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

КУРСОВАЯ РАБОТА

по дисциплине «Web Технологии»

Тема: Разработка игры на языке JavaScript

Студент гр. 0303 _____ Болкунов В. О.

Преподаватель _____ Беляев С. А.

Санкт-Петербург

2022

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент: Болкунов В. О.

Группа 0303

Тема работы: разработка игры на языке JavaScript

Исходные данные:
Необходимо создать игру на «чистом» JavaScript (ES6)

Содержание пояснительной записки: содержание, введение, модули игры, пользовательский интерфейс, тестирование, заключение, список использованных источников.

Предполагаемый объем пояснительной записки:
Не менее 13 страниц.

Дата выдачи задания: 01.09.2022

Дата сдачи реферата: 02.12.2022

Дата защиты реферата: 02.12.2022

Студент _____ Болкунов В. О.

Преподаватель _____ Беляев С. А.

АННОТАЦИЯ

В данной курсовой работе описан процесс и особенности реализации игры на чистом языке JavaScript, совместимой с редактором наборов спрайтов и карт «Tiled». В результате были успешно спроектированы и реализованы модули игры и непосредственно веб-приложение, позволяющее пользователю запустить игру в браузере.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: изучить возможности языка JavaScript (ES6) и создать игру соответствующую требованиям.

Основные требования:

1. Минимум 2 уровня игры
2. Реализованы все менеджеры в соответствии с учебным пособием (УП)
3. Есть таблица рекордов
4. Есть препятствия
5. Есть «интеллектуальные» противники и «бонусы»
6. Используются tiles с редактором Tiled (www.mapeditor.org) в соответствии с УП

1. МОДУЛИ ИГРЫ

1.1. Модуль ядра

В данном модуле представлены основные классы для использования в других модулях.

1.1.1. Вектор

Класс вектора (Vec) в двухмерном пространстве – основа модели игры, на которой строится всё взаимодействие объектов, содержит ряд методов для вычислений и векторных операций.

1.1.2. Объект набора тайлов

Объект набора тайлов (TileSetObject) – класс для представления объектов из набора тайлов редактора Tiled, преобразует сырой JSON в свойства объекта и также загружает спрайт объекта.

1.1.3. Набор тайлов

TileSet – класс непосредственно представляющий набор тайлов из редактора Tiled, загружает файл с тайлсетом и также загружает все спрайты. Данный класс является аналогом менеджера спрайтов.

1.2. Модуль карты

Данный модуль отвечает за загрузку и отображение клеток (тайлов) и карты.

1.2.1. Тайл

Класс тайла (Tile) – единица карты игры.

1.2.2. Карта игры

Класс карты игры (GameMap). Позволяет отобразить карту и получить клетку по её координатам или индексам. Данный класс является аналогом менеджера карты.

1.3. Модуль движка

В данном модуле описаны объекты игры и сценарии их взаимодействия между собой и с картой игры, в том числе физика и звук. В целом данный модуль вместе с объектами игры и классом движка является аналогом менеджера физики.

1.3.1 Фигуры объектов

Фигуры позволяют движку обрабатывать столкновения и наложения объектов. Базовый класс фигур – Shape. Основные фигуры игры: прямоугольники (Rect) и окружности (Circle).

1.3.2. Классы объектов игры

В данный модуль внутри модуля движка вынесены классы объектов игры и фабрика для создания объектов из сырого JSON-а, полученного с карты.

1.3.2.1 Базовый объект игры

Базовый объект игры (GameObject) содержит информацию об объекте, размещённом на карте в редакторе Tiled, в т.ч. его свойства. Также содержит физические параметры объекта (позиция, скорость, угол поворота) и основные методы для отображения и обновления объекта.

1.3.2.2. Существо

Существо (Entity) – класс объектов врагов и игрока, способен подбирать/выбрасывать оружие и атаковать.

1.3.2.3. Бонус

Объекты бонусов (Bonus) – влияют на существа которые их подбирают. Конкретно в игре реализован 1 вид бонусов – бонусы на увеличение здоровья.

1.3.2.4. Оружие

Объекты оружия (Weapon) которые существа могут брать/выкидывать и атаковать с их помощью.

1.3.2.5. Снаряд

Снаряд (Projectile) – объект выпускаемый оружием дальнего боя. При попадании наносит урон объектам.

1.3.2.6. Выход

Выход (Exit) – объект для перехода на следующий уровень (подразумевается что на карте он один).

1.3.3 Звук

Класс звука (Sound), позволяет загружать и проигрывать звуки.

1.3.4 Менеджер звука

Менеджер звука – глобальный объект, который содержит аудио-контекст и список звуков игры (которые он позволяет загрузить).

1.3.5 Класс движка

Объект движка (Engine) – обрабатывает события игры и связывает объекты игры с внешними модулями.

1.4. Модуль игры

Данный модуль объединяет в себе движок, карту, объекты и их отрисовку. Позволяет пользователю взаимодействовать с игрой.

1.4.1. Менеджер событий

Класс менеджера событий (EventManager) отлавливает события с клавиатуры и мыши игрока, сохраняя их в своём состоянии, в дальнейшем в классе игры эти данные используются для синхронного управления существом игрока.

1.4.2. Контроллер противника

EnemyController – реализация интеллекта противника, он способен находить и подбирать оружие, преследовать и атаковать игрока, либо убегать от него если оружие отсутствует.

1.4.3 Игра

Класс игры (Game) отвечает за уровень игры запускает внутри себя цикл обработки движка и передаёт ему действия игрока и контроллеров противников.

2. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС

Интерфейс пользователя реализован с использованием чистого html и css и использует JavaScript для перехода между отображениями (вход, игра, сообщение о конце игры и таблица рекордов) и их анимаций.

3. ТЕСТИРОВАНИЕ

На рисунке 1 изображена стартовая страница с вводом имени игрока.

Рисунок 1: начальная страница

На рисунках 2, 3, 4 изображены уровни игры.

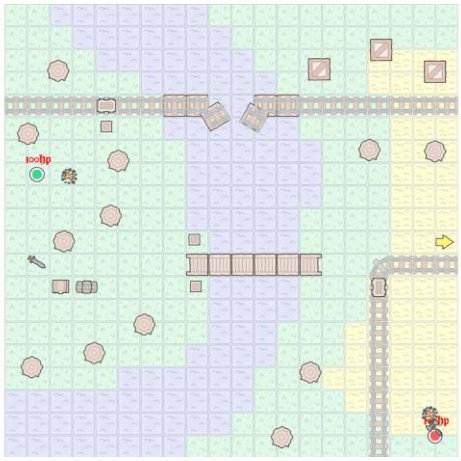

Рисунок 2: уровень 1

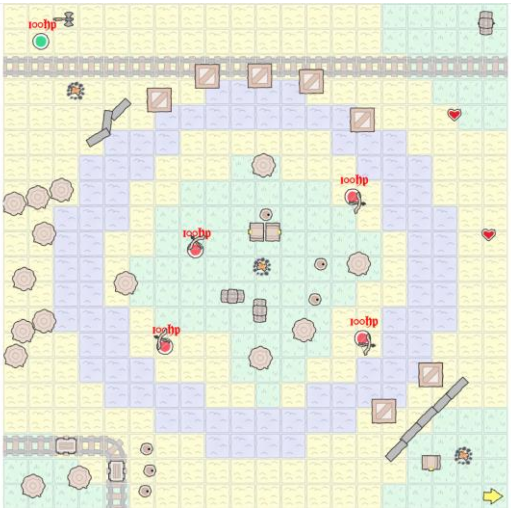

Рисунок 3: уровень 2

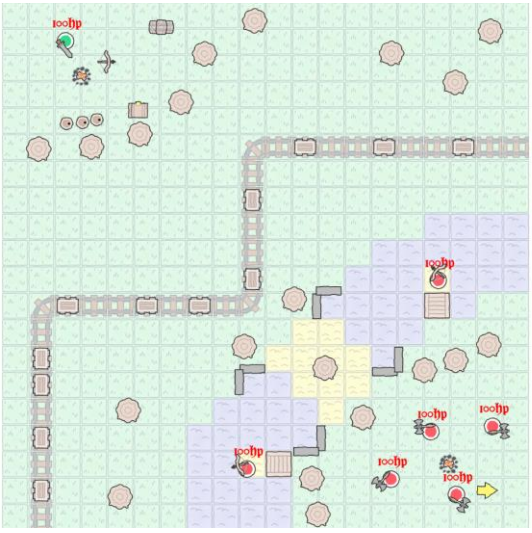

Рисунок 4: уровень 3

На рисунке 5 изображён пример таблицы рекордов.

Таблица рекордов:

Имя игрока	Время
Anatolyi	20.674 сек.
qwe	24.774 сек.
йцу	26.004 сек.
123	39.74 сек.
Влад	40.156 сек.
I	42.192 сек.
Владик	44.649 сек.
q	44.843 сек.

Рисунок 5: таблица рекордов

ЗАКЛЮЧЕНИЕ

В результате выполнение курсовой работы была спроектирована игра и полностью реализована на языке JavaScript. Игра совместима с редактором карт Tiled, в котором можно размещать препятствия и врагов, обладающих простым интеллектом. По окончании игры, приложение позволяет просмотреть таблицу рекордов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Беляев С.А. Разработка игр на языке JavaScript: Издательство Лань, 2022.
2. https://learn.javascript.ru/
3. https://doc.mapeditor.org/en/stable/

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Файл ./core/Vec.js

```
export class Vec {
  /** @type {number} */
  x;
  /** @type {number} */
  y;

  /** @param {number} x
   * @param {number} y */
  constructor(x = 0, y = 0) {
    this.x = x;
    this.y = y;
  }

  /** @param {number} a
   * @return {Vec} */
  static fromAngle(a) {
    return new Vec(Math.cos(a), Math.sin(a));
  }

  /** @return{Vec} */
  sign() {
    return new Vec(...this.flat().map(Math.sign));
  }

  /** @return{Vec} */
  abs() {
    return new Vec(...this.flat().map(Math.abs));
  }

  /** @return{[number, number]} */
  flat() {
    return [this.x, this.y];
  }

  /** @return {Vec} */
  neg() {
    return new Vec(-this.x, -this.y);
  }

  /** @param {Vec} v
   * @return {Vec} */
  add(v) {
    return new Vec(this.x + v.x, this.y + v.y);
  }

  /** @param {Vec} v
   * @return {Vec} */
  diff(v) {
    return this.add(v.neg());
  }

  /** @param {number | Vec} k
   * @return {Vec} */
  mult(k) {
    return typeof k === "number"
      ? new Vec(this.x * k, this.y * k)
      : new Vec(this.x * k.x, this.y * k.y);
```

```
  }

  /** @param {Vec} v
   * @return {number} */
  dot(v) {
    return this.x * v.x + this.y * v.y;
  }

  /** @return {number} */
  len2() {
    return this.x ** 2 + this.y ** 2;
  }

  /** @return {number} */
  len() {
    return this.len2() ** (1 / 2);
  }

  /** @return {Vec} */
  norm() {
    let len = this.len();
    return len !== 0 ? new Vec(this.x, this.y).mult(1 / len) : new Vec();
  }

  /** @param {Vec|number} v
   * @return {Vec} */
  rot(v) {
    if (typeof v === "number") {
      return this.rot(Vec.fromAngle(v));
    } else {
      v = v.norm();
      return new Vec(this.x * v.x - this.y * v.y, this.x * v.y + this.y * v.x);
    }
  }

  /** @param {Vec} v
   * @return {number} */
  proj(v) {
    return this.dot(v) / this.len();
  }

  /** @param {Vec} v
   * @return {Vec} */
  vecProj(v) {
    return this.norm().mult(this.proj(v));
  }

  /** @param {Vec} v
   * @return {Vec} */
  compare(v) {
    return new this.diff(v).sign();
  }

  /** @param {Vec} v
   * @return {number} */
  range(v) {
    return this.diff(v).len();
  }
}

export const axisX = new Vec(1, 0);
export const axisY = new Vec(0, 1);
```

Файл ./core/TileSetObject.js

```
import { Vec } from "./Vec.js";

export class TileSetObject {
  /** @type {number} */
  id;
  /** @type {string} */
  tclass;
  /** @type {string} */
  imagePath;
  /** @type {any} */
  props;
  /** @type {Vec} */
  size;
  /** @type {HTMLImageElement} */
  sprite;

  /** @param {any} obj */
  constructor(obj) {
    let props = {};
    for (let prop of obj.properties) {
      props[prop.name] = prop.value;
    }
    [this.id, this.tclass, this.imagePath, this.size, this.props] = [
      obj.id,
      obj.class,
      obj.image,
      new Vec(obj.imagewidth, obj.imageheight),
      props,
    ];
    this.sprite = new Image(...this.size.flat());
  }

  /** @returns {Promise<void>} */
  async load() {
    return new Promise((resolve) => {
      this.sprite.src = this.imagePath;
      this.sprite.onload = () => {
        resolve();
      };
    });
  }
}
```

Файл ./core/TileSet.js

```
import { TileSetObject } from "./TileSetObject.js";
import { Vec } from "./Vec.js";

export class TileSet {
  /** @type {string} */
  assetsPath;
  /** @type {string} */
  tsFile;
  /** @type {TileSetObject[]} */
  tiles = [];
  /** @type {Vec} */
  size;

  /**
   * @param {string} assetsPath
```

```
   * @param {string} tsFile
   */
  constructor(assetsPath, tsFile) {
    this.tsFile = tsFile;
    this.assetsPath = assetsPath;
  }

  /** @returns {Promise<void>} */
  async load() {
    let data = await (await fetch(`${this.assetsPath}/${this.tsFile}`)).json();
    this.size = new Vec(data.tileheight, data.tilewidth);
    await Promise.all(
      (this.tiles = data.tiles.map(
        (t) =>
          new TileSetObject({ ...t, image: `${this.assetsPath}/${t.image}` })
      )).map((s) => s.load())
    );
  }

  /** @param {number} id
   * @return {TileSetObject | null} */
  get(id) {
    return this.tiles.find((t) => t.id === id) ?? null;
  }
}
```

Файл ./core/index.js

```
export * from "./Vec.js";
export * from "./TileSetObject.js";
export * from "./TileSet.js";

const radCoef = Math.PI / 180;

/** @param {number} angle
 * @returns {number} */
export function rad(angle) {
  return angle * radCoef;
}
```

Файл ./map/Tile.js

```
export class Tile {
  /** @type {boolean} */
  passable;
  /** @type {TileSetObject} */
  tile;
  /** @type {Vec} */
  pos;

  /** @param {TileSetObject} tile
   * @param {Vec} pos */
  constructor(tile, pos) {
    this.tile = tile;
    this.passable = tile.props.passable;
    this.pos = pos;
  }

  /** @returns {Vec} */
  getRealPos() {
    this.pos.mult(this.tile.size);
```

```
  }
}
```

Файл ./map/GameMap.js

```
import { Vec } from "../core";
import { Tile } from "./Tile.js";

export class GameMap {
  /** @type {Tile[][]} */
  field;
  /** @type {TileSet} */
  ts;
  /** @type {Vec} */
  size;
  /** @type {Vec} */
  tilesSize;

  /** @param {TileSet} ts
   * @param {any} field */
  constructor(ts, field) {
    this.size = new Vec(field.width, field.height);
    this.tilesSize = new Vec(field.tilewidth, field.tileheight);
    this.ts = ts;
    this.field = [];
    for (let i = 0; i < this.size.y; i++) {
      this.field.push([]);
      for (let j = 0; j < this.size.x; j++) {
        const id = field.data[i * this.size.x + j] - 1;
        this.field[i].push(new Tile(ts.get(id), new Vec(j, i)));
      }
    }
  }

  /** @returns {Vec} */
  getRealSize() {
    return this.size.mult(this.tilesSize);
  }

  /** @param {Vec} v
   * @return {Vec} */
  getIdx(v) {
    return new Vec(
      Math.floor(v.x / this.ts.size.x),
      Math.floor(v.y / this.ts.size.y)
    );
  }

  /** @param {Vec} v
   * @param {boolean} indexes
   * @return {Tile | null} */
  get(v, indexes = false) {
    if (!indexes) v = this.getIdx(v);
    return v.x < 0 || v.y < 0 ? null : this.field?.at(v.y)?.at(v.x) ?? null;
  }

  /** @param{CanvasRenderingContext2D} ctx */
  draw(ctx) {
    for (let y = 0; y < this.size.y; y++) {
      for (let x = 0; x < this.size.x; x++) {
        ctx.drawImage(
          this.get(new Vec(x, y), true)?.tile.sprite,
```

```
          x * this.ts.size.x,
          y * this.ts.size.y
        );
      }
    }
  }
}
```

Файл ./map/index.js

```
export * from "./Tile.js";
export * from "./GameMap.js";

/** @param {string} mapFile
 * @return {Promise<[any, any]>} [tiles, objects] */
export async function parseMap(mapFile) {
  const data = await (await fetch(`${mapFile}`)).json();
  return [
    {
      ...data?.layers?.find((e) => e.name === "field"),
      tilewidth: data.tilewidth,
      tileheight: data.tileheight,
    },
    data?.layers?.find((e) => e.name === "objects"),
  ];
}
```

Файл ./engine/shapes.js

```
import { rad, Vec } from "../core";

/** @enum {string} */
export const ShapeTypes = {
  RECT: "rect",
  CIRCLE: "circle",
};

export class Shape {
  /** @type {GameObject} */
  obj;

  /** @param {GameObject} obj */
  constructor(obj) {
    this.obj = obj;
  }

  /** @param {Vec} dot
   * @returns {boolean}
   * @abstract */
  inside(dot) {
    return false;
  }

  /** @param {Vec} vec
   * @returns {Vec}
   * @abstract */
  getBorderDot(vec) {
    return new Vec();
  }
}
```

```
export class Circle extends Shape {
  /** @override */
  inside(dot) {
    return this.obj.pos.range(dot) <= this.obj.props.radius;
  }

  /** @override */
  getBorderDot(vec) {
    return vec.norm().mult(this.obj.props.radius);
  }
}

export class Rect extends Shape {
  /** @override */
  inside(dot) {
    let v = this.obj.pos.diff(dot);
    return (
      Math.abs(this.obj.rot.proj(v)) <= this.obj.props.width / 2 &&
      Math.abs(this.obj.rot.rot(rad(90)).proj(v)) <= this.obj.props.height / 2
    );
  }

  getBorderDot(vec) {
    let nvec = vec
      .norm()
      .mult((this.obj.props.width ** 2 + this.obj.props.height ** 2) ** (1 / 2))
      .abs();
    return vec
      .sign()
      .mult(
        new Vec(
          Math.min(this.obj.rot.proj(nvec), this.obj.props.width / 2),
          Math.min(
            this.obj.rot.rot(rad(90)).proj(nvec),
            this.obj.props.height / 2
          )
        ).rot(this.obj.rot)
      );
  }
}
```

Файл ./engine/game_objects/GameObject.js

```
import { rad, Vec } from "../../core";

export class GameObject {
  /** @type {Vec} */
  pos;
  /** @type {Vec} */
  rot;
  /** @type {Vec} */
  size;
  /** @type {ObjectTypes} */
  type;
  /** @type {TileSetObject} */
  tsObj;
  /** @type {any} */
  props;
  /** @type {Vec} */
  velocity = new Vec();
  /** @type {Shape} */
  shape;
```

```
/** @type {boolean} */
solid;

/** @param {TileSetObject} tsObj
 * @param {any} obj */
constructor(obj, tsObj) {
  this.size = new Vec(obj.width, obj.height);
  this.rot = Vec.fromAngle(rad(obj.rotation));
  this.pos = new Vec(obj.x, obj.y).add(
    this.size.mult(0.5).rot(rad(obj.rotation - 90))
  );
  this.tsObj = tsObj;
  this.props = { ...tsObj.props };
  this.type = this.props.type;
  this.solid = this.props.solid;
}

/** @param{CanvasRenderingContext2D} ctx */
draw(ctx) {
  ctx.save();
  ctx.transform(
    ...this.rot.flat(),
    -this.rot.y,
    this.rot.x,
    ...this.pos.flat()
  );
  ctx.drawImage(
    this.tsObj.sprite,
    ...this.size.mult(-0.5).flat(),
    ...this.size.flat()
  );
  ctx.restore();
}

/** @param {Vec} dot
 * @param {number} range
 * @returns {boolean} */
isNear(dot, range) {
  return this.pos.range(dot) < range;
}

/** @param {Vec} velocity */
move(velocity) {
  this.velocity = velocity;
}

/** @param {Vec} rot */
rotate(rot) {
  this.rot = rot;
}

/** @param {Engine} engine */
update(engine) {
  if (this.solid && this.props.hp !== undefined && this.props.hp <= 0)
    engine.destroy(this);

  this.pos = this.pos.add(this.velocity);
}

/** @param {Engine} engine
 * @param {number} value */
receiveDamage(engine, value) {
  if (this.props?.hp) {
```

```
      this.props.hp -= value;
      engine.playSound(engine.sm.sounds.punch, this.pos);
    }
  }
}
```

Файл ./engine/game_objects/Entity.js

```
import { GameObject } from "./GameObject.js";
import { axisX, axisY, Vec } from "../../core";
import { solidCollisionsUpdate } from "./index.js";
import { WeaponTypes } from "./Weapon.js";

export const MAX_DROP_RANGE = 96;
export const BASE_ATTACK_DELAY = 700;
export const ATTACK_ANIM_DELAY = 75;
export const MELEE_ATTACKING_ANGLE = 0.2;
export const MELEE_ATTACKING_ROT = Vec.fromAngle(-MELEE_ATTACKING_ANGLE);
export const MELEE_ATTACKING_ROT_INV = Vec.fromAngle(MELEE_ATTACKING_ANGLE);

export class Entity extends GameObject {
  /** @type {Weapon} */
  weapon = null;
  /** @type {boolean} */
  attacking = false;
  /** @type {number} */
  attackAnim = 0;
  /** @type {Vec} */
  attackingRot = Vec.fromAngle(0);

  /** @override */
  update(engine) {
    solidCollisionsUpdate(
      this,
      engine.solid.filter(
        (o) => o.pos.diff(this.pos).len2() <= this.size.add(o.size).len2()
      )
    );

    if (!engine.map.get(this.pos.add(axisX.vecProj(this.velocity)))?.passable)
      this.velocity.x = 0;
    if (!engine.map.get(this.pos.add(axisY.vecProj(this.velocity)))?.passable)
      this.velocity.y = 0;

    if (this.weapon?.props?.range === WeaponTypes.MELEE && this.attacking) {
      this.attackingRot =
        this.attackAnim === 0
          ? Vec.fromAngle(0)
          : this.attackingRot.rot(
              this.attackAnim === 1
                ? MELEE_ATTACKING_ROT
                : MELEE_ATTACKING_ROT_INV
            );
      this.rotate(this.rot);
    }

    super.update(engine);
    if (this.props?.hp <= 0) {
      this.dropWeapon(engine, this.pos);
    }
  }
```

```
  /** @override */
  rotate(rot) {
    rot = rot.rot(this.attackingRot);
    super.rotate(rot);
  }

  /** @override */
  move(velocity) {
    if (velocity.len() > this.props.speed)
      velocity = velocity.norm().mult(this.props.speed);
    super.move(velocity);
  }

  /** @override */
  draw(ctx) {
    super.draw(ctx);
    ctx.fillText(
      this.props.hp + "hp",
      ...this.pos.diff(this.size.mult(1 / 2)).flat()
    );

    this.weapon?.drawWithOwner(ctx);
  }

  /** @param {Engine} engine */
  attack(engine) {
    if (!this.attacking) {
      this.attackAnim = +(this.attacking = true);
      this?.weapon?.attack(engine);
      setTimeout(() => (this.attacking = false), BASE_ATTACK_DELAY);
      setTimeout(() => {
        this.attackAnim = -1;
        setTimeout(() => (this.attackAnim = 0), ATTACK_ANIM_DELAY);
      }, ATTACK_ANIM_DELAY);
    }
  }

  /** @param {Engine} engine
   *  @param {Vec} pos */
  dropWeapon(engine, pos) {
    if (this.weapon) {
      engine.playSound(engine.sm.sounds.drop, this.pos);
      this.weapon.owner = null;
      this.weapon.pos =
        pos.range(this.pos) <= MAX_DROP_RANGE
          ? pos
          : this.pos.add(pos.diff(this.pos).norm().mult(MAX_DROP_RANGE));
      this.weapon = null;
    }
  }
}
```

Файл ./engine/game_objects/Bonus.js

```
import { GameObject } from "./GameObject.js";
import { ObjectTypes } from "./index.js";

/** @enum {number} */
export const BonusEffects = {
  HEAL: "heal",
};
```

```
export class Bonus extends GameObject {
  /** @override */
  update(engine) {
    super.update(engine);

    let receiver = engine.objects
      .filter((o) => o.type === ObjectTypes.ENTITY)
      .filter((o) => o.shape.inside(this.pos))
      .at(0);

    if (receiver) {
      engine.playSound(engine.sm.sounds.drink, this.pos);
      switch (this.props.effect) {
        case BonusEffects.HEAL:
          receiver.props.hp += this.props.value;
          engine.destroy(this);
          break;
      }
    }
  }
}
```

Файл ./engine/game_objects/Weapon.js

```
import { GameObject } from "./GameObject.js";
import { ObjectTypes } from "./index.js";
import { rad } from "../../core";
import { Projectile } from "./Projectile.js";

export const WEAPON_ANGLE = rad(45);
export const WEAPON_TRANSLATION = 20;

/** @enum {string} */
export const WeaponTypes = {
  MELEE: "melee",
  RANGE: "range",
};

export class Weapon extends GameObject {
  /** @type {Entity} */
  owner = null;

  /** @override */
  update(engine) {
    super.update(engine);

    let receiver = engine.objects
      .filter((o) => o.type === ObjectTypes.ENTITY)
      .filter((o) => o.shape.inside(this.pos))
      .at(0);

    if (!this.owner && receiver && !receiver?.weapon) {
      this.owner = receiver;
      receiver.weapon = this;
      engine.playSound(engine.sm.sounds.grab, this.pos);
    }

    // console.log(this.owner?.attackAnim);
    if (this.owner) {
      this.pos = this.owner.pos.add(
        this.owner.rot.rot(WEAPON_ANGLE).norm().mult(WEAPON_TRANSLATION)
      );
```

```
      this.rotate(this.owner.rot);
    }
  }

  draw(ctx) {
    if (!this.owner) super.draw(ctx);
  }

  /** @param {CanvasRenderingContext2D} ctx */
  drawWithOwner(ctx) {
    super.draw(ctx);
  }

  /** @param {Engine} engine */
  attack(engine) {
    switch (this.props.range) {
      case WeaponTypes.RANGE:
        engine.playSound(engine.sm.sounds.bow_shoot, this.pos);
        engine.add(Projectile.createProjectile(engine, this));
        break;
      case WeaponTypes.MELEE:
        engine.playSound(engine.sm.sounds.hit, this.pos);
        engine.objects
          .filter(
            (o) =>
              o !== this.owner &&
              o.pos.range(this.owner.pos) <=
                this.props.dist + this.owner.size.len()
          )
          .filter(
            (o) =>
              this.owner.rot.dot(o.pos.diff(this.owner.pos).norm()) >=
              Math.cos(rad(this.props.angle))
          )
          .forEach((o) => {
            o.receiveDamage(engine, this.props.damage);
          });
        break;
    }
  }
}
```

Файл ./engine/game_objects/Projectile.js

```
import { GameObject } from "./GameObject.js";

export class Projectile extends GameObject {
  /** @type {Weapon} */
  source;

  /** @param {Engine} engine
   * @param {Weapon} source
   * @returns {Projectile} */
  static createProjectile(engine, source) {
    let tsObj = engine.map.ts.get(source.props.ammoId);
    let projectile = new Projectile(
      { width: tsObj.size.x, height: tsObj.size.y },
      tsObj
    );
    projectile.pos = source.pos;
    projectile.rot = source.owner.rot;
    projectile.velocity = source.owner.rot.mult(projectile.props.speed);
```

```
    projectile.source = source;
    return projectile;
  }

  /** @override */
  update(engine) {
    super.update(engine);

    if (
      this.pos.x < 0 ||
      this.pos.y < 0 ||
      this.pos.x > engine.map.getRealSize().x ||
      this.pos.y > engine.map.getRealSize().y
    )
      engine.destroy(this);

    let receiver = engine.objects
      .filter((o) => o.solid)
      .filter((o) => o.shape.inside(this.pos))
      .at(0);

    if (receiver) {
      engine.destroy(this);
      receiver.receiveDamage(engine, this.source.props.damage);
      engine.playSound(engine.sm.sounds.arrow_impact, this.pos);
    }
  }
}
```

Файл ./engine/game_objects/Exit.js

```
import { GameObject } from "./GameObject.js";

export class Exit extends GameObject {
  /** @param {Entity} player
   * @returns {boolean} */
  isPlayerStepped(player) {
    return player.shape.inside(this.pos);
  }
}
```

Файл ./engine/game_objects/index.js

```
import { GameObject } from "./GameObject.js";
import { Exit } from "./Exit.js";
import { Entity } from "./Entity.js";
import { Bonus } from "./Bonus.js";
import { Weapon } from "./Weapon.js";
import { Projectile } from "./Projectile.js";
import { Circle, Rect, Shape, ShapeTypes } from "../shapes.js";

export * from "./GameObject.js";
export * from "./Exit.js";
export * from "./Bonus.js";
export * from "./Weapon.js";
export * from "./Projectile.js";
export * from "./Entity.js";

/** @param {GameObject} obj
 * @param {GameObject[]} objects */
export function solidCollisionsUpdate(obj, objects) {
```

```
  if (obj.solid && Math.abs(obj.velocity.len2()) > 0)
    objects.forEach((n) => {
      let borderDot = obj.shape.getBorderDot(n.pos.diff(obj.pos));
      if (n.shape.inside(obj.pos.add(obj.velocity).add(borderDot)))
        obj.velocity = obj.velocity.add(
          obj.pos.add(borderDot).diff(n.pos).norm().mult(obj.velocity.len())
        );
    });
}

/** @enum {string} */
export const ObjectTypes = {
  OBJECT: "object",
  EXIT: "exit",
  ENTITY: "entity",
  BONUS: "bonus",
  WEAPON: "weapon",
  PROJECTILE: "projectile",
};

/** @param {any} obj
 * @param {TileSet} ts */
export function createGameObject(obj, ts) {
  let tsObj = ts.get(obj.gid - 1);
  let o = new [GameObject, Exit, Entity, Bonus, Weapon, Projectile][
    [
      ObjectTypes.OBJECT,
      ObjectTypes.EXIT,
      ObjectTypes.ENTITY,
      ObjectTypes.BONUS,
      ObjectTypes.WEAPON,
      // ObjectTypes.PROJECTILE,
    ].indexOf(tsObj.props.type)
  ](obj, tsObj);
  o.shape = new ([Rect, Circle][
    [ShapeTypes.RECT, ShapeTypes.CIRCLE].indexOf(tsObj.props.shape)
  ] ?? Shape)(o);
  return o;
}
```

Файл ./engine/Sound.js

```
export class Sound {
  /** @type {string} */
  path;
  /** @type {AudioBuffer} */
  audio;
  /** @type {AudioContext} */
  ctx;

  /** @param {string} path
   * @param {AudioContext} ctx */
  constructor(ctx, path) {
    this.path = path;
    this.ctx = ctx;
  }

  /** @returns {Promise<void>} */
  async load() {
    this.audio = await this.ctx.decodeAudioData(
      await (await fetch(this.path)).arrayBuffer()
    );
```

```
  }

  /** @param {number} volume
   * @returns {Promise<void>} */
  async play(volume) {
    return new Promise((resolve) => {
      let s = this.ctx.createBufferSource(),
        g = this.ctx.createGain();
      g.gain.value = volume;
      s.buffer = this.audio;
      s.connect(g).connect(this.ctx.destination);
      s.start();
      s.onended = function () {
        resolve();
      };
    });
  }
}
```

Файл ./engine/SoundManager.js

```
import { Sound } from "./Sound.js";

export const SoundManager = {
  /** @type {AudioContext} */
  ctx: new AudioContext(),

  /** @enum {string} */
  soundsPaths: {
    step: "step.wav",
    grab: "grab.wav",
    drop: "drop.wav",
    hit: "hit.wav",
    punch: "punch.wav",
    bow_shoot: "bow_shoot.wav",
    arrow_impact: "arrow_impact.wav",
    drink: "drink.wav",
    dead: "dead.mp3",
    win: "win.wav",
  },

  sounds: {},

  /** @param {string} path */
  async load(path) {
    this.sounds = {
      step: new Sound(this.ctx, `${path}/${this.soundsPaths.step}`),
      grab: new Sound(this.ctx, `${path}/${this.soundsPaths.grab}`),
      drop: new Sound(this.ctx, `${path}/${this.soundsPaths.drop}`),
      hit: new Sound(this.ctx, `${path}/${this.soundsPaths.hit}`),
      punch: new Sound(this.ctx, `${path}/${this.soundsPaths.punch}`),
      bow_shoot: new Sound(this.ctx, `${path}/${this.soundsPaths.bow_shoot}`),
      arrow_impact: new Sound(
        this.ctx,
        `${path}/${this.soundsPaths.arrow_impact}`
      ),
      drink: new Sound(this.ctx, `${path}/${this.soundsPaths.drink}`),
      dead: new Sound(this.ctx, `${path}/${this.soundsPaths.dead}`),
      win: new Sound(this.ctx, `${path}/${this.soundsPaths.win}`),
    };
    await Promise.all(Object.values(this.sounds).map((s) => s.load(this.ctx)));
```

```
  },
};
```

Файл ./engine/Engine.js

```
import { SoundManager } from "./SoundManager.js";

export class Engine {
  /** @type {GameMap}*/
  map;
  /** @type {GameObject[]}*/
  objects;
  /** @type {GameObject[]} */
  solid;
  /** @type {GameObject[]} */
  nonSolid;
  /** @type {typeof SoundManager} */
  sm = SoundManager;
  /** @type {Entity} */
  player;

  /** @param {GameMap} map
   * @param {GameObject[]} objects */
  constructor(map, objects) {
    this.map = map;
    this.objects = objects;
    this.calcSolids();
    this.player = objects.find((o) => o.props.entity === "player");
  }

  /** @param {Sound} sound
   * @param {Vec} pos
   * @returns {Promise<void>} */
  async playSound(sound, pos) {
    await sound.play(
      Math.min(
        1,
        Math.max(
          0.3,
          1 -
            (2 * this.player.pos.diff(pos).len2()) /
              this.map.getRealSize().len2()
        )
      )
    );
  }

  calcSolids() {
    this.solid = this.objects.filter((o) => o.props.solid === true);
    this.nonSolid = this.objects.filter((o) => o.props.solid === false);
  }

  /** @param {GameObject} obj */
  destroy(obj) {
    this.objects.splice(this.objects.indexOf(obj), 1);
    this.calcSolids();
  }

  /** @param {GameObject} obj */
  add(obj) {
    this.objects.push(obj);
    this.calcSolids();
```

```
  }

  update() {
    this.objects.forEach((o) => {
      o.update(this);
    });
  }
}
```

Файл ./engine/index.js

```
export * from "./game_objects";
export * from "./shapes.js";
export * from "./SoundManager.js";
export * from "./Engine.js";
```

Файл ./game/EventManager.js

```
import { Vec } from "../core";

/** @enum {number} */
export const KeyEvents = {
  UP: 0,
  DOWN: 1,
  LEFT: 2,
  RIGHT: 3,
  SPACE: 4,
};

/** @param {KeyboardEvent} ev
 * @returns {KeyEvents} */
export function getKeyEvent(ev) {
  return [
    KeyEvents.UP,
    KeyEvents.LEFT,
    KeyEvents.DOWN,
    KeyEvents.RIGHT,
    KeyEvents.SPACE,
  ][["w", "a", "s", "d", " "].indexOf(ev.key)];
}

export class InputState {
  /** @type {Vec} */
  mousePos = new Vec();
  /** @type {boolean} */
  mouseClick = false;
  /** @type {Set<KeyEvents>} */
  moves = new Set();
}

export class EventManager {
  /** type {InputState} */
  state = new InputState();

  /** @param {CanvasRenderingContext2D} ctx */
  constructor(ctx) {
    ctx.canvas.addEventListener("mousemove", (ev) => {
      let rect = ctx.canvas.getBoundingClientRect();
      this.state.mousePos = new Vec(
        ev.clientX - rect.left,
        ev.clientY - rect.top
```

```
      );
    });
    ctx.canvas.addEventListener("mousedown", (ev) => {
      this.state.mouseClick = true;
    });
    ctx.canvas.addEventListener("mouseup", (ev) => {
      this.state.mouseClick = false;
    });
    window.addEventListener("keydown", (ev) => {
      this.state.moves.add(getKeyEvent(ev));
    });
    window.addEventListener("keyup", (ev) => {
      this.state.moves.delete(getKeyEvent(ev));
    });
  }
}
```

Файл ./game/EnemyController.js

```
import { ObjectTypes, WeaponTypes } from "../engine";
import { Vec } from "../core";

export const VISIBILITY_RANGE = 600;
export const WEAPON_FIND_RANGE = 400;
export const RANGE_ATTACK = 500;
export const MELEE_ATTACK = 90;
export const RUN_RANGE = 200;
export const BYPASS_RANGE = 150;
export const MIN_BYPASS_SPEED = 0.3;
export const ATTACK_DELAY = 500;

export class EnemyController {
  /** @type {Engine} */
  engine;
  /** @type {Entity} */
  entity;
  /** @type {boolean} */
  attackDelay = false;

  /** @param {Engine} engine
   * @param {Entity} entity */
  constructor(engine, entity) {
    this.engine = engine;
    this.entity = entity;
  }

  /** @returns {boolean} */
  findWeapon() {
    let weapon = this.engine.objects
      .filter((o) => o.type === ObjectTypes.WEAPON && !o.owner)
      .filter((o) => o.pos.range(this.entity.pos) <= WEAPON_FIND_RANGE)
      .sort(
        (a, b) => a.pos.range(this.entity.pos) - b.pos.range(this.entity.pos)
      )
      .at(0);
    if (weapon) {
      this.entity.velocity = weapon.pos
        .diff(this.entity.pos)
        .norm()
        .mult(this.entity.props.speed);
      return true;
    } else return false;
```

```
}

runAway() {
  this.entity.move(this.entity.pos.diff(this.engine.player.pos));
}

attack() {
  this.entity.rotate(this.engine.player.pos.diff(this.entity.pos).norm());
  if (!this.attackDelay) {
    setTimeout(() => {
      this.entity.attack(this.engine);
      this.attackDelay = false;
    }, ATTACK_DELAY);
    this.attackDelay = true;
  }
}

/** @param {number} range */
shoot(range) {
  if (range <= RANGE_ATTACK) {
    this.attack();
  } else
    this.entity.move(
      this.engine.player.pos
        .diff(this.entity.pos)
        .norm()
        .mult(this.entity.props.speed)
    );
}

/** @param {number} range */
hit(range) {
  if (range >= MELEE_ATTACK)
    this.entity.move(
      this.engine.player.pos
        .diff(this.entity.pos)
        .norm()
        .mult(range / MELEE_ATTACK)
    );
  else this.attack();
}

bypass() {
  this.engine.solid.forEach((o) => {
    let range = o.pos.diff(this.entity.pos).len();
    if (range <= BYPASS_RANGE && range > 0 && o !== this.entity)
      this.entity.move(
        this.entity.velocity.add(
          this.entity.pos
            .diff(o.pos)
            .norm()
            .mult(Math.min(this.entity.size.len() / range, MIN_BYPASS_SPEED))
        )
      );
  });
}

update() {
  this.entity.velocity = new Vec();
  let range = this.engine.player.pos.range(this.entity.pos);

  if (!this.entity.weapon) {
    if (!this.findWeapon() && range <= RUN_RANGE) this.runAway();
```

```
    } else {
      if (range <= VISIBILITY_RANGE) {
        if (this.entity.weapon?.props?.range === WeaponTypes.RANGE)
          this.shoot(range);
        else this.hit(range);
      }
    }
    this.bypass();

    if (range < VISIBILITY_RANGE)
      this.entity.rotate(this.engine.player.pos.diff(this.entity.pos).norm());
    else if (this.entity.velocity.len2())
      this.entity.rotate(this.entity.velocity.norm());
  }
}
```

Файл ./game/Game.js

```
import { Engine, ObjectTypes } from "../engine";
import { Vec } from "../core";
import { EventManager, KeyEvents } from "./EventManager.js";
import { EnemyController } from "./EnemyController.js";

export class Game {
  /** @type {CanvasRenderingContext2D} */
  ctx;
  /** @type {TileSet} */
  ts;
  /** @type {GameMap} */
  map;
  /** @type {GameObject[]} */
  objects;
  /** @type {Entity} */
  player;
  /** @type {Exit} */
  exit;
  /** @type {EnemyController[]} */
  enemies;
  /** @type {EventManager} */
  em;
  /** @type {number} */
  mapScale;

  /** @param {CanvasRenderingContext2D} ctx
   * @param {TileSet} ts
   * @param {GameMap} map
   * @param {GameObject[]} objects */
  constructor(ctx, ts, map, objects) {
    this.ctx = ctx;
    this.ts = ts;
    this.map = map;
    this.objects = objects;
    this.exit = objects.find((o) => o.type === ObjectTypes.EXIT);
    this.em = new EventManager(ctx);

    this.mapScale = Math.max(
      ctx.canvas.width / map.getRealSize().x,
      ctx.canvas.height / map.getRealSize().y
    );
    ctx.scale(this.mapScale, this.mapScale);

    this.engine = new Engine(map, objects);
```

```
    this.enemies = objects
      .filter((o) => o.props.entity === "enemy")
      .map((o) => new EnemyController(this.engine, o));

    this.player = this.engine.player;
  }

  /** @returns {Promise<boolean>} */
  async startGame() {
    return new Promise((resolve) => {
      let gameCycle = setInterval(() => {
        let mousePos = this.em.state.mousePos.mult(1 / this.mapScale);
        this.player.rot = mousePos.diff(this.player.pos).norm();
        this.player.velocity = new Vec(
          this.em.state.moves.has(KeyEvents.RIGHT) -
            this.em.state.moves.has(KeyEvents.LEFT),
          this.em.state.moves.has(KeyEvents.DOWN) -
            this.em.state.moves.has(KeyEvents.UP)
        )
          .norm()
          .mult(this.player.props.speed);

        if (this.em.state.moves.has(KeyEvents.SPACE))
          this.player.dropWeapon(this.engine, mousePos);
        if (this.em.state.mouseClick) this.player.attack(this.engine);

        this.enemies.forEach((e) => e.update());
        this.engine.update();

        if (this.player.props.hp <= 0) {
          resolve(false);
          clearInterval(gameCycle);
        } else if (this.exit.isPlayerStepped(this.player)) {
          resolve(true);
          clearInterval(gameCycle);
        }
      }, 10);
      this.draw();
    });
  }

  restoreCanvas() {
    this.ctx.scale(1 / this.mapScale, 1 / this.mapScale);
  }

  draw() {
    this.ctx.clearRect(0, 0, ...this.map.getRealSize().flat());
    this.map.draw(this.ctx);
    this.engine.nonSolid.forEach((o) => o.draw(this.ctx));
    this.engine.solid.forEach((o) => o.draw(this.ctx));
    requestAnimationFrame(this.draw.bind(this));
  }
}
```

Файл ./game/index.js

```
export * from "./Game.js";
export * from "./EventManager.js";
```

Файл ./util.js

```
/** @param {string} name
 * @param {number} time */
export function saveRecord(name, time) {
  let records = getRecords();
  records.push([name, time]);
  localStorage.setItem("game.records", JSON.stringify(records));
}

/** @returns {Array<[string, number]>} */
export function getRecords() {
  return JSON.parse(localStorage.getItem("game.records") ?? "[]");
}

/** @param {HTMLTableElement} records */
export function renderRecords(records) {
  let table = records.querySelector("table");
  for (let i = 0; i < table.tBodies.length; i++) {
    table.tBodies.item(i).remove();
  }
  let body = table.createTBody();
  for (let [name, time] of getRecords().sort((a, b) => a[1] - b[1])) {
    let row = document.createElement("tr"),
      nameCol = document.createElement("td"),
      timeCol = document.createElement("td");
    nameCol.appendChild(document.createTextNode(name));
    timeCol.appendChild(document.createTextNode(`${time} сек.`));
    row.appendChild(nameCol);
    row.appendChild(timeCol);
    body.appendChild(row);
  }
}

/** @param {HTMLElement} elem */
export function hide(elem) {
  elem.style.display = "none";
}

/** @param {HTMLElement} elem */
export function show(elem) {
  elem.style.display = "block";
}

/** @param {HTMLElement} elem */
export function disappear(elem) {
  elem.style.opacity = "0";
}

/** @param {HTMLElement} elem */
export function appear(elem) {
  elem.style.opacity = "1";
}

/** @param {number} time
 * @returns {Promise<void>} */
export function delay(time) {
  return new Promise((resolve) => {
    setTimeout(() => resolve(), time);
  });
}
```

Файл ./index.js

```
import { TileSet } from "./core";
import { createGameObject, SoundManager } from "./engine";
import { GameMap, parseMap } from "./map";
import { Game } from "./game";
import {
  appear,
  delay,
  disappear,
  hide,
  renderRecords,
  saveRecord,
  show,
} from "./util.js";

const ANIM_DELAY = 300;

const canvas = document.getElementById("canvas"),
  input = document.getElementById("inputSection"),
  end = document.getElementById("end"),
  winMsg = document.getElementById("winMsg"),
  gameOverMsg = document.getElementById("gameOverMsg"),
  records = document.getElementById("records");

const ctx = canvas.getContext("2d");
let size = Math.min(window.innerWidth, window.innerHeight);
[ctx.canvas.width, ctx.canvas.height] = [size, size];
ctx.font = "30px KJV1611";
ctx.fillStyle = "red";
ctx.save();

const levels = ["level1.tmj", "level2.tmj", "level3.tmj"];
const assets = "./assets";

[input, canvas, end, records].map((e) => {
  disappear(e);
  hide(e);
});
[winMsg, gameOverMsg].map(hide);

let playerName = "";

/** @param {TileSet} ts */
async function startGame(ts) {
  let time = Date.now();
  let res = false;
  for (let i = 0; i < levels.length; i++) {
    const [field, objects] = await parseMap(`${assets}/${levels[i]}`);
    let game = new Game(
      ctx,
      ts,
      new GameMap(ts, field),
      objects.objects.map((o) => createGameObject(o, ts))
    );
    appear(canvas);
    res = await game.startGame();
    disappear(canvas);
    await delay(ANIM_DELAY);
    game.restoreCanvas();
    if (!res) break;
  }
  hide(canvas);
  if (res) {
    show(winMsg);
```

```
    SoundManager.sounds.win.play(0.5);
    saveRecord(playerName, (Date.now() - time) / 1000);
  } else {
    show(gameOverMsg);
    SoundManager.sounds.dead.play(0.5);
  }
  show(end);
  await delay(ANIM_DELAY);
  appear(end);
}

(async function () {
  await SoundManager.load(`${assets}/sounds`);
  const ts = new TileSet(assets, "tileset.tsj");
  await ts.load();

  show(input);
  appear(input);
  show(document.body);
  input.querySelector("button").addEventListener("click", (ev) => {
    playerName = input.querySelector("input").value;
    disappear(input);
    setTimeout(() => {
      hide(input);
      show(canvas);
      startGame(ts);
    }, ANIM_DELAY);
  });

  end.querySelector("button").addEventListener("click", (ev) => {
    disappear(end);
    setTimeout(() => {
      hide(end);
      show(records);
      appear(records);
      renderRecords(records);
    }, ANIM_DELAY);
  });
})();
```

Файл ./index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Game</title>

    <link rel="stylesheet" href="main.css">
</head>
<body>
<canvas id="canvas"></canvas>

<section id="end">
    <div class="center">
        <div>
            <span id="gameOverMsg">Игра окончена</span>
            <span id="winMsg">Победа</span>
            <br>
            <button>Перейти к рекордам</button>
        </div>
    </div>
```

```
</section>

<section id="inputSection">
    <div class="center">
        <div>
            <label for="nameInput">
                <span>В</span>ведите имя игрока:
            </label>
            <br>
            <input type="text" id="nameInput">
            <br>
            <button><span>Н</span>ачать</button>
        </div>
    </div>
</section>

<section id="records">
    <span>Т</span>аблица рекордов:
    <table>
        <thead>
        <tr>
            <td><span>И</span>мя игрока</td>
            <td><span>В</span>ремя</td>
        </tr>
        </thead>
    </table>
</section>

<script src="index.js" type="module"></script>
</body>
</html>
```

Файл ./main.css

```
@font-face {
    font-family: 'KJV1611';
    font-style: normal;
    font-weight: normal;
    src: url("./KJV1611.otf") format("opentype");
}

body {
    margin: 0;
    font-family: 'KJV1611', sans-serif;
    text-align: center;
    font-size: 2em;
    color: saddlebrown;
    display: none;
}

* {
    font: inherit;
    color: inherit;
}

canvas {
    margin: auto;
}

button {
    padding: 0.3em;
    background-color: antiquewhite;
```

```
    border-radius: 0.3em;
    transition: all 0.1s ease-in-out;
}

button:hover {
    cursor: pointer;
}

button:active {
    background-color: bisque;
    scale: 0.98;
}

.center {
    display: block;
    margin: auto;
}

body > * {
    transition: all 0.29s ease-in-out;
}

.center {
    display: flex;
    align-items: center;
    justify-content: center;
    height: 100vh;
}

#end {
    font-size: 2em;
}

#end span {
    font-size: 1.5em;
}

#winMsg {
    color: forestgreen;
}

#gameOverMsg {
    color: firebrick;
}

#inputSection {
    font-size: 2em;
}

#inputSection > div > div > *, #end > div > div > * {
    margin-bottom: 1.5em;
}

#records {
    padding: 2em;
}

#records > table {
    margin: auto;
    border-spacing: 1em;
}

#records > table th, #records > table td {
```

```
    padding: 0.5em;
}

#records span, #inputSection span {
    color: firebrick;
}
```