных и функций, а также реализация основной логики игры
game_contoller.js	Вспомогательный код, который отслеживает нажатие заданных клавиш пользователем и передаёт их в основную программу
Game_render.js	Перерисовка и отображение игрового процесса на холсте, расположенного на странице

Одними из основных элементов index.html являются элемент `canvas` и блок `div`, содержащий кнопку для начала игры и строку для отображения счёта игрока (рисунок 3.2). В файле index.html последовательно подключаются сценарии на языке JavaScript, в которых прописана основная логика игры:

```
<script src='game_main.js'></script>
<script src='game_controller.js'></script>
<script src='game_render.js'></script>
```

Файл game_main.js должен быть встроен в документ в первую очередь, так как он содержит основные переменные и функции, которые используются в других скриптах, подключаемых позже.

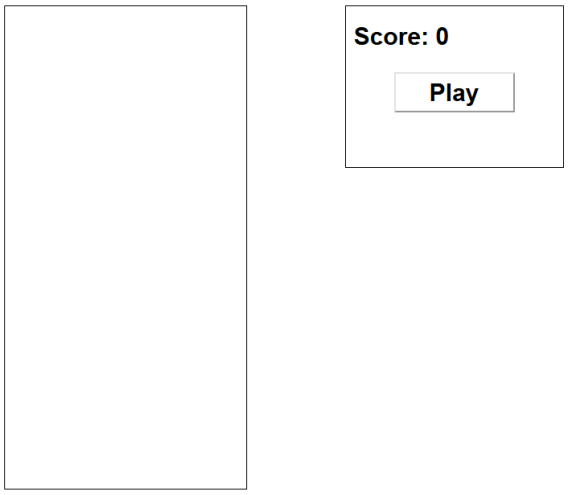

Рисунок 3.2 – Игровое поле

Для написания игрового приложения были использованы следующие функции и основные переменные представленные в таблице 4 и таблице 5.

Таблица 4 – Переменные

Переменная	**Описание переменной**
COLS	Количество столбцов игрового поля
ROWS	Количество строк игрового поля
Field	Поле игры
GameOver	Логическая переменная состояния игры
Interval	Ссылка на метод setInterval(tick, 500)
intervalRender	Ссылка на метод setInterval(render, 40)
currentShape	Активная фигура
current	Позиция активной фигуры по оси X
currentY	Позиция активной фигуры по оси Y
Freezed	Состояние движения активной фигуры
Figures	Массив возможных фигур
colors	Массив цветов
Score	Набранные очки

Keys	Массив с кодами клавиш управления
BLOCK_W	Ширина одной клетки на игровом поле в пикселях
BLOCK_H	Высота одной клетки на игровом поле в пикселях

Таблица 5 – Функции

Функция	**Описание функции**
initialization()	Подготовка игрового поля
createNewShape()	Создание новой фигуры на поле
freeze()	Фиксирование текущей фигуры на поле
rotate(current)	Поворот фигуры current
checkCollisions(offsetX, offsetY, newCurrent)	Проверка на столкновение
tick()	Обработка падения фигуры, и проверка конца игры
newGame()	Создание новой игры
clearLines()	Проверка строк игрового поля и их очистка
keyPress(key)	Обработка нажатия клавиши
clearAllIntervals()	Остановка функций setInterval()
drawBlock(x, y)	Рисование фигуры на заданной позиции
render()	Отображение состояния поля и фигур

Игра начинается с нажатия на кнопку «Play». При этом вызывается функция `newGame()`, которая запускает игровой цикл и состоит из следующих шагов:

1) Вызов функции `initialization()` – очистка игрового поля путем присваивания элементам массива `field` нулей.
2) Вызов функции `createNewShape()` – присваивание переменной `currentShape` случайной фигуры из массива `figures`.

3) Начальная инициализация переменных `gameOver` и `score`.
4) Очистка запущенных функций `setinterval()`, если таковые имеются.
5) Вызов функций `tick()` и `render()` с интервалами 40 и 500 миллисекунд соответственно, что приводит к циклическому обновлению как графической, так и логической составляющей игры.

Таким образом создается случайная фигура, взятая из массива `Figures`, которая размещается в переменной `currentShape` и располагается на верхней линии игрового поля (рисунок 3.3). Данная фигура является активной и имеет координаты, заданные переменными `currentX` и `currentY`, которые изначально равны 4 и 0 соответственно. Все эти действия описываются в функции `createNewShape()`:

```
function createNewShape() {
     // генерация случайного числа
    var id = Math.floor( Math.random() * 7 );
     //инициализация активной фигуры, из массива
фигур figures
    currentShape = figures[id];

    // отмена заморозки фигуры
    freezed = false;
    // начальная позиция новой фигуры
    currentX = 4;
    currentY = 0;
}
```

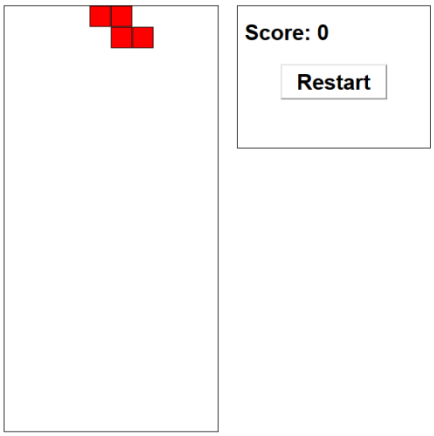

Рисунок 3.3 – Появление первой фигуры на поле

При нажатии пользователем на клавиши происходит вызов события `onkeydown`, для которого определён обработчик:

```
window.onkeydown = function( e ) {
    var keys = {
        65: 'left',
        68: 'right',
        83: 'down',
        87: 'rotate',
        69: 'drop'
    };
    if ( typeof keys[ e.keyCode ] != 'undefined' ) {
        keyPress( keys[ e.keyCode ] );
        render();
    }
};
```

В данном обработчике определён массив `keys` с кодами клавиш, которые соответствуют клавишам: «W», «A», «S», «D», «E». При нажатии любой клавиши обработчику передаётся её код, после чего происходит проверка наличия данной клавиши в массиве `keys`. В случае если клавиша относится к данному списку, она автоматически передаётся в функцию `keyPress()`, которая в свою очередь совершает определённые действия над

фигурой с учетом переданной в неё клавиши. Основная задача данной функции – выполнить смещение фигуры или её поворот, путём изменения переменных `currentX`, `currentY`, или изменения самого массива фигуры currentShape с помощью функции `rotate()` .

Функция `keyPress()` выглядит следующим образом:

```
function keyPress( key ) {
    switch ( key ) {
        case 'left':
            if ( checkCollisions( -1 ) ) {
                //смещение фигуры влево
                --currentX;
            }
            break;
        case 'right':
            if ( checkCollisions( 1 ) ) {
                //смещение фигуры вправо
                ++currentX;
            }
            break;
        case 'down':
            if ( checkCollisions( 0, 1 ) ) {
                //смещение фигуры вниз (ускорение)
                ++currentY;
            }
            break;
        case 'rotate':
            //поворот фигуры
            var rotated = rotate( currentShape );
            if ( checkCollisions( 0, 0, rotated ) )
{
```

```
                currentShape = rotated;
            }
            break;
        case 'drop':
            //мгновение падение фигуры
            while( checkCollisions(0, 1) ) {
                ++currentY;
            }
            tick();
            break;
    }
}
```

Отметим, что до изменения позиции фигуры на поле необходимо обработать её столкновения с препятствиями. Любая попытка изменения состояния фигуры на поле сопровождается вызовом функции `checkCollisions(offsetX, offsetY, newCurrent)`. Параметры, которые предаются в данную функцию, зависят от типа перемещения фигуры.

Параметр `offsetX` может принимать следующие значения:

- -1, если смещение происходит влево
- 0, если фигура не смещается по оси X
- 1, если смещение происходит вправо

Параметр `offsetY` может принимать значения:

- 0, если фигура не смещается по оси Y
- 1, если фигура смещается вниз

Третий параметр `newCurrent` принимает значение фигуры, развернутой на 90 градусов, с помощью функции `rotate(currentShape)`, и используется в случае, когда смещения по осям не происходит, но фигуру пытаются повернуть.

В случае если смещение или разворот фигуры на заданные значения возможен, происходит ее сдвиг в соответствующую сторону. В противном

случае фигура остается в изначальном положении. Рассмотрим основные приемы, которые использовались для реализации функции `checkCollisions(offsetX, offsetY, newCurrent)`.

```
offsetX = offsetX || 0;
offsetY = offsetY || 0;
newCurrent = newCurrent || currentShape;
```

Данный код нужен для инициализации параметров, передаваемых в функцию, в случае, если они не были указаны. Например, при передвижении фигуры влево функция вызывается со следующими параметрами: `checkCollisions( -1 );`

После инициализации запускается цикл, перебирающий значения массива `newCurrent`, в которых хранятся ячейки фигуры, и последовательно проводится ряд проверок с помощью оператора `if`:

- `typeof field[ y + offsetY ] == 'undefined'` – проверка на выход за границы поля по оси Y
- `typeof field[ y + offsetY ][ x + offsetX ] == 'undefined'` – проверка на выход за границы поля по оси X
- `field[ y + offsetY ][ x + offsetX ]` – проверка на наличие элемента фигуры на поле по заданным координатам
- `offsetY == 1 && freezed` – проверка на окончание игры

Если хотя бы одно из условий оказалось верным, функция `checkCollisison()` возвращает значение `false`, что означает, что фигура не может быть перемещена на заданную позицию, так как для нее существует препятствие, и значение `true` в противном случае.

Функция `tick()` с заданным интервалом опускает фигуру вниз. Если при этом активная фигура столкнулась с фигурой, закреплённой на поле, произойдут следующие действия:

- Активная фигура закрепляется на текущей позиции с помощью функции `freeze()`.
- Происходит освобождение заполненных линий и смещение фигур на позицию ниже с помощью `clearLines()`.
- Происходит проверка свободного места для появления новой фигуры.
- Создаётся новая фигура с помощью функции `createNewShape()`.

Если свободного места для создания новой фигуры не оказалось, то на экран выводится сообщение о завершении игры и очки, набранные игроком (рисунок 3.4).

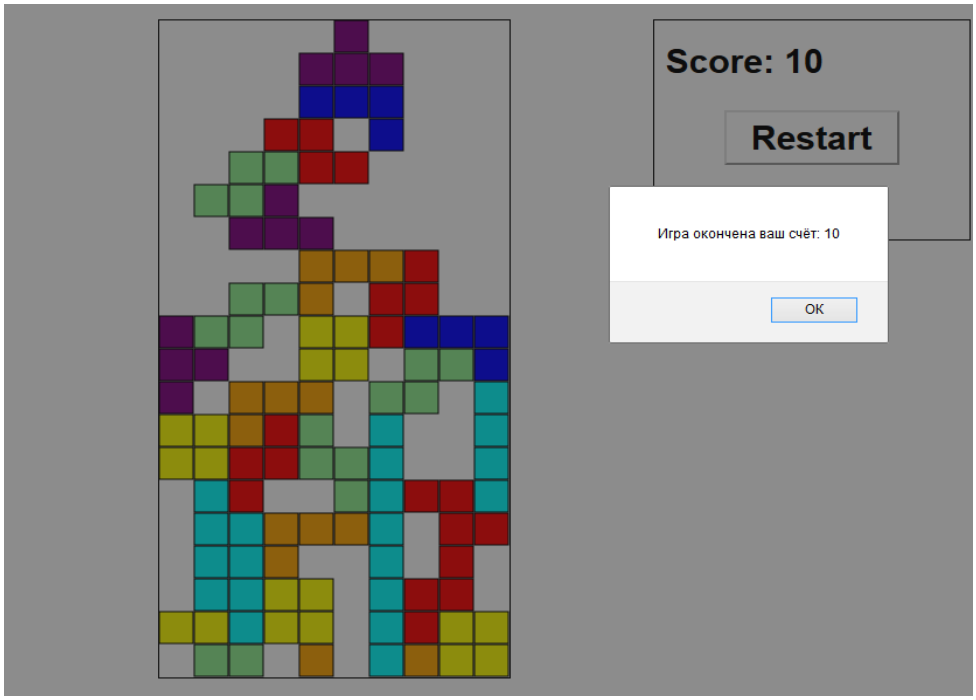

Рисунок 3.4 – Завершение игры

Важное значение имеет функция `render()` из файла game_render.js, которая совместно с функцией `tick()` вызывается циклически с определённым интервалом. Основной задачей данной функции является отображение игрового процесса на экране.

Каждый вызов функции `render()` сопровождается очисткой игрового поля. Данный эффект достигается с помощью получения контекста элемента

canvas и вызова его метода `clearRect(0, 0, W, H)`, очищающего заданную область на экране.

Для прорисовки активной фигуры и фигур, уже закрепленных на игровом поле, используются два цикла:

```
    //проририсовка поля
    for ( var x = 0; x < COLS; ++x ) {
            for ( var y = 0; y < ROWS; ++y ) {
                if ( field[ y ][ x ] ) {
                    ctx.fillStyle = colors[ field[ y ][
x ] - 1 ];
                    drawBlock( x, y );
                }
            }
        }

    //прорисовка активной фигуры
    for ( var y = 0; y < 4; ++y ) {
            for ( var x = 0; x < 4; ++x ) {
                if ( currentShape[ y ][ x ] ) {
                    ctx.fillStyle =
colors[currentShape[ y ][ x ] - 1 ];
                    drawBlock( currentX + x, currentY +
y );
                }
            }
        }
```

Данные циклы очень схожи, основная их задача заключается в том, чтобы проверить элементы массивов `field` и `currentShape` на наличие в них цифр, соответствующих блокам фигур. Если клетка массива не пуста, начинается ее прорисовка на экране с помощью функции `drawBlock()`:

```
    function drawBlock( x, y ) {
        ctx.fillRect( BLOCK_W * x, BLOCK_H * y, BLOCK_W
- 1 , BLOCK_H - 1 );
        ctx.strokeRect( BLOCK_W * x, BLOCK_H * y,
BLOCK_W - 1 , BLOCK_H - 1 );
    }
```

Эта функция получает в качестве параметров координаты фигуры в массиве и с помощью методов `fillRect()` и `strokeRect()` создает прямоугольник, который является составной частью фигуры. Размер этого прямоугольника зависит от экрана устройства.

ЗАКЛЮЧЕНИЕ

В настоящей работе была изучена история развития языка гипертекстовой разметки HTML, были описаны основные задачи и области применения современного стандарта HTML5, а также рассмотрены возможности HTML5 для работы с 2D- и 3D-графикой на веб-страницах. Графические технологии HTML5 были использованы для реализации браузерной версии известной игры «Тетрис» на языке JavaScript. Подобные игры (так называемые инди-игры), создаваемые в одиночку или небольшим коллективом без финансовой поддержки крупных корпораций, сейчас часто распространяются в социальных сетях, являясь полезным инструментом для привлечения пользователей.

Технологии HTML5 открывают новые возможности и перспективы в области веб-разработки, они будут оставаться актуальными и в будущем, так как позволяют использовать браузер в качестве универсальной платформы для графических, игровых и мультимедийных приложений, которые пишутся на языке JavaScript и работают на любых мобильных и десктопных устройствах.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1 Дронов В. А. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. А Дронов – СПб. : БХВ-Петербург, 2011. – 416 с.

2 Кудряшев А. В. Введение в современные веб-технологии / А. В Кудряшев, П. А Светашков – М.: ИНТУИТ.РУ, 2010. – 241с.

3 Иванов В. П. Трехмерная компьютерная графика / Под ред. Г. М Полищука – М.: Радио и связь, 1995. – 224с.

4 HTML 5 Динамичная графика [Электронный ресурс]:[сайт] – Ресурс о программировании. – [Б. м. : б. и.] – Режим доступа: http://html5insight.ru/post/9135528558/html5-canvas-and-svg/. – Загл. с экрана.

5 Виды компьютерной графики [Электронный ресурс]:[сайт] – Информационный портал. – [Б. м. : б. и.] – Режим доступа: https://poznayka.org/s60094t1.html. – Загл. с экрана.

6 Гультяев А.К. Проектирование и дизайн пользовательского интерфейса – С - Пб: Корона-Принт, 2010г. – 349с.

7 Графика в HTML5 [Электронный ресурс]:[сайт] – Ресурс о программировании. – [Б. м. : б. и.] – Режим доступа: http://html5ru.com/html5/grafika-v-html5. – Загл. с экрана.

8 HTML5 Canvas Анимация [Электронный ресурс]:[сайт] – Ресурс о программировании. – [Б. м. : б. и.] – Режим доступа: https://professorweb.ru/my/html/html5/level4/4_8.php. – Загл. с экрана

9 Учебник HTML5 графики [Электронный ресурс]:[сайт] – Ресурс о программировании. – [Б. м. : б. и.] – Режим доступа: https://msiter.ru/tutorials/svg. – Загл. с экрана.

10 Shankar A.R. Pro HTML5 Games – Apress, 2012г. – 357с

ПРИЛОЖЕНИЕ А (обязательное)

Листинг 1. (Создание 3D куба) **3DCube.html**

```
<!DOCTYPE>
<html>
<head>
<script src="babylon.js"></script>
</head>
<body>
<canvas id="testCanvas" width="500" height="500" >
</canvas>
</body>
</html>

<script>
var canva = document.getElementById("testCanvas",
true);
var babylonEngine = new BABYLON.Engine(canva);
var babylonScene = new BABYLON.Scene(babylonEngine);
babylonScene.clearColor = new
BABYLON.Color3(255,255,255);

var babylonCamera = new
BABYLON.ArcRotateCamera("camera", 0, Math.PI/2,10,new
BABYLON.Vector3(0,0,0),babylonScene);
babylonCamera.attachControl(canva);
var cube = BABYLON.Mesh.CreateBox("cube",3,
babylonScene);

var babylonLight = new
BABYLON.HemisphericLight("hLight", new
BABYLON.Vector3(0,1,2),babylonScene);


//var material = new BABYLON.StandardMaterial("mat",
babylonScene);
//cube.material = material;
//material.diffuseColor = BABYLON.Color3.Red();

babylonEngine.runRenderLoop(function(){
    babylonScene.render();
});
</script>
```

Продолжение ПРИЛОЖЕНИЯ А

Листинг 2 (Создание анимации) **animation.html**

```
<!DOCTYPE>
<html>
<head>
<script src="babylon.js"></script>
</head>
<body>
<canvas id="testCanvas" width="500" height="500"
style="border:1px solid black" >
</canvas>
</body>
</html>

<script>
var canva = document.getElementById("testCanvas");
var context = canva.getContext("2d");

var circlePos_x = 100;
var circlePos_y = 150;
var circleRadius = 25;
var i =0;

window.onload = function(){
    setTimeout("draw()", 20);
}

function draw(){
context.clearRect(0,0,canva.width, canva.height);
context.beginPath();

context.arc(circlePos_x, circlePos_y, circleRadius,0,
2*Math.PI);
context.fillStyle = "red";
context.fill();
context.lineWidth = 1;
context.stroke();

circlePos_x +=1;
circlePos_y +=1;
if(circlePos_x >=500 && circlePos_y >= 500)
    return;
console.log(i++);
setTimeout("draw()",20);
```

```
}
</script>
```

Содержание файла svg_graphic.html

```
<!DOCTYPE>
<html>
<body>
<svg width="500" height="200" >
<circle cx="75" cy="75" r="50" stroke="black" stroke-
width="3" fill="yellow" />
<rect x="175" y="25" width="100" height="100"
stroke="black" stroke-width="2" fill="blue"/>
<polygon points="320,125 370,25 420,125" stroke="black"
stroke-width="1" fill="#00ff00"/>
</svg>
</body>
</html>
```

Содержание файла canvas_graphic.html

```
<!DOCTYPE>
<html>
<body>
<canvas id="testCanvas"width="200" height="200" >
</canvas>
</body>
</html>

<script>
var canvas = document.getElementById("testCanvas");
var ctx = canvas.getContext("2d");

ctx.fillStyle = "#ff0000";
ctx.fillRect(20, 20, 100, 100);
</script>
```

ПРИЛОЖЕНИЕ Б (обязательное)

Содержание файла index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML5 TETRIS</title>
        <link rel='stylesheet' href='style.css' />
    </head>
    <body>

        <canvas id="canvas" width='300'
height='600'></canvas>
        <div class="panel">
         <span>Score: <span
id="score">0</span></span><br><br>
         <button id="playbutton"
onclick="newGame();">Play</button>
        </div>

        <script src='game_main.js'></script>
        <script src='game_controller.js'></script>
        <script src='game_render.js'></script>
    </body>
</html>
```

Содержание файла style.css

```
canvas {
    margin-top: 10px;
    margin-left: 10%;
    border: 1px solid black;
}

#playbutton {
    width: 150px;
    height: 50px;
    display: block;
    margin: auto;
    margin-top: 10px;
    font-family: Arial, sans-serif;
    font-size: 30px;
    font-weight: bold;
    background-color: white;
```

```
    cursor: pointer;
}



.panel span{
    display: inline-block;
    font-family: Arial, sans-serif;
    font-size: 30px;
    font-weight: bold;
    margin-top: 10px;
}


.panel{
    width: 250px;
    height: 200px;
    border: 1px solid black;
    float: right;
    margin-right: 35%;
    margin-top: 10px;
    padding-left: 10px;
    padding-right: 10px;
}
```

Содержание файла game_main.js

```
 var COLS = 10, ROWS = 20;   //строки и столбцы
var field = [];     //игровое поле
var gameOver;
var interval;       //ссылки на интервал
var intervalRender;
var currentShape; // активная фигура
var currentX, currentY; // позиция активной фигуры
var freezed; // заморожена ли активная фигура
var score;  //счет

//массив всевозможных фигур
var figures = [
    [                           //I
        [1,0,0,0],
        [1,0,0,0],
        [1,0,0,0],
        [1,0,0,0]
    ],
```

Продолжение ПРИЛОЖЕНИЯ Б

```
    [                                   //L
        [2,2,2,0],
        [2,0,0,0],
        [0,0,0,0],
        [0,0,0,0]
    ],
    [                                   //J
        [3,3,3,0],
        [0,0,3,0],
        [0,0,0,0],
        [0,0,0,0]
    ],
    [                                   //O
        [4,4,0,0],
        [4,4,0,0],
        [0,0,0,0],
        [0,0,0,0]
    ],
    [                                   //S
        [5,5,0,0],
        [0,5,5,0],
        [0,0,0,0],
        [0,0,0,0]
    ],
    [                                   //Z
        [0,6,6,0],
        [6,6,0,0],
        [0,0,0,0],
        [0,0,0,0]
    ],
    [                                   //T
        [0,7,0,0],
        [7,7,7,0],
        [0,0,0,0],
        [0,0,0,0]
    ],
];
//массив цветов
var colors = [
    'cyan', 'orange', 'blue', 'yellow', 'red',
'lightgreen', 'purple'
];
```

Продолжение ПРИЛОЖЕНИЯ Б

```
// Функция, создающая новую фигуру и помещающая ее в
глобальную переменную-массив current размером 4x4
// 4x4 для того, чтобы можно было свободно вращать
function createNewShape() {
    var id = Math.floor( Math.random() * 7 );
    currentShape = figures[id];

    // после создания отменяем заморозку фигуры
    freezed = false;
    // начальная позиция фигуры
    currentX = 4;
    currentY = 0;
}

// очистка поля и заполнение его нулями
function initialization() {
    for ( var y = 0; y < ROWS; ++y ) {
        field[ y ] = [];
        for ( var x = 0; x < COLS; ++x ) {
            field[ y ][ x ] = 0;
        }
    }
}

// падение фигур, создание новых при необходимости,
очистка заполненных строк
function tick() {
    if ( checkCollisions( 0, 1 ) ) {
        ++currentY;
    }
    // если произошло столкновение
    else {
        freeze();
        clearLines();
        checkCollisions(0, 1);

        if (gameOver) {
            clearAllIntervals();
            return false;
        }
        createNewShape();
    }
}
```

Продолжение ПРИЛОЖЕНИЯ Б

```
// получение позиции фигуры и фиксирование её на поле
(заморозка)
function freeze() {
    for ( var y = 0; y < 4; ++y ) {
        for ( var x = 0; x < 4; ++x ) {
            if ( currentShape[ y ][ x ] ) {
                field[ y + currentY ][ x + currentX ] =
currentShape[ y ][ x ];
            }
        }
    }
    freezed = true;
}

// получение фигуры, развернутой на 90 градусов
function rotate( current ) {
    var newCurrent = [];
    for ( var y = 0; y < 4; ++y ) {
        newCurrent[ y ] = [];
        for ( var x = 0; x < 4; ++x ) {
            newCurrent[ y ][ x ] = currentShape[ 3 - x
][ y ];   //поворт фигуры в массиве 4x4
        }
    }

    return newCurrent;
}

// проверка строк на заполненность и их очистка
function clearLines() {
    //проверка строк на заполненность
    for ( var y = ROWS - 1; y >= 0; --y ) {
        var rowFilled = true;
        for ( var x = 0; x < COLS; ++x ) {
            if ( field[ y ][ x ] == 0 ) {
                rowFilled = false;
                break;
            }
        }
        //очистка строки и увеличение очков
        if ( rowFilled ) {
            score+=10;
            document.getElementById("score").innerHTML
= score;
```

```
            for ( var yy = y; yy > 0; --yy ) {
                for ( var x = 0; x < COLS; ++x ) {
                    field[ yy ][ x ] = field[ yy - 1 ][
x ];
                }
            }
            ++y;
        }
    }
}

// обработка нажатия клавиши
function keyPress( key ) {
    switch ( key ) {
        case 'left':
            if ( checkCollisions( -1 ) ) {
                //смещение фигуры влево
                --currentX;
            }
            break;
        case 'right':
            if ( checkCollisions( 1 ) ) {
                //смещение фигуры вправо
                ++currentX;
            }
            break;
        case 'down':
            if ( checkCollisions( 0, 1 ) ) {
                //смещение фигуры вниз (ускорение)
                ++currentY;
            }
            break;
        case 'rotate':
            //поворот фигуры
            var rotated = rotate( currentShape );
            if ( checkCollisions( 0, 0, rotated ) ) {
                currentShape = rotated;
            }
            break;
        case 'drop':
            //мгновение падение фигуры
            while( checkCollisions(0, 1) ) {
                ++currentY;
            }
```

```
            tick();
            break;
    }
}

// проверка столкновений на заданных координатах
function checkCollisions( offsetX, offsetY, newCurrent
) {
    // инициализация параметров
    offsetX = offsetX || 0;
    offsetY = offsetY || 0;
    offsetX = currentX + offsetX;
    offsetY = currentY + offsetY;
    newCurrent = newCurrent || currentShape;

    //проверка на выходы за границы поля и наличия
других фигур по заданным координатам
    for ( var y = 0; y < 4; ++y ) {
        for ( var x = 0; x < 4; ++x ) {
            if ( newCurrent[ y ][ x ] ) {
                if ( typeof field[ y + offsetY ] ==
'undefined'
                  || typeof field[ y + offsetY ][ x +
offsetX ] == 'undefined'
                  || field[ y + offsetY ][ x + offsetX
] ) {

                    if (offsetY == 1 && freezed) {
                        gameOver = true; // завершение
игры в случае если верхняя ячейка поля занята и для
новой фигуры нет места

                        alert("Игра окончена ваш счёт:
"+score);

document.getElementById('playbutton').disabled = false;
                    }
                    return false;
                }
            }
        }
    }
    return true;
}
```

Продолжение ПРИЛОЖЕНИЯ Б

```
// создание новой игры
function newGame() {
    document.getElementById("playbutton").innerHTML =
'Restart';

    initialization();
    createNewShape();
    gameOver = false;
    score = 0;
    //очистка старых интервалов и установка новых
    clearAllIntervals();
    intervalRender = setInterval( render, 40 );
    interval = setInterval( tick, 500 );

}
//очистка ссылок на  интервалы
function clearAllIntervals(){
    clearInterval( interval );
    clearInterval( intervalRender );
}
```

Содержание файла game_contoller.js

```
// обработчик нажатия клавиш
window.onkeydown = function( e ) {
    var keys = {
        65: 'left',
        68: 'right',
        83: 'down',
        87: 'rotate',
        69: 'drop'
    };
    if ( typeof keys[ e.keyCode ] != 'undefined' ) {
        keyPress( keys[ e.keyCode ] );
        render();
    }
};
```

Содержание файла game_render.js

```
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext( '2d' );
var W = 300, H = 600;
var BLOCK_W = W / COLS, BLOCK_H = H / ROWS;

// отрисовка отдельного фрагмента(квадрата) на задданой
позиции
```

Продолжение ПРИЛОЖЕНИЯ Б

```
function drawBlock( x, y ) {
    ctx.fillRect( BLOCK_W * x, BLOCK_H * y, BLOCK_W - 1
, BLOCK_H - 1 );
    ctx.strokeRect( BLOCK_W * x, BLOCK_H * y, BLOCK_W -
1 , BLOCK_H - 1 );
}

// отрисовка поля и фигур
function render() {
    //очистка экрана
    ctx.clearRect( 0, 0, W, H );

    ctx.strokeStyle = 'black';
    //отрисовка закрепленных фигур на игровом поле
    for ( var x = 0; x < COLS; ++x ) {
        for ( var y = 0; y < ROWS; ++y ) {
            if ( field[ y ][ x ] ) {
                ctx.fillStyle = colors[ field[ y ][ x ]
- 1 ];
                drawBlock( x, y );
            }
        }
    }

    ctx.fillStyle = 'red';
    ctx.strokeStyle = 'black';
    //отрисовка активной фигуры
    for ( var y = 0; y < 4; ++y ) {
        for ( var x = 0; x < 4; ++x ) {
            if ( currentShape[ y ][ x ] ) {
                ctx.fillStyle = colors[ currentShape[ y
][ x ] - 1 ];
                drawBlock( currentX + x, currentY + y
);
            }
        }
    }
}
```

Отзыв на бакалаврскую работу
«Применение технологий HTML 5 для разработки компьютерных игр»
студента 4 курса факультета математики и информационных технологий
А.А. Пархачева

В данной бакалаврской работе была поставлена задача изучения технологий современного стандарта HTML5, позволяющих с помощью языка JavaScript разрабатывать графические приложения (в частности, компьютерные игры), работающие в веб-браузерах на любых современных устройствах.

В качестве демонстрации графических возможностей HTML5 автором работы была реализована браузерная версия известной игры «Тетрис».

В целом бакалаврская работа соответствует заявленной теме и раскрывает исследуемые технологии. В ходе выполнения бакалаврской работы А.А. Пархачев показал себя грамотным специалистом, способным решать поставленные задачи.

Считаю, что бакалаврская работа А.А. Пархачева заслуживает оценки "отлично".

Руководитель дипломной работы
к.ф. – м.н., доцент кафедры
фундаментальной информатики А.В. Попов

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.П. ОГАРЁВА"

ОТЗЫВ РЕЦЕНЗЕНТА
О ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Студента Пархачева Андрея Александровича
Факультет математики и информационных технологий
Кафедра фундаментальной информатики и информационных технологий
Группа 402
Направление подготовки Фундаментальная информатика и информационные технологии
Квалификация (степень) бакалавр
Наименование темы: Применение технологий HTML5 для разработки компьютерных игр
Рецензент Сухарев Л. А., МГУ им. Н.П. Огарева, кандидат физико-математических наук, доцент

ОЦЕНКА ВЫПУСКНОЙ РАБОТЫ

№ п/п	Показатели		Оценка				
			5	4	3	2	0*
1.	Актуальность тематики работы		+				
2.	Степень полноты обзора состояния вопроса и корректность постановки задачи			+			
3.	Уровень и корректность использования в работе методов исследования, математического моделирования			+			
4.	Степень комплексности работы, применение в ней знаний естественнонаучных, социально-экономических, общепрофессиональных и специальных дисциплин		+				
5.	Ясность, четкость, последовательность и обоснованность изложения		+				
6.	Применение современного математического и программного обеспечения, компьютерных технологий в работе		+				
7.	Качество оформления (общий уровень грамотности, стиль изложения, качество иллюстраций, соответствие требованиям стандарта)		+				
8.	Оригинальность и новизна полученных результатов (научных, конструкторских и технологических решений)		+				
9.	Тип работы	фундаментальная с оригинальными результатами					
		реферативная					
		прикладная	+				
10.	Рекомендации	к опубликованию					
		к внедрению	+				
ИТОГОВАЯ ОЦЕНКА			*отлично*				

* – не оценивается (трудно оценить)

Отмеченные достоинства:

Работа изложена ясно и четко, прослеживается логическая связь между разделами. Процесс проектирования и реализации веб-приложения браузерной игры описан довольно подробно, что делает работу доступной для понимания.

Работа имеет практическое применение – методы, использованные автором при реализации игры «Тетрис», могут быть применены при разработке других браузерных HTML5-приложений.

Отмеченные недостатки:

Недостаточно полно описаны мультимедийные возможности технологий HTML5, не приведены практические примеры их использования.

Заключение:

Указанный недостаток не умаляет достоинств работы, которая удовлетворяет требованиям к выпускным квалификационным работам. Считаю, что выполненная работа может быть допущена к защите и заслуживает оценки «отлично», а ее автор, Пархачев Андрей Александрович, заслуживает присвоения степени бакалавра по направлению подготовки «Фундаментальная информатика и информационные технологии».

10 июня 2019 г. Рецензент _____
(подпись)

Заявление о самостоятельном характере выполнения работы

Я, Пархачев Андрей Александрович, обучающийся 4 курса направления подготовки 02.03.02 – Фундаментальная информатика и информационные технологии, заявляю, что в моей работе на тему Применение технологий HTML5 для разработки компьютерных игр, представленной в Государственную экзаменационную комиссию для публичной защиты, не содержится элементов неправомерных заимствований.

Все прямые заимствования из печатных и электронных источников, а также ранее защищённых письменных работ, кандидатских и докторских диссертаций имеют соответствующие ссылки.

Я ознакомлен с действующим в Университете Положением о проверке работ обучающихся ФГБОУ ВО «МГУ им. Н.П. Огарёва» на наличие заимствований, в соответствии с которым обнаружение неправомерных заимствований является основанием для отрицательного отзыва руководителя работы.

04.06.2019

Подпись обучающегося | Дата

Работа представлена для проверки в Системе «Антиплагиат.ВУЗ»

04.06.2019

Дата представления работы | *подпись руководителя*

ОТЧЕТ
о результатах проверки работы обучающегося
на наличие заимствований

Ф.И.О. автора работы Пархачев Андрей Александрович

Тема работы Применение технологий HTML5 для разработки компьютерных игр

Руководитель канд. физ.-мат. наук Попов А. В.

Представленная работа прошла проверку на наличие заимствований в системе «Антиплагиат.ВУЗ»

Результаты автоматической проверки: оригинальность – 90,13 %
цитирования – 0,62%
заимствования – 9,25 %

Результаты анализа полного отчета на наличие заимствований:
правомерные заимствования: да, 9,25%

корректные цитирования: да, 0,62%

неправомерные заимствования: нет

признаки обхода системы: нет

Общее заключение об итоговой оригинальности работы и возможности ее допуска к защите: Итоговая оригинальность работы составляет 90,13%, Пархачев А.А. допускается к защите выпускной квалификационной работы (в форме бакалаврской работы)

Руководитель работы
канд. физ.-мат. наук 04.06.2019 А. В. Попов
подпись, дата

Заведующему кафедрой
фундаментальной информатики
А. Г. Смольянову
студента 4 курса
очной формы обучения
на бесплатной основе
направления подготовки
02.03.02 – Фундаментальная
информатика и информационные
технологии факультета математики
и информационных технологий
ФГБОУ ВО «МГУ им. Н.П.Огарёва»
Пархачева Андрея Александровича

заявление.

Прошу разместить мою выпускную квалификационную работу на тему «Применение технологий HTML5 для разработки компьютерных игр» в электронной библиотечной системе университета в полном объеме.

21.06.2019
Дата

Подпись