**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В. И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**КУРСОВАЯ РАБОТА**

**по дисциплине «Web-технологии»**

**Тема: Разработка игры на JavaScript**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1381 |  | Возмитель В. Е. |
| Преподаватель |  | Беляев С. А. |

Санкт-Петербург

2023

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ**

|  |  |
| --- | --- |
| Студент Возмитель В. Е. | |
| Группа 1381 | |
| Тема работы: разработка игры на JavaScript | |
| Исходные данные:  Необходимо выполнить курсовую работу в соответствии с учебным пособием Беляев С.А. "Разработка игр на языке JavaScript", используя стандарт ES6 языка программирования JavaScript. Проект должен соответствовать следующим условиям:   1. Реализация не менее двух уровней игры. 2. Работоспособность всех менеджеров, описанных в учебном пособии. 3. Наличие таблицы рекордов. 4. Присутствие препятствий в уровнях. 5. Программная реализация "интеллектуальных" противников и "бонусов". 6. Использование tiles с использованием редактора Tiled, согласно   указаниям в учебном пособии. | |
| Содержание пояснительной записки:  «Содержание», «Введение», «Заключение», «Список использованных источников» | |

|  |  |  |
| --- | --- | --- |
| Предполагаемый объем пояснительной записки:  Не менее 20 страниц. | | |
| Дата выдачи задания: 01.09.2023 | | |
| Дата сдачи реферата: 28.11.2023 | | |
| Дата защиты реферата: 28.11.2023 | | |
| Студент |  | Возмитель В. Е. |
| Преподаватель |  | Беляев С.А. |

**АННОТАЦИЯ**

Курсовой проект представляет собой 2D-игру «Танки», созданный с использованием HTML5 и JavaScript стандарта ES6. В проекте используется модульная структура, которая включает в себя менеджеры событий, игровых объектов, управление уровнями, физическое взаимодействие и звук. Исходный код содержит несколько менеджеров, таких как MapManager, EventsManager, GameManager, PhysicManager, SpriteManager, SoundManager и RecordManager, а также основные компоненты игры - Player и Enemy. Проект также включает функциональность для обработки событий клавиш, управления персонажем, обработки коллизий, взаимодействия с уровнями и звукового сопровождения.

**SUMMARY**

The course project is a 2D game ”Tanks", created using HTML5 and JavaScript ES6 standard. The project uses a modular structure that includes event managers, game objects, level management, physical interaction and sound. The source code contains several managers, such as MapManager, EventsManager, GameManager, PhysicManager, SpriteManager, SoundManager and RecordManager, as well as the main components of the game - Player and Enemy. The project also includes functionality for key event handling, character control, collision handling, level interaction, and soundtrack.

**СОДЕРЖАНИЕ**

|  |  |  |
| --- | --- | --- |
|  | Введение | 6 |
| 1. | Реализация менеджеров | 7 |
| 1.1. | MapManager | 7 |
| 1.2. | EventsManager | 8 |
| 1.3. | GameManager | 9 |
| 1.4. | PhysicManager | 10 |
| 1.5. | SpriteManager | 11 |
| 1.6. | SoundManager | 12 |
| 1.7. | Recorder | 12 |
| 2. | Реализация игровых сущностей | 13 |
| 2.1. | Player | 13 |
| 2.2. | Tank | 13 |
| 2.3. | Rocket | 13 |
| 2.4. | Bonus | 14 |
| 3. | Реализация клиентской части | 15 |
| 3.1. | Таблица рекордов | 15 |
| 3.2. | Начальная страница | 15 |
| 3.3. | Главная страница | 15 |
| 3.4. | Страница-таблица | 16 |
| 4. | Реализация карты в Tiled | 18 |
|  | Заключение | 19 |
|  | Список использованных источников | 20 |
|  | Приложение А. Исходный код программ. | 21 |

**ВВЕДЕНИЕ**

Цель данной работы заключается в разработке 2D-игры «Танки» на языке JavaScript (ES6), соответствуя учебному пособию. Главные задачи включают создание не менее двух уровней игры, реализацию всех рекомендованных менеджеров (MapManager, SpriteManager, PhysicManager, EventsManager, SoundManager, GameManager), таблицы рекордов, добавление препятствий и интеллектуальных противников, а также использование tiles с редактором Tiled для построения уровней. Достижение этих целей требует применения методов, таких как использование стандарта ES6, различных менеджеров для управления игровыми компонентами, работа с локальным хранилищем, реализация алгоритмов обнаружения столкновений и управления движением противников, а также интеграция с редактором Tiled для удобного создания уровней с использованием tiles.

## РЕАЛИЗАЦИЯ МЕНЕДЖЕРОВ

* 1. **MapManager.**

В данном учебном пособии объект для управления картой (или менеджера карты) называется mapManager.

Менеджер карты обеспечивает правильную и полную загрузку карты, представленной в виде JSON-файла, созданного в Tiled. В этом менеджере реализованы методы для обработки данных JSON файла, для отрисовки этих данных на canvas, а также вспомогательные методы.

Менеджер содержит следующие свойства:

* mapData: данные карты;
* tLayer: слой тайлов карты;
* xCount, yCount: количество тайлов по горизонтали и вертикали;
* tSize: размеры одного тайла;
* mapSize: размеры всей карты;
* tilesets: список тайлсетов, используемых на карте;
* imgLoadCount: количество загруженных изображений тайлсетов;
* imgLoaded, jsonLoaded: флаги, указывающие на загрузку изображений и данных карты;
* view: видимая область карты;

Класс также содержит следующие методы:

* createEnemy(): создает вражеского танка на заданных координатах и добавляет его в массив entities игры;
* parseMap(tilesJSON): парсит данные карты и загружает изображения тайлсетов;
* draw(ctx): отрисовывает карту на основе загруженных данных;
* loadMap(path): загружает данные карты по заданному пути;
* parseEntities(): разбирает слой объектов карты и создает соответствующие объекты в игре.

Таким образом, класс MapManager предоставляет функциональность для работы с картой игры, загрузки ее данных, отображения и создания объектов на карте.

* 1. **GameManeger**

GameManager играет роль в общей организации игрового процесса. Он согласовывает действия всех менеджеров, обрабатывает изменения состояния игры, управляет жизненным циклом игры, включая инициализацию, завершение уровней, а также обработку победы и поражения. Кроме того, этот менеджер создает и управляет экземплярами игровых объектов, включая персонажей и врагов.

Классы, определенные в коде:

* Bonus - класс, описывающий бонусы, которые игрок может собирать в игре.
* Player - класс, описывающий игрока.
* Rocket - класс, описывающий снаряды, которые может выпускать игрок.
* Tank - класс, описывающий танки, противников, с которыми игрок будет сражаться.

Переменные, определенные в коде:

* recorder - переменная, которая используется для записи результатов игры.
* ctx - контекст рисования на холсте.
* canvas - холст, на котором происходит отрисовка игры.
* eventsManager - менеджер событий, отвечающий за обработку пользовательских действий.
* mapManager - менеджер карты, отвечающий за загрузку и отрисовку карты игрового мира.
* spriteManager - менеджер спрайтов, отвечающий за загрузку и отображение спрайтов игровых сущностей.
* gameManager - экземпляр класса GameManager, который управляет игровым процессом.

Методы, определенные в классе GameManager:

* initPlayer(obj) - метод, который инициализирует игрока.
* end() - метод, который вызывается при окончании игры и записывает результаты игры.
* kill(obj) - метод, который добавляет объект в список объектов, которые будут удалены в следующем обновлении игры.
* update() - метод, который обновляет состояние игровых объектов и отрисовывает игровой мир.
* draw(ctx) - метод, который отрисовывает игровые объекты на холсте.
* loadAll(jsonPath, mapNum) - метод, который загружает карту игрового мира и настраивает менеджеры карты и спрайтов.
* play() - метод, который запускает игровой процесс и обновление игрового мира.
  1. **SpriteManager**

Класс SpriteManager позволяет удобно управлять спрайтами, загружать их изображения и данные из JSON-файла, а также отрисовывать спрайты на канвасе.

* Конструктор класса инициализирует два логических флага: imgLoaded и jsonLoaded, а также создает новое изображение.
* Метод parseAtlas принимает JSON-строку с информацией о спрайтах и парсит ее, записывая полученные значения в массив спрайтов.
* Метод loadAtlas загружает асинхронно JSON-файл и вызывает метод parseAtlas при успешной загрузке. Также он вызывает метод loadImg для загрузки изображения спрайтов.
* Метод loadImg загружает изображение спрайтов асинхронно и устанавливает флаг imgLoaded в true при успешной загрузке.
* Метод drawSprite отвечает за отрисовку спрайта на указанном контексте. Если изображение или JSON-файл еще не загружены, то метод вызывает себя через 100 мс. Если спрайт находится вне видимой области карты, то он не рисуется.
* Метод getSprite возвращает объект спрайта по его имени.
  1. **EventsMabager**

Класс EventsManager предоставляет интерфейс для регистрации и обработки событий в игре. Он связывает клавиши и манипуляции мышью с определенными действиями, которые затем можно использовать в основной части игры для выполнения соответствующей логики.

В конструкторе класса определены два массива: bind и action. Массив bind используется для связывания клавиш с определенными действиями (например, клавиша 87 - 'W' связана с действием 'up'). Массив action представляет собой список текущих активных действий в игре.

Метод setup принимает в качестве аргумента элемент canvas, на котором будут происходить события.

Внутри метода определены обработчики событий:

* при нажатии кнопки мыши (mousedown) вызывается функция onMouseDown, которая устанавливает значение 'true' для действия 'fire'.
* при отпускании кнопки мыши (mouseup) вызывается функция onMouseUp, которая устанавливает значение 'false' для действия 'fire'.
* при нажатии клавиши на клавиатуре (keydown) вызывается функция onKeyDown, которая проверяет, есть ли связанное действие для нажатой клавиши, и если есть, то устанавливает значение 'true' для этого действия в массиве action.
* при отпускании клавиши (keyup) вызывается функция onKeyUp, которая также проверяет, есть ли связанное действие для отпущенной клавиши, и если есть, то устанавливает значение 'false' для этого действия в массиве action.

Все обработчики событий используют стрелочные функции для сохранения контекста (this).

* 1. **PhysicsManager**

PhysicManager отвечает за обработку физических взаимодействий в игре. Этот менеджер управляет столкновениями игровых объектов, определяет их движение и обновляет их положение в соответствии с физическими правилами. Он содержит следующие свойства:

* player\_x и player\_y представляют координаты игрока на карте.
* playerHasMoved - флаг, указывающий был ли игрок перемещен.
* ind - объект с индексами для различных элементов на карте, таких как игрок, стены и враги.
* addNewEnemy - флаг, указывающий нужно ли добавить нового врага.

Класс также имеет несколько методов:

* findPlayer(playerX, playerY, startX, startY) - асинхронный метод, который находит путь от заданных начальных координат startX и startY до координат playerX и playerY на карте. Он использует алгоритм поиска в ширину (bfs) для поиска кратчайшего пути. Возвращает массив с координатами клеток, через которые нужно пройти для достижения цели.
* update(obj) - метод, обновляющий координаты объекта на карте. Принимает объект с координатами (pos\_x, pos\_y), направлением (move\_x, move\_y) и скоростью (speed). Проверяет возможность перемещения объекта на новые координаты и обрабатывает события соприкосновения с другими объектами или элементами карты. Возвращает строку-сигнал "stop", "move" или "break", указывающую состояние объекта после обновления.
* entityAtXY(obj, x, y) - метод, проверяющий наличие других объектов в заданных координатах (x, y) на карте. Возвращает найденный объект или null, если объект не найден.
  1. **SoundManager**

SoundManager управляет звуковыми эффектами игры, инициализирует звуковые контексты, загружает звуковые файлы и предоставляет возможность воспроизводить звуки с различными настройками.

SoundManager обеспечивает звуковую обратную связь для игровых событий, таких как столкновения, победы и поражения.

Этот менеджер тесно сотрудничает с GameManager, реагируя на различные игровые ситуации и воспроизводя соответствующие звуковые эффекты. SoundManager также связан с другими менеджерами, чтобы реагировать на игровые события и воспроизводить звуки в зависимости от действий пользователя.

* 1. **Recorder**

Recorder отвечает за управление таблицей рекордов в игре. Он добавляет время прохождения игры и далее сортирует по возрастанию.

Для каждого уровня игры своя таблица рекордов. Так записи не будут перемешиваться в таблицах. Recorder связан с GameManager для обновления рекордов при достижении новых результатов.

## РЕАЛИЗАЦИЯ ИГРОВЫХ СУЩНОСТЕЙ

В данном разделе представлено описание реализации ключевых игровых сущностей, таких как Entity, Player, Tank, Bonus, . Класс Entity предоставляет базовый шаблон для всех игровых сущностей, определяя их базовые характеристики, такие как позиция и размер.

* 1. **Player**

Класс игрока наследуется от класса Entity, наследуя размеры и координаты. В классе Player есть поля. отвечающие за направление движения игрока, также скорость, игровые деньги и количество игровых жизней.

Реализованы следующие методы:

* draw(ctx) – отрисовка игрока, в зависимости от его будущего направления.
* update() – обновление позиции игрока на карте.
* onTouchEntity(obj) – столкновение с другими объектами карты.
* kill() – смерть игрока.
* fire() – стрельбаю.
  1. **Tank**

Класс Tank имеет те же поля и методы, что и игрок. Но добавлен метод Attack() для обработки движения и атаки на игрока.

Вражеский танк обладает возможностью периодически стрелять, при этом двигаясь к игроку.

* 1. **Rocket**

Класс Rocket наследуется от класса Entity и имеет следующие свойства: name, move\_x, move\_y speed.

Он имеет следующие методы:

* draw(ctx): метод отрисовки ракеты на холсте ctx в зависимости от значения свойств move\_x и move\_y.
* update(): метод обновления ракеты.
* onTouchEntity(obj): метод вызывается при соприкосновении с другим объектом. Если имя объекта соответствует "enemy" или "player", вызывается метод kill у объекта obj, а затем метод kill у текущего объекта Rocket.
* onTouchMap(): метод вызывается при соприкосновении с картой. Вызывается метод kill у текущего объекта Rocket.
* kill(): метод вызывается для "убийства" ракеты.
  1. **Bonus**

Класс Bonus также наследуется от класса Entity и имеет следующие методы:

* draw(ctx): метод отрисовки бонуса (монетки) на холсте ctx. Используется объект spriteManager для отрисовки спрайтов.
* kill(): метод вызывается для "убийства" бонуса. Если звуковой менеджер загружен и звук уже воспроизводится, то вызывается метод play у объекта soundManager и проигрывается звуковой файл. Затем объект удаляется.

1. **РЕАЛИЗАЦИЯ КЛИЕНТСКОЙ ЧАСТИ**
   1. **Таблица рекордов.**

Recorder отвечает за управление таблицей рекордов в игре.

В зависимости от уровня игры, который выберет клиент, будет создана соответствующая уровню таблица рекордов.

В таблицу рекордов записывается время прохождения игрока карты. При достижении всех трёх монет клиента перенаправляет на страницу, где будет записан его результат. Результат записывается в виде времени и между собой результаты сортируются по возрастанию. При поражении игрока в таблицу будет записано нулевое время.

Также для каждой таблицы есть кнопка для ее очистки.

* 1. **Начальная страница.**

Начальная страница имеет кнопку войти, где пользователь может ввести своё имя и выбрать уровень игры.

В ситуации, когда пользователь не ввел своё имя, то программа подставит на место имени в таблице строку “Player”.



Рисунок 1. Начальная страница.

* 1. **Главная страница.**

Главная страница игры включает в себя элемент canvas. Данный элемент используется для отображения игрового поля. Скрипт игры, подключенный в виде модуля, импортирует необходимые классы, такие как MapManager, EventsManager, GameManager, PhysicManager, SpriteManager, RecordManager, SoundManager. В самом начале работы скрипта создаются экземпляры каждого из этих менеджеров, а также устанавливаются связи между ними. Происходит инициализация звукового менеджера, загрузка массива звуков, а также создание объекта карты, менеджера спрайтов, игрового менеджера, менеджера физики, менеджера событий, и менеджера записей рекордов.



Рисунок 2. Главная страница (1уровень).



Рисунок 3. Главная страница (2уровень).

* 1. **Страница-таблица.**

Данная страница информирует о завершении игры и показывает результаты ее прохождения.

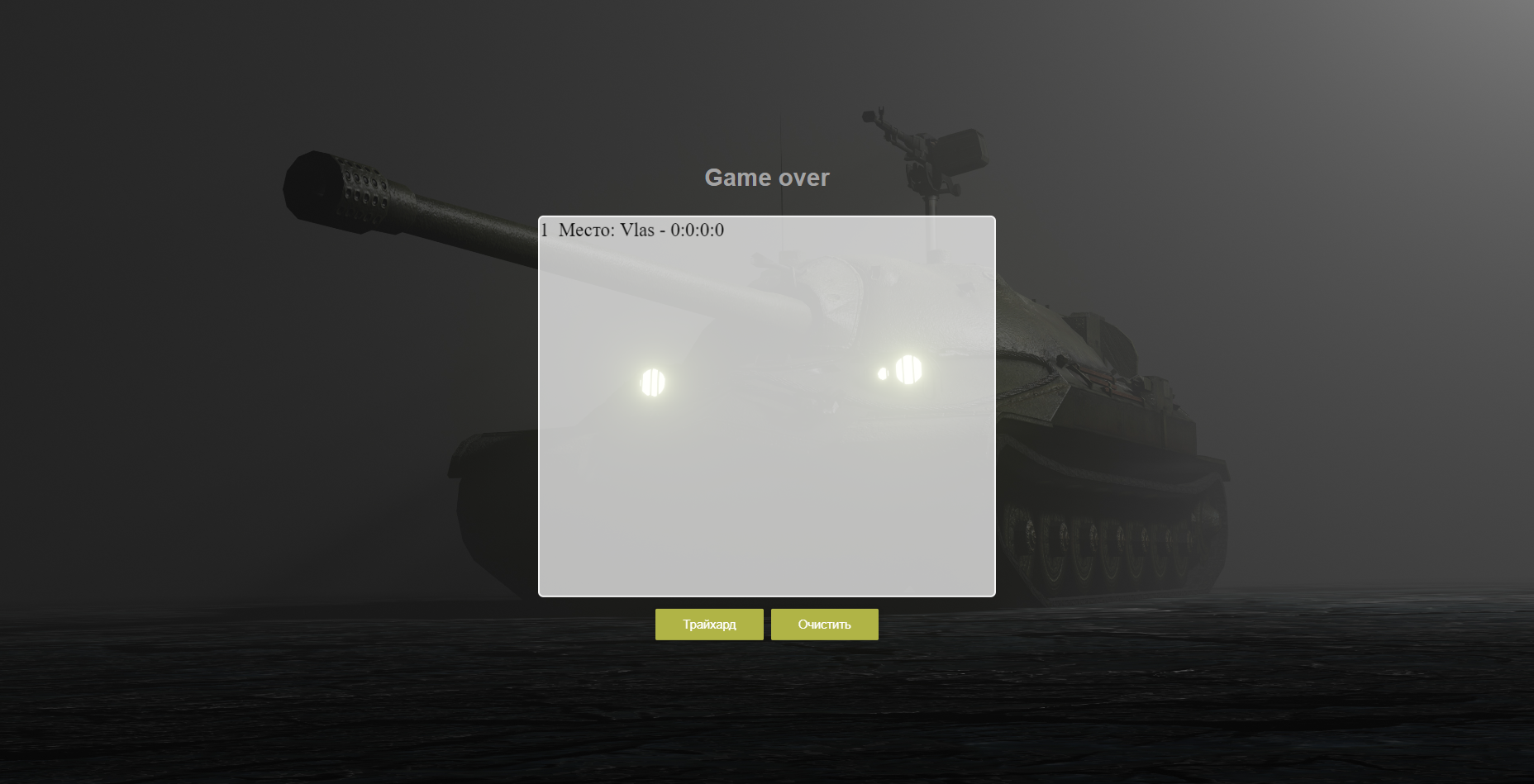
****

Рисунок 4. Страница-таблица.

1. **РЕАЛИЗАЦИЯ КАРТЫ В TILED**

Для создания уровней в игре использовался редактор карт Tiled. Каждый уровень представляет собой JSON-файл, содержащий информацию о тайлах, их расположении и свойствах. MapManager отвечает за загрузку и обработку карт. При инициализации он считывает JSON-файл уровня, извлекает информацию о тайлах и их свойствах. С помощью SpriteManager отображаются соответствующие изображения, а PhysicManager определяет физические свойства тайлов. Это обеспечивает простоту добавления и изменения уровней, а также позволяет использовать разнообразные тайлы для создания интересного игрового мира.

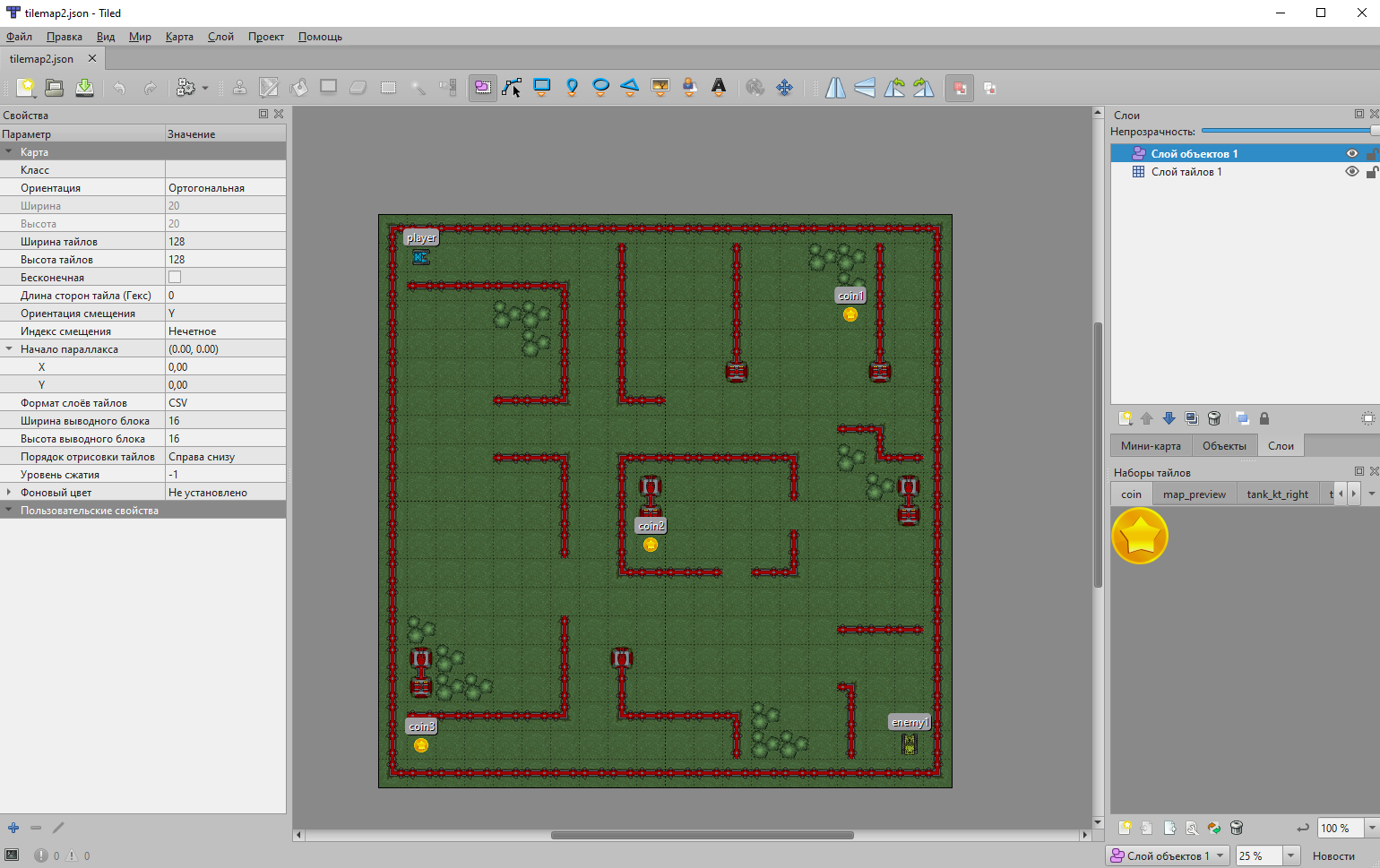


Рисунок 5. Редактор карт Tiled.

## ЗАКЛЮЧЕНИЕ.

При разработке курсового проекта, направленного на создание игры на языке JavaScript по стандартам ES6, была достигнута основная цель: создание интересного и функционального игрового приложения. Реализована двухуровневая игра с препятствиями, умными противниками и бонусами. Весь код структурирован с использованием различных обработчиков для управления различными аспектами игры.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ.

1. Беляев С. А. Разработка игр на языке JavaScript: Учебное пособие. — СПб.: Издательство «Лань», 2016. — 128 с.: ил. — (Учебники для вузов. Специальная литература). ISBN 9785811419807

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММ**

1. **MapManager.js**

*import {gameManager} from "./main.js";*

*export class MapManager{*

*constructor() {*

*this.mapData = null;*

*this.tLayer = null;*

*this.xCount = 0;*

*this.yCount = 0;*

*this.tSize = {x: 128, y: 128};*

*this.mapSize = {x: 2560, y: 2560};*

*this.tilesets = [];*

*this.imgLoadCount = 0;*

*this.imgLoaded = false;*

*this.jsonLoaded = false;*

*this.view = {x: 0, y: 0, w: 2560, h: 2560};*

*this.mazeMap = []*

*}*

*createEnemy(){*

*let obj = new gameManager.factory["Tank"](gameManager);*

*obj.name = "enemy1";*

*obj.pos\_x = 2329;*

*obj.pos\_y = 2419 - 80;*

*obj.size\_x = 70;*

*obj.size\_y = 94;*

*gameManager.entities.push(obj);*

*}*

*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;*

*let tmpMazeMap = [...this.mapData.layers[0].data]; //???? 0 элемент ?????*

*let h = this.mapData.layers[0].height;*

*let w = this.mapData.layers[0].width;*

*for(let i = 0; i < h; i++){*

*let row = []*

*for(let j = 0; j < w; j++){*

*row.push(tmpMazeMap[w\*i +j]);*

*}*

*this.mazeMap.push(row);*

*}*

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

*let img = new Image();*

*img.onload = () => {*

*this.imgLoadCount++;*

*if (this.imgLoadCount === this.mapData.tilesets.length) {*

*this.imgLoaded = true;*

*}*

*}*

*img.src = this.mapData.tilesets[i].image;*

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

*let ts = {*

*firstgid: t.firstgid,*

*image: img,*

*name: t.name,*

*xCount: Math.floor(t.imagewidth / this.tSize.x),*

*yCount: Math.floor(t.imageheight / this.tSize.y)*

*}*

*this.tilesets.push(ts);*

*}*

*this.jsonLoaded = true;*

*}*

*draw(ctx) {*

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

*setTimeout(() => {*

*this.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;*

*break;*

*}*

*}*

*}*

*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, tile.px, tile.py, this.tSize.x, this.tSize.y, pX, pY, this.tSize.x, this.tSize.y);*

*}*

*}*

*}*

*}*

*loadMap = (path) => {*

*return new Promise((resolve, reject) => {*

*const request = new XMLHttpRequest();*

*request.onreadystatechange = () => {*

*if (request.readyState === 4) {*

*if (request.status === 200) {*

*this.parseMap(request.responseText);*

*resolve();*

*} else {*

*reject(new Error("Ошибка загрузки карты"));*

*}*

*}*

*};*

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

*request.send();*

*});*

*}*

*parseEntities() {//разбор слоя на типы objectgroup*

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

*setTimeout(() => {*

*this.parseEntities();*

*}, 100);*

*} else*

*for (let j = 0; j < this.mapData.layers.length; j++) //проверка всех слоев*

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

*let entities = this.mapData.layers[j]; //слой с объектами следует "разобрать"*

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

*let e = entities.objects[i];*

*try {*

*let obj = new gameManager.factory[e.type]; //в соответсвии с типом моздаем экземпляр объекта*

*obj.name = e.name;*

*obj.pos\_x = e.x;*

*obj.pos\_y = e.y-80;*

*obj.size\_x = e.width;*

*obj.size\_y = e.height;*

*//помещаем в массив объектов*

*gameManager.entities.push(obj);*

*if (obj.name === "player")*

*gameManager.initPlayer(obj);*

*} catch (ex) {*

*console.log("Error while creating: [" + e.gid + "]" + e.type + "," + ex)*

*}*

*}*

*}*

*}*

*getTilesetIdx(x, y) {*

*//получить блок по координатам по карте*

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

*return this.tLayer.data[idx];*

*}*

*centerAt(x, y) {*

*if (x < this.view.w / 2)//центрирование по горизонтали*

*this.view.x = 0;*

*else if (x > this.mapSize.x - this.view.w / 2)*

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

*else*

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

*if (y < this.view.h / 2)//центрирование по вертикали*

*this.view.y = 0;*

*else if (y> this.mapSize.y - this.view.h / 2)*

*this.view.y = this.mapSize.y - this.view.h;*

*else*

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

*}*

*getTile (tileIndex) {*

*let tile = {*

*img: null,*

*px: 0,*

*py: 0*

*};*

*let tileset = this.getTileset(tileIndex);*

*tile.img = tileset.image;*

*let id = tileIndex - tileset.firstgid;*

*let x = id % tileset.xCount;*

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

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

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

*return tile;*

*}*

*getTileset(tileIndex) {*

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

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

*return this.tilesets[i];*

*}*

*return null;*

*}*

*isVisible(x, y, width, height) {*

*return !(x + width < this.view.x || y + height < this.view.y || x > this.view.x + this.view.w || y > this.view.y + this.view.h);*

*//my(comp) change*

*}*

*}*

1. **Entity.js**

*import {ctx, gameManager, mapManager, physicManager, soundManager, spriteManager} from "./main.js";*

*export class Entity{*

*constructor() {*

*this.pos\_x = 0;*

*this.pos\_y = 0;*

*this.size\_x = 0;*

*this.size\_y = 0;*

*}*

*}*

*export class Player extends Entity{*

*constructor() {*

*super();*

*this.lifetime = 100;*

*this.move\_x = 0;*

*this.move\_y = 0;*

*this.last\_position = "";*

*this.speed = 20;*

*this.cash = 0;*

*}*

*draw(ctx){*

*switch (true){*

*case(this.move\_x === 0 && this.move\_y === -1):*

*spriteManager.drawSprite(ctx, "tank\_kt\_up", this.pos\_x, this.pos\_y);*

*this.last\_position = "up";*

*break;*

*case (this.move\_x === 0 && this.move\_y === 1):*

*spriteManager.drawSprite(ctx, "tank\_kt\_down", this.pos\_x, this.pos\_y);*

*this.last\_position = "down";*

*break;*

*case (this.move\_x === 1 && this.move\_y === 0):*

*spriteManager.drawSprite(ctx, "tank\_kt\_right", this.pos\_x, this.pos\_y);*

*this.last\_position = "right";*

*break;*

*case (this.move\_x === -1 && this.move\_y === 0):*

*spriteManager.drawSprite(ctx, "tank\_kt\_left", this.pos\_x, this.pos\_y);*

*this.last\_position = "left";*

*break;*

*default:*

*switch (this.last\_position){*

*case ("up"):*

*spriteManager.drawSprite(ctx, "tank\_kt\_up", this.pos\_x, this.pos\_y);*

*this.last\_position = "up"*

*break;*

*case ("down"):*

*spriteManager.drawSprite(ctx, "tank\_kt\_down", this.pos\_x, this.pos\_y);*

*this.last\_position = "down"*

*break;*

*case ("right"):*

*spriteManager.drawSprite(ctx, "tank\_kt\_right", this.pos\_x, this.pos\_y);*

*this.last\_position = "right"*

*break;*

*case ("left"):*

*spriteManager.drawSprite(ctx, "tank\_kt\_left", this.pos\_x, this.pos\_y);*

*this.last\_position = "left"*

*break;*

*default:*

*spriteManager.drawSprite(ctx, "tank\_kt\_right", this.pos\_x, this.pos\_y);*

*this.last\_position = "right"*

*break;*

*}*

*}*

*}*

*update(){*

*physicManager.update(this)*

*}*

*onTouchEntity(obj){*

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

*this.cash += 1;*

*obj.kill()*

*}*

*}*

*kill(){*

*this.lifetime = 0;*

*gameManager.kill(this);*

*gameManager.end();*

*}*

*fire(){*

*let r = new Rocket();*

*r.size\_x = 64;*

*r.size\_y = 64;*

*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 + this.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 + this.size\_y;*

*break;*

*default: return;*

*}*

*if(soundManager.loaded && soundManager.isPlaying)*

*soundManager.play('../public/fire.mp3', { volume: 0.2, looping: false });*

*gameManager.entities.push(r);*

*}*

*}*

*export class Tank extends Entity{*

*constructor() {*

*super();*

*this.lifetime = 100;*

*this.move\_x = 0;*

*this.move\_y = 0;*

*this.speed = 20;*

*this.last\_position = "";*

*this.LastPosition = {x: 0, y: 0};*

*this.lastPlayerPosX = 1;*

*this.lastPlayerPosY = 1;*

*this.canShoot = true;*

*}*

*draw(ctx){*

*switch (true){*

*case(this.move\_x === 0 && this.move\_y === -1):*

*spriteManager.drawSprite(ctx, "tank\_ter\_up", this.pos\_x, this.pos\_y);*

*this.last\_position = "up";*

*break;*

*case (this.move\_x === 0 && this.move\_y === 1):*

*spriteManager.drawSprite(ctx, "tank\_ter\_down", this.pos\_x, this.pos\_y);*

*this.last\_position = "down";*

*break;*

*case (this.move\_x === 1 && this.move\_y === 0):*

*spriteManager.drawSprite(ctx, "tank\_ter\_right", this.pos\_x, this.pos\_y);*

*this.last\_position = "right";*

*break;*

*case (this.move\_x === -1 && this.move\_y === 0):*

*spriteManager.drawSprite(ctx, "tank\_ter\_left", this.pos\_x, this.pos\_y);*

*this.last\_position = "left";*

*break;*

*default:*

*switch (this.last\_position){*

*case ("up"):*

*spriteManager.drawSprite(ctx, "tank\_ter\_up", this.pos\_x, this.pos\_y);*

*this.last\_position = "up"*

*break;*

*case ("down"):*

*spriteManager.drawSprite(ctx, "tank\_ter\_down", this.pos\_x, this.pos\_y);*

*this.last\_position = "down"*

*break;*

*case ("right"):*

*spriteManager.drawSprite(ctx, "tank\_ter\_right", this.pos\_x, this.pos\_y);*

*this.last\_position = "right"*

*break;*

*case ("left"):*

*spriteManager.drawSprite(ctx, "tank\_ter\_left", this.pos\_x, this.pos\_y);*

*this.last\_position = "left"*

*break;*

*default:*

*spriteManager.drawSprite(ctx, "tank\_ter\_up", this.pos\_x, this.pos\_y);*

*this.last\_position = "up"*

*break;*

*}*

*}*

*}*

*update(){*

*physicManager.update(this)*

*}*

*onTouchEntity(obj){*

*if(!obj.name.startsWith("coin")){*

*obj.kill()*

*}*

*}*

*kill(){*

*this.lifetime = 0;*

*gameManager.kill(this);*

*physicManager.addNewEnemy = true;*

*}*

*async Attack() {*

*let playerX = physicManager.player\_x;*

*let playerY = physicManager.player\_y;*

*let ePX = Math.floor(this.pos\_x / 128);*

*let ePY = Math.floor(this.pos\_y / 128);*

*let res = await physicManager.findPlayer(playerX, playerY, ePX, ePY);*

*if (res === -1) {*

*physicManager.playerHasMoved = true;*

*return;*

*}*

*let path = res.map(coords => {*

*const [x, y] = coords.split('x').map(Number);*

*return {x, y};*

*});*

*for (const nextStep of path) {*

*const dx = nextStep.x - Math.floor(Math.round(this.pos\_x / 128));*

*const dy = nextStep.y - Math.floor(Math.round(this.pos\_y / 128));*

*if ((dx !== 0 && dy !== 0) || (dx > 1) || (dx < -1) || dy > 1 || dy < -1) {*

*this.move\_x = 0;*

*this.move\_y = 0;*

*physicManager.playerHasMoved = true;*

*}*

*this.move\_x = dx;*

*this.move\_y = dy;*

*this.fire();*

*while (Math.abs((this.pos\_x) - nextStep.x \* 128 - 32) > 10 || Math.abs((this.pos\_y) - nextStep.y \* 128 - 32) > 10) {*

*if (this.lifetime <= 0) return;*

*await new Promise(resolve => setTimeout(resolve, 1));*

*}*

*this.move\_x = 0;*

*this.move\_y = 0;*

*if (physicManager.player\_x !== this.lastPlayerPosX || physicManager.player\_y !== this.lastPlayerPosY) {*

*this.lastPlayerPosX = physicManager.player\_x;*

*this.lastPlayerPosY = physicManager.player\_y;*

*physicManager.playerHasMoved = true;*

*return;*

*}*

*}*

*}*

*fire(){*

*if(this.move\_x!==0 && this.move\_y!==0) return;*

*if(!this.canShoot) return;*

*let r = new Rocket();*

*r.size\_x = 64;*

*r.size\_y = 64;*

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

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

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

*console.log(r.move\_x, r.move\_y)*

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

*case -1:*

*r.pos\_x = this.pos\_x - this.size\_x - r.size\_y / 2;*

*r.pos\_y = this.pos\_y + r.size\_y / 8;*

*break;*

*case 1:*

*r.pos\_x = this.pos\_x + this.size\_x + r.size\_y / 2;*

*r.pos\_y = this.pos\_y + r.size\_y / 2 - r.size\_y / 2;*

*break;*

*case -2:*

*r.pos\_x = this.pos\_x + r.size\_x / 8;*

*r.pos\_y = this.pos\_y - r.size\_y - r.size\_y / 2;*

*break;*

*case 2:*

*console.log("down")*

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

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

*break;*

*default:*

*return;*

*}*

*if(soundManager.loaded && soundManager.isPlaying)*

*soundManager.play('../public/fire.mp3', { volume: 0.2, looping: false });*

*gameManager.entities.push(r);*

*this.canShoot = false;*

*setTimeout(() => {*

*this.canShoot = true;*

*}, 1500);*

*}*

*}*

*export class Rocket extends Entity{*

*constructor() {*

*super();*

*this.name = "";*

*this.move\_x = 0;*

*this.move\_y = 0;*

*this.speed = 32;*

*}*

*draw(ctx) {*

*switch (true) {*

*case(this.move\_x === 0 && this.move\_y === -1):*

*spriteManager.drawSprite(ctx, "rocket\_up", this.pos\_x, this.pos\_y);*

*this.last\_position = "up";*

*break;*

*case (this.move\_x === 0 && this.move\_y === 1):*

*spriteManager.drawSprite(ctx, "rocket\_down", this.pos\_x, this.pos\_y);*

*this.last\_position = "down";*

*break;*

*case (this.move\_x === 1 && this.move\_y === 0):*

*spriteManager.drawSprite(ctx, "rocket\_right", this.pos\_x, this.pos\_y);*

*this.last\_position = "right";*

*break;*

*case (this.move\_x === -1 && this.move\_y === 0):*

*spriteManager.drawSprite(ctx, "rocket\_left", this.pos\_x, this.pos\_y);*

*this.last\_position = "left";*

*break;*

*}*

*}*

*update(){*

*physicManager.update(this);*

*}*

*onTouchEntity(obj){*

*if(obj.name.match(/enemy/) ||*

*obj.name.match(/player/)*

*){*

*obj.kill()*

*}*

*this.kill();*

*}*

*onTouchMap(){*

*this.kill();*

*}*

*kill(){*

*gameManager.kill(this);*

*}*

*}*

*export class Bonus extends Entity{*

*draw(ctx){*

*spriteManager.drawSprite(ctx, "coin", this.pos\_x, this.pos\_y);*

*}*

*kill(){*

*if(soundManager.loaded && soundManager.isPlaying)*

*soundManager.play('../public/bonus.mp3', { volume: 0.2, looping: false });*

*gameManager.kill(this);*

*}*

*}*

1. **EventsManager.js**

*export class EventsManager{*

*constructor() {*

*this.bind = [];*

*this.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) => {*

*this.action["fire"] = true;*

*}*

*onMouseUp = (event) => {*

*this.action["fire"] = false;*

*}*

*onKeyDown = (event) =>{*

*let action = this.bind[event.keyCode];*

*if (action)*

*this.action[action] = true;*

*}*

*onKeyUp = (event) =>{*

*let action = this.bind[event.keyCode];*

*if (action)*

*this.action[action] = false;*

*}*

*}*

1. **GameManager.js**

*import {Bonus, Player, Rocket, Tank} from "./Entity.js";*

*import {*

*recorder,*

*ctx,*

*canvas,*

*eventsManager,*

*mapManager,*

*spriteManager,*

*gameManager,*

*physicManager*

*} from "./main.js"*

*export class GameManager{*

*constructor() {*

*this.factory = {};*

*this.entities = [];*

*this.fireNum = 0;*

*this.player = null;*

*this.laterKill = [];*

*this.time = "";*

*this.mapNum = ""*

*}*

*initPlayer(obj) {*

*this.player = obj;*

*}*

*end(){*

*const timerElement = document.getElementById('timer');*

*this.time = timerElement.textContent;*

*if(this.player.lifetime === 0){*

*recorder.record.score = "0:0:0:0"*

*}*

*else{*

*recorder.record.score = this.time;*

*}*

*recorder.record.map = this.mapNum;*

*recorder.writeResults()*

*window.location.href = "http://localhost:3000/end";*

*}*

*kill(obj){*

*this.laterKill.push(obj);*

*}*

*update(){*

*let cnt = 1;*

*if(this.player === null)*

*return;*

*if(this.player.cash === 1){*

*this.end();*

*return;*

*}*

*this.player.move\_x = 0;*

*this.player.move\_y = 0;*

*if(eventsManager.action["down"] && eventsManager.action["right"] ||*

*eventsManager.action["down"] && eventsManager.action["left"] ||*

*eventsManager.action["up"] && eventsManager.action["right"] ||*

*eventsManager.action["up"] && eventsManager.action["left"]) cnt++*

*if(cnt === 1){*

*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(function(e) {*

*try {*

*if(e.name.startsWith("enemy")) {*

*// console.log(e.name, e.type)*

*if (physicManager.playerHasMoved) {*

*physicManager.playerHasMoved = false;*

*e.Attack();*

*}*

*}*

*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.draw(ctx);*

*mapManager.centerAt(this.player.pos\_x, this.player.pos\_y);*

*this.draw(ctx);*

*if(physicManager.addNewEnemy){*

*mapManager.createEnemy();*

*physicManager.addNewEnemy = false;*

*physicManager.playerHasMoved = true;*

*}*

*}*

*draw(ctx){*

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

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

*}*

*}*

*loadAll(jsonPath, mapNum){*

*this.mapNum = mapNum;*

*mapManager.loadMap(jsonPath)*

*.then(() => {*

*spriteManager.loadAtlas("./src/atlas.json", "../public/images/spritesheet.png");*

*this.factory['Player'] = Player;*

*this.factory['Tank'] = Tank;*

*this.factory['Bonus'] = Bonus;*

*this.factory['Rocket'] = Rocket;*

*mapManager.parseEntities();*

*mapManager.draw(ctx);*

*eventsManager.setup(canvas);*

*physicManager.setData(mapManager.mazeMap);*

*})*

*.catch(error => {*

*console.error("Ошибка загрузки карты:", error);*

*});*

*}*

*play(){*

*setInterval(this.updateWorld, 100)*

*}*

*updateWorld(){*

*gameManager.update();*

*}*

*}*

1. **PhysicManager.js**

*import {gameManager, mapManager} from "./main.js";*

*export class PhysicManager {*

*constructor() {*

*this.player\_x = 1;*

*this.player\_y = 1;*

*this.map = [];*

*this.playerHasMoved = true;*

*this.ind = {*

*player : -1,*

*wall : [233, 238, 239, 259, 299, 293, 220, 258, 278],*

*noth: 1,*

*enemy: -2*

*}*

*this.addNewEnemy = false;*

*}*

*setData(map){*

*this.map = map;*

*this.map[this.player\_y][this.player\_x] = this.ind.player;*

*}*

*async findPlayer(playerX, playerY, startX, startY) { //async???*

*let targetX = playerX;*

*let targetY = playerY;*

*if(this.ind.wall.includes(this.map[targetY][targetX])){*

*console.log("игрок в стене");*

*return -1;*

*}*

*let start = {col:startX, row:startY};*

*let startKey=`${startX}x${startY}`;*

*const q = [];*

*const parentForCell = {};*

*q.push(start);*

*while(q.length>0){*

*const {col, row} = q.shift();*

*const curKey = `${col}x${row}`;*

*const neighbors = [*

*{col, row: row-1 },*

*{col: col+1,row},*

*{col, row: row+1},*

*{col: col-1, row},*

*]*

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

*const nRow = neighbors[i].row;*

*const nCol = neighbors[i].col;*

*if(nRow<0 || nRow > this.map.length-1){*

*continue;*

*}*

*if(nCol<0 || nCol > this.map[nRow].length-1){*

*continue;*

*}*

*if(this.ind.wall.includes(this.map[nRow][nCol])){*

*continue;*

*}*

*const key = `${nCol}x${nRow}`;*

*if(key in parentForCell) {*

*continue;*

*}*

*parentForCell[key] = {*

*key: curKey,*

*}*

*q.push(neighbors[i]);*

*}*

*}*

*const path = [];*

*let currentKey = `${targetX}x${targetY}`;*

*while(currentKey !== startKey){*

*path.push(currentKey);*

*const {key} = parentForCell[currentKey];*

*currentKey = key;*

*}*

*// path.push(currentKey);*

*path.reverse();*

*return path;*

*}*

*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 / 2, newY + obj.size\_y / 2);*

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

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

*obj.onTouchEntity(e);*

*if(ts !== 219 && obj.onTouchMap){*

*obj.onTouchMap(ts);*

*}*

*if(ts === 219 && e === null){*

*obj.pos\_x = newX;*

*obj.pos\_y = newY;*

*if(obj.name==='player' && (( Math.floor(obj.pos\_x/128)!==this.player\_x) || ( Math.floor(obj.pos\_y/128)!==this.player\_y))){*

*let X = [Math.round(newX/128), Math.ceil(newX/128), Math.floor(newX/128)]*

*let Y = [Math.round(newY/128), Math.ceil(newY/128), Math.floor(newY/128)]*

*for(let y of Y){*

*for(let x of X){*

*if(!this.ind.wall.includes(this.map[y][x])){*

*this.player\_x = x;*

*this.player\_y = y;*

*}*

*}*

*}*

*}*

*}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 < e.pos\_x ||*

*y + obj.size\_y < e.pos\_y ||*

*x > e.pos\_x + e.size\_x ||*

*y > e.pos\_y + e.size\_y)*

*continue;*

*return e;*

*}*

*}*

*return null;*

*}*

*}*

1. **Recorder.js**

*export default class Score {*

*record = {*

*place: 0,*

*score: "",*

*name: '',*

*map: ""*

*}*

*constructor(name, score, place) {*

*this.record.name = name;*

*if (!this.record.name){*

*this.record.name = 'Player';*

*}*

*this.record.score = score;*

*this.record.place = place;*

*}*

*writeResults() {*

*let score\_table;*

*if(localStorage['cw.lvl'] === "1")*

*score\_table = JSON.parse(localStorage.getItem('таблицаРекордов1')) || [];*

*else*

*score\_table = JSON.parse(localStorage.getItem('таблицаРекордов2')) || [];*

*score\_table.push(this.record);*

*score\_table.sort((a, b) => {*

*const timePartsA = a.score.split(':');*

*const timePartsB = b.score.split(':');*

*// в миллисекунды*

*const timeInMillisecondsA =*

*parseInt(timePartsA[1]) \* 3600000 +*

*parseInt(timePartsA[2]) \* 60000 +*

*parseInt(timePartsA[3]) \* 1000 +*

*parseInt(timePartsA[4])*

*const timeInMillisecondsB =*

*parseInt(timePartsB[1]) \* 3600000 +*

*parseInt(timePartsB[2]) \* 60000 +*

*parseInt(timePartsB[3]) \* 1000 +*

*parseInt(timePartsB[4])*

*// Сравниваем значения времени в миллисекундах*

*console.log(timePartsA, timePartsB)*

*return timeInMillisecondsA - timeInMillisecondsB;*

*});*

*for(let i = 0; i < score\_table.length; i++){*

*score\_table[i].place = i + 1;*

*}*

*if(localStorage['cw.lvl'] === "1")*

*localStorage.setItem('таблицаРекордов1', JSON.stringify(score\_table));*

*else*

*localStorage.setItem('таблицаРекордов2', JSON.stringify(score\_table));*

*}*

*}*

1. **SoundManager.js**

*export class SoundManager{*

*constructor() {*

*this.clips= {};*

*this.context = null;*

*this.gainNode = null;*

*this.loaded = false;*

*this.isPlaying = false*

*}*

*init() {*

*this.context = new (window.AudioContext || window.webkitAudioContext)();*

*this.gainNode = this.context.createGain();*

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

*this.context.resume().then(() => {*

*this.loaded = true;*

*});*

*}*

*load(path, callback){*

*if(this.clips[path]){*

*callback(this.clips[path]);*

*return;*

*}*

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

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

*this.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'; // Corrected line*

*request.onload = () => {*

*this.context.decodeAudioData(*

*request.response,*

*buffer => {*

*clip.buffer = buffer;*

*clip.loaded = true;*

*callback(clip);*

*},*

*error => {*

*console.error('Error decoding audio data:', error);*

*}*

*);*

*};*

*request.send();*

*}*

*loadArray(array, callback) {*

*let loadedClips = 0;*

*array.forEach((path) => {*

*if (this.clips[path]) {*

*loadedClips++;*

*if (loadedClips === array.length && callback) {*

*callback();*

*}*

*return;*

*}*

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

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

*this.play(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 = () => {*

*this.context.decodeAudioData(*

*request.response,*

*(buffer) => {*

*clip.buffer = buffer;*

*clip.loaded = true;*

*loadedClips++;*

*if (loadedClips === array.length && callback) {*

*callback();*

*}*

*},*

*(error) => {*

*console.error('Error decoding audio data', error);*

*}*

*);*

*};*

*this.isPlaying = true;*

*request.send();*

*});*

*}*

*play(path, settings){*

*if(!this.loaded){*

*setTimeout(()=>{*

*this.play(path, settings);*

*}, 1000);*

*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 = this.context.createBufferSource();*

*sound.buffer = sd.buffer;*

*sound.connect(this.gainNode);*

*sound.loop = looping;*

*this.gainNode.gain.value = volume;*

*sound.start(0);*

*return true;*

*}*

*toggleMute(){*

*const context = this.context;*

*if (context.state === 'running') {*

*context.suspend().then(() => {*

*this.isPlaying = false;*

*});*

*} else if (context.state === 'suspended') {*

*context.resume().then(() => {*

*this.isPlaying = true;*

*});*

*}*

*}*

*stopAll(){*

*this.gainNode.disconnect();*

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

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

*}*

*}*

1. **Sprites.js**

*import {mapManager} from "./main.js";*

*export class SpriteManager{*

*constructor() {*

*this.image = new Image();*

*this.sprites = [];*

*this.imgLoaded = false;*

*this.jsonLoaded = false;*

*}*

*parseAtlas(atlasJSON) {*

*const atlas = JSON.parse(atlasJSON);*

*for (let name in atlas.frames) {*

*let frame = atlas.frames[name].frame;*

*this.sprites.push({*

*name: name,*

*x: frame.x,*

*y: frame.y,*

*w: frame.w,*

*h: frame.h*

*});*

*}*

*this.jsonLoaded = true;*

*}*

*loadAtlas(atlasJson, atlasImg) {*

*const request = new XMLHttpRequest();*

*request.onreadystatechange = () => {*

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

*this.parseAtlas(request.responseText);*

*}*

*};*

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

*request.send();*

*this.loadImg(atlasImg);*

*}*

*loadImg(imgName) {*

*this.image.onload = () => {*

*this.imgLoaded = true;*

*};*

*this.image.src = imgName;*

*}*

*drawSprite(ctx, name, x, y) {*

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

*setTimeout(() => {*

*this.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(*

*this.image,*

*sprite.x,*

*sprite.y,*

*sprite.w,*

*sprite.h, x,*

*y, sprite.w,*

*sprite.h*

*);*

*}*

*}*

*getSprite(name) {*

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

*let s = this.sprites[i];*

*if (s.name === name)*

*return s;*

*}*

*return null;*

*}*

*}*

1. **Register.js**

*import {SoundManager} from "./SoundManager.js"*

*export const soundManager = new SoundManager();*

*const loadMusic = () => {*

*const musicPaths = ['../public/back.mp3', '../public/fire.mp3', '../public/bonus.mp3'*

*,'../public/end.mp3', '../public/start.mp3'];*

*soundManager.loadArray(musicPaths, () => {});*

*}*

*document.getElementById("Enter").addEventListener("click", function() {*

*soundManager.init();*

*loadMusic();*

*soundManager.play('../public/start.mp3', { volume: 0.2, looping: false });*

*});*

1. **Main.js**

*import { MapManager } from "./MapManager.js"*

*import { SpriteManager } from "./Sprites.js"*

*import {GameManager} from "./GameManager.js";*

*import {EventsManager} from "./EventsManager.js";*

*import {PhysicManager} from "./PhysicManager.js";*

*import {SoundManager} from "./SoundManager.js"*

*import Score from "./recorder.js";*

*const loadMusic = () => {*

*const musicPaths = ['../public/back.mp3', '../public/fire.mp3', '../public/bonus.mp3'*

*,'../public/end.mp3', '../public/start.mp3'];*

*soundManager.loadArray(musicPaths, () => {*

*if(soundManager.loaded)soundManager.play('../public/back.mp3', { volume: 0.2, looping: true });*

*});*

*}*

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

*export var ctx = canvas.getContext("2d");*

*canvas.width = 2560;*

*canvas.height = 2560;*

*export const mapManager = new MapManager()*

*export const spriteManager = new SpriteManager()*

*export const gameManager = new GameManager();*

*export const eventsManager = new EventsManager();*

*export const physicManager = new PhysicManager()*

*export const soundManager = new SoundManager();*

*export const recorder = new Score(localStorage['cw.user\_name'], "", 0);*

*if(localStorage['cw.lvl'] === '1'){*

*soundManager.init();*

*loadMusic();*

*gameManager.loadAll("./src/tilemap1.json", "1 map");*

*gameManager.play();*

*}*

*else if(localStorage['cw.lvl'] === '2'){*

*soundManager.init();*

*loadMusic();*

*gameManager.loadAll("./src/tilemap2.json", "2 map");*

*gameManager.play();*

*}*