МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
КАФЕДРА МО ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Web-технологии»

Тема: ТЕТРИС НА JAVASCRIPT

Студент гр. 1303 _____Коренев Д. А.

Преподаватель _____Беляев С. А.

Санкт-Петербург

2023

Цель работы.

Целью работы является изучение работы web-сервера nginx со статическими файлами и создание клиентских JavaScript web-приложений.

Задание.

Необходимо создать web-приложение – игру в тетрис. Основные требования:

– сервер – nginx, протокол взаимодействия – HTTPS версии не ниже 2.0;

– отображается страница для ввода имени пользователя с использованием HTML-элементов <input>;

– статическая страница отображает «стакан» для тетриса с использованием HTML-элемента <canvas>, элемент используется для отображения следующей фигуры, отображается имя пользователя;

– фигуры в игре – классические фигуры тетриса (7 шт. тетрамино);

– случайным образом генерируется фигура и начинает падать в «стакан»

– пользователь имеет возможность двигать фигуру влево и вправо, повернуть на 90 и «уронить»;

– если собралась целая «строка», она должна исчезнуть;

– при наборе некоторого заданного числа очков увеличивается уровень, что заключается в увеличении скорости игры;

– пользователь проигрывает, когда стакан «заполняется», после чего ему отображается локальная таблица рекордов;

– вся логика приложения написана на JavaScript.

Необязательно: оформление с использованием CSS. Постарайтесь сделать такую игру, в которую вам будет приятно играть.

Помните, когда-то эта игра была хитом! Преимуществом будет использование звукового сопровождения событий: падение фигуры,

исчезновение «строки».

Выполнение работы.

Для удобства выполнения работы она была разделена на несколько этапов:

1. Настройка nginx
2. Написание логики игры
    I. Логика ввода имени пользователя
    II. Логика самой игры
3. Написание css стилей

Настройка nginx:

Для начала скачен архив nginx и распакован в рабочую директорию проекта. Для работы с https2 необходимы закрытый и открытый ключи, для генерации ключей использовалась команда:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout example.key -out example.csr
```

Эти ключи перемещены в папку nginx. Конфиг nginx был изменен, смотреть приложение А. Следует обратить внимание на следующие вещи:

Web-сервер настроен на 443 порт - стандартный порт для защищенной связи веб-браузера – с протоколом SSL. Так же необходимо “включить” http2 протокол с помощью строки `http2 on;`. Чтобы убедиться в том, что используется именно этот протокол, можно посмотреть в настройках разработчика – рисунок 1.

Status	Protocol	Type
200	h2	font
200	chrome-extension	script
200	h2	script
200	h2	document
200	h2	stylesheet
200	h2	jpeg

Рисунок 1 http2

Строка `error_page 405 =200 $uri;` необходима для обработки 405 ошибки (она заменяется на код 200, т.е. успешный) для корректного отправки данных на сервер.

Два блока location: для игрового поля и входа в игру.

Написание логики игры:

Ввод имени пользователя:

Файлы login.html код и login.js представлены в приложении Б. Функция saveName записывает значение, введенное в поле input, в localStorage по ключу “name”. Функция getName получает это значение из localStorage. На кнопки подтверждения ввода накладывается лисенер, который вызывает функцию saveName. При запуске скрипта функция getName сразу же вызывается, чтобы вписать раннее введенное имя пользователя в поле ввода. Поле ввода находится внутри формы form с атрибутом action="game_layout.html", поэтому после нажатия на кнопку, открывается окно game_layout.html

Логика игры:

Файл game_layout.html и game.js отвечают за отображение игрового поля, список игроков, текущее значение очков уровня и тп. Представлены в приложении В. После файла скрипта game.js создается объекты:

- config типа Config, с аргументами: значение, возвращаемое функций getName (аналогичная функции из login.js) и объектом Level
- storage типа Storage
- controller типа Controller
- Looper, с аргументами конструктора: функция stopGame, config, storage, controller.

Вызывается функция отображения результатов – showResult, у объекта looper вызывается метод initGame.

Функция showResult()

Класс Looper представлен в приложении Г. В методе initGame получаются элементы из html страницы, устанавливаются нужные значения (например имя игрока) и лисенеры нажатий. В конце вызывается метод startGame, который запускает фрагмент кода в определённом интервале. Важные методы класса Looper:

- restartGame – перезапускает игру, все данные очищаются (принимают начальные значения)
- increaseLevel – повышает уровень игры. Повышение уровня ведет за собой ускорение игры: фигура снижается быстрее. Это сделано путем уменьшения задержки у функции setInterval
- finishGame – заканчивает процесс игры, добавляет результат в список имеющихся, обращается к game, вызывая метод stopGame
- updateView – обновляет изображение состояния “стакана”, а так же количество очков.

За хранение и управление фигурами отвечает класс Playground. В нем есть поле ground – двумерный массив с заданными из конструктора значениями. Методы класса, реализующие его функциональность:

- insertFigureAvailable – возвращает true если заданную фигуру можно вставить по заданной координате, false в противном случае
- setFigure - устанавливает заданную фигуру в качестве текущей
- moveFigure – передвигает фигуру в заданном направлении, если это возможно
- rotateFigure – поворачивает текущую фигуру если это возможно

- fixFigure – устанавливает текущую фигуру в поле (т.е. блокирует ее передвижение), и вызывает метод deleteFullLines
- deleteFullLines – определяет полные линии и удаляет их с помощью метода deleteLine(row), начисляет очки и повышает при необходимости уровень
- getView – возвращает двумерный массив, который несет в себе информацию об уже поставленных фигурах, а также о текущей фигуре.

Управление фигурой установлено в классе Controller, которому можно сообщить объект, к которому будут применяться действия движения (в моем случае это Playground). В нем есть метод listen, который добавляет лисенер нажатий (тип – ‘keydown’) и в зависимости от кнопки, вызывает у playground метод moveFigure, с соответствующим аргументом: MoveLeft, MoveRight, MoveDrop, MoveRotate, а после нажатия обновляет картинку.

За генерацию фигур отвечает класс FigureGenerator. У него есть два метода:

- getRandomFigure – генерирует новую рандомную фигуру
- getNext – возвращает новую фигуру, генерирует следующую и отображает ее в html странице.

Также у этого класса есть поле figureAsset – массив заготовленных матриц – тетрамино.

Класс Figure является сущностью фигуры тетрамино. В конструкторе принимает матрицу (двумерный массив) – татрамино, устанавливает необходимые значения в поля matrix, height, width, size. У него есть два метода:

- rotate – присваивает в поле matrix перевёрнутую по часовой стрелке матрицу
- getRotateMatrix – вращает матрицу фигуры по часовой стрелке и возвращает ее, вращение реализовано путем создания нового двумерного массива и сменой индексов рядов и колонок.

Прочие классы проекта:

- ColorConverter – имеет метод конвертации значений матрицы в нужный цвет
- Level – управляет уровнем игры
- Move, от которого наследуются классы MoveDown, MoveLeft, MoveRight, MoveDrop, MoveRotate
- Result – класс результат, хранит в себе количество очков, имя игрока, уровень игры
- Config – хранит в себе имя игрока и уровень игры

Написание CSS стилей.

Стили для login.html:

Опишу некоторые атрибуты, на которые важно обратить внимание:

- background-image: Устанавливает фоновое изображение с именем 'background.jpg'
- background-size: Устанавливает размер фонового изображения как "cover", что означает, что изображение будет масштабироваться так, чтобы полностью покрыть задний фон
- backdrop-filter: Применяет размытие с радиусом 5px
- display: Устанавливает свойство display в "flex" для элемента body, что позволяет центрировать его содержимое по горизонтали и вертикали.

- justify-content и align-items: Устанавливают значения "center" для выравнивания содержимого по центру горизонтали и вертикали.
- height: Устанавливает высоту элемента body на 100% высоты видимой области
- установить цвет можно с помощью RGBA, например RGBA(77, 23, 47, 0.8) для контейнера с классом login-container
- padding: Добавляет внутренний отступ
- margin: Устанавливает внешний отступ
- задать свой кастомный шрифт можно следующим образом:

```
@font-face {
    font-family: 'game_font';
    src:url('press-start-2p-regular.ttf')
format('truetype');
    font-weight: normal;
    font-style: normal;
}
```

Скриншот login экрана изображен на рисунке 2.

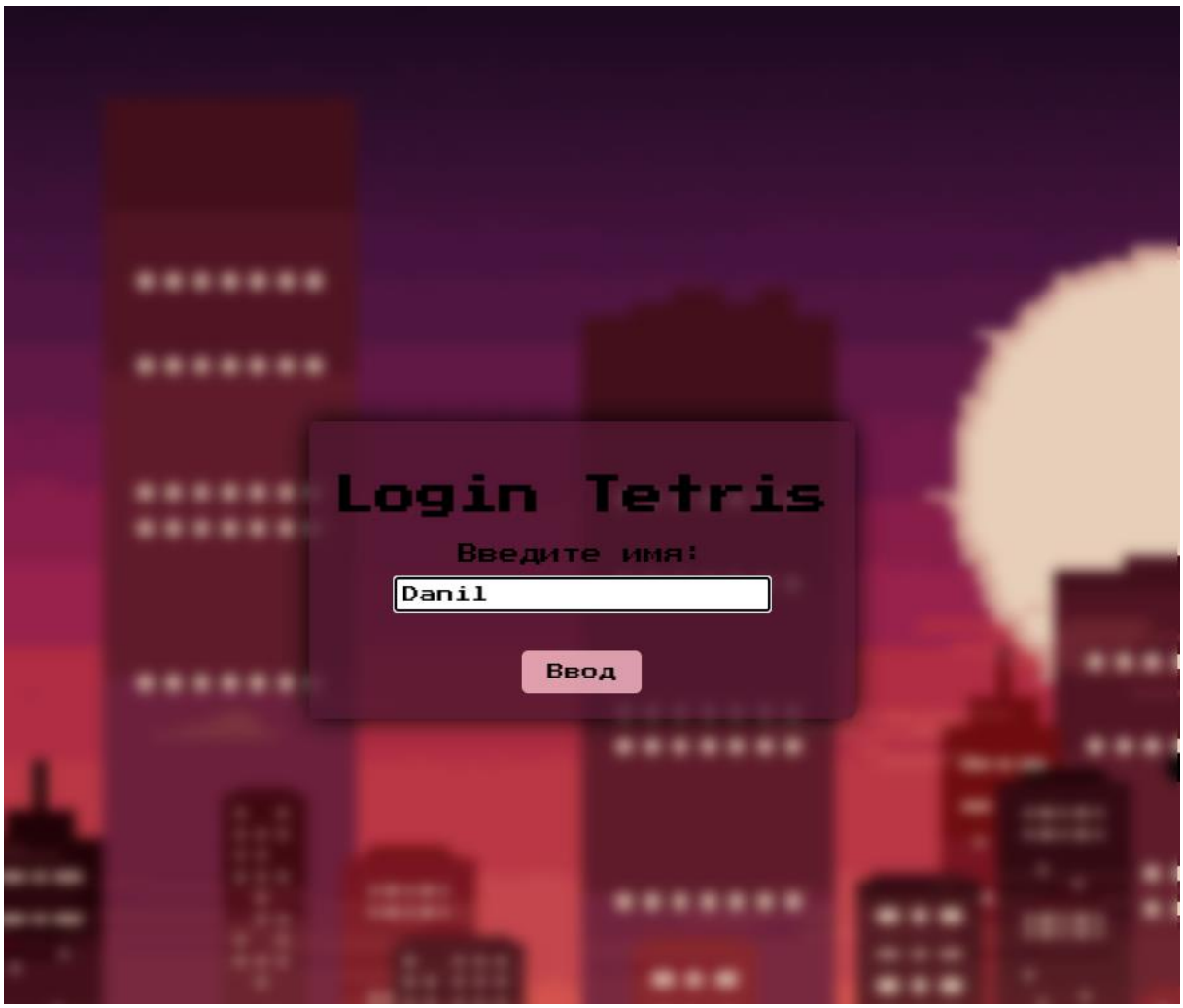

Рисунок 2 экран login

Стили для game_layout.html:

Если необходимо задать атрибут для всех тегов определенного имени, то используется имя тега, если для всех тегов одного класса, то используется .”имя класса”, например .column, если для определенного тега с id, то используется #”id”, например #currentPlayer.

Опишу некоторые атрибуты, на которые важно обратить внимание:

- width: fit-content устанавливает ширину элемента так, чтобы она соответствовала содержимому элемента
- gap устанавливает промежуток (отступ) между элементами внутри контейнера.

Скриншот game_layout экрана изображен на рисунке 3.

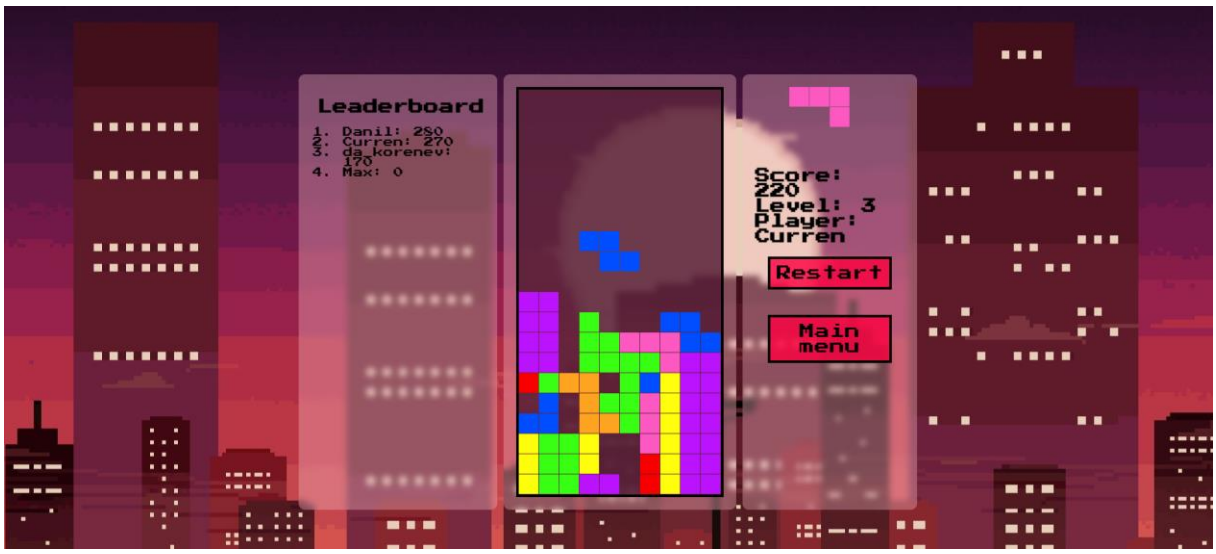

Рисунок 3 экран game_layout

UseCase диаграмма.

UseCase диаграмма представлена на рисунке 4.

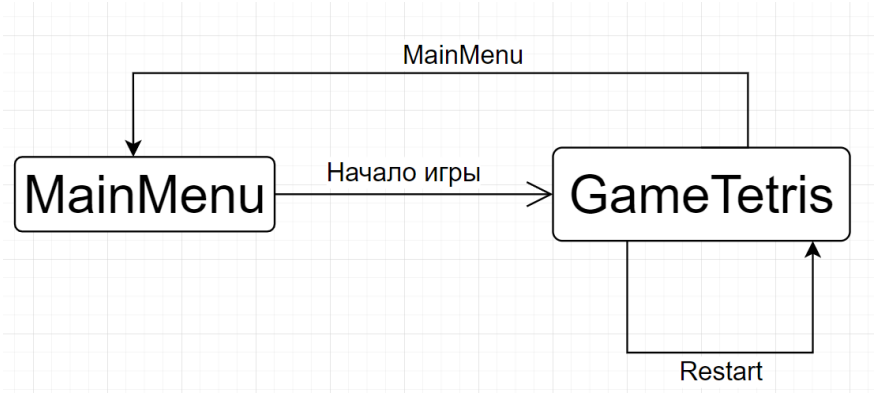

Рисунок 4 UseCase диаграмма

Вывод.

В ходе выполнения работы получен опыт создания и настройки web-сервера *nginx* со статическими файлами и использованием протокола HTTPS 2.0. Создано клиентское web-приложение – игра “Тетрис”, на языке JavaScript.

ПРИЛОЖЕНИЕ А

КОНФИГ NGINX

```
worker_processes  1;

error_log  logs/error.log;
error_log  logs/error.log  notice;
error_log  logs/error.log  info;

pid logs/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include mime.types;
    default_type application/octet-stream;

    sendfile on;
    keepalive_timeout 65;

    server {
        listen 443 ssl;
        http2 on;
        server_name localhost;

        ssl_certificate example.csr;
        ssl_certificate_key example.key;

        error_page 405 =200 $uri;

        location /game_layout.html {
            root
C:/Users/danil/WebstormProjects/tetris_game/game;
            index  game_layout.html;
        }

        location / {
            root
C:/Users/danil/WebstormProjects/tetris_game/game;
            index  login.html;
        }

        error_page 500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }
    }
}
```

ПРИЛОЖЕНИЕ Б
LOGIN.HTML И LOGIN.JS

login.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Login Tetris</title>
    <link rel="stylesheet" href="login.css">
</head>
<body>
<div class="login-container">
    <h1>Login Tetris</h1>
    <form action="game_layout.html" method="post">
        <label>
            Введите имя:<br>
            <input id="username" placeholder="Имя
пользователя"><br>
        </label>
        <input type="submit" value="Ввод" id="submit">
    </form>
</div>
    <script type="module" src="domain/login.js"></script>
</body>
</html>
```

login.js:

```
function saveName() {
    let name = document.getElementById('username').value;
    console.log(`${name}`)
    localStorage.setItem('name', name);
}

function getName() {
    return localStorage.getItem('name');
}

const nameInput = document.getElementById('username');
nameInput.value = getName();

const submit = document.getElementById('submit')
submit.addEventListener('click', saveName)
```

ПРИЛОЖЕНИЕ В

GAME_LAYOUT.HTML И GAME.JS

game_layout.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Tetris game</title>
    <link rel="stylesheet" href="style.css">
    <script type="module" src="presentation/game.js"></script>
</head>
<body>
    <div class="container">
        <div class="column">
            <h2>Leaderboard</h2>
            <ol id="leaderboard" style="text-align: left;"></ol>
        </div>
        <div class="column">
            <canvas width="320" height="640" id="game"></canvas>
            <div id="gameOverOverlay" class="gameOverOverlay">
                <div class="gameOverText">GAME<br>OVER</div>
            </div>
        </div>
        <div class="column">
            <div class="canvasNextFigureContainer">
                <canvas class="canvasNextFigure" width="128"
height="128" id="next_figure" style="width: fit-content"></canvas>
            </div>
            <div id="score">Score: 0</div>
            <div id="level">Level: 1</div>
            <div id="current_player">Name: Player 1</div>
            <div class="buttonContainer">
                <button class="buttonControl" id="button_restart"
style="width: fit-content">Restart</button>
                <button class="buttonControl" id="button_main_menu"
style="width: fit-content">Main menu</button>
            </div>
        </div>
    </div>
</body>
</html>
```

game.js:

```
import {Looper} from "../domain/Looper.js";
import {Storage} from "../data/Storage.js";
import {Controller} from "../domain/objects/Controller.js";
import {Config} from "../domain/objects/Config.js";
import {Level} from "../domain/objects/Level.js";

let config = new Config(getName(), new Level())
let storage = new Storage()
let controller = new Controller()
const limitLeaderboard = 30

let looper = new Looper(
    stopGame,
```

```
    config,
    storage,
    controller
)

function getName() {
    return localStorage.getItem('name');
}

function stopGame(){
    showResult()
}

function showResult(){
    let records = storage.getAllRecords().sort((a, b) => b.score -
a.score)
    const leaderboard = document.getElementById('leaderboard');
    leaderboard.innerHTML = '';


    for(let i = 0; i < Math.min(records.length, limitLeaderboard);
i++){
        const newRecord = document.createElement('li');
        newRecord.textContent = `${records[i].player}:
${records[i].score}`;
        leaderboard.appendChild(newRecord)
    }
}

showResult()
looper.initGame()
```

ПРИЛОЖЕНИЕ Г
LOOPER.JS

```
import {Playground} from "./Playground.js";
import {FigureGenerator} from "./FigureGenerator.js";
import {MoveDown} from "./objects/Move.js";
import {ColorConverter} from "./objects/ColorConverter.js";
import {Result} from "./objects/Result.js";

export class Looper {

    color = new ColorConverter()
    intervalId = null
    storage = null
    constructor(stopGame, config, storage, controller) {
        this.stopGame = stopGame
        this.config = config
        this.storage = storage
        this.controller = controller

        this.playground = new Playground(this)
        this.figureGenerator = new FigureGenerator()
        this.controller.setPlayground(this.playground)

        let levelDocument = document.getElementById('level')
        levelDocument.textContent                                    =
this.formatLevel(this.config.level.timeSpeed)
    }

    setTimeSpeed() {
        return Math.round(1000 / this.config.level.timeSpeed)
    }

    initGame () {
        this.canvas = document.getElementById('game')
        this.contextCanvas = this.canvas.getContext('2d')
        this.scoreDocument = document.getElementById('score')
        this.playerDocument                                          =
document.getElementById('current_player')
        this.playerDocument.textContent                              =
this.formatPlayer(this.config.username)

        this.restartGameButton                                       =
document.getElementById('button_restart')
        this.restartGameButton.addEventListener("click",     ()     =>
{this.restartGame()})

        this.mainMenuButton                                          =
document.getElementById('button_main_menu')
        this.mainMenuButton.addEventListener('click',      ()      =>
{this.mainMenu()})

        this.controller.listen(this)

        this.playground.setFigure(
            this.figureGenerator.getNext(),
            Math.round(Math.random()*(this.playground.width-2))
        )
```

```
        this.startGame(this.setTimeSpeed())
    }

    startGame(time){
        const                    gameOverOverlay                    =
document.getElementById('gameOverOverlay')
        gameOverOverlay.style.display = 'none'
        this.intervalId = setInterval(() => {
            if    (this.playground.figure    ===    null){    //фигура
отсутсвует -> сгенерировать новую
                let              resultGenerateFigure              =
this.playground.setFigure(
                    this.figureGenerator.getNext(),
                    Math.floor(this.playground.width/2)
                )
                if (resultGenerateFigure === false) {
                    this.finishGame()
                }
            } else {
                this.playground.moveFigure(MoveDown)
            }
            this.updateView()
            console.log(this.setTimeSpeed())
        }, time)
    }

    restartGame(){
        clearInterval(this.intervalId)
        this.config.level.setDefault()
        this.playground.restart()
        this.updateView()
        this.startGame(this.setTimeSpeed())
    }

    mainMenu() {
        clearInterval(this.intervalId)
        window.location.replace('https://localhost/')
    }

    increaseLevel(){
        this.config.level.timeSpeed+=1
        let levelDocument = document.getElementById('level')
        levelDocument.textContent                                    =
this.formatLevel(this.config.level.timeSpeed)
        clearInterval(this.intervalId)
        this.startGame(this.setTimeSpeed())
    }

    finishGame() {
        const                    gameOverOverlay                    =
document.getElementById('gameOverOverlay')
        gameOverOverlay.style.display = 'flex'

        clearInterval(this.intervalId)
        let    result    =    new    Result(this.playground.score,
this.config.username, this.config.level)
        this.storage.addNewResult(result)
```

```
        this.stopGame()
    }

    updateView() {
        this.grid = 32
        let view = this.playground.getView();
        this.contextCanvas.clearRect(0,     0,     this.canvas.width,
this.canvas.height)
        for (let row = 0; row < this.playground.height; row++){
            for  (let  column  =  0;  column  <  this.playground.width;
column++){
                if (view[row][column] !== 0) {
                    this.contextCanvas.fillStyle                    =
this.color.color[view[row][column]]
                    this.contextCanvas.fillRect(column  *  this.grid,
row * this.grid, this.grid - 1, this.grid - 1);
                }
            }
        }
        this.scoreDocument.textContent                                  =
this.formatScore(this.playground.score)
    }

    formatScore(score) {
        return `Score: ${score}`
    }

    formatLevel(level) {
        return `Level: ${level}`
    }

    formatPlayer(player){
        return `Player: ${player}`
    }
}
```