**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

Курсовая РАБОТА

**по дисциплине «Web-технологии»**

Тема: Разработка игры на языке JavaScript

|  |  |  |
| --- | --- | --- |
| Студент гр. 0303 |  | Калмак Д.А. |
| Преподаватель |  | Беляев С.А. |

Санкт-Петербург

2022

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Калмак Д.А. | | |
| Группа 0303 | | |
| Тема работы: Разработка игры на языке JavaScript | | |
| Исходные данные:  В соответствии с учебным пособием на «чистом» JavaScript (ES6):  Минимум 2 уровня игры, реализованы все менеджеры в соответствии с учебным пособием (УП), есть таблица рекордов, есть препятствия, есть «интеллектуальные» противники и «бонусы», используются tiles с редактором Tiled в соответствии с УП. | | |
| Содержание пояснительной записки:  «Содержание», «Введение», «Форма входа», «Map manager», «Sprite manager», «Events manager», «Physic manager», «Game manager», «Sound manager», «Классы», «Рекорды», «Заключение», «Список использованных источников» | | |
| Предполагаемый объем пояснительной записки:  Не менее 43 страниц. | | |
| Дата выдачи задания: 01.09.2022 | | |
| Дата сдачи реферата: 02.12.2022 | | |
| Дата защиты реферата: 02.12.2022 | | |
| Студент |  | Калмак Д.А. |
| Преподаватель |  | Беляев С.А. |

**Аннотация**

Разработана игра на языке JavaScript. Игрок должен пройти все уровни, уничтожив всех врагов на каждом уровне. Во время прохождения начисляются очки, которые пойдут в рейтинговую таблицу. Для создания карты для игры использовался редактор Tiled. Для работы игры реализованы менеджеры. Пользователь может играть в игру в браузерах.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 6 |
| 1. | Форма входа | 7 |
| 1.1. | Реализация формы входа | 7 |
| 2. | Map manager | 8 |
| 2.1. | Поля | 8 |
| 2.2. | Методы | 8 |
| 3. | Sprite manager | 10 |
| 3.1. | Поля | 10 |
| 3.2. | Методы | 10 |
| 4. | Events manager | 11 |
| 4.1. | Поля | 11 |
| 4.2. | Методы | 11 |
| 5. | Physic manager | 12 |
| 5.1. | Методы | 12 |
| 6. | Game manager | 13 |
| 6.1. | Поля | 13 |
| 6.2 | Методы | 13 |
| 7. | Sound manager | 16 |
| 7.1. | Поля | 16 |
| 7.2. | Методы | 16 |
| 8. | Классы | 17 |
| 8.1. | Entity | 17 |
| 8.2. | Player | 17 |
| 8.3. | Boxer | 18 |
| 8.4. | Rocket | 18 |
| 8.5. | Bonus | 19 |
| 9. | Рекорды | 20 |
| 9.1. | Реализация рекордов | 20 |
|  | Заключение | 21 |
|  | Список использованных источников | 22 |
|  | Приложение А. Название приложения | 23 |

**введение**

Цель работы.

Разработать игру на языке JavaScript.

Задачи.

* Создать минимум 2 уровня игры.
* Реализовать все менеджеры в соответствии с учебным пособием (УП). Создать таблица рекордов.
* На карте разместить препятствия.
* Создать «интеллектуальных» противников и «бонусы».
* Использовать tiles с редактором Tiled в соответствии с УП.

**1. ФОРМА ВХОДА**

**1.1. Реализация формы входа**

Создана форма входа. С помощью <form action="main5.html" method="get"> после ввода имени пользователя в <input placeholder="Имя пользователя"> и нажатия <input type="submit"> происходит переход на основную форму, при этом сохраняется имя пользователя. С помощью функции store() имя помещается в локальное хранилище. (см. рис. 1)

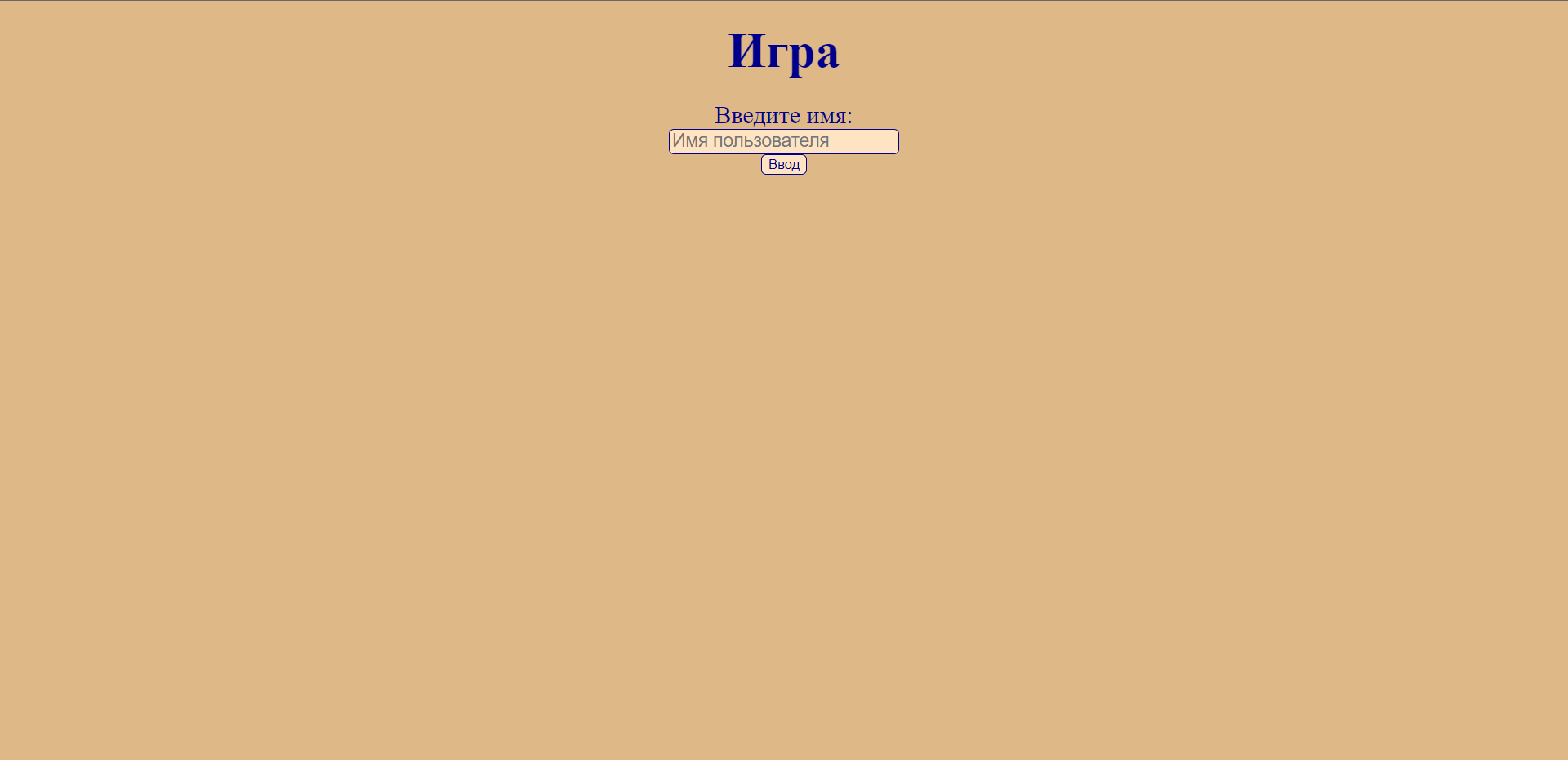
****

Рисунок 1 – Форма входа

**2. Map MANAGER**

**2.1. Поля**

mapData: null - переменная для хранения карты

tLayer: null - переменная для хранения ссылки на блоки карты

tSet: null – переменная для хранения тайлсета

xCount: 0 - количество блоков по горизонтали

yCount: 0 - количество блоков по вертикали

tSize: {x: 32, y: 32} - размер блока

mapSize: {x: 22, y: 22} - размер карты в пикселях (вычисляется)

tilesets: new Array() - массив описаний блоков карты

pict: [] – массив тайлов

imgLoadCount: 0 – количество загруженных изображений

imgLoaded: false – флаг, что все изображения загружены

jsonLoaded: false – флаг, что карта загружена

view: {x: 0, y: 0, w: 704, h: 704} – переменные для видимой области

**2.2. Методы**

parseMap(tilesJSON) – парсинг карты tilesJSON. Получает количество блоков по горизонтали и вертикали, размер блоков, высчитывает размер карты. С помощью асинхронного запроса также загружаем тайлы. Информация о них записывается в массив pict. Когда все изображения загружены, флаг imgLoaded переводится в true. После разбора карты jsonLoaded переводится в true.

loadMap(path) – асинхронная загрузка карты по путю path. При успешной загрузке переход к парсингу.

draw(ctx) – отрисовка карты в canvas. Отрисовка происходит, когда все флаги переведены в true. В карте ищется слой tilelayer и с него отрисовываются блоки карты с помощью метода canvas drawImage.

getTile(tileIndex) – получение блока по индексу tileIndex из тайлсета. Получаем тайлсет по индексу tileIndex. Находим нужный тайл по id равному tileIndex – tileset.firstgid.

getTileset(tileIndex) – получение тайлсета по индексу tileIndex. Проходим циклом с индексацией i по тайлсетам по убыванию. Если tileIndex больше или равен начального номера блоков в tilesets[i], то именно этот набор блоков нужен.

isVisible(x, y, width, height) – проверка видимой зоны. Возвращает false, если вне видимой зоны, и true, если в видимой зоне.

parseEntities() – парсинг сущностей. Парсинг происходит, когда все флаги переведены в true. В карте ищется слой objectgroup и с него собирается информация о сущностях. Создается объект фабрики gameManager класса сущности. Получает информацию об имени сущности, ее координатах, размерах. Созданный объект помещается в массив entities gameManager. Если сущность класса Boxer, то переменная num\_enemies в gameManager увеличивается на единицу. Если сущность класса игрока, то происходит инициализация игрока в gameManager.

getTilesetIdx(x, y) – получение индекса блока в массиве data, используя размеры блоков и количество блоков по горизонтали.

centerAt(x, y) – центрирование области view относительно положения игрока (x, y). Центрируется по горизонтали, затем центрируется по вертикали.

**3. SPRITE MANAGER**

**3.1. Поля**

sprites: new Array() – массив объектов для отображения

imgLoadCount: 0 – количество загруженных изображений

imgLoaded: false – флаг, что все изображения загружены

jsonLoaded: false – флаг, что json спрайтов загружен

**3.2. Методы**

loadAtlas(atlasJson) - асинхронная загрузка файла со спрайтами по путю path. При успешной загрузке переход к парсингу.

parseAtlas(atlasJSON) – парсинг файла со спрайтами. С помощью асинхронного запроса также загружаем спрайты. Информация о них записывается в массив sprites. Информация содержит имя спрайта, путь к нему, размеры. Когда все изображения загружены, флаг imgLoaded переводится в true. После разбора карты jsonLoaded переводится в true.

drawSprite(ctx, name, x, y) – отрисовка сущности с именем name в canvas по координатам x, y. Используется метод canvas drawImage.

getSprite(name) – поиск спрайта с именем name в массиве sprites методом find.

**4. EVENTS MANAGER**

**4.1. Поля**

bind: [] – массив для сопоставления клавиш действиям

action: [] – массив для действий

**4.2. Методы**

setup(canvas) – настройка сопоставления. w - двигаться вверх, a - двигаться влево, s - двигаться вниз, d - двигаться вправо, пробел – выстрелить. Добавлены контроли нажатия мыши, нажатия клавиш на клавиатуре с помощью метода addEventListener.

onMouseDown(event) – обработка нажатия клавиши мыши. Происходит выстрел, значение в массиве переводится в true.

onMouseUp(event) - обработка отпускания клавиши мыши. Значение в массиве переводится в false.

onKeyDown(event) - обработка нажатия клавиши клавиатуры. Значение в массиве по индексу действия переводится в true.

onKeyUp(event) - обработка отпускания клавиши клавиатуры. Значение в массиве по индексу действия переводится в false.

**5. PHYSIC MANAGER**

**5.1. Методы**

update(obj) – обновление состояния объекта. Если объект не движется, то возвращается “stop”. Высчитываются новые координаты объекта по направлению движения объекта и его скорости. Получается индекс блока по новым координатам, скорректированным для правильного отображения на карте. Получается возможная сущность по новым координатам, также скорректированным. Если есть по новым координатам сущность и у объекта есть метод взаимодействия с сущностями, то вызывается метод у объекта. Если по новым координатам есть препятствие и у объекта есть метод взаимодействия с картой, то вызывается метод у объекта. Если нет препятствий и нет сущностей, то объекту присваиваются новые координаты. Возвращается “move”, иначе возвращается “break”.

entityAtXY(obj, x, y) – проверка столкновения объекта с сущностями. Проходит циклом массив сущностей entities gameManager. Анализируются объекты, если имена объекта и сущности не совпадают, то происходит проверка столкновения. Если объект и сущность не столкнулись, то с помощью continue осуществляется переход к очередной итерации цикла for для анализа следующего объекта. Если обнаружено столкновение, то возвращается сущность, иначе возвращается null.

**6. GAME MANAGER**

**6.1. Поля**

factory: {} – фабрика объектов на карте

entities: [] – массив объектов на карте

fireNum: 0 – идентификатор выстрела

player: null – указатель на объект игрока

laterKill: [] – массив отложенного уничтожения объектов

num\_enemies: 0 – количество врагов

num: 0 – уровень

score: 0 – очки

name: null – имя игрока

game\_over: false – флаг конца игры

**6.2. Методы**

initPlayer(obj) – инициализация игрока. В поле player записывается obj.

kill(obj) – убийство объекта. Добавление obj в массив отложенного удаления laterKill.

update() – обновление игрового процесса и игровой информации. Если игрока нет, то update() не производится. По массиву действий action eventsManager определяются действия игрока. Для каждого объекта на карте в массиве entities вызывается метод update(), если он есть. Обрабатывается массив отложенного уничтожения объектов, и по индексу удаляются объекты из массива entities. Длина массива отложенного уничтожения объектов laterKill обнуляется. Отрисовывается карта. Отрисовывается информация об игроке, его здоровье, его очках. Если игрок прошел первый уровень, проигрывается музыка перехода на следующий уровень, а также появляется надпись, сообщающая это. Загружается второй уровень. Инициализация игрока сбрасывается, массив объектов entities опустошается, уровень меняется. Если игрок прошел второй уровень, проигрывается победная музыка, а также появляется надпись, сообщающая это. Инициализация игрока сбрасывается, массив объектов entities опустошается, сохраняется рекорд, производится переход на страницу рекордов.

show\_info() – отрисовка информации. Отрисовывает очки, имя игрока, количество здоровья с помощью метода canvas fillText.

save\_record(name, score) – сохранение рекорда игрока. Сохраняет информацию о рекордах игрока с именем name, набравшим score очков. Если игрок сыграл первый раз, то о нем записывается информация в локальное хранилище: имя игрока и его очки, - иначе записывается информация лучшей попытки игрока.

draw(ctx) – отрисовка объектов на карте. Циклом вызывается метод draw для каждого объекта в массиве entities.

loadAll(namem) – инициализация менеджеров для работы игры. Обнуляется счетчик загруженных изображений mapManager.imgLoadCount, флаги загруженных изображений и файла карты переводятся в false. Загружается карта, загружаются спрайты. Инициализируется фабрика factory gameManager с такими классами, как Player, Boxer, Bonus, Rocket. Парсятся объекты на карте. Отрисовывается карта. Настраивается менеджер событий eventsManager. Получается имя игрока.

play() – запуск обновления игрового мира раз в 100 мс.

updateWorld() – обновление игрового мира. Вызывается метод update gameManager.

Игровой процесс представлен на рис. 2.

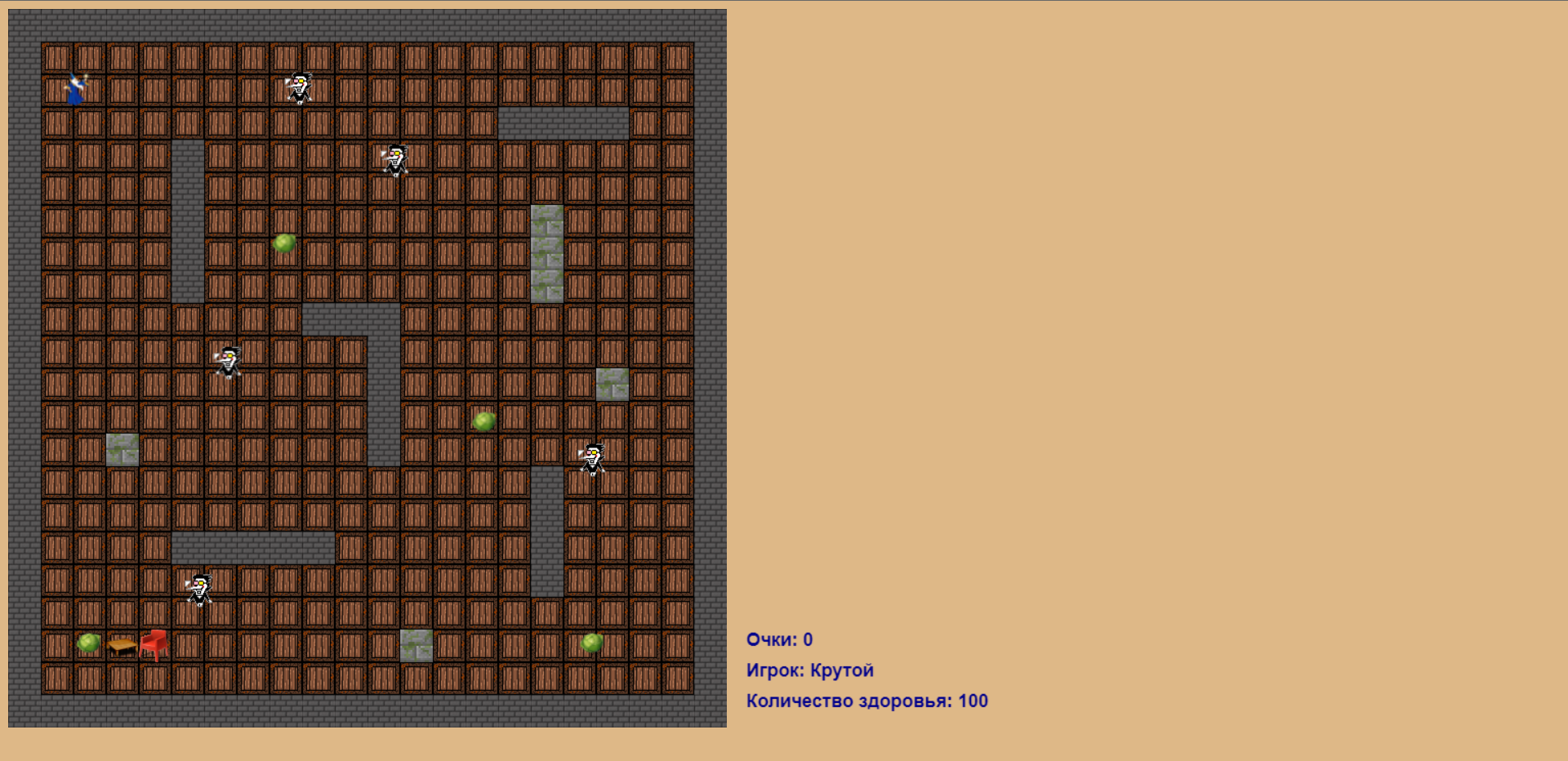


Рисунок 2 – Игровой процесс

**7. SOUND MANAGER**

**7.1. Поля**

clips: {} – хранение звуковых эффектов

context: null - аудиоконтекст

gainNode: null – главный узел

loaded: false – флаг загрузки всех звуков

**7.2. Методы**

init() – инициализация менеджера звука.

load(path, callback) – загрузка одного аудиофайла. Асинхронно загружается звук.

loadArray(array) – загрузка массива звуков. Асинхронно загружается каждый звук из массива array. Когда все звуки загружены, флаг loaded переводится в true

play(path, settings) – проигрывание файла. Проигрывается звук по пути path с настройками settings, которые могут включать громкость volume и зацикливание looping. По умолчанию громкость volume 1, а зацикливание looping false.

playWorldSound(path, x, y) – проигрывание файла с учетом координат игрока. Проигрывает звук на расстоянии не дальше 80% от размера карты, а также уменьшается громкость в зависимости от расстояния от игрока.

toggleMute() – приостановить проигрывание звуков и возобновить проигрывание звуков. gainNode.value присваивается 0 и 1 соответственно.

stopAll() – отключить проигрывание звуков. Отключение gainNode от колонок и пересоздание gainNode.

**8. Классы**

**8.1. Entity**

Класс сущности. Содержит поля:

pos\_x – координата положения x. По умолчанию 0.

pos\_y – координата положения y. По умолчанию 0.

size\_x – размер по x. По умолчанию 0.

size\_y – размер по y. По умолчанию 0.

**8.2. Player**

Класс игрока. Расширяет класс сущности. Содержит дополнительные поля:

lifetime – количество здоровья. По умолчанию 100.

move\_x – сторона движения по x. По умолчанию 0.

move\_y – сторона движения по y. По умолчанию 0.

speed – скорость движения. По умолчанию 10.

Содержит методы:

draw (ctx) – отрисовка с помощью метода drawSprite spriteManager.

update () – обновление с помощью метода update physicManager.

onTouchEntity (obj) – обработка столкновения с сущностью. Если это бонус, то к здоровью прибавляется 50. Бонус уничтожается.

kill () – обрабатывается убийство игрока. Проигрывается звук. Флаг конца игры переводится в true.

fire () – обрабатывается выстрел игрока. Создается ракета класса Rocket с помощью фабрики factory gameManager. Координаты появления ракеты рассчитываются с помощью текущего направления движения игрока. Выстрел сопровождается звуком. Объект ракеты добавляется в массив entities gameManager.

**8.3. Boxer**

Класс боксера, врага. Расширяет класс сущности. Содержит дополнительные поля:

lifetime – количество здоровья. По умолчанию 100.

move\_x – сторона движения по x. По умолчанию 0.

move\_y – сторона движения по y. По умолчанию 0.

speed – скорость движения. По умолчанию 5.

Содержит методы:

draw (ctx) – отрисовка с помощью метода drawSprite spriteManager.

update () – обновление с помощью метода update physicManager. Движется к игроку.

onTouchEntity (obj) – обработка столкновения с сущностью. Если это игрок, то от здоровья игрока отнимается 10. Если количество здоровья игрока на нуле, то игрок уничтожается. Сопровождается звуком.

kill () – обрабатывается убийство боксера. Проигрывается звук. Количество врагов num\_enemies gameManager на карте уменьшается на 1. Игроку начисляется 100 очков.

fire () – обрабатывается выстрел игрока. Создается ракета класса Rocket с помощью фабрики factory gameManager. Координаты появления ракеты рассчитываются с помощью текущего направления движения игрока. Выстрел сопровождается звуком. Объект ракеты добавляется в массив entities gameManager.

**8.4. Rocket**

Класс ракеты, снаряды игрока. Расширяет класс сущности. Содержит дополнительные поля:

move\_x – сторона движения по x. По умолчанию 0.

move\_y – сторона движения по y. По умолчанию 0.

speed – скорость движения. По умолчанию 10.

Содержит методы:

draw (ctx) – отрисовка с помощью метода drawSprite spriteManager.

update () – обновление с помощью метода update physicManager.

onTouchEntity (obj) – обработка столкновения с сущностью. Если это враг, игрок или ракета, то сущность уничтожается. Ракета уничтожается.

onTouchMap (idx) – обрабатывается столкновение с препятствием. Ракета уничтожается. Сопровождается звуком.

kill () – обрабатывается уничтожение ракеты.

**8.5. Bonus**

Класс бонуса для игрока. Расширяет класс сущности.

Содержит методы:

draw (ctx) – отрисовка с помощью метода drawSprite spriteManager.

kill () – обрабатывается уничтожение бонуса. Проигрывается звук. Количество очков игрока уменьшается на 50.

**9. РЕКОРДЫ**

**9.1. Реализация рекордов**

Из локального хранилища достаются пары имя и очки и сохраняются в словарь. Словарь сортируется по убыванию очков. Содержание словаря отображается на странице в виде “Имя \* очков”. Рекорды представлены на рис. 3.

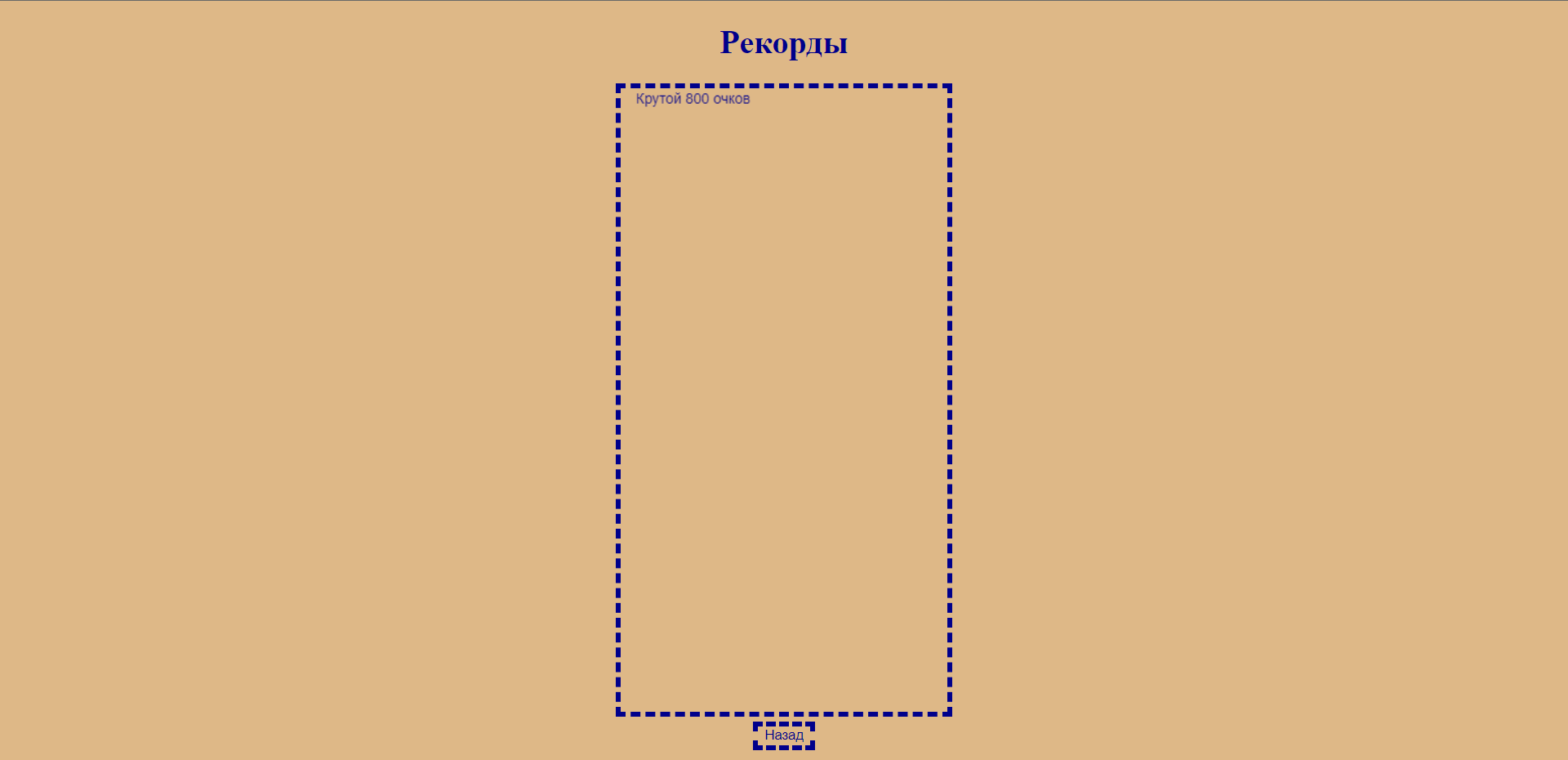


Рисунок 3 - Рекорды

**заключение**

Таким образом, разработана игра на языке JavaScript. Созданы 2 уровня игры. Реализованы все менеджеры в соответствии с учебным пособием (УП). Создана таблица рекордов. На карте размещены препятствия. Созданы «интеллектуальных» противники и «бонусы». Использован tiles с редактором Tiled в соответствии с УП.

**список использованных источников**

1. С. А. Беляев РАЗРАБОТКА ИГР НА ЯЗЫКЕ JAVASCRIPT Лань: Издательство, 2022. 150 с.

**приложение А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

**login.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Ввод пользователя</title>

<style>

body{

background-color: burlywood;

}

h1{

text-align:center;

color: darkblue;

}

form{

text-align:center;

}

input[type=submit] {

color: darkblue;

background-color: bisque;

border: thin solid darkblue;

border-radius: 5px;

}

.label input {

color: darkblue;

background-color: bisque;

border: thin solid darkblue;

border-radius: 5px;

font-size: 18px;

}

</style>

</head>

<body>

<h1><font size="7" face="Times New Roman">Игра</font></h1>

</body>

<form action="main5.html" method="get">

<label class="label">

<font size=5 face="Times New Roman" color="#00008b">Введите имя:</font><br>

<input placeholder="Имя пользователя"><br>

</label>

<input type="submit" value="Ввод" onclick="store(document.querySelector('input'))">

</form>

<script>

function store(source) {

localStorage["game.username"] = source.value;

}

</script>

</html>

**main5.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Игра</title>

<script src="mapManager.js"></script>

<script src="spriteManager.js"></script>

<script src="eventsManager.js"></script>

<script src="physicManager.js"></script>

<script src="gameManager.js"></script>

<script src="soundManager.js"></script>

<style>

body {

background-color: burlywood;

}

</style>

</head>

<body>

<canvas id="canvasId" width="704" height="704"></canvas>

<canvas id='score' width='270' height='100'></canvas>

<script>

let canvas = document.getElementById("canvasId");

let ctx = canvas.getContext("2d");

const score = document.getElementById('score');

const ctx2 = score.getContext('2d');

ctx2.font = 'bold 18px Arial';

ctx2.fillStyle = "darkblue";

class Entity {

constructor() {

this.pos\_x = 0;

this.pos\_y = 0;

this.size\_x = 0;

this.size\_y = 0;

}

}

class Player extends Entity {

constructor() {

super();

this.lifetime = 100;

this.move\_x = 0;

this.move\_y = 0;

this.speed = 10;

}

draw (ctx) {

spriteManager.drawSprite(ctx, "player.png", this.pos\_x, this.pos\_y);

}

update () {

physicManager.update(this)

}

onTouchEntity (obj) {

if (obj.name.match(/star[\d]/)) {

this.lifetime += 50;

obj.kill();

}

}

kill () {

console.log("yes");

//soundManager.play('effects/gameover.mp3');

gameManager.game\_over = true;

}

fire () {

let r = new gameManager.factory["Rocket"]();

r.size\_x = 32;

r.size\_y = 32;

r.name = "rocket" + (++gameManager.fireNum);

r.move\_x = this.move\_x;

r.move\_y = this.move\_y;

switch (this.move\_x + 2 \* this.move\_y) {

case -1:

r.pos\_x = this.pos\_x - r.size\_x;

r.pos\_y = this.pos\_y;

break;

case 1:

r.pos\_x = this.pos\_x + r.size\_x;

r.pos\_y = this.pos\_y;

break;

case -2:

r.pos\_x = this.pos\_x;

r.pos\_y = this.pos\_y - r.size\_y;

break;

case 2:

r.pos\_x = this.pos\_x;

r.pos\_y = this.pos\_y + r.size\_y;

break;

default: return;

}

//soundManager.play('effects/shot.mp3');

gameManager.entities.push(r);

}

}

class Boxer extends Entity {

constructor() {

super();

this.lifetime = 100;

this.move\_x = 0;

this.move\_y = 0;

this.speed = 5;

}

draw (ctx) {

spriteManager.drawSprite(ctx, "boxer.png", this.pos\_x, this.pos\_y);

}

update () {

if (gameManager.player.pos\_x > this.pos\_x) this.move\_x = 1;

else this.move\_x = -1

if (gameManager.player.pos\_y > this.pos\_y) this.move\_y = 1;

else this.move\_y = -1;

physicManager.update(this);

}

onTouchEntity (obj) {

if (obj.name.match(/player/)) {

//soundManager.play('effects/boxer.mp3');

if (obj.lifetime > 0) {

obj.lifetime -= 10;

}

if (obj.lifetime <= 0) {

obj.kill();

}

}

}

kill () {

soundManager.play('effects/killenemy.mp3', {volume: 0.5});

gameManager.kill(this);

gameManager.num\_enemies -= 1;

gameManager.score += 100;

}

}

class Rocket extends Entity {

constructor() {

super();

this.move\_x = 0;

this.move\_y = 0;

this.speed = 10;

}

draw (ctx) {

spriteManager.drawSprite(ctx, "rocket3.png", this.pos\_x, this.pos\_y);

}

update () {

physicManager.update(this);

}

onTouchEntity (obj) {

//console.log(this.name + " yes touch " + obj.name);

if (obj.name.match(/enemy[\d\*]/) || obj.name.match(/player/) || obj.name.match(/rocket[\d\*]/)) {

obj.kill();

}

this.kill();

}

onTouchMap (idx) {

this.kill();

soundManager.playWorldSound('effects/boom.mp3', this.pos\_x, this.pos\_y);

}

kill () {

gameManager.kill(this);

}

}

class Bonus extends Entity {

draw (ctx) {

spriteManager.drawSprite(ctx, "star.png", this.pos\_x, this.pos\_y);

}

kill () {

soundManager.play('effects/takebonus.mp3');

gameManager.kill(this);

gameManager.score -= 50;

}

}

let soundarray = ['effects/boom.mp3', 'effects/boxer.mp3', 'effects/gameover.mp3', 'effects/killenemy.mp3',

'effects/newlevel.mp3', 'effects/shot.mp3', 'effects/takebonus.mp3', 'effects/victory.mp3'];

soundManager.init();

soundManager.loadArray(soundarray);

let map1 = "map/tilemap.tmj";

gameManager.loadAll(map1);

gameManager.play();

</script>

</body>

</html>

**mapManager.js**

let mapManager = {

mapData: null,

tLayer: null,

tSet: null,

xCount: 0,

yCount: 0,

tSize: {x: 32, y: 32},

mapSize: {x: 22, y: 22},

tilesets: new Array(),

pict: [],

imgLoadCount: 0,

imgLoaded: false,

jsonLoaded: false,

view: {

x: 0,

y: 0,

w: 704,

h: 704,

},

parseMap(tilesJSON) {

this.mapData = JSON.parse(tilesJSON);

this.xCount = this.mapData.width;

this.yCount = this.mapData.height;

this.tSize.x = this.mapData.tilewidth;

this.tSize.y = this.mapData.tileheight;

this.mapSize.x = this.xCount \* this.tSize.x;

this.mapSize.y = this.yCount \* this.tSize.y;

for (let i = 0; i < this.mapData.tilesets.length; i++) {

let imageSrc = "map/" + (this.mapData.tilesets[i].source);

let request = new XMLHttpRequest();

request.onreadystatechange = () => {

if(request.readyState === 4 && request.status === 200) {

mapManager.tSet = JSON.parse(request.responseText).tiles;

for (let t\_im of mapManager.tSet) {

let img = new Image();

img.src = 'map/' + t\_im.image;

img.onload = () => {

mapManager.imgLoadCount++;

if (mapManager.imgLoadCount === mapManager.tSet.length) {

mapManager.imgLoaded = true;

}

mapManager.pict.push({img: img, id: t\_im.id});

}

}

}

}

request.open('GET', imageSrc, true);

request.send();

let t = this.mapData.tilesets[i];

let ts = {

firstgid: t.firstgid,

imageSource: imageSrc,

name: t.name,

xCount: Math.floor(32 / mapManager.tSize.x),

yCount: Math.floor(32 / mapManager.tSize.y)

};

this.tilesets.push(ts);

}

this.jsonLoaded = true;

},

loadMap(path) {

let request = new XMLHttpRequest();

request.onreadystatechange = () => {

if (request.readyState === 4 && request.status === 200) {

mapManager.parseMap(request.responseText);

}

};

request.open("GET", path, true);

request.send();

},

draw(ctx) {

if (!mapManager.imgLoaded || !mapManager.jsonLoaded) {

setTimeout(() => { mapManager.draw(ctx); }, 100);

} else {

if (this.tLayer === null) {

for (let id = 0; id < this.mapData.layers.length; id++) {

let layer = this.mapData.layers[id];

if (layer.type === 'tilelayer') {

this.tLayer = layer;

}

for (let i = 0; i < this.tLayer.data.length; i++) {

if (this.tLayer.data[i] !== 0) {

let tile = this.getTile(this.tLayer.data[i]);

let pX = (i % this.xCount) \* this.tSize.x;

let pY = Math.floor(i / this.xCount) \* this.tSize.y;

if (!this.isVisible(pX, pY, this.tSize.x, this.tSize.y)) {

continue;

}

pX -= this.view.x;

pY -= this.view.y;

ctx.drawImage(tile.img, pX, pY);

}

}

}

}

}

},

getTile(tileIndex) {

let tile = {

img: null,

px: 0, py: 0,

index: tileIndex,

};

let tileset = this.getTileset(tileIndex);

tile.img = this.pict.filter(el => el.id === tile.index - tileset.firstgid)[0].img;

let id = tileIndex - tileset.firstgid;

let x = id % tileset.xCount;

let y = Math.floor(id / tileset.xCount);

tile.px = x \* mapManager.tSize.x;

tile.py = y \* mapManager.tSize.y;

return tile;

},

getTileset(tileIndex) {

for (let i = mapManager.tilesets.length - 1; i >= 0; i--)

if (mapManager.tilesets[i].firstgid <= tileIndex) {

return mapManager.tilesets[i];

}

return null;

},

isVisible(x, y, width, height){

if (x + width <= this.view.x || y + height <= this.view.y ||

x >= this.view.x + this.view.w ||

y >= this.view.y + this.view.h) {

return false;

}

return true;

},

parseEntities() {

if(!mapManager.imgLoaded || !mapManager.jsonLoaded){

setTimeout(() => {mapManager.parseEntities();}, 100);

} else {

for (let j = 0; j < this.mapData.layers.length; j++) {

if (this.mapData.layers[j].type === 'objectgroup') {

let entities = this.mapData.layers[j];

for (let i = 0; i < entities.objects.length; i++) {

let e = entities.objects[i];

try{

let obj = new gameManager.factory[e.class]();

obj.name = e.name;

obj.pos\_x = e.x;

obj.pos\_y = e.y - e.height;

obj.size\_x = e.width;

obj.size\_y = e.height;

gameManager.entities.push(obj);

if (e.class === "Boxer") {

gameManager.num\_enemies += 1;

}

if(obj.name === 'player'){

console.log('Плеер найден');

gameManager.initPlayer(obj);

}

} catch(ex) {

console.log('Error while creating: [' + e.gid + '] ' + e.class + ', ' + ex);

}

}

}

}

}

},

getTilesetIdx(x, y) {

let wX = x;

let wY = y;

let idx = Math.floor(wY/this.tSize.y) \* this.xCount + Math.floor(wX/this.tSize.x);

return this.tLayer.data[idx];

},

centerAt(x, y) {

if(x < this.view.w/2){

this.view.x = 0;

} else if(x > this.mapSize.x \* this.tSize.x - this.view.w/2){

this.view.x = this.mapSize.x \* this.tSize.x - this.view.w;

} else {

this.view.x = (x - (this.view.w/2));

}

if(y < this.view.h/2){

this.view.y = 0;

} else if(y > this.mapSize.y \* this.tSize.y - this.view.h/2){

this.view.y = (this.mapSize.y \* this.tSize.y - this.view.h);

} else {

this.view.y = (y - (this.view.h/2));

}

}

}

**spriteManager.js**

let spriteManager = {

image: new Image(),

sprites: new Array(),

imgLoadCount: 0,

imgLoaded: false,

jsonLoaded: false,

loadAtlas(atlasJson) {

let request = new XMLHttpRequest();

request.onreadystatechange = () => {

if (request.readyState === 4 && request.status === 200) {

spriteManager.parseAtlas(request.responseText);

}

};

request.open("GET", atlasJson, true);

request.send();

},

parseAtlas(atlasJSON) {

let atlas = JSON.parse(atlasJSON).tiles;

for (let im of atlas) {

let img = new Image();

img.src = 'map/' + im.image;

img.onload = () => {

spriteManager.imgLoadCount++;

if (spriteManager.imgLoadCount === atlas.length) {

spriteManager.imgLoaded = true;

}

this.sprites.push({name: im.image, src: img, w: im.imagewidth, h: im.imageheight});

}

this.jsonLoaded = true;

}

},

drawSprite(ctx, name, x, y) {

if (!this.imgLoaded || !this.jsonLoaded) {

setTimeout(() => { spriteManager.drawSprite(ctx, name, x, y); }, 100);

} else {

let sprite = this.getSprite(name);

if (!mapManager.isVisible(x, y, sprite.w, sprite.h))

return;

x -= mapManager.view.x;

y -= mapManager.view.y;

ctx.drawImage(sprite.src, x, y);

}

},

getSprite(name) {

return this.sprites.find(el => el.name === name);

}

};

**eventsManager.js**

let eventsManager = {

bind: [],

action: [],

setup(canvas) {

this.bind[87] = 'up';

this.bind[65] = 'left';

this.bind[83] = 'down';

this.bind[68] = 'right';

this.bind[32] = 'fire';

canvas.addEventListener("mousedown", this.onMouseDown);

canvas.addEventListener("mouseup", this.onMouseUp);

document.body.addEventListener("keydown", this.onKeyDown);

document.body.addEventListener("keyup", this.onKeyUp);

},

onMouseDown(event) {

eventsManager.action["fire"] = true;

},

onMouseUp(event) {

eventsManager.action["fire"] = false;

},

onKeyDown(event) {

let action = eventsManager.bind[event.keyCode];

if (action)

eventsManager.action[action] = true;

},

onKeyUp(event) {

let action = eventsManager.bind[event.keyCode];

if (action)

eventsManager.action[action] = false;

}

};

**physicManager.js**

let physicManager = {

update(obj) {

if (obj.move\_x === 0 && obj.move\_y === 0)

return "stop";

let newX = obj.pos\_x + Math.floor(obj.move\_x \* obj.speed);

let newY = obj.pos\_y + Math.floor(obj.move\_y \* obj.speed);

let ts = mapManager.getTilesetIdx(newX + obj.size\_x / 4, newY + obj.size\_y / 4);

let e = this.entityAtXY(obj, newX, newY);

if (e!== null && obj.onTouchEntity)

obj.onTouchEntity(e);

if (ts !== 0 && obj.onTouchMap)

obj.onTouchMap(ts);

if (ts === 0 && e === null) {

obj.pos\_x = newX;

obj.pos\_y = newY;

} else

return "break";

return "move";

},

entityAtXY(obj, x, y) {

for (let i = 0; i < gameManager.entities.length; i++) {

let e = gameManager.entities[i];

if (e.name !== obj.name) {

if (x + obj.size\_x / 1.3 < e.pos\_x || y + obj.size\_y < e.pos\_y || x > e.pos\_x + e.size\_x || y > e.pos\_y + e.size\_y / 1.2)

continue;

return e;

}

}

return null;

}

};

**gameManager.js**

let gameManager = {

factory: {},

entities: [],

fireNum: 0,

player: null,

laterKill: [],

num\_enemies: 0,

num: 0,

score: 0,

name: null,

game\_over: false,

initPlayer(obj) {

this.player = obj;

},

kill(obj) {

this.laterKill.push(obj);

},

update() {

if (this.player === null)

return;

this.player.move\_x = 0;

this.player.move\_y = 0;

if (eventsManager.action["up"]) this.player.move\_y = -1;

if (eventsManager.action["down"]) this.player.move\_y = 1;

if (eventsManager.action["left"]) this.player.move\_x = -1;

if (eventsManager.action["right"]) this.player.move\_x = 1;

if (eventsManager.action["fire"]) this.player.fire();

this.entities.forEach((e) => {

try {

e.update();

} catch(ex) {}

});

for (let i = 0; i < this.laterKill.length; i++) {

let idx = this.entities.indexOf(this.laterKill[i]);

if (idx > -1)

this.entities.splice(idx, 1);

}

if (this.laterKill.length > 0)

this.laterKill.length = 0;

mapManager.tLayer = null;

mapManager.draw(ctx);

this.draw(ctx);

this.show\_info();

if (gameManager.num\_enemies === 0 && gameManager.num === 1) {

soundManager.play('effects/victory.mp3');

ctx.font = 'bold 35px Arial';

ctx.fillStyle = "black";

ctx.fillText("Победа!", 300, 352);

gameManager.player = null;

gameManager.entities = [];

gameManager.num += 1;

gameManager.save\_record(gameManager.name, gameManager.score);

setTimeout(() => {window.open("records.html", "\_self");}, 6000);

}

if (gameManager.num\_enemies === 0 && gameManager.num === 0) {

soundManager.play('effects/newlevel.mp3');

ctx.font = 'bold 35px Arial';

ctx.fillStyle = "white";

ctx.fillText("Переход на второй уровень!", 100, 352);

let map2 = "map/tilemap2.tmj";

gameManager.player = null;

gameManager.entities = [];

gameManager.num += 1;

setTimeout(() => {gameManager.loadAll(map2);}, 2500);

}

if (this.game\_over) {

ctx.font = 'bold 35px Arial';

if (gameManager.num === 0)

ctx.fillStyle = "white";

else ctx.fillStyle = "black";

ctx.fillText("Игра закончена!", 200, 352);

gameManager.player = null;

gameManager.entities = [];

gameManager.save\_record(gameManager.name, gameManager.score);

setTimeout(() => {window.open("records.html", "\_self");}, 1500);

}

},

show\_info() {

ctx2.clearRect(0, 0, score.width, score.height);

ctx2.fillText('Очки: ' + gameManager.score, 15, 20);

ctx2.fillText('Игрок: ' + gameManager.name, 15, 50);

ctx2.fillText('Количество здоровья: ' + gameManager.player.lifetime, 15, 80);

},

save\_record(name, score) {

if (!localStorage[name]) {

localStorage[name] = score;

}

if (score > localStorage[name]) {

localStorage[name] = score;

}

},

draw(ctx) {

for (let e = 0; e < this.entities.length; e++)

this.entities[e].draw(ctx);

},

loadAll(namem) {

mapManager.imgLoadCount = 0;

mapManager.imgLoaded = false;

mapManager.jsonLoaded = false;

mapManager.loadMap(namem);

spriteManager.loadAtlas("map/tileset.tsj");

gameManager.factory['Player'] = Player;

gameManager.factory['Boxer'] = Boxer;

gameManager.factory['Bonus'] = Bonus;

gameManager.factory['Rocket'] = Rocket;

mapManager.parseEntities();

mapManager.draw(ctx);

eventsManager.setup(canvas);

gameManager.name = localStorage.getItem("game.username");

},

play() {

setInterval(this.updateWorld, 100);

},

updateWorld() {

gameManager.update();

}

};

**soundManager.js**

let soundManager = {

clips: {},

context: null,

gainNode: null,

loaded: false,

init() {

this.context = new AudioContext();

this.gainNode = this.context.createGain ? this.context.createGain() : this.context.createGainNode();

this.gainNode.connect(this.context.destination);

},

load(path, callback) {

if (this.clips[path]) {

callback(this.clips[path]);

return;

}

let clip = {path: path, buffer: null, loaded: false};

clip.play = (volume, loop) => {

soundManager.play(this.path, {looping: loop ? loop : false,

volume: volume ? volume: 1});

};

this.clips[path] = clip;

let request = new XMLHttpRequest();

request.open('GET', path, true);

request.responseType = 'arraybuffer';

request.onload = () => {

soundManager.context.decodeAudioData(request.response, (buffer) => {

clip.buffer = buffer;

clip.loaded = true;

callback(clip);

});

};

request.send();

},

loadArray(array) {

for (let i = 0; i < array.length; i++) {

soundManager.load(array[i], () => {

if (array.length === Object.keys(soundManager.clips).length) {

for (let sd in soundManager.clips) {

if (!soundManager.clips[sd].loaded) return;

soundManager.loaded = true;

}

}

});

}

},

play(path, settings) {

if(!soundManager.loaded){

setTimeout(() => {soundManager.play(path, settings);}, 100);

return;

}

let looping = false;

let volume = 1;

if (settings) {

if(settings.looping)

looping = settings.looping;

if(settings.volume)

volume = settings.volume;

}

let sd = this.clips[path];

if(sd === null)

return false;

let sound = soundManager.context.createBufferSource();

sound.buffer = sd.buffer;

sound.connect(soundManager.gainNode);

sound.loop = looping;

soundManager.gainNode.gain.value = volume;

this.context.resume().then(sound.start(0));

return true;

},

playWorldSound(path, x, y) {

if (gameManager.player === null)

return;

let viewSize = Math.max(mapManager.view.w, mapManager.view.h) \* 0.8;

let dx = Math.abs(gameManager.player.pos\_x - x);

let dy = Math.abs(gameManager.player.pos\_y - y);

let distance = Math.sqrt(dx \* dx + dy \* dy);

let norm = distance / viewSize;

if (norm > 1)

norm = 1;

let volume = 1.0 - norm;

if (!volume)

return;

soundManager.play(path, {looping: false, volume: volume});

},

toggleMute() {

if (this.gainNode.gain.value > 0)

this.gainNode.gain.value = 0;

else

this.gainNode.gain.value = 1;

},

stopAll() {

this.gainNode.disconnect();

this.gainNode = this.context.createGainNode(0);

this.gainNode.connect(this.context.destination);

}

}

**records.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Рекорды</title>

<style>

body{

background-color: burlywood;

}

h1{

text-align:center;

color: darkblue;

}

canvas {

margin-left: auto;

margin-right: auto;

display: flex;

background: burlywood;

border: thick dashed darkblue;

}

button {

margin-left: auto;

margin-right: auto;

margin-top: 5px;

display: flex;

background: burlywood;

border: thick dashed darkblue;

}

</style>

</head>

<body>

<h1><font face="Times New Roman">Рекорды</font></h1>

<canvas id='records' width='320' height=610></canvas>

<button id="back" onclick="window.open('main5.html', '\_self')"><font color="#00008b">Назад</font></button>

</body>

<script>

const canvas = document.getElementById('records');

const ctx = canvas.getContext('2d');

function clear\_window(){

ctx.clearRect(0, 0, canvas.width, canvas.height);

}

function show\_records() {

clear\_window();

let records = []

records = localStorage;

//console.log(records);

let records\_dict = Object.keys(records).map((key) => {

return [key, records[key]];

});

records\_dict.sort((a\_el, b\_el) => {

return b\_el[1] - a\_el[1];

});

ctx.font = '14px Arial';

ctx.fillStyle = "darkblue";

let i\_sh = 0;

for (let i\_s = 0; i\_s < records\_dict.length; i\_s++) {

if (records\_dict[i\_s][0] !== "game.username") {

ctx.fillText(records\_dict[i\_s][0] + ' ' + records\_dict[i\_s][1] + ' ' + 'очков', 15, 15 \* (i\_sh + 1));

i\_sh += 1;

}

}

}

show\_records();

</script>

</html>