МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по курсовой работе

по дисциплине «WEB-технологии»

ТЕМА: СОЗДАНИЕ ИГРЫ НА ЯЗЫКЕ JAVASCRIPT

Студент гр. 8304 _____ Сергеев А. Д.

Преподаватель _____ Беляев С. А.

Санкт-Петербург

2020

Оглавление

Цель работы:

Разработать игру на языке JavaScript.

Выполнение работы:

На html элементе для рисования при помощи JavaScript прорисовываются кадры игры, которые в свою очередь строятся относительно игрового состояния. Разработанную игру можно разделить на основные блоки:

- Менеджер управления картой
- Менеджер управления физикой игры
- Менеджер событий
- Менеджер звукового сопровождения
- Менеджер спрайтов
- Менеджер игры

Описание менеджеров:

1. **Менеджер управления картой:**

   Загрузка карты из JSON-файла:

```
export async function load_map (path) {
    tiles_layer = null;
    tile_sets = [];
    view = { x: 0, y: 0, w: window.innerWidth, h: window.innerHeight };
    world_speed = 32;
    world_speedup = 0.001;
    world_run = 0;
    world_offset = 0;

    map_data = JSON.parse(await load_file(path));
    x_count = map_data['width'];
    y_count = map_data['height'];
    t_size.x = map_data['tilewidth'];
    t_size.y = map_data['tileheight'];
    tile_size = Math.ceil( x: window.innerHeight / y_count);
    m_size.x = x_count * t_size.x;
    m_size.y = y_count * t_size.y;

    map_size.x = x_count * tile_size;
    map_size.y = y_count * tile_size;

    floor.l = map_size.y - tile_size;
    floor.h = tile_size;

    for (const tile_set of map_data['tilesets']) {
        const img = await load_img(tile_set.image);
        const t = tile_set;
        const ts = {
            firstgid: t.firstgid,
            image: img,
            name: t.name,
            xCount: Math.floor( x: t['imagewidth'] / map_data['tilewidth']),
            yCount: Math.floor( x: t['imageheight'] / map_data['tileheight'])
        };
        tile_sets.push(ts);
    }
}
```

Отрисовка карты:

```
export function draw_map (ctx) {
    if (tiles_layer === null) tiles_layer = map_data['layers'].find(e => { return e.type === 'tilelayer' });
    for (let i = 0; i < tiles_layer.data.length; i++) if (tiles_layer.data[i] !== 0) {
        const tile = get_tile(tiles_layer.data[i]);
        let pX = (i % x_count) * tile_size;
        let pY = Math.floor( x: i / x_count) * tile_size;
        if (!isVisible(pX, pY, tile_size, tile_size)) continue;
        ctx.drawImage(tile.img, tile.px, tile.py, t_size.x, t_size.y, pX, pY, tile_size, tile_size);
    }
}
```

2. Менеджер спрайтов

Парсинг:

```
export async function load_atlas (atlasJson, atlasImg) {
    sprites = [];
    image = await load_img(atlasImg);

    const atlas = JSON.parse(await load_file(atlasJson));
    for (const frame of atlas)
        sprites.push({
            name: frame.name,
            x: frame.x,
            y: frame.y,
            w: frame.width,
            h: frame.height
        });
}
```

Отрисовка:

```
export function drawSprite (ctx, name, x, y, w, h, curFrame) {
    if (!isVisible(x, y, w, h)) return;
    const sprite = sprites.find(e => { return e.name === name; });
    ctx.drawImage(image, sprite.x + curFrame * t_size.x, sprite.y, t_size.x, t_size.y, x, y, w, h);
}

export function drawSpriteRotated (ctx, name, x, y, w, h, curFrame, angle) {
    if (!isVisible(x, y, w, h)) return;
    const sprite = sprites.find(e => { return e.name === name; });
    const tx = x + w / 2;
    const ty = y + h / 2;
    ctx.translate(tx, ty);
    ctx.rotate(angle);
    ctx.drawImage(image, sprite.x + curFrame * t_size.x, sprite.y, t_size.x, t_size.y * (h / tile_size),
        -w / 2, -h / 2, w, h);
    ctx.rotate(-angle);
    ctx.translate(-tx, -ty);
}
```

3. Менеджер событий:

```
// declarations
const immediates = ['ArrowUp', 'ArrowDown', 'w', 's'];
export let callback = null;
export let escaped = false;



// initiator
export function init_events () {
    callback = undefined;
    escaped = false;
    document.body.addEventListener( type: "keydown", onKeyDown);
}



// setters
export function set_callback(func) { callback = func; }
export function escape () { escaped = true; }
```

4. **Менеджер физики игры:**

Обновление состояния игрока:

```
export function update_player (player) {
    if (player.flying) { // 100 times slower than real
        player.fly += 0.015;
        const new_speed = player.y_speed_prev + player.y_speedup * player.fly;
        const new_coord = player.pos_y + gravity * new_speed * 0.015 * dpc_y;

        if (new_coord > floor.l - player.size_y) {
            player.pos_y = floor.l - player.size_y;
            if (gravity === 1) player.flying = false;
            player.y_speed = player.y_speed_prev = player.fly = 0;
        } else if (new_coord < floor.h) {
            player.pos_y = floor.h;
            if (gravity === -1) player.flying = false;
            player.y_speed = player.y_speed_prev = player.fly = 0;
        } else {
            player.y_speed = new_speed;
            player.pos_y = new_coord;
        }
    }

    player.y_speedup += 0.0001;
}
```

Взаимодействие с врагами:

```
export function entityAtXY (obj, x, y, any) {
    for (const entity of entities)
        if (((entity.name !== obj.name) || any) && (!(x + obj.size_x < entity.pos_x ||
            y + obj.size_y < entity.pos_y || x > entity.pos_x + entity.size_x || y > entity.pos_y + entity.size_y)))
            return entity;
    return null;
}

export function inColumn (obj) {
    for (const entity of entities)
        if (!(obj.pos_x + obj.size_x < entity.pos_x || obj.pos_x > entity.pos_x + entity.size_x))
            return entity;
    return null;
}
```

5. Менеджер звукового сопровождения:

```
// declarations
let clips = {};
let audio = null;



// initiators
export function load_audios (array) {
    clips = {};
    audio = null;
    for (const item of array) clips[item] = new Audio( src: "files/" + item);
}



// functions
export function play (sound, repet, volume) {
    if (audio) audio.pause();
    audio = clips[sound];
    audio.play().catch(_ => {});
    audio.onended = function () { if (repet) play(sound, repet, volume); };
}

export function stop () {
    if (audio) audio.pause();
}
```

6. Игровой цикл:

```
function update (ctx) {
    update_world(ctx);
    for (const entity of entities) entity.update();

    if (escaped) {
        end_game(ctx);
        return;
    }

    emit_wall();
    draw_map(ctx);
    for (const entity of entities) entity.draw(ctx);
    print_score_and_prompts(ctx);
}
```

Карта игры:

Используются карты, созданные независимо от программного кода и сохраненные в формате JSON. Карты создавалась в редакторе TiledMapEditor.

Уровень 1:

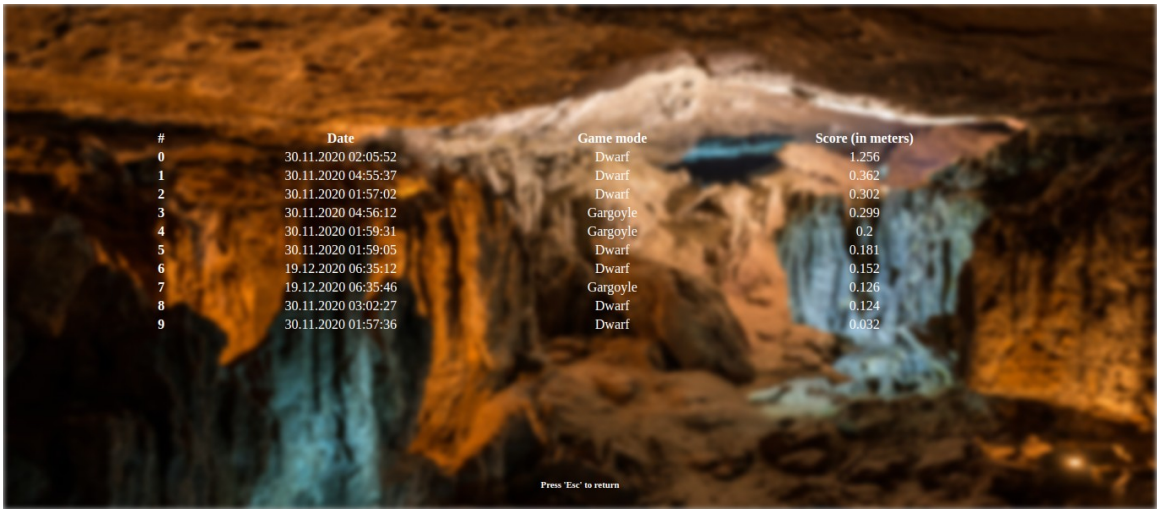
Уровень 2:

Тестрирование игры:

- Начальное окно игры

- Отображение количества набранных очков во время игры

- Обновление таблицы рекордов



#	Date	Game mode	Score (in meters)
0	30.11.2020 02:05:52	Dwarf	1.256
1	30.11.2020 04:55:37	Dwarf	0.362
2	30.11.2020 01:57:02	Dwarf	0.302
3	30.11.2020 04:56:12	Gargoyle	0.299
4	30.11.2020 01:59:31	Gargoyle	0.2
5	30.11.2020 01:59:05	Dwarf	0.181
6	19.12.2020 06:35:12	Dwarf	0.152
7	19.12.2020 06:35:46	Gargoyle	0.126
8	30.11.2020 03:02:27	Dwarf	0.124
9	30.11.2020 01:57:36	Dwarf	0.032

Вывод:

Таким образом была реализована игра при помощи JavaScript с использованием архитекруры, основанной на разделении функций и областей ответственности между различными менеджерами.