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. 8383 _____ Переверзев Д.Е.

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:**

Парсинг карты:

```
parseMap: function (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;
  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;
      }
    }
  }
  this.jsonLoaded = true;
},

parseEntities: function () {
  if (!mapManager.jsonLoaded) {
    setTimeout(function () {
      mapManager.parseEntities();
    }, 100);
  } else {
    for (let j = 0; j < this.mapData.layers.length; j++) {
      if (this.mapData.layers[j].type === "objectgroup") {
        const entities = this.mapData.layers[j];
        for (let i = 0; i < entities.objects.length; i++) {
          const e = entities.objects[i];
          try {
            const obj = gameManager.genObj(e);
            if (obj.name === "Player") {
              const lastPlayer = gameManager.player;
              if (lastPlayer) {
                obj.money < lastPlayer.money &&
                  (obj.money = lastPlayer.money);
                obj.hp < lastPlayer.hp && (obj.hp = lastPlayer.hp);
                obj.mp < lastPlayer.mp && (obj.mp = lastPlayer.mp);
              }
              gameManager.player = obj;
            }
          } catch (ex) {
            console.log(
              "Ошибка создания: [" + e.gid + "]" + e.type + "," + ex
            );
          }
        }
      }
    }
  }
},
```

Отрисовка карты:

```
draw: function (ctx) {
  if (!spriteManager.imgLoaded || !spriteManager.jsonLoaded) return;
  ctx.rect(0, 0, this.view.w, this.view.h);
  ctx.fillStyle = "#222222";
  ctx.fill();
  if (!mapManager.jsonLoaded) {
    setTimeout(function () {
      mapManager.draw(ctx);
    }, 100);
  } else {
    for (let i = 0; i < this.tLayer.data.length; i++) {
      if (this.tLayer.data[i] !== 0) {
        const pX = (i % this.xCount) * this.tSize.x;
        const pY = Math.floor(i / this.xCount) * this.tSize.y;
        spriteManager.drawSprite(
          ctx,
          spriteManager.getSpriteBySpriteId(this.tLayer.data[i]),
          pX,
          pY
        );
      }
    }
  }
},
```

2. Менеджер спрайтов

Парсинг:

```
parseAtlas: function (atlasJSON) {
  const atlas = JSON.parse(atlasJSON);
  const sorted = atlas.frames.sort((f1, f2) =>
    f1.frame.y !== f2.frame.y
      ? f1.frame.y - f2.frame.y
      : f1.frame.x - f2.frame.x
  );
  sorted.forEach((obj) => {
    const frame = obj.frame;
    this.sprites.push({
      name: obj.filename,
      x: frame.x,
      y: frame.y,
      w: frame.w,
      h: frame.h,
      solid: obj.filename.toLowerCase().includes("wall"),
    });
  });
  this.jsonLoaded = true;
},
```

Отрисовка:

```
drawSprite: function (ctx, sprite, x, y, r = 0) {
  if (!this.imgLoaded || !this.jsonLoaded) {
  } else {
    if (!mapManager.isVisible(x, y, sprite.w, sprite.h)) return;
    x -= mapManager.view.x;
    y -= mapManager.view.y;

    if (r === 0) {
      ctx.drawImage(
        this.image,
        sprite.x,
        sprite.y,
        sprite.w,
        sprite.h,
        Math.round(x),
        Math.round(y),
        sprite.w,
        sprite.h
      );
    } else {
      ctx.save();
      ctx.translate(x + sprite.w / 2, y + sprite.h / 2);
      ctx.rotate((r / 180) * Math.PI + Math.PI / 2);
      ctx.drawImage(
        this.image,
        sprite.x,
        sprite.y,
        sprite.w,
        sprite.h,
        -sprite.w / 2,
        -sprite.h / 2,
        sprite.w,
        sprite.h
      );
      ctx.restore();
    }

  }
},
```

3. Менеджер событий:

```
const eventsManager = {
    bind: [],
    action: [],

    setup: function (canvas) {
        this.bind[27] = 'esc'
        this.bind[87] = 'up'
        this.bind[65] = 'left'
        this.bind[83] = 'down'
        this.bind[68] = 'right'
        this.bind[16] = 'nitro'
        this.bind[18] = 'slow'
        canvas.addEventListener('mousedown', this.onMouseDown)
        canvas.addEventListener('mouseup', this.onMouseUp)
        canvas.addEventListener('mousemove', this.onMouseMove)
        document.body.addEventListener('keydown', this.onKeyDown)
        document.body.addEventListener('keyup', this.onKeyUp)
    },

    onMouseMove: function (event) {
        eventsManager.action.fire && (eventsManager.action.fire = event)
    },

    onMouseDown: function (event) {
        eventsManager.action.fire = event
    },

    onMouseUp: function (event) {
        eventsManager.action.fire = false
    },

    onKeyDown: function (event) {
        const action = eventsManager.bind[event.keyCode]
        if (action) {
            event.preventDefault ? event.preventDefault(): event.returnValue=false
            eventsManager.action[action] = true
        }
    },

    onKeyUp: function (event) {
        const action = eventsManager.bind[event.keyCode]
        if (action) {
            event.preventDefault ? event.preventDefault(): event.returnValue=false
            eventsManager.action[action] = false
        }
    }
}
```

4. Менеджер физики игры:

Обработка:

```
update: function (entity) {
    if (entity.move_x === 0 && entity.move_y === 0) return
    entity.direction = vectorAngle(entity.move_x, entity.move_y)
    const oldX = entity.pos_x
    const oldY = entity.pos_y
    const newX = oldX + Math.round(entity.move_x * entity.speed)
    const newY = oldY + Math.round(entity.move_y * entity.speed)
    const newCenterX = newX + entity.size_x / 2
    const newCenterY = newY + entity.size_y / 2
    let spriteId = null
    let destinationSprite = null
    let destinationEntity = null
    if (newCenterX > 0 && newCenterX < mapManager.mapSize.x && newCenterY > 0 && newCenterY < mapManager.mapSize.y) {
        spriteId = mapManager.getSpriteId(newCenterX, newCenterY)
        destinationSprite = spriteManager.getSpriteBySpriteId(spriteId)
        destinationEntity = this.entityAtXY(entity, newX, newY)
    }
    if (destinationEntity !== null) {
        if (entity.solid) { destinationEntity.onEntityCollision(entity) }
        if (destinationEntity.solid) { entity.onEntityCollision(destinationEntity) }
    }
    if (!destinationEntity?.solid) {
        if (destinationSprite === null) {
            entity.onCollision(null)
        } else if (destinationSprite.solid) {
            entity.onCollision(spriteId)
        } else {
            entity.pos_x = newX
            entity.pos_y = newY
            entity.onMoved(oldX, oldY)
        }
    }
},
```

Взаимодействие с врагами:

```
update (filter) {
    filter.include(TransformComponent.name).include(HealthComponent.name).include(ColliderComponent.name).forEach(entity => {
        const health = entity.HealthComponent
        const collider = entity.ColliderComponent
        for (const otherEntity of collider.triggers) {
            if (BulletComponent.name in otherEntity) {
                health.hp = Math.max(0, health.hp - otherEntity.BulletComponent.damage)
                otherEntity.isNull = true
                soundManager.play('hurt2', 1, false)
            }
        }
        if (health.hp <= 0) {
            entity.isNull = true
            config.kills++
            soundManager.play('hurt', 1, false)
        }
    })
}
```

5. Менеджер рекордов

```
const recordManager = {

    getLS: function (name) {
        let data = localStorage.getItem(name)
        try { return JSON.parse(data) || {} }
        catch (error) {
            this.setLS({})
            return data || {}
        }
    },
    setLS: function (name, data) {
        localStorage.setItem(name, JSON.stringify(data))
    },
    get: function () {
        return this.getLS("records_cw")
    },
    set: function (records) {
        return this.setLS("records_cw", records)
    },
    add: function (name, money) {
        this.set({ ...this.get(), [name]: money })
    }
}
```

6. 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, {
      looping: loop || false,
      volume: volume || 0.3,
    });
  };
  this.clips[path] = clip;
  const request = new XMLHttpRequest();
  request.open("GET", path, true);
  request.responseType = "arraybuffer";
  request.onload = function () {
    soundManager.context.decodeAudioData(request.response, function (buffer) {
      clip.buffer = buffer;
      clip.loaded = true;
      callback(clip);
    });
  };
  request.send();
},
```

Проигрывание звука

```
play: function (path, settings) {
  if (!soundManager.loaded) {
    setTimeout(function () {
      soundManager.play(path, settings);
    }, 200);
    return;
  }
  let looping = false;
  let volume = 0.3;
  if (settings) {
    if (settings.looping) {
      looping = settings.looping;
    }
    if (settings.volume) {
      volume = settings.volume;
    }
  }
  const sd = this.clips[path];
  if (sd === null) return false;
  const sound = soundManager.context.createBufferSource();
  soundManager.context.resume();
  sound.buffer = sd.buffer;
  sound.connect(soundManager.gainNode);
  sound.loop = looping;
  soundManager.gainNode.gain.value = volume;
  // sound.resume();
  sound.start(0);
  return true;
},
```

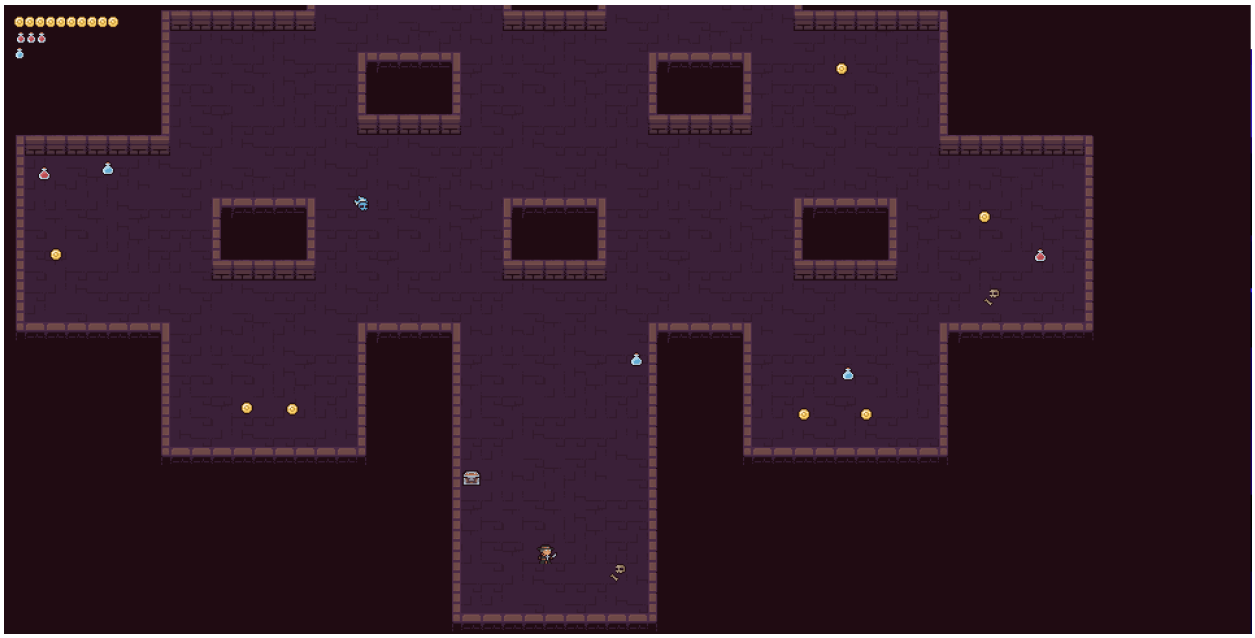
7. Игровой цикл:

```
update: function () {
  if (this.player === null) return;
  HTMLManager.updateAll();
  if (eventsManager.action.esc) this.end_game();
  this.entities.forEach(function (entity) {
    entity.update();
  });
  for (let i = 0; i < this.forRemove.length; i++) {
    const idx = this.entities.indexOf(this.forRemove[i]);
    if (idx > -1) {
      this.entities.splice(idx, 1);
    }
  }
  this.forRemove.length = 0;
  mapManager.centerAt(this.player?.pos_x || 0, this.player?.pos_y || 0);
  mapManager.draw(this.ctx);
  for (let e = 0; e < this.entities.length; e++) {
    this.entities[e].draw(this.ctx);
  }
  if (this.isGameOver()) {
    if (this.level + 1 === this.levels_path.length) {
      this.endGame();
    } else {
    }
  }
  BarManager.updateAll(this.ctx);
},
```

Карта игры:

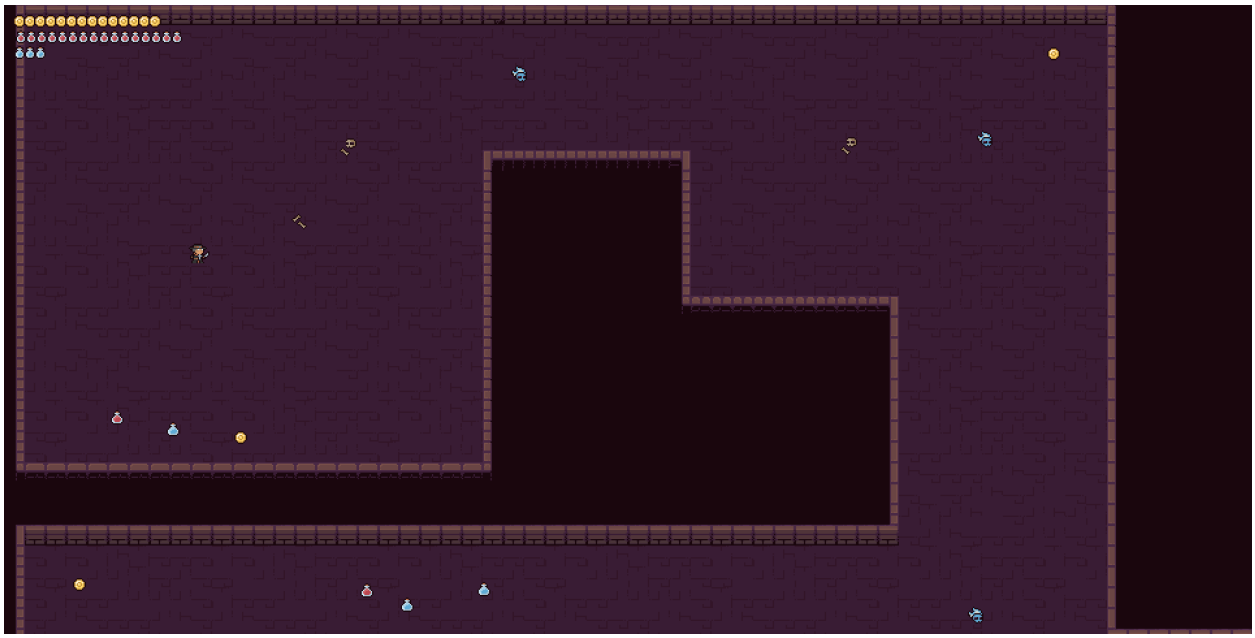
Будем использовать карты, созданные независимо от программного кода и сохраненные в формате JSON. Карта создавалась в TiledMapEditor. Выбирается размер карты, загружается набор тайлов, из которых и строится карта.

Уровень 1:

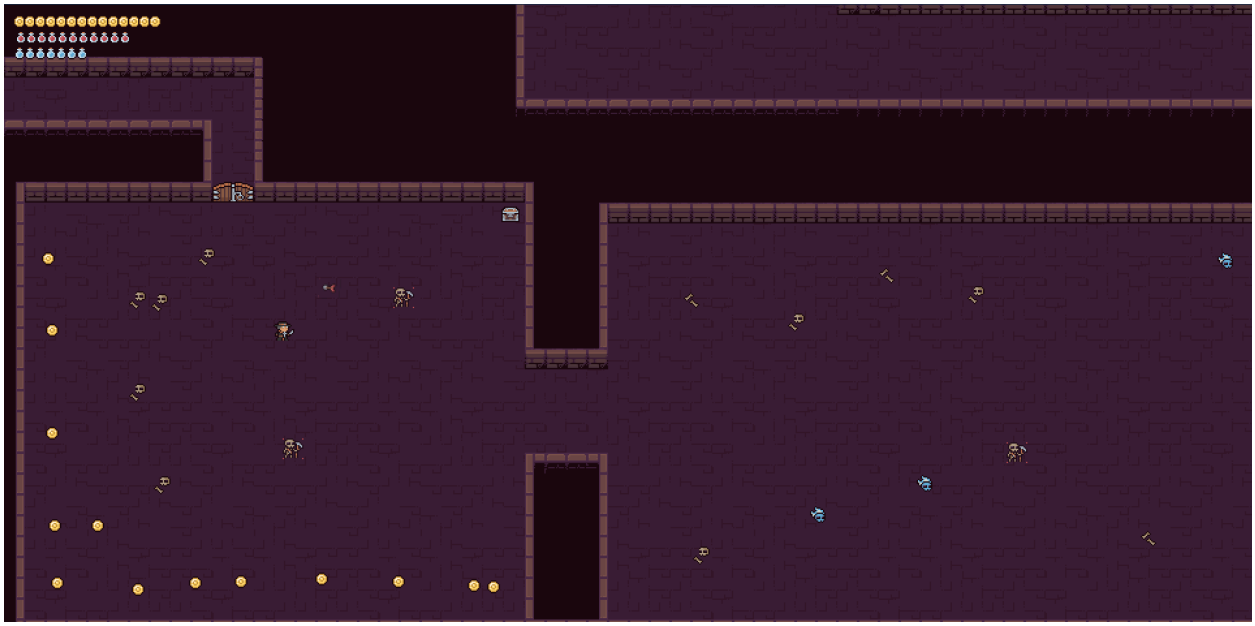


Уровень 2:

Старт:



Финиш



Тестрирование игры:

- Таблица рекордов

Рекорды

Ник игрока	Максимальное набранное кличество монет
fix1209	48
fox	1209
просто прохожий	31
Примите курсач, пожалуйста	14

Рекорды посмотрели, можно и поиграть

Вывод:

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