МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный электротехнический
университет «ЛЭТИ» им. В.И. Ульянова (Ленина)

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КУРСОВОЙ РАБОТЫ

Учебно-методическое пособие

Санкт-Петербург
2016

УДК 004.432

Методические указания по выполнению курсовой работы: учеб.-метод. пособие / сост.: С.А. Беляев. СПб. 2016. 32 с.

Представлены материалы по дисциплине «Web-технологии». Рассматриваются вопросы разработки веб-приложений с использованием HTML, JavaScript и CSS. Приводятся требования к выполнению курсовой работы.

Предназначено для студентов направлений «Программная инженерия» и «Прикладная математика и информатика».

Одобрено
Методической комиссией факультета компьютерных технологий и информатики
в качестве учебно-методического пособия

ВВЕДЕНИЕ

В учебно-методическом пособии приводятся методические указания по выполнению курсовой работы по дисциплине «Web-технологии».

Информационные технологии (операционные системы, программное обеспечение общего и специализированного назначения, информационные справочные системы) и материально-техническая база, используемые при осуществлении образовательного процесса по дисциплине, соответствуют требованиям федерального государственного образовательного стандарта высшего образования.

Для обеспечения образовательного процесса по дисциплине используются следующие информационные технологии:

1. Операционные системы: Microsoft Windows 10, Microsoft Windows 7, Ubuntu Desktop.

2. Программное обеспечение общего и специализированного назначения: NetBeans IDE, IntelliJ IDEA Community Edition.

3. Информационные справочные системы: международная ассоциация сетей «Интернет», электронные библиотечные системы и ресурсы удаленного доступа библиотеки СПбГЭТУ «ЛЭТИ».

Образовательный процесс обеспечивается на следующей материально-технической базе (один из трех вариантов):

1) персональный компьютер (системный блок RAMEC STORM, монитор LG L1953S, клавиатура, мышь, ИБП APC), персональный компьютер (системный блок RAMEC STORM, монитор LG L1953S, клавиатура, мышь, ИБП APC), сервер (RAMEC, монитор LG L1953S, клавиатура, мышь, ИПБ APC), проектор Mitsubishi XD430U, экран проекционный настенный;

2) Персональный компьютер (системный блок RAMEC STORM, монитор LG L1953S, клавиатура, мышь, ИБП APC), персональный компьютер (системный блок RAMEC STORM, монитор LG L222WS, клавиатура, мышь, ИБП APC), принтер HP Laser Jet, принтер HP Color Laser Jet, сервер SuperMicro, сканер планшетный HP LaserJet, проектор Mitsubishi XD430U, экран проекционный настенный подпружиненный ScreenMedia Goldview 213x213 MW;

3) Персональный компьютер (системный блок Hewlett-Packard, монитор Samsung SyncMaster 913N, клавиатура, мышь), персональный компьютер (си-

стемный блок Hewlett-Packard, монитор Samsung SyncMaster E2220, клавиатура, мышь), проектор Vivitek D555, компьютер-сервер (системный блок Universal KOMPUMIR, монитор Samsung SyncMaster 913N, клавиатура, мышь), кондиционер QuattroClima Industriale QA-RWD.

1. КУРСОВАЯ РАБОТА. РАЗРАБОТКА ИГРЫ НА ЯЗЫКЕ JAVASCRIPT

1.1. Цель и задачи

Целью работы является изучение особенностей построения интерактивных web-приложений, формирование навыков по использованию языка JavaScript, освоение особенностей построения интерактивных приложений на JavaScript.

Для достижения поставленной цели требуется решить следующие задачи:

1) Определиться с жанром игры.

2) Создать или найти в сети Интернет необходимые графические элементы.

3) Создать необходимые карты игры с использованием «тайлов».

4) Создать меню уровней игры.

5) Обеспечить выполнение игры, переход по уровням и отображение результатов.

6) Подготовить отчёт о выполнении курсовой работы.

1.2. Основные теоретические сведения

Основные теоретические сведения приведены в книге Беляева С.А. «Разработка игр на языке JavaScript» (СПб.: Лань, 2016).

1.3. Общая формулировка задачи

Необходимо создать игру с использованием HTML, CSS и JavaScript 2016 без использования дополнительных фреймворков.

1) Создание меню.

2) Создание менеджера управления картой.

3) Создание менеджера объектов.

4) Создание менеджера взаимодействия с пользователем.

5) Создание логики поведения объектов.

6) Создание менеджера игры.

7) Создание менеджера звука.

8) Создание механизма перехода между уровнями и отображения результатов игры.

1.4. Пример выполнения задания

Canvas

Для отображения нашей игры воспользуемся гибким html-элементом (canvas) с размером 1365x658 пикселей

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>SnowGame</title>
    <link rel="stylesheet" href="styleGame.css">
</head>
<body>
<div id="Score"> Очков: </div>
<canvas id="canvas" width="1365"
height="658"></canvas>
<script src="javascriptG.js"></script>
<div id="menu_">
        <div id="wintext"></div>
        <div id="WinScore"></div>
        <a href="htmlG.html" >Попробуйте еще?</a>
</div>
</body>
</html>
```

Игровые менеджеры

Для функционирования игры, разделим js-код на блоки, которые будут реализовывать свои функции, условно назовем этим блоки:

«Менеджер работы с картой»

«Менеджер работы со спрайтами»

«Менеджер объектов»

«Менеджер звука»

«Менеджер физики»

«Менеджер событий»

«Игровой менеджер»

«Менеджер работы с картой»

```
var mapManager = {
    mapData: null, // Информация о карте
    tLayer: null, //Текущий тайл
    xCount: 0,
    yCount: 0,
    imgLoadCount: 0, // количество загруженных
изображений
    imgLoaded: false, // изображения не загружены
    jsonLoaded: false, // json не загружен
    tSize: {x: 50, y: 50}, //Размер тайла
    mapSize: {x: 20000, y: 750}, // Размер карты
    tilesets: [], //Наборы тайлов
    view: {x: 0, y: 0, w: 1365, h: 750}, //Видимая
часть
```

Функции *loadMap* и *parseMap*

Карту будет необхоидмо хранить внутри данного менеджера, для этого будет создано поле mapData.

При помощи функции loadMap мы загрузим карту, а при помощи функции parseMap обработаем её и поместим в наше поле.

```
// ajax-загрузка карты
loadMap: function (path) {
    var request = new XMLHttpRequest();
    request.onreadystatechange = function () {
        if (request.readyState === 4 && re-
quest.status === 200) {
            mapManag-
er.parseMap(request.responseText);
        }
    };
    request.open("GET", path, true);
    request.send();
},
```

```
    //Разбираем карту
    parseMap: function (tilesJSON) {
        this.mapData = JSON.parse(tilesJSON);
//разобрать JSON
        this.xCount = this.mapData.width; // сохранение
ширины
        this.yCount = this.mapData.height; // сохранение
высоты
        this.tSize.x = this.mapData.tilewidth; //
сохранение размера тайла
        this.tSize.y = this.mapData.tileheight; //
сохранение размера тайла
        this.mapSize.x = this.xCount * this.tSize.x; //
вычисление размера карты
        this.mapSize.y = this.yCount * this.tSize.y;
        for (var i = 0; i <
this.mapData.tilesets.length; i++) {
            var img = new Image(); // создаем переменную
для хранения изображений
            img.onload = function () { // при загрузке
изображения
                mapManager.imgLoadCount++;
                if (mapManager.imgLoadCount === mapMan-
ager.mapData.tilesets.length) {
                    mapManager.imgLoaded = true; //
загружены все изображения
                }
            };
            img.src = this.mapData.tilesets[i].image; //
задание пути к изображению
            var t = this.mapData.tilesets[i]; //забираем
tileset из карты
            var ts = { // создаем свой объект tileset
                firstgid: t.firstgid, // с него
начинается нумерация в data
                image: img,
```

```
                name: t.name, // имя элемента рисунка
                xCount: Math.floor(t.imagewidth / mapManager.tSize.x), // горизонталь
                yCount: Math.floor(t.imageheight / mapManager.tSize.y) // вертикаль
            }; // конец объявления ts
            this.tilesets.push(ts); // сохраняем tileset в массив
        } // окончание цикла for
        this.jsonLoaded = true; // когда разобран весь json
    },
```

В качестве карты, будем использовать файл, созданный независимо от программного кода и сохраненные в формате JSON –

Наша карта будет представлять из себя 2D игровое поле, состоящее из тайлов и объектов, описанных в данном файле.

Так же к проекту приложен файл Tiles.png необходимый для отображения тайлов.

Функции *draw, getTile, getTileset, isVisible, getTilesetIdx*

Функция *draw* предназначена для отображения карты на холсте, контекст (ctx) ей передается в качестве параметра.

Отрисовка карты есть совокупность операций отрисовки ее тайлов.

```
    // отрисовка карты в контексте канваса
    draw: function (ctx) {
        // если карта не загружена, то повторить прорисовку через 100 мс
        if (!mapManager.imgLoaded || !mapManager.jsonLoaded) {
            setTimeout(function () {
                mapManager.draw(ctx);
            }, 100);
        } else {
            var layerCount = 0;
            if (this.tLayer === null) {// проверка, что tLayer настроен
```

```
                for (var id = 0; id <
this.mapData.layers.length; id++) {
                    // проходим по всем layer карты
                    var layer = this.mapData.layers[id];
                    if (layer.type === "tilelayer") {
                        this.tLayer = layer;
                        //break;
                    }
                }
            }
            for (var i = 0; i < this.tLayer.data.length;
i++) { // проходим по всей карте
                if (this.tLayer.data[i] !== 0) { // если
данных нет, то пропускаем
                    var tile =
this.getTile(this.tLayer.data[i]); // получение блока
по индексу
                    var pX = (i % this.xCount) *
this.tSize.x; // вычисляем x в пикселях
                    var pY = Math.floor(i / this.xCount)
* this.tSize.y;
                    // не рисуем за пределами видимой
зоны
                    if (!this.isVisible(pX, pY,
this.tSize.x, this.tSize.y))
                        continue;
                    // сдвигаем видимую зону
                    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); //
                    //отрисовка в контексте
                }
            }
        }
```

```
    },
```

Потому создадим еще одну функцию *getTile*, возвращающая ообъект блока по ее индексу в функцию отрисовки карты.

```
    getTile: function (tileIndex) {
        var tile = {
            img: null, // изображение tileset
            px: 0, py: 0 // координаты блока в tileset
        };
        var tileset = this.getTileset(tileIndex);
        tile.img = tileset.image; // изображение
искомого tileset
        var id = tileIndex - tileset.firstgid; // индекс
блока в tileset
        // блок прямоугольный, остаток от деления на
xCount дает x в tileset
        var x = id % tileset.xCount;
        var y = Math.floor(id / tileset.xCount);
        tile.px = x * mapManager.tSize.x;
        tile.py = y * mapManager.tSize.y;
        return tile; // возвращаем тайл для отображения
    },
```

И функции, *getTileset*, *getTilesetIdx* для работы с блоками тайлов

```
    getTileset: function (tileIndex) {
        for (var i = mapManager.tilesets.length - 1; i
>= 0; i--) {
            // в каждом tilesets[i].firstgid записано
число, с которого начинается нумерация блоков
            if (mapManager.tilesets[i].firstgid <=
tileIndex) {
                // если индекс первого блока меньше ,
либо равен искомому, значит этот tileset и нужен
                return mapManager.tilesets[i];
            }
        }
        return null;
    },
```

```
    getTilesetIdx: function (x, y)
    {
        var wX = x;
        var wY = y;
        var idx = Math.floor(wY / this.tSize.y) *
this.xCount + Math.floor(wX / this.tSize.x);  //При
помощи размера тайла и координат находим, его номер
        return this.tLayer.data[idx];
    },
```

Так же для того чтобы отрисовывать не всю карту, а только её видимую часть (созданное поле view) создадим функцию *isVisible* для проверки нахождения объекта в области видимости.

```
    isVisible: function (x, y, width, height) {
        // не рисуем за пределами видимой зоны
        return !(x + width < this.view.x || y + height <
this.y || x > this.view.x + this.view.w || y >
this.view.y + this.view.h);
    },
```

Функция *centerAt.*

Так как игрок будет двигаться, то будем проводить центрирование видимой области относительно игрока, при помощи данной функции.

```
    centerAt: function (x, y) {
        if (x < this.view.w / 2) // Центрирование по
горизонтали
            this.view.x = 0;      //Если персонаж
находится до середины видимой части карты, видимая
часть стоит на месте
        else if (x > this.mapSize.x - this.view.w / 2)
            this.view.x = this.mapSize.x - this.view.w;
//Передвигаем видимую часть
        else
            this.view.x = x - (this.view.w / 2);
    },
```

Функции *parseEnteties*.

Так же файл json хранит в себе информацию о слое объектов, поэтому при помощи данной функции, мы будем его обрабатывать.

```
    //Разбирает слой объектов
    parseEntities: function () {
        if (!mapManager.imgLoaded ||
!mapManager.jsonLoaded) { //Если картинки или карта не
загружена
            setTimeout(function () {
                mapManager.parseEntities(); //Ждем пока
они загружаются
            }, 100);
        } else
            for (var j = 0; j <
this.mapData.layers.length; j++) // просмотр всех слоев
                if (this.mapData.layers[j].type === 'ob-
jectgroup') {
                    var entities =
this.mapData.layers[j]; // слой с объектами следует
разобрать
                    for (var i = 0; i < enti-
ties.objects.length; i++) { //Просматриваем все объекты
                        var e = entities.objects[i];
//Получае объект
                        try {
                            var obj = Ob-
ject.create(gameManager.factory[e.type]); //Создаем
объект нужного типа
                            obj.name = e.name;
//Устанавливаем имя
                            obj.pos_x = e.x;   //  x
                            obj.pos_y = e.y;   //    y
                            obj.size_x = e.width; //
ширину
                            obj.size_y =
e.height;//высоту объекта
```

```
                            if (obj.name === 'Santa') {
//Если объект - Санта
                                obj.dirSprite =
'SantaSprite';
                                obj.curFrame = 0;
                                gameManag-
er.initPlayer(obj);
                            }
                            if (obj.name === 'Gift') {
// Если объект - подарок
                                obj.dirSprite =
'GiftSprite';
                                obj.curFrame = 0;
                            }
                            gameManag-
er.entities.push(obj);
                        } catch (ex) {
                            console.log("Error while
creating: [" + e.gid + "]" + e.type + " " + ex);
                        }
                    }
                }
    },
```

«Менеджер работы со спрайтами»

Данный менеджер используется для отображения объектов.

```
var spriteManager = {
    image: new Image(),
    sprites: [],
    imgLoaded: false,
    jsonLoaded: false,
```

Функции *loadAtlas, loadImg, parseAtlas.*

Для различных анимаций будем использовать один файл с множеством картинок. Этот файл мы будем загружать при помощи функции *loadImg.*

```
    loadImg: function (imgName) {
        this.image.onload = function () {
            spriteManager.imgLoaded = true;
```

```
        };
        this.image.src = imgName;
        this.image.id= imgName;
    },
```

Размещение множества изображений в одном файле позволяет существенно уменьшить время их загрузки в браузер. При этом одновременно с файлом изображения необходимо сформировать описание, в котором будет храниться информация о размещении каждого отдельного изображения. Общее описание при этом принято называть «атлас», а каждое отдельное изображение — «спрайт». Такое описание будет хранится в файле atlas.json

Аналогично менеджеру карт, в менеджере отображения необходимо преобразовать json-текст в javascript-объект. Для этого определим функцию *loadAtlas* для загрузки файла и функцию *parseAtlas* для разбора json файла.

```
    loadAtlas: function (atlasJson, atlasImg) {
        var request = new XMLHttpRequest();
        request.onreadystatechange = function () {
            if (request.readyState === 4 && re-
quest.status === 200) {
                spriteManag-
er.parseAtlas(request.responseText);
            }
        };
        request.open("GET", atlasJson, true);
        request.send();
        this.loadImg(atlasImg);
    },
    parseAtlas: function (atlasJSON) {
        var atlas = JSON.parse(atlasJSON);
        for (var name in atlas.frames) { // проход по
всем именам в frames
            var frame = atlas.frames[name].frame; //
получение спрайта и сохранение в frame
            this.sprites.push({name: name, x: frame.x,
y: frame.y, w: frame.w, h: frame.h}); // сохранение
характеристик frame в виде объекта
        }
```

```
        this.jsonLoaded = true; // атлас разобран
    }
```

Функции, *getSprite, drawSprite.*

Менеджер спрайтов создан для отрисовки изображений, по этому его основными функциями будут – *getSrpite* для получения необходимой информации о спрайте. И *drawSprite* для обрезания нужного кадра и его отрисовки.

```
    drawSprite: function (ctx, name, x, y, curFrame) {
        // если изображение не загружено, то повторить
запрос через 100 мс
        if (!this.imgLoaded || !this.jsonLoaded) set-
Timeout(function () {spriteManager.drawSprite(ctx,
name, x, y, 1);}, 100);
        else
        {
            var sprite = this.getSprite(name); //
получить спрайт по имени
            if (name.match("Santa"))
            {
                if (!mapManager.isVisible(x, y, 86,
111)) return; // не рисуем за пределами видимой зоны
            }
            else
            {
                if (!mapManager.isVisible(x, y,
sprite.w, sprite.h)) return; // не рисуем за пределами
видимой зоны
            }
            x -= mapManager.view.x;
            y -= mapManager.view.y;
            // отображаем спрайт на холсте
            if (name.match(/Santa/))
                ctx.drawImage(this.image, sprite.x +
curFrame * 154, sprite.y, 86, 111, x, y, 86, 111);
            else ctx.drawImage(this.image,  sprite.x +
curFrame * 48, sprite.y, 38, 41, x, y, 38, 41);
        }
```

```
    },
    // получить спрайт по имени
    getSprite: function (name) {
        for (var i = 0; i < this.sprites.length; i++) {
            var s = this.sprites[i];
            if (s.name === name)
                return s;
        }
        return null; // не нашли спрайт
    }
```

«Менеджер объектов»

В игре у нас будут присутствовать 2 типа объектов: Santa, который будет передвигаться по карте и собирать подарки. И Gift – сам подарок, который может собрать Санта.

Entity

Для более удобной работы воспользуемся похожими на наследование (как на объектно-ориентированных языках, таких как Java или C++, C#,), но другими механизмами.

```
    var Entity = {
        pos_x: 0, pos_y: 0, // позиция объекта
        size_x: 0, size_y: 0, // размеры объекта
        extend: function (extendProto) { // расширение
сущности
            var object = Object.create(this); //
создание нового объекта
            for (var property in extendProto) { // для
всех свойств нового объекта
                if (this.hasOwnProperty(property) ||
typeof object[property] === 'undefined') {
                    // если свойства отсутствуют в
родительском объекте, то добавляем
                    object[property] = extend-
Proto[property];
                }
            }
            return object;
```

```
        }
    };
```

Функция extend реализует наследование путем создается новая переменная property, которая последовательно перебирает все поля и функции объекта extendProto. Значение поля или функции с именем property копируется из extendProperty в object.

Player.

```
    var Player = Entity.extend({
        Score: 0,
        dirSprite: null,
        curFrame: 0,
        speed_x: 26,
        speed_y: 0,
        draw: function (ctx)
        {// прорисовка объекта
                spriteManager.drawSprite(ctx,
this.dirSprite, this.pos_x, this.pos_y, this.curFrame);
                this.curFrame++;
                if (this.curFrame > 14)
                    this.curFrame = 0;
        },
        update: function ()
        {
            var result = physicManager.update(this);

        },
        onTouchEntity: function (obj) {
            soundManager.play('music/gift.wav', {loop-
ing: false, volume: 1});
            this.Score=this.Score+5;
            document.getElementById("Score").innerHTML =
'Очки: '+this.Score;
            obj.kill();
        },
        kill: function () {
            gameManager.kill(this);
```

```
        },
    });
```

Gift.

```
    var Gift = Entity.extend({
        dirSprite: null,
        curFrame: 0,
        draw: function (ctx) {
            // прорисовка объекта
            spriteManager.drawSprite(ctx,
this.dirSprite, this.pos_x, this.pos_y, this.curFrame);
            this.curFrame++;
            if (this.curFrame > 14)
                this.curFrame = 0;
        },
        kill: function () {
            gameManager.kill(this);
        }
    });
```

У Санты и подарка, есть функции:

draw: описывает отрисовку объекта и изменение кадра спрайта

kill: что делать в случае смерти объекта

onTouchEntity: что делать в случае столкновении

update: что делать при каждом такте игры

«Менеджер событий»

Используется для взаимодействия с пользователем. Сопоставляем коды клавиш действиям с помощью массива. И устанавливаем обработчики событий нажатия клавиш.

```
    var eventsManager = {
        bind: [], // сопоставление клавиш действиям
        action: [], // действия
        setup: function () { // настройка сопоставления
            this.bind[32] = 'up'; // Space
            // контроль событий клавиатуры
            document.body.addEventListener("keydown",
this.onKeyDown);
```

```
            document.body.addEventListener("keyup",
this.onKeyUp);
        },
        //При нажатии кнопки
        onKeyDown: function (event) {
            var action = eventsManag-
er.bind[event.keyCode];
                    if (action) // проверка на action
=== true
                            eventsManager.action[action] =
true; // выполняем действие
        },
        //При отпуске кнопки
        onKeyUp: function (event) {
            var action = eventsManag-
er.bind[event.keyCode];
            if (action)
                eventsManager.action[action] = false; //
отменили действие
        }
    };
```

onKeyDown: Активирует событие, отведенное для нажатой кнопки

onKeyUp: Отключает событие, отведенное кнопки, которую мы отпустили

«Менеджер физики объектов»

Отвечает за перемещение объектов на карте (у нас двигается только Санта).

Функция *update.*

Выполняется при каждом обновлении игры, нам необходимо обеспечить передвижение санты по карте, для этого каждый игровой такт, мы будем прибавлять к его положению, его скорости, так же проверять не находится ли он в воздухе, и в случае этого воздействовать на него ускорением свободного падения, а так же проверять на столкновения с тайлами и объектами (проверять коллизии).

```
    update: function (obj) {
```

```
        var newX = obj.pos_x + Math.floor(obj.speed_x);
//Получаем новые значение  x y
        var newY = obj.pos_y + Math.floor(obj.speed_y);
        if(obj.name.match("Santa")) // Для санты
        {
            if(newY<438) obj.speed_y=obj.speed_y+6;
//Если Санта не на земле , уменьшаем его скорость по
y(ускорение свободного падения)
            else
            {
                obj.speed_y=0; //Если упал на землю,
скорость по y=0
                newY=438;
            }
        }
        // анализ на столкновения(коллизии)
        var ts = mapManager.getTilesetIdx(newX +
obj.size_x-15, newY + obj.size_y-5); //Находим id тайла
с которым взаимодействуем
        var e = this.entityAtXY(obj, newX, newY); //
Проверяем столкнулись ли мы с объектом
        if (e !== null && obj.onTouchEntity) // если
столкнулись с объектом(подарком)
            obj.onTouchEntity(e); // собираем подарок
   if(((ts>29)&(ts<32))|((ts>37)&(ts<40))|((ts>44)&(ts<
49))|((ts>52)&(ts<57))|((ts>60)&(ts<65))|(obj.pos_x>199
50)) //При столкновении с елкой/концом карты
        {
            document
.getElementById('menu_').style.display = 'inline';
//Завершаем игру
            document
.getElementById('canvas').style.display = 'none';
            document
.getElementById('Score').style.display='none';
            gameManager.player = null;
```

```
            if(obj.pos_x<900) document
.getElementById('wintext').innerHTML = 'Space -
прыжок';
            else
            {
                if (obj.pos_x < 10000) document
.getElementById('wintext').innerHTML = 'Рано или поздно
получится';
                else
                {
                    if(obj.pos_x<15000) document
.getElementById('wintext').innerHTML = 'Вы почти
справились!';
                    else if(obj.pox_x==20000) document
.getElementById('wintext').innerHTML = 'Вы выиграли!';
                }
            }
            Document
.getElementById('WinScore').innerHTML = 'Очки:
'+obj.Score;
            Document
.getElementById('wintext').style.display = 'block';
            obj.kill();
            gameManager.player=null;
            gameManager.stopAll();
        }
        obj.pos_y=newY; //Если все столкновения
обработаны, обновляем координаты
        obj.pos_x=newX;
    }
```

Функция *entityAtXY*.

Функция выполняет поиск объекта по координатам, для проверки столкновений

```
    entityAtXY: function (obj, x, y)
    {
```

```
        for (var i = 0; i < gameManager.entities.length;
i++) {
            var e = gameManager.entities[i]; //
рассматриваем все объекты на карте
            if (e.name !== obj.name) { // имя не
совпадает
                if (x + obj.size_x < e.pos_x || // не
пересекаются
                    y + obj.size_y < e.pos_y ||
                    x > e.pos_x + e.size_x ||
                    y > e.pos_y + e.size_y)
                    continue;
                return e; // найден объект
            }
        }
        return null; // объект не найден
    }
```

«Менеджер звука»

Имеет методы инициализации, проигрывания, остановки проигрывания.

Работа с аудиофайлом осуществляется с использованием специального аудиоконтекста, для загрузки файла создана функция loadSound, для проигрывания — функция playSound.

this.context = new AudioContext();

На основании встроенного объекта Web Audio API (AudioContext) создается новая переменная context, которая будет использоваться в функциях loadSound и playSound.

Функция loadSound загружает звук из файла (url) с помощью асинхронного запроса с использованием AJAX технологий.

```
    var soundManager = {
        clips: {}, // звуковые эффекты
        context: null, // аудиоконтекст
        gainNode: null, // главный узел
        loaded: false, // все звуки загружены
        init: function () {
            // инициализация
            this.context = new AudioContext();
```

```
            this.gainNode = this.context.createGain ?
this.context.createGain() :
this.context.createGainNode();
this.gainNode.connect(this.context.destination);
        },
        load: function (path, callback) { // загрузка
одного аудиофайла
            if (this.clips[path]) {
                callback(this.clips[path]);
                return;
            }
            var clip = {path: path, buffer: null, load-
ed: false};
            clip.play = function (volume, loop) {
                soundManager.play(this.path,{looping:
loop?loop:false, volume: volume ? volume:1});
            };
            this.clips[path] = clip;
            var request = new XMLHttpRequest();
            request.open("GET", path, true);
            request.responseType = 'arraybuffer';
            request.onload = function () {
                soundManag-
er.context.decodeAudioData(request.response, function
(buffer) {
                    clip.buffer = buffer;
                    clip.loaded = true;
                    callback(clip);
                });
            };
            request.send();
        },
        loadArray: function (array) {
            // загрузка массива звуков
            for (var i = 0; i < array.length; i++) {
```

```
                soundManager.load(array[i], function ()
{
                    if (array.length === Ob-
ject.keys(soundManager.clips).length) {
                        for (var sd in soundManag-
er.clips)
                            if (!soundManag-
er.clips[sd].loaded) return;
                        soundManager.loaded = true;
                    }
                });
            }
        },
        //запуск звука
        play: function (path, settings) {
            if (!soundManager.loaded) {
                setTimeout(function () {
                    soundManager.play(path,settings);
                }, 200);
                return;
            }
            var looping = false;
            var volume = 1;
            if (settings) {
                if (settings.looping)
                    looping = settings.looping;
                if (settings.volume)
                    volume = settings.volume;
            }
            var sd = this.clips[path];
            if (sd === null) return false;
            // создаем нвоый экземпляр проигрывателя
BufferSOurce
            var sound = soundManag-
er.context.createBufferSource();
            sound.buffer = sd.buffer;
```

```
            sound.connect(soundManager.gainNode);
            sound.loop = looping;
            soundManager.gainNode.gain.value = volume;
            sound.start(0);
            return true;
        },
        //Заканчивает всю музыку
        stopAll: function () {
            this.gainNode.disconnect();
        }
```

«Менеджер игры»

```
    var gameManager = { // менеджер игры
        factory: {}, // фабрика объектов на карте
        entities: [], // объекты на карте
        player: null, // указатель на объект игрока
        laterKill: [], // отложенное уничтожение
объектов
```

Обеспечивает инициализацию, загрузку всех необходимых ресурсов, хранение и управление всеми объектами игры, регулярное обновление и отображение пользователю игрового мира. Для корректной работы всех менеджеров игры они должны быть корректно инициализированы в правильной последовательности, для этого должна быть вызвана функция загрузки (loadAll).

```
    loadAll: function () {
        document.getElementById("Score").innerHTML =
'Очки: 0';
        soundManager.init();
        soundManager.loadArray(['music/gift.wav' , 'mu-
sic/JingleBellRock.mp3']);//Загрузка музыки
        mapManager.loadMap("map.json"); // загрузка
карты
        spriteManager.loadAtlas("atlas.json", "sprite-
sheet.png"); // загрузка атласа
        gameManager.factory['Santa'] = Player; //
инициализация фабрики
        gameManager.factory['Gift'] =  Gift;
```

```
        mapManager.parseEntities(); // разбор сущностей
карты
        mapManager.draw(ctx); // отобразить карту
        eventsManager.setup(); // настройка событий
        soundManager.play('music/JingleBellRock.mp3',
{looping: true, volume: 0.75});
    }
    initPlayer: function (obj)
    {
        this.player = obj;
    }
```

Игра регулярно обновляется. Обновление игры может происходить с частотой несколько раз в секунду. Чем чаще оно происходит, тем более плавные движения на экране наблюдает пользователь, тем большие требования предъявляются к компьютеру, на котором выполняется программа. В моей игре я выбрала частоту – 50 миллисекунд

```
    play: function () {
        gameManager.loadAll();
        setInterval(updateWorld, 50);
    }
```

Функция updateWorld, описанная вне менеджера, выполняет вызов обновления игрового мира.

```
    function updateWorld() {
        gameManager.update();
    }
```

Для уничтожения объектов используем объект отложенного удаления объектов, тогда оба

объекты поместят друг друга в массив laterKill и после выполнения всех действий текущего такта достаточно проверить и удалить хранящиеся в нем объекты.

```
     kill: function (obj) {
        this.laterKill.push(obj);
    }
```

Каждый так вызываются функции обновления, удаления и отрисовки, обработка нажатых клавиш (в нашем случае это пробел и его действие – увеличить скорость персонажа по у, если он стоит на земле).

```
    update: function () {
        if (this.player === null)
            return;
        if ((eventsManag-
er.action["up"])&(this.player.pos_y==438))
            this.player.speed_y = -66;
        //обновление информации по всем объектам на
карте
        this.entities.forEach(function (e) {
            try {
                e.update();
            } catch (ex) {
            }
        });
        // удаление всех объектов попавших в laterKill
        for (var i = 0; i < this.laterKill.length; i++)
{
            var idx =
this.entities.indexOf(this.laterKill[i]);
            if (idx > -1)
                this.entities.splice(idx, 1); //
удаление из массива 1 объекта
        }
        if (this.laterKill.length > 0) // очистка
массива laterKill
            this.laterKill.length = 0;
        mapManager.draw(ctx); // Перерисовываем карту
        mapManager.centerAt(this.player.pos_x,
this.player.pos_y); //центрируем карту
        this.draw(ctx);//Перерисовываем объекты
    },
    draw: function (ctx) {
        for (var e = 0; e < this.entities.length; e++) {
            this.entities[e].draw(ctx);
        }}
```

Список литературы

1. Гоше Хуан Диего. HTML5 Для профессионалов, СПб.:Питер, 2015.
2. Беляев С.А. Разработка игр на языке JavaScript, СПб.: Лань, 2016

Содержание

Учебно-методическое пособие

Беляев Сергей Алексеевич

Методические указания по выполнению курсовой работы

Издание публикуется в авторской редакции

СПбГЭТУ «ЛЭТИ»
197376, Санкт-Петербург, ул. Проф. Попова, 5