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. 1304 _____ Стародубов М.В.

Преподаватель _____ Беляев С.А.

Санкт-Петербург
2023

Цель работы.

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 <*div*> 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» (описание правил см., например, *https://ru.wikipedia.org/wiki/Тетрис*);

– 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*.

Выполнение работы.

Исходный код приложения приведен в приложении А.

На странице *index.html* расположена форма, содержащая поле ввода имети пользователя и кнопку начала игры. Кнопка является неактивной, когда поля ввода не заполнено. После нажатия на кнопку введенное в поле ввода имя

пользователя сохраняется в локальное хранилище и происходит переход на страницу *main.html.*

В файле *main.html* с помощью элемента *canvas* на странице отображается «стакан», в котором падают тетрамино, имя пользователя, текущий счет, а также элементы *div*, в которых отображается следующая и сохраненная фигуры.

Реализован набор классов, каждый из которых хранит информацию о конкретном тетрамино.

Для хранения данных игры реализован класс *TetrisData*. В данном классе расположены поля, хранящие информацию о высоте и ширине игрового поля («стакана»), двумерный массив игрового поля, текущее, следующее и сохраненное тетрамино, позиция текущего тетрамино и его «тени», а также информация для подсчета очков (общее число собранных рядов, текущий уровень и общее число очков).

Для взаимодействия с данными реализован класс *TetrisModel*, реализующий методы, позволяющие перемещать текушее тетрамино в «стакане» в стороны, вниз, а также «уронить» его в самый низ или повернуть его на 90° по часовой стрелке, или против часовой стрелки. Реализован метод, производящий проверку позиции текущего тетрамино, возвращающий истинное логическое значение, если текущее тетрамино не выходит за границы «стакана», и элементы текущего тетрамино не находятся поверх элементов тетрамино, расположенных в стакане, если после изменения положения текущей фигуры данный метод возвращает ложное значение, то данное изменение отменяется. Для автоматического падения тетрамино используестя метод, позволяющий запустить таймер вызова функции падения текущего тетрамино. Когда текущее тетрамино не может совершить движение вниз, вызывается темод, присваивающий текущие позиции элементов тетрамино на соответствующие позиции «стакана», после чего вызываются методы, позволяющие удалить с поля собранные ряды и подсчитать полученное число очков. Для получения объекта случайного тетрамино и нахождения позиции «тени» текущего тетримино также реализованы соответствующие методы. Если

стакан заполнен, то вызывается метод завершения игры, реализующий запись текущего счета в общую таблицу рекордов и совершающий переход на страницу *score.html*.

Для отрисовки сохраненных данных игры реализован класс *TetrisView*. Для обновления отображаемой информации используется метод *update.* Для отображения содержимого «стакана», текущего тетрамино и его «тени» на элементе *canvas* происходит отрисовка изображений соответствующих элементов тетрамино. Отрисовка следующего и сохраненного тетрамино происходит с помощью 4-х элементов *div,* которые в зависимости от типа тетрамино изменяют свое положение и стиль.

Сохранение текущего рекуорда происходит с помощью локального хранилища. В локальном хранилище по ключу *tetris.score_table* в виде *JSON*’а хранится массив структур, содержащих поля *name* и *score*.

Выводы.

В ходе выполнения работы было изучено, как настроить *web*-сервер *nginx*, изучены основы языка программирвоания *JavaScript*, а также язык гипертекстовой разметки *HTML* и каскадной таблицы стилей *CSS*. С использование полученных знаний реализована игра тетрис.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: config.nginx

```
server
{
    listen 8443 ssl http2;

    server_name testdomain.localhost www.testdomain.localhost;

    ssl_certificate /etc/ssl/certs/testdomain.localhost.crt;
    ssl_certificate_key /etc/ssl/private/testdomain.localhost.key;

    location /
    {
        root /home/zephyro/LETI/web-technologies/lab/lab-1/src;
        index index.html;
    }
}
```

Название файла: index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="styles.css">
    <title>Tetris</title>
    <script src="preview_script.js" type="module"></script>
</head>
<body>
<div class="preview">
         <img  src="assets/images/preview-picture.png"  alt="Tetris"
width="800px" height="549px">
    <form id="form" action="main.html" method="get">
        <br><br>
        <label>
                        <input  class="username_input"  id="username"
placeholder="Enter your username...">
        </label>
        <br><br>
                  <input  class="start_button"  disabled="disabled"
id="start_button" type="submit" value=">START">
    </form>
</div>
</body>
</html>
```

Название файла: main.html

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

```
        <link rel="stylesheet" href="styles.css">
        <title>Tetris</title>
        <script src="main_script.js" type="module"></script>
    </head>
    <body>
    <div class="main">
            <img  src="assets/images/main-background.png"  alt="Tetris"
style="position: fixed; top: 0; left: 0; z-index: -1;">
        <label>
              <input class="username" style="top: 47px; left: 560px;"
id="username" disabled value="">
        </label>
        <div class="custom_text" style="top: 128px; left: 530px; width:
260px; text-align: right;" id="score">
            0
        </div>
        <div class="custom_text" style="top: 198px; left: 562px; width:
190px; text-align: center;">
            SAVED
        </div>
        <div class="custom_text" style="top: 462px; left: 562px; width:
190px; text-align: center;">
            NEXT
        </div>
        <canvas class="field" id="field">
        </canvas>
         <div class="tetromino_display" style="left: 575px; top:530px;"
id="next">
        </div>
         <div class="tetromino_display" style="left: 575px; top:265px;"
id="saved">
        </div>
    </div>
    </body>
    </html>
```

Название файла: score.html

```
    <!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <link rel="stylesheet" href="styles.css">
        <title>Tetris</title>
        <script src="score_script.js" type="module"></script>
    </head>
    <body>
    <div class="main">
            <img  src="assets/images/score-background.png"  alt="Tetris"
style="position: fixed; top: 0; left: 0; z-index: -1;">
        <div class="custom_text"
             style="top: 85px; left: 205px; width: 350px; height: 40px;
text-align: center; font-size: 50px;">
            TOP-SCORE
        </div>
          <div class="username" style="top: 630px; left: 285px; text-
align: center;" id="restart">
            >RESTART
```

```
        </div>
        <div class="score_digits">
            0
        </div>
        <div class="score_digits" style="top: 235px;">
            1
        </div>
        <div class="score_digits" style="top: 275px;">
            2
        </div>
        <div class="score_digits" style="top: 315px;">
            3
        </div>
        <div class="score_digits" style="top: 355px;">
            4
        </div>
        <div class="score_digits" style="top: 395px;">
            5
        </div>
        <div class="score_digits" style="top: 435px;">
            6
        </div>
        <div class="score_digits" style="top: 475px;">
            7
        </div>
        <div class="score_digits" style="top: 515px;">
            8
        </div>
        <div class="score_digits" style="top: 555px;">
            9
        </div>
        <div id="names"
              style=" position: fixed; top: 195px; left: 160px; width:
265px; height: 410px;">
        </div>
        <div id="scores"
              style=" position: fixed; top: 195px; left: 450px; width:
265px; height: 410px;">
        </div>
    </div>
    </body>
    </html>
```

Название файла: styles.css

```
    @font-face
    {
        font-family: "Half Bold Pixel-7";
                            src:     local("Half     Bold     Pixel-7"),
url("assets/fonts/half_bold_pixel-7.ttf");
    }

    .preview
    {
        position: absolute;
        top: 50%;
        left: 50%;
        transform: translate(-50%, -50%);
```

```
    text-align: center;
    background-color: #8cad28;
    color: #214231;
    font-family: "Half Bold Pixel-7", monospace;
    width: 800px;
    height: 720px;
    outline: 10px solid;
    border-radius: 5px;
}

.start_button
{
    text-align: center;
    width: 110px;
    height: 25px;
    color: #214231;
    font-size: 25px;
    font-family: "Half Bold Pixel-7", monospace;
    border: 0;
    background-color: #8cad28;
}

.start_button:hover
{
    color: #597926;
}

.start_button:active
{
    color: #597926;
}

.start_button:disabled
{
    color: #597926;
}

.username_input
{
    background-color: #8cad28;
    font-size: 25px;
    color: #214231;
    font-family: "Half Bold Pixel-7", monospace;
    border-bottom-width: 3px;
    border-bottom-color: #597926;
    border-left: 0;
    border-right: 0;
    border-top: 0;
}

.main
{
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    text-align: center;
    background-color: #214231;
```

```
    color: #214231;
    font-family: "Half Bold Pixel-7", monospace;
    width: 800px;
    height: 720px;
    outline: 10px solid;
    border-radius: 5px;
}

.username
{
    position: fixed;
    text-align: center;
    width: 191px;
    height: 25px;
    color: #214231;
    font-size: 30px;
    font-family: "Half Bold Pixel-7", monospace;
    border: 0;
    background-color: #8cad28;
}

.username:hover
{
    color: #597926;
}

.username:active
{
    color: #597926;
}

.custom_text
{
    position: fixed;
    text-align: center;
    width: 191px;
    height: 25px;
    color: #214231;
    font-size: 30px;
    font-family: "Half Bold Pixel-7", monospace;
    border: 0;
    background-color: #8cad28;
}

.field
{
    position: fixed;
    left: 80px;
    width: 400px;
    height: 720px;
}

.empty_cell
{
    position: absolute;
    width: 40px;
    height: 40px;
}
```

```
.tetromino_display
{
    position: fixed;
    width: 165px;
    height: 165px;
}

.tetromino_I
{
    background-image: url("assets/images/tetromino-I.png");
    position: absolute;
    width: 40px;
    height: 40px;
}

.tetromino_J
{
    background-image: url("assets/images/tetromino-J.png");
    position: absolute;
    width: 40px;
    height: 40px;
}

.tetromino_L
{
    background-image: url("assets/images/tetromino-L.png");
    position: absolute;
    width: 40px;
    height: 40px;
}

.tetromino_S
{
    background-image: url("assets/images/tetromino-S.png");
    position: absolute;
    width: 40px;
    height: 40px;
}

.tetromino_Z
{
    background-image: url("assets/images/tetromino-Z.png");
    position: absolute;
    width: 40px;
    height: 40px;
}

.tetromino_O
{
    background-image: url("assets/images/tetromino-O.png");
    position: absolute;
    width: 40px;
    height: 40px;
}

.tetromino_T
{
```

```
    background-image: url("assets/images/tetromino-T.png");
    position: absolute;
    width: 40px;
    height: 40px;
}

.score_digits
{
    position: fixed;
    text-align: center;
    color: #214231;
    font-size: 30px;
    font-family: "Half Bold Pixel-7", monospace;
    border: 0;
    background-color: #8cad28;
    width: 80px;
    height: 40px;
    top: 195px;
    left: 60px;
}

.name_text
{
    position: relative;
    text-align: left;
    width: 265px;
    height: 40px;
    color: #214231;
    font-size: 30px;
    font-family: "Half Bold Pixel-7", monospace;
    border: 0;
    background-color: #8cad28;
}

.score_text
{
    position: relative;
    text-align: right;
    width: 265px;
    height: 40px;
    color: #214231;
    font-size: 30px;
    font-family: "Half Bold Pixel-7", monospace;
    border: 0;
    background-color: #8cad28;
}
```

Название файла: preview_script.js

```
let username_element = document.getElementById("username")
let start_button_element = document.getElementById("start_button")
let form_element = document.getElementById("form")

username_element.addEventListener("input", checkUsernameInput)
form_element.addEventListener("submit", saveUsername)

function checkUsernameInput(event)
{
```

```
    if (username_element.value.length === 0)
    {
        start_button_element.setAttribute("disabled", "disable")
    } else
    {
        start_button_element.removeAttribute("disabled")
    }
}

function saveUsername(event)
{
    localStorage["tetris.username"] = username_element.value
    if (localStorage["tetris.score_table"] === undefined)
    {
        localStorage["tetris.score_table"] = JSON.stringify([])
    }
}
```

Название файла: main_script.js

```
import {TetrisData} from "./TetrisData.js";
import {TetrisView} from "./TetrisView.js";
import {TetrisModel} from "./TetrisModel.js";

let username_element = document.getElementById("username")
username_element.value = localStorage["tetris.username"]

let data = new TetrisData()
let view = new TetrisView(data)
let model = new TetrisModel(data, view)

document.addEventListener("keydown", proceedKeyDown)

function proceedKeyDown(event)
{
    switch (event.key)
    {
        case 'k':
        case 'ArrowUp':
            model.dropCurrentTetromino()
            break
        case 'j':
        case 'ArrowDown':
            model.moveCurrentTetrominoDown()
            break
        case 'h':
        case 'ArrowLeft':
            model.moveCurrentTetrominoLeft()
            break
        case 'l':
        case 'ArrowRight':
            model.moveCurrentTetrominoRight()
            break
        case 'f':
        case ' ':
            model.rotateCurrentTetrominoRight()
            break
        case 'd':
```

```
                model.rotateCurrentTetrominoLeft()
                break
            case 'Shift':
                model.switchToSavedTetromino()
                break
            default:
                break;
        }
    }
```

Название файла: TetrisData.js

```
    /* A class that implements data storage
     */
    export class TetrisData
    {
        constructor()
        {
            this.field_width = 10
            this.field_height = 18
            this.field = new Array(this.field_height)
            for (let i = 0; i < this.field_height; i++)
            {
                                                this.field[i]    =    new
Array(this.field_width).fill("empty_cell")
            }
            this.next_tetromino = undefined
            this.saved_tetromino = undefined
            this.current_tetromino = undefined
            this.current_tetromino_position = {x: 0, y: 0}
            this.shadow_tetromino_position = {x: 0, y: 0}
            this.removed_rows_amount = 0
            this.level = 0
            this.score = 0
        }
    }
```

Название файла: TetrisModel.js

```
    import {TetrominoI, TetrominoJ, TetrominoL, TetrominoO, TetrominoS,
TetrominoT, TetrominoZ} from "./Tetromino.js";

    const maximum_level = 20
    const different_tetromino_amount = 7
    const level_time_step = 50

    /*  A  class  that  implements  data  changes  in  response  to  user
requests
     */
    export class TetrisModel
    {
        constructor(data, view)
        {
            this.data = data
            this.view = view
            this.data.current_tetromino = this.getRandomTetromino()
            this.data.next_tetromino = this.getRandomTetromino()
            this.save_allowed = true
```

```
            this.restartFallTimeout()
            this.calculateShadowTetrominoPosition()
            this.view.update()
        }

        moveCurrentTetrominoDown()
        {
            this.restartFallTimeout()
            this.data.current_tetromino_position.y += 1
            if (!this.checkCurrentTetrominoPosition())
            {
                this.data.current_tetromino_position.y -= 1
                this.placeCurrentTetromino()
                this.switchToNextTetromino()
            }
            this.view.update()
        }

        moveCurrentTetrominoRight()
        {
            this.data.current_tetromino_position.x += 1
            if (!this.checkCurrentTetrominoPosition())
            {
                this.data.current_tetromino_position.x -= 1
            }
            this.calculateShadowTetrominoPosition()
            this.view.update()
        }

        moveCurrentTetrominoLeft()
        {
            this.data.current_tetromino_position.x -= 1
            if (!this.checkCurrentTetrominoPosition())
            {
                this.data.current_tetromino_position.x += 1
            }
            this.calculateShadowTetrominoPosition()
            this.view.update()
        }

        checkCurrentTetrominoPosition()
        {
                                            for     (let     point     of
this.data.current_tetromino.current_state)
            {
                                              let    x    =    point.x    +
this.data.current_tetromino_position.x
                                              let    y    =    point.y    +
this.data.current_tetromino_position.y
                if (y < 0 && !(x < 0 || x >= this.data.field_width || y
>= this.data.field_height))
                {
                    continue
                }
                     if (x < 0 || x >= this.data.field_width || y >=
this.data.field_height ||
                    this.data.field[y][x] !== "empty_cell")
                {
```

```
                    return false
                }
            }
            return true
        }

        getRandomTetromino()
        {
                                switch    (Math.floor(Math.random()    *
different_tetromino_amount))
            {
                case 0:
                    return new TetrominoI()
                case 1:
                    return new TetrominoJ()
                case 2:
                    return new TetrominoL()
                case 3:
                    return new TetrominoS()
                case 4:
                    return new TetrominoZ()
                case 5:
                    return new TetrominoO()
                case 6:
                    return new TetrominoT()
            }
        }

             restartFallTimeout(timeout  =  1000  -  level_time_step  *
this.data.level)
        {
            clearTimeout(this.timeout_id)
                               this.timeout_id    =    setTimeout(()    =>
this.moveCurrentTetrominoDown(), timeout)
        }

        calculateShadowTetrominoPosition()
        {
            let temporary = {...this.data.current_tetromino_position}
            while (this.checkCurrentTetrominoPosition())
            {
                this.data.current_tetromino_position.y += 1
            }
            this.data.current_tetromino_position.y -= 1
                                this.data.shadow_tetromino_position    =
this.data.current_tetromino_position
            this.data.current_tetromino_position = temporary
        }

        dropCurrentTetromino()
        {
            this.restartFallTimeout()
                               this.data.current_tetromino_position    =
this.data.shadow_tetromino_position
            this.placeCurrentTetromino()
            this.switchToNextTetromino()
            this.view.update()
        }
```

```
        placeCurrentTetromino()
        {
            this.save_allowed = true
                                                for (let point of
this.data.current_tetromino.current_state)
            {
                                                  let x = point.x +
this.data.current_tetromino_position.x
                                                  let y = point.y +
this.data.current_tetromino_position.y
                                                   this.data.field[y][x] =
this.data.current_tetromino.name
            }
            this.removeCompleteRows()
        }

        removeCompleteRows()
        {
            let i = 0
            let removed_rows_amount = 0
            while (i < this.data.field_height)
            {
                let flag = false
                for (let j = 0; j < this.data.field_width; j++)
                {
                            flag = flag || this.data.field[i][j] ===
"empty_cell"
                }
                if (!flag)
                {
                    removed_rows_amount += 1
                    this.removeRow(i)
                }
                i += 1
            }
            this.calculateScore(removed_rows_amount)
        }

        removeRow(remove_row_index)
        {
            let temporary = new Array(remove_row_index)
            for (let i = 0; i < remove_row_index; i++)
            {
                temporary[i] = new Array(this.data.field_width)
                for (let j = 0; j < this.data.field_width; j++)
                {
                    temporary[i][j] = this.data.field[i][j]
                    this.data.field[i][j] = "empty_cell"
                }
            }
            for (let i = 0; i < remove_row_index; i++)
            {
                for (let j = 0; j < this.data.field_width; j++)
                {
                    this.data.field[i + 1][j] = temporary[i][j]
                }
            }
```

```
        }

        calculateScore(removed_rows_amount)
        {
            let base = 0
            switch (removed_rows_amount)
            {
                case 1:
                    base = 40
                    break
                case 2:
                    base = 100
                    break
                case 3:
                    base = 300
                    break
                case 4:
                    base = 1200
                    break
                default:
                    break
            }
            this.data.score += base * (this.data.level + 1)
            this.data.removed_rows_amount += removed_rows_amount
            // level increases every ten removed rows
                                                    this.data.level      =
Math.min(Math.floor(this.data.removed_rows_amount / 10), maximum_level)
        }

        switchToNextTetromino()
        {
            this.data.current_tetromino_position = {x: 0, y: 0}
            this.data.current_tetromino = this.data.next_tetromino
            this.data.next_tetromino = this.getRandomTetromino()
            if (!this.checkCurrentTetrominoPosition())
            {
                this.gameOver()
            }
            this.calculateShadowTetrominoPosition()
        }

        switchToSavedTetromino()
        {
            if (!this.save_allowed)
            {
                return
            }
            this.save_allowed = false
            if (this.data.saved_tetromino === undefined)
            {
                this.data.saved_tetromino = this.data.current_tetromino
                this.switchToNextTetromino()
                this.view.update()
                return
            }
            this.data.current_tetromino_position = {x: 0, y: 0}
            let temporary = this.data.current_tetromino
            this.data.current_tetromino = this.data.saved_tetromino
```

```
            this.data.saved_tetromino = temporary
            this.calculateShadowTetrominoPosition()
            this.view.update()
        }

        rotateCurrentTetrominoRight()
        {
            this.data.current_tetromino.rotateRight()
            if (!this.checkCurrentTetrominoPosition())
            {
                this.data.current_tetromino.rotateLeft()
            }
            this.calculateShadowTetrominoPosition()
            this.view.update()
        }

        rotateCurrentTetrominoLeft()
        {
            this.data.current_tetromino.rotateLeft()
            if (!this.checkCurrentTetrominoPosition())
            {
                this.data.current_tetromino.rotateRight()
            }
            this.calculateShadowTetrominoPosition()
            this.view.update()
        }

        gameOver()
        {
            console.log(localStorage["tetris.score_table"])
                                                let     temporary     =
JSON.parse(localStorage["tetris.score_table"])
            temporary.push({
                name: localStorage["tetris.username"],
                score: this.data.score
            })
                                    localStorage["tetris.score_table"]   =
JSON.stringify(temporary)
            window.location = "score.html"
        }
    }
```

Название файла: TetrisView.js

```
    const tetromino_size = 4
    const cell_size = 40

    /* A class that implements rendering
     */
    export class TetrisView
    {
        constructor(data)
        {
            this.data = data
            this.field_element = document.getElementById("field")
                   this.field_element.width  =  this.data.field_width  *
cell_size
```

```
                this.field_element.height  =  this.data.field_height  *
cell_size
            this.field_context = this.field_element.getContext('2d')
            this.tetromino_images = new Map()
            this.initializeTetrominoImages()
            this.next = new Array(tetromino_size)
            this.initializeNext()
            this.saved = new Array(tetromino_size)
            this.initializeSaved()
            this.score = document.getElementById("score")
        }

        initializeTetrominoImages()
        {
            for (let value of 'IJLOSTZ')
            {
                    this.tetromino_images.set(`tetromino_${value}`,  new
Image())
                 this.tetromino_images.get(`tetromino_${value}`).src =
`./assets/images/tetromino-${value}.png`
            }
        }

        initializeNext()
        {
            let next_element = document.getElementById("next")
            for (let i = 0; i < this.next.length; i++)
            {
                this.next[i] = document.createElement("div")
                this.next[i].setAttribute("class", "empty_cell")
                next_element.appendChild(this.next[i])
            }
        }

        initializeSaved()
        {
            let saved_element = document.getElementById("saved")
            for (let i = 0; i < this.saved.length; i++)
            {
                this.saved[i] = document.createElement("div")
                this.saved[i].setAttribute("class", "empty_cell")
                saved_element.appendChild(this.saved[i])
            }
        }

        update()
        {
            this.updateField()
            if (this.data.current_tetromino !== undefined)
            {
                this.updateShadowTetromino()
                this.updateCurrentTetromino()
            }
            if (this.data.next_tetromino !== undefined)
            {
                this.updateNextTetromino()
            }
            if (this.data.saved_tetromino !== undefined)
```

```
            {
                this.updateSavedTetromino()
            }
            this.score.innerText = `${this.data.score}`
        }

        updateField()
        {
                                    this.field_context.clearRect(0,    0,
this.field_element.width, this.field_element.height)
            for (let i = 0; i < this.data.field_height; i++)
            {
                for (let j = 0; j < this.data.field_width; j++)
                {
                            if (this.data.field[i][j] === "empty_cell")
continue
                    let x = j * cell_size
                    let y = i * cell_size
                     this.field_context.drawImage(this.tetromino_images.
get(this.data.field[i][j]), x, y, cell_size, cell_size)
                }
            }
        }

        updateCurrentTetromino()
        {
                                            for     (let     point     of
this.data.current_tetromino.current_state)
            {
                                              let    x    =    (point.x    +
this.data.current_tetromino_position.x) * cell_size
                                              let    y    =    (point.y    +
this.data.current_tetromino_position.y) * cell_size
                this.field_context.drawImage(this.tetromino_images.get(
this.data.current_tetromino.name), x, y, cell_size, cell_size)
            }
        }

        updateShadowTetromino()
        {
                                            for     (let     point     of
this.data.current_tetromino.current_state)
            {
                                              let    x    =    (point.x    +
this.data.shadow_tetromino_position.x) * cell_size
                                              let    y    =    (point.y    +
this.data.shadow_tetromino_position.y) * cell_size
                this.field_context.globalAlpha = 0.5
                 this.field_context.drawImage(this.tetromino_images.get(
this.data.current_tetromino.name), x, y, cell_size, cell_size)
                this.field_context.globalAlpha = 1
            }
        }

        updateNextTetromino()
        {
            for (let i = 0; i < this.next.length; i++)
            {
```

```
                                        this.next[i].setAttribute("class",
this.data.next_tetromino.name)
                // in the top and left positions added offset relative
to fixation position
                                            let   top   =   cell_size   *
(this.data.next_tetromino.current_state[i].y + 1)
                                           let   left   =   cell_size   *
(this.data.next_tetromino.current_state[i].x - 3)
                this.next[i].style.top = `${top}px`
                this.next[i].style.left = `${left}px`
            }
        }

        updateSavedTetromino()
        {
            for (let i = 0; i < this.saved.length; i++)
            {
                                       this.saved[i].setAttribute("class",
this.data.saved_tetromino.name)
                // in the top and left positions added offset relative
to fixation position
                                            let   top   =   cell_size   *
(this.data.saved_tetromino.current_state[i].y + 1)
                                           let   left   =   cell_size   *
(this.data.saved_tetromino.current_state[i].x - 3)
                this.saved[i].style.top = `${top}px`
                this.saved[i].style.left = `${left}px`
            }
        }
    }
```

Название файла: Tetromino.js

```
    /*  A  set  of  classes  that  store  information  about  various
tetrominoes and have a single interface
     */
    export class TetrominoI
    {
        constructor()
        {
            this.name = "tetromino_I"
            this.states = []
            this.states[0] = [{x: 3, y: 1}, {x: 4, y: 1}, {x: 5, y: 1},
{x: 6, y: 1}]
             this.states[1] = [{x: 4, y: -1}, {x: 4, y: 0}, {x: 4, y:
1}, {x: 4, y: 2}]
            this.current_state_index = 0
        }

        get current_state()
        {
            return this.states[this.current_state_index]
        }

        rotateRight()
        {
             this.current_state_index = (this.current_state_index + 1) %
this.states.length
```

```
        }

        rotateLeft()
        {
             this.current_state_index = (this.current_state_index - 1 +
this.states.length) % this.states.length
        }
    }

    export class TetrominoJ
    {
        constructor()
        {
            this.name = "tetromino_J"
            this.states = []
            this.states[0] = [{x: 3, y: 1}, {x: 4, y: 1}, {x: 5, y: 1},
{x: 5, y: 2}]
            this.states[1] = [{x: 4, y: 0}, {x: 4, y: 1}, {x: 4, y: 2},
{x: 3, y: 2}]
            this.states[2] = [{x: 3, y: 0}, {x: 3, y: 1}, {x: 4, y: 1},
{x: 5, y: 1}]
            this.states[3] = [{x: 4, y: 0}, {x: 5, y: 0}, {x: 4, y: 1},
{x: 4, y: 2}]
            this.current_state_index = 0
        }

        get current_state()
        {
            return this.states[this.current_state_index]
        }

        rotateRight()
        {
            this.current_state_index = (this.current_state_index + 1) %
this.states.length
        }

        rotateLeft()
        {
             this.current_state_index = (this.current_state_index - 1 +
this.states.length) % this.states.length
        }
    }

    export class TetrominoL
    {
        constructor()
        {
            this.name = "tetromino_L"
            this.states = []
            this.states[0] = [{x: 3, y: 1}, {x: 4, y: 1}, {x: 5, y: 1},
{x: 3, y: 2}]
            this.states[1] = [{x: 4, y: 0}, {x: 4, y: 1}, {x: 4, y: 2},
{x: 3, y: 0}]
            this.states[2] = [{x: 5, y: 0}, {x: 3, y: 1}, {x: 4, y: 1},
{x: 5, y: 1}]
            this.states[3] = [{x: 4, y: 0}, {x: 5, y: 2}, {x: 4, y: 1},
{x: 4, y: 2}]
```

```
            this.current_state_index = 0
        }

        get current_state()
        {
            return this.states[this.current_state_index]
        }

        rotateRight()
        {
             this.current_state_index = (this.current_state_index + 1) %
this.states.length
        }

        rotateLeft()
        {
             this.current_state_index = (this.current_state_index - 1 +
this.states.length) % this.states.length
        }
    }

    export class TetrominoS
    {
        constructor()
        {
            this.name = "tetromino_S"
            this.states = []
             this.states[0] = [{x: 4, y: 1}, {x: 5, y: 1}, {x: 3, y: 2},
{x: 4, y: 2}]
             this.states[1] = [{x: 3, y: 0}, {x: 3, y: 1}, {x: 4, y: 1},
{x: 4, y: 2}]
            this.current_state_index = 0
        }

        get current_state()
        {
            return this.states[this.current_state_index]
        }

        rotateRight()
        {
             this.current_state_index = (this.current_state_index + 1) %
this.states.length
        }

        rotateLeft()
        {
             this.current_state_index = (this.current_state_index - 1 +
this.states.length) % this.states.length
        }
    }

    export class TetrominoZ
    {
        constructor()
        {
            this.name = "tetromino_Z"
            this.states = []
```

```
            this.states[0] = [{x: 3, y: 1}, {x: 4, y: 1}, {x: 4, y: 2},
{x: 5, y: 2}]
            this.states[1] = [{x: 4, y: 0}, {x: 4, y: 1}, {x: 3, y: 1},
{x: 3, y: 2}]
            this.current_state_index = 0
        }

        get current_state()
        {
            return this.states[this.current_state_index]
        }

        rotateRight()
        {
             this.current_state_index = (this.current_state_index + 1) %
this.states.length
        }

        rotateLeft()
        {
             this.current_state_index = (this.current_state_index - 1 +
this.states.length) % this.states.length
        }
    }

    export class TetrominoO
    {
        constructor()
        {
            this.name = "tetromino_O"
            this.states = []
            this.states[0] = [{x: 4, y: 1}, {x: 5, y: 1}, {x: 4, y: 2},
{x: 5, y: 2}]
            this.current_state_index = 0
        }

        get current_state()
        {
            return this.states[this.current_state_index]
        }

        rotateRight()
        {
             this.current_state_index = (this.current_state_index + 1) %
this.states.length
        }

        rotateLeft()
        {
             this.current_state_index = (this.current_state_index - 1 +
this.states.length) % this.states.length
        }
    }

    export class TetrominoT
    {
        constructor()
        {
```

```
            this.name = "tetromino_T"
            this.states = []
            this.states[0] = [{x: 3, y: 1}, {x: 4, y: 1}, {x: 5, y: 1},
{x: 4, y: 2}]
            this.states[1] = [{x: 3, y: 1}, {x: 4, y: 1}, {x: 4, y: 0},
{x: 4, y: 2}]
            this.states[2] = [{x: 3, y: 1}, {x: 4, y: 1}, {x: 5, y: 1},
{x: 4, y: 0}]
            this.states[3] = [{x: 4, y: 0}, {x: 4, y: 1}, {x: 5, y: 1},
{x: 4, y: 2}]
            this.current_state_index = 0
        }

        get current_state()
        {
            return this.states[this.current_state_index]
        }

        rotateRight()
        {
             this.current_state_index = (this.current_state_index + 1) %
this.states.length
        }

        rotateLeft()
        {
             this.current_state_index = (this.current_state_index - 1 +
this.states.length) % this.states.length
        }
    }
```

Название файла: scjore_script.js

```
    let restart_element = document.getElementById("restart")
    restart_element.addEventListener("click", restart)
    document.addEventListener("keydown", restartEnter)

    function restart(event)
    {
        window.location = "main.html"
    }

    function restartEnter(event)
    {
        if (event.key === "Enter")
        {
            restart(event)
        }
    }

    let score_table = JSON.parse(localStorage["tetris.score_table"])
    score_table.sort(compare)

    function compare(left, right)
    {
        if (left.score < right.score)
        {
            return 1
```

```
    }
    return -1
}

let names_element = document.getElementById("names")
let scores_element = document.getElementById("scores")

for (let i = 0; i < Math.min(10, score_table.length); i++)
{
    let current_name = document.createElement("div")
    current_name.setAttribute("class", "name_text")
    current_name.innerText = score_table[i].name
    names_element.appendChild(current_name)
    let current_score = document.createElement("div")
    current_score.setAttribute("class", "score_text")
    current_score.innerText = score_table[i].score
    scores_element.appendChild(current_score)
}
```