

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

Холковский К.В.

Преподаватель

Беляев С.А.

Санкт-Петербург

2020

Оглавление

Цель работы:	3
Выполнение работы:	3
Описание менеджеров:	3
1. Менеджер управления картой:	3
2. Менеджер спрайтов	5
3. Менеджер событий:	6
4. Менеджер физики игры:	7
5. Менеджер звукового сопровождения:	8
6. Игровой цикл:	9
Карта игры:	12
Тестирование игры:	12

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:

```
loadMap(path) {  
    let request = new XMLHttpRequest();  
    request.onreadystatechange = function(){  
        if(request.readyState === 4 && request.status === 200)  
        {  
            mapManager.parseMap(request.responseText);  
        }  
    };  
    request.open( method: "GET", path, async: true);  
    request.send();  
},
```

Парсинг карты:

```
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++) {
    const img = new Image();
    img.onload = () => {
      this.imgLoadCount += 1;
      if (this.imgLoadCount === this.mapData.tilesets.length) {
        this.imgLoaded = true;
      }
    };
    img.src = this.mapData.tilesets[i].image;
    const t = this.mapData.tilesets[i];
    const ts = {
      firstgid: t.firstgid,
      image: img,
      name: t.name,
      xCount: Math.floor(x: t.imagewidth / this.tSize.x),
      yCount: Math.floor(x: t.imageheight / this.tSize.y)
    };
    this.tilesets.push(ts);
  }
  this.jsonLoaded = true;
},
```

Отрисовка карты:

```
draw(ctx) {
  if (!this.imgLoaded || !this.jsonLoaded) {
    setTimeout( handler: function () {
      mapManager.draw(ctx);
    }, timeout: 100);
  }
  else {
    if (this.tLayer === null)
      for (let id = 0; id < this.mapData.layers.length; id++) {
        const 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) {
        const tile = this.getTile(this.tLayer.data[i]);
        let pX = (i % this.xCount) * this.tSize.x;
        let pY = Math.floor( x: i / this.xCount) * this.tSize.y;
        ctx.drawImage(tile.img, tile.px, tile.py,
          this.tSize.x, this.tSize.y, pX, pY, this.tSize.x, this.tSize.y)
      }
    }
  }
},
```

2. Менеджер спрайтов

Парсинг:

```
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;
},
```

Отрисовка:

```
drawSprite(ctx, name, x, y)
{
  if(!this.imgLoaded || !this.jsonLoaded)
  {
    setTimeout( handler: function(){spriteManager.drawSprite(ctx, name, x, y);}, timeout: 100);
  }
  else
  {
    const sprite = this.getSprite(name);
    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);
  }
},
```

3. Менеджер событий:

```
let eventsManager = {
  bind: [],
  action: [],
  setup: function(){
    this.bind[37] = 'left';
    this.bind[39] = 'right';
    this.bind[38] = 'up';
    this.bind[40] = 'down';
    this.bind[32] = 'space';

    document.body.addEventListener( type: "keydown", listener: e =>this.onKeyDown(e));
    document.body.addEventListener( type: "keyup", listener: e => this.onKeyUp(e));
  },
  onKeyDown: function(event) {
    const action = eventsManager.bind[event.keyCode];
    if (action && gameManager.started)
      eventsManager.action[action] = true;
  },
  onKeyUp: function(event){
    const action = eventsManager.bind[event.keyCode];
    if (action)
      eventsManager.action[action] = false;
  },
};
```

4. Менеджер физики игры:

Координирование и взаимодействие с препятствиями:

```
let physicManager = {
  update: function (obj){
    var newX = obj.pos_x + obj.move_x * obj.speed;
    var newY = obj.pos_y + obj.move_y * obj.speed;

    var tileIdx = Math.max(mapManager.getTilesetIdIdx(newX, newY),
      mapManager.getTilesetIdIdx(x: newX + obj.size_x, newY),
      mapManager.getTilesetIdIdx(newX, y: newY + obj.size_y),
      mapManager.getTilesetIdIdx(x: newX + obj.size_x, y: newY + obj.size_y));

    var entAtXY = this.entityAtXY(obj, newX, newY);

    if(tileIdx !== 1 && obj.onTouchMap) {
      obj.onTouchMap(tileIdx);
      return;
    }
    //console.log(obj.name + ", "+ entAtXY);
    if(entAtXY !== null && obj.onTouchEntity)
      obj.onTouchEntity(entAtXY);

    if (tileIdx === 1 && entAtXY === null) {
      obj.pos_x = newX;
      obj.pos_y = newY;
    }
    else
      return;
    return;
  },
}
```

Взаимодействие с врагами:

```
entityAtXY: function(obj, x, y) {
    for(let i = 0; i < gameManager.entities.length; i++)
    {
        const 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;
};
```

5. Менеджер звукового сопровождения:

Звуковые файлы загружаются при открытии загрузки скрипта и проигрываются при нужном событии. Звук загружается следующим образом:

```
load: function(path, callback){
    if(this.clips[path]){
        callback(this.clips[path]);
        return;
    }
    const clip = {path: path, buffer: null, loaded: false};
    clip.play = function(volume, loop) {
        soundManager.play(this.path, {settings: {looping: loop ? loop : false, volume: volume ? volume : 1}});
    };
    this.clips[path] = clip;
    let request = new XMLHttpRequest();
    request.open( method: 'GET', path, async: true);
    request.responseType = 'arraybuffer';
    request.onload = function(){
        soundManager.context.decodeAudioData(request.response, {successCallback: function(buffer : AudioBuffer) {
            clip.buffer = buffer;
            clip.loaded = true;
            callback(clip);
        }});
    };
    request.send();
},
```


Пример, в случае, если игрок стреляет:

```
if (eventsManager.action["space"] && this.is_bullet !== 0) {  
    let obj = Object.create(this.factory["bullet"]);  
    obj.size_x = 50;  
    obj.size_y = 13;  
    obj.pos_x = this.player.pos_x + obj.size_x;  
    obj.pos_y = this.player.pos_y + obj.size_y + 10;  
    obj.name = "bullet";  
    obj.move_x = 1  
    gameManager.entities.push(obj);  
    soundManager.play( path: "sound/shoot.mp3")  
  
    this.is_bullet--  
}
```

6. Игровой цикл:

```

update()
{
    if(!this.bonus_placed)
    {
        let obj = Object.create(this.factory["Bonus"]);
        obj.pos_x = 404;
        obj.pos_y = 284;
        gameManager.entities.push(obj);
        gameManager.entities.reverse();

        let obj2 = Object.create(this.factory["Bonus"]);
        obj2.pos_x = 364;
        obj2.pos_y = 484;

        gameManager.entities.push(obj2);
        gameManager.entities.reverse();

        let obj3 = Object.create(this.factory["Range"]);
        obj3.pos_x = 364;
        obj3.pos_y = 284;
        gameManager.entities.push(obj3);
        gameManager.entities.reverse();

        let obj4 = Object.create(this.factory["Range"]);
        obj4.pos_x = 404;
        obj4.pos_y = 484;

        gameManager.entities.push(obj4);
        gameManager.entities.reverse();

        this.bonus_placed = true;
    }

    if(this.player === null)
        return;
    this.player.move_x = 0;
    this.player.move_y = 0;
}

```

```

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["space"] && this.is_bullet !== 0) {
        let obj = Object.create(this.factory["bullet"]);
        obj.size_x = 50;
        obj.size_y = 13;
        obj.pos_x = this.player.pos_x + obj.size_x;
        obj.pos_y = this.player.pos_y + obj.size_y + 10;
        obj.name = "bullet";
        obj.move_x = 1
        gameManager.entities.push(obj);
        soundManager.play( path: "sound/shoot.mp3")

        this.is_bullet--
    }

    this.entities.forEach(function (e) {
        e.update()
    })
}

```

```

        for(let i = 0; i < this.laterKill.length; i++) {
            const idx = this.entities.indexOf(this.laterKill[i]);
            if(idx > -1){
                let check = this.entities[idx].name;
                if (check === "bullet")
                    this.is_bullet++
                this.entities.splice(idx, 1);
            }
        }

        let flag = 0
        this.entities.forEach(function (e) {
            if(e.name === "Enemy"){
                flag++
            }
        })

        if(flag === 0 && this.num_maked_enemys === this.lvlNum*15)
            this.onGameOver( result: "win")

        if(this.laterKill.length > 0)
            this.laterKill.length = 0;

        mapManager.draw(ctx);
        this.draw(ctx);
    },

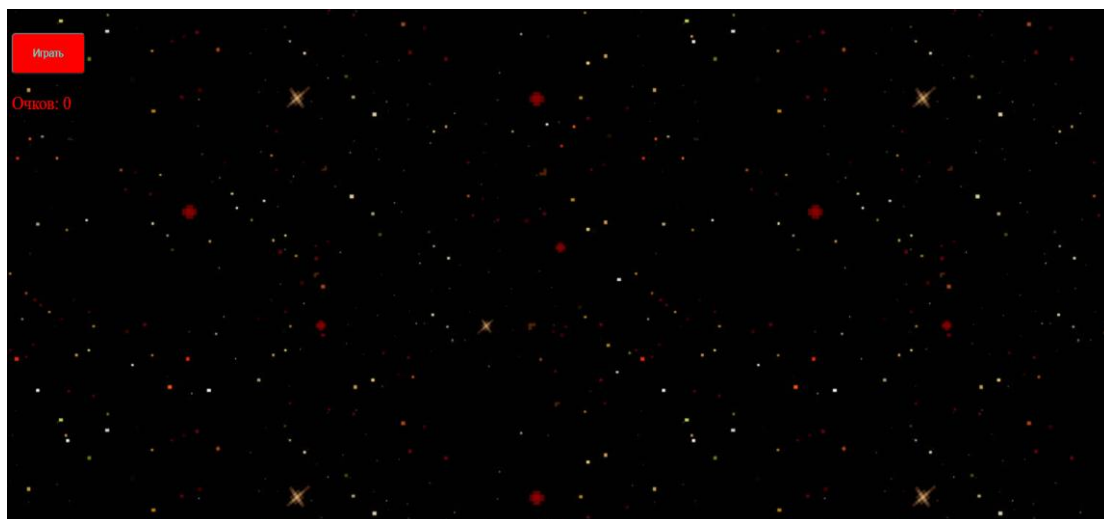
```

Карта игры:

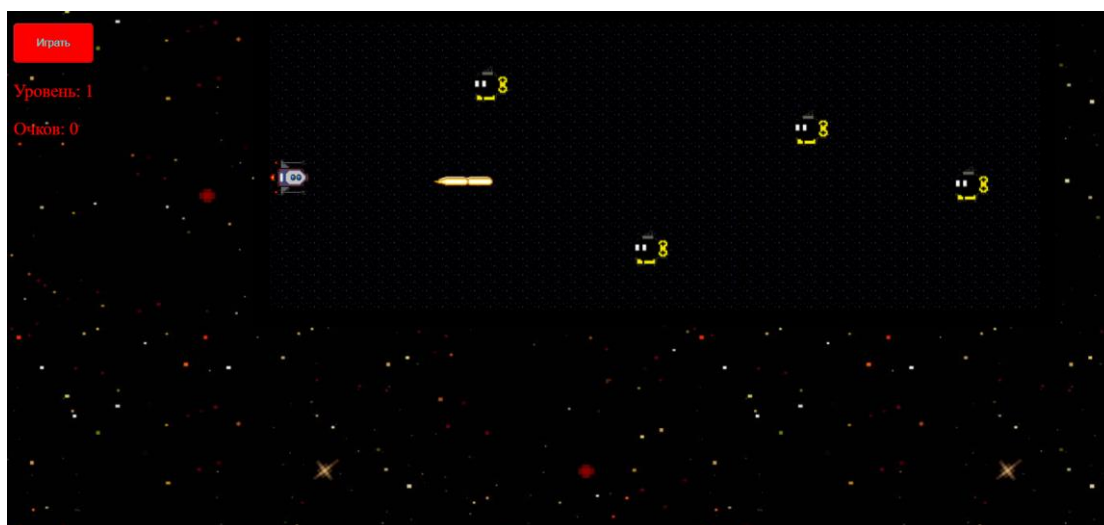
Будем использовать карты, созданные независимо от программного кода и сохраненные в формате JSON. Карта создавалась в TiledMapEditor. Выбирается размер карты, загружается набор тайлов, из которых и строится карта.

Тестирование игры:

- Главное окно игры



- Первый уровень



- Второй уровень



Вывод:

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