МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Веб-технологии»

Тема: Тетрис на JavaScript

|  |  |  |
| --- | --- | --- |
| Студент гр. 0304 |  | Асташёнок М.С. |
| Преподаватель |  | Беляев С.А. |

Санкт-Петербург

2022

Цель и задачи:

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

Для достижения поставленной цели требуется решить следующие задачи:

– генерация открытого и закрытого ключей для использования шифрования (<https://www.openssl.org/>);

– настройка сервера nginx для работы по протоколу HTTPS; – разработка интерфейса web-приложения;

– обеспечение ввода имени пользователя;

– обеспечение создания новой фигуры для тетриса по таймеру и ее движение;

– обеспечение управления пользователем падающей фигурой;

– обеспечение исчезновения ряда, если он заполнен;

– по окончании игры – отображение таблицы рекордов, которая хранится в браузере пользователя.

Общая формулировка задачи:

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;

– 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.

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 задания.

Верстка сайта.

Первым делом был сверстан *index.html* файл, который содержит в себе необходимые для функционирования элементы, а также *style.css*, который добавляет к html-элементам стили. Тело (body) файла index.html содержит в себе 2 контейнера: main\_menu\_wrapper для отображения и работы с главным меню и gameplay\_wrapper для игрового экрана.

В ходе верстки был получен следующий результат:



Рисунок 1. – главное меню игры.

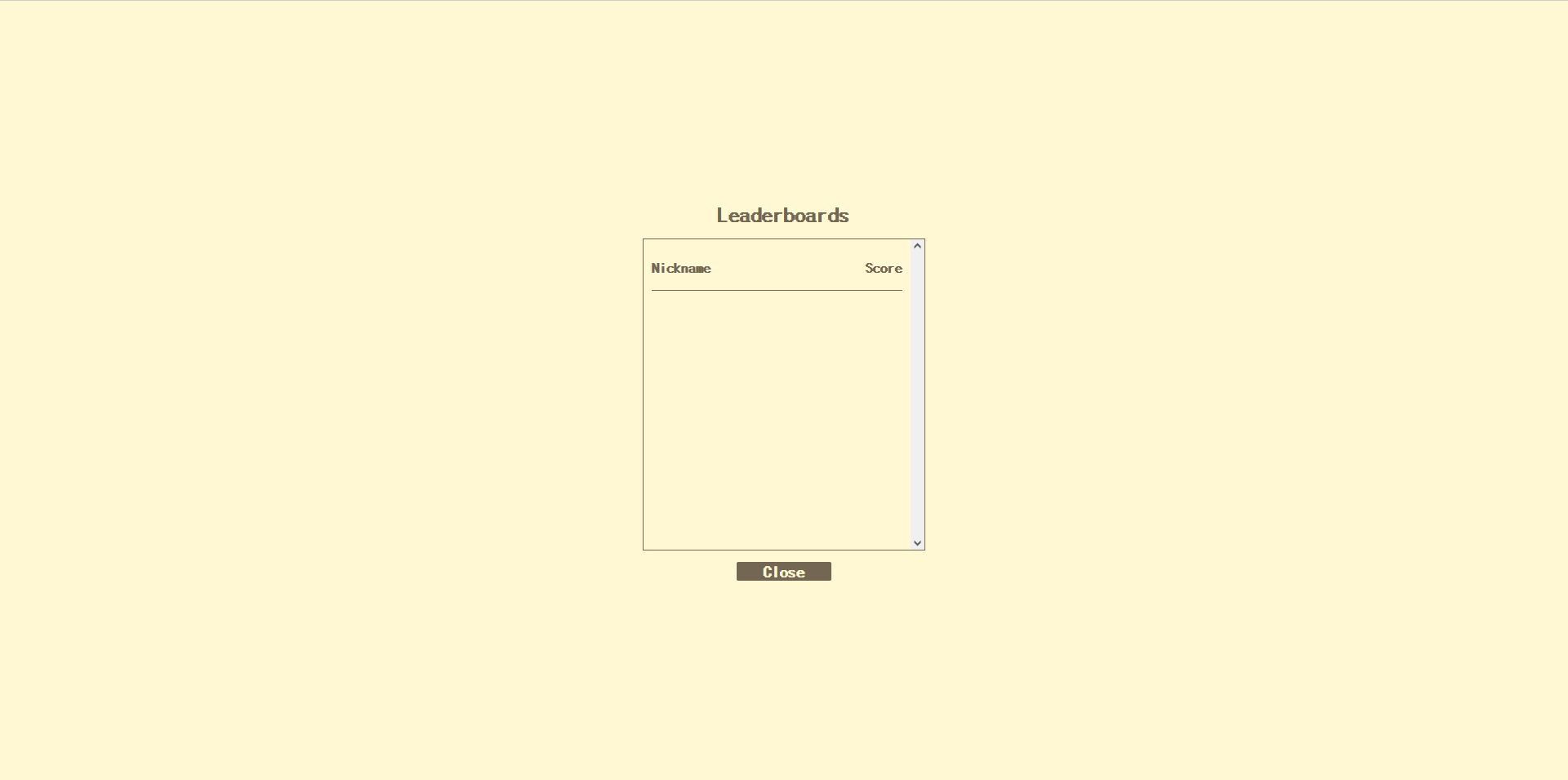


Рисунок 2. – Таблица рекордов (данная таблица пуста, т.к. в браузере пока еще нет записей об играх.



Рисунок 3. – Игровой экран в самом начале игры.

Для стилизации был выбран особый шрифт, загруженный из Интернета под названием Undertale-Battle-Font.ttf.

Добавление JS-скриптов.

Т.к. вся игра помещается в один html файл, было необходимо обработать взаимодействие с различными элементами: кнопками, формами и контейнерами. Для этого был написан и подключен файл *menu\_interaction.js,* который содержит в себе методы взаимодействия с элементами на html-странице.

Шаблоны для игры, т.е. формы тетрамино *tetraminos*, их цвета *colors* и их названия *labels* содержатся в скрипте *templates.js*.

Скрипт *tetramino\_interaction.js* отвечает за создание тетрамино, случайную генерацию их последовательности, проверку на валидность перемещения, расположение тетрамино на поле.

Скрипт *controls.js* отвечает за обработку нажатий клавиш игроком. Также содержит в себе функции по обработке положения фигуры в соответствии с нажатой клавишей.

Для задания игровых правил, изменения уровней сложности, запуске игрового цикла, был написан и подключен скрипт *gameplay\_rules.js*.

Для отрисовки поля и следующего тетрамино был написан и подключен скрипт *graphics.js.*

Подключение NGINX сервера и работа с HTTPS протоколом.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="./public/style.css">

    <title>Tetris</title>

</head>

<body>

    <div id="main\_menu\_wrapper" >

        <div id="main\_menu">

            <h1>Hello, Tetris!</h1>

            <input type="text" id="nickname\_input" placeholder="Enter your nickname">

            <button type="submit" id="nickname\_btn">Play</button>

            <button type="submit" id="leaderboards\_btn">Leaderboards</button>

        </div>

        <div id="leaderboards\_wrapper" style="display: none">

            <h2>Leaderboards</h2>

            <table id="leaderboards\_table">

                <thead>

                    <tr>

                        <th>

                            Nickname

                        </th>

                        <th>

                            Score

                        </th>

                    </tr>

                </thead>

                <tbody id="leaderboards\_table\_body"></tbody>

            </table>

            <button id="leaderboards\_close\_button">Close</button>

        </div>

    </div>

    <div id="gameplay\_wrapper" style="display: none">

        <h1>Hello, Tetris</h1>

        <div id="gameplay\_interface">

            <div id="gameplay\_stats">

                <button type="submit" id="back\_to\_menu">Go to menu</button>

                <h3 id="players\_nickname"></h3>

                <p id="current\_score"></p>

                <p id="current\_level"></p>

                <p>Следующая фигура:</p>

                <canvas id="next\_shape"></canvas>

                <p>Управление:</p>

                <ul id="controls\_description">

                    <li>Клавиша "W" - Поворот на 90 градусов</li>

                    <li>Клавиша "A" - Движение влево</li>

                    <li>Клавиша "S" - Рывок вниз</li>

                    <li>Клавиша "D" - Движение вправо</li>

                </ul>

            </div>

            <canvas id="playground"></canvas>

        </div>

    </div>

    <script src="./scripts/graphics.js"></script>

    <script src="./scripts/templates.js"></script>

    <script src="./scripts/menu\_interaction.js"></script>

    <script src="./scripts/tetramino\_interaction.js"></script>

    <script src="./scripts/gameplay\_rules.js"></script>

    <script src="./scripts/controls.js"></script>

</body>

</html>

Style.css

/\* FONT SETUP \*/

@font-face {

    font-family: 'UBF';

    src: url("Undertale-Battle-Font.ttf");

}

/\* COMMON STYLES \*/

\* {

    padding: 0;

    margin: 0;

    font-family: "UBF";

}

body {

    min-height: 100vh;

    position: relative;

    background-color: #fff8d3;

}

/\* MAIN MENU STYLES \*/

#main\_menu\_wrapper {

    box-sizing: border-box;

    position: absolute;

    top: 50%;

    left: 50%;

    transform: translate(-50%, -50%);

    height: 100%;

    width: 100%;

    display: flex;

    flex-direction: column;

    justify-content: space-around;

    align-items: center;

    color: #736754;

}

#main\_menu {

    height: 25%;

    width: 20%;

    display: flex;

    flex-direction: column;

    justify-content: space-around;

    align-items: center;

    border-radius: 3px;

}

#main\_menu\_wrapper h1 {

    font-weight: 700;

    font-size: 30px;

}

#nickname\_input {

    background: none;

    outline: none;

    border: none;

    font-size: 18px;

    padding: 5px 10px;

    border-radius: 10px;

    color: #736754;

    border-bottom: 1px solid #736754;

}

#nickname\_btn,

#leaderboards\_btn {

    background: none;

    outline: none;

    border: none;

    background-color: #736754;

    width: 45%;

    height: 15%;

    font-weight: 700;

    color: #fff8d3;

    font-size: 18px;

}

#nickname\_btn:hover,

#leaderboards\_btn:hover {

    background-color: #fff8d3;

    color: #736754;

    border: 1px solid #736754;

    transition: .2s;

    cursor: pointer;

}

#leaderboards\_wrapper {

    position: absolute;

    top: 50%;

    left: 50%;

    transform: translate(-50%, -50%);

    z-index: 2;

    background-color: #fff8d3;

    width: 20%;

    height: 50%;

    display: flex;

    flex-direction: column;

    justify-content: space-around;

    align-items: center;

}

#leaderboards\_table {

    height: 80%;

    width: 90%;

    border: 1px solid #736754;

    display: flex;

    flex-direction: column;

    overflow: hidden;

    overflow-y: scroll;

    padding: 10px;

}

#leaderboards\_table thead {

    border-bottom: 1px solid #736754;

    padding-bottom: 2px;

}

#leaderboards\_table tr {

    width: 100%;

    height: 10%;

    display: flex;

    justify-content: space-between;

    margin: 15px 0;

}

#leaderboards\_close\_button {

    background: none;

    outline: none;

    border: none;

    background-color: #736754;

    width: 30%;

    height: 5%;

    font-size: 18px;

    font-weight: 700;

    color: #fff8d3;

    border-radius: 2px;

}

#leaderboards\_close\_button:hover {

    background-color: #fff8d3;

    color: #736754;

    border: 1px solid #736754;

    transition: .2s;

    cursor: pointer;

}

/\* GAMEPLAY MENU STYLES \*/

#gameplay\_wrapper {

    box-sizing: border-box;

    position: absolute;

    top: 50%;

    left: 50%;

    transform: translate(-50%, -50%);

    display: flex;

    flex-direction: column;

    justify-content: space-between;

    align-items: center;

    color:#736754;

}

#gameplay\_wrapper h1 {

    font-size: 2em;

    margin-bottom: 1em;

}

#gameplay\_interface {

    box-sizing: border-box;

    width: 100%;

    height: 90%;

    display: flex;

    align-items: center;

    justify-content: space-between;

}

#gameplay\_stats {

    box-sizing: border-box;

    width: 40%;

    height: 100%;

    display: flex;

    flex-direction: column;

    justify-content: center;

    font-size: 16px;

}

#next\_shape {

    box-sizing: border-box;

    border: 1px solid #736754;

    width: 128px;

    height: 128px;

}

#gameplay\_stats \* {

    margin-bottom: 15px;

}

#playground {

    width: 320px;

    height: 640px;

    border: 1px solid #736754;

    background-image: linear-gradient( 45deg, #ffeec7 25%, transparent 25%, transparent 74%, #ffeec7 75%, #ffeec7), linear-gradient( 45deg,#ffeec7 25%, transparent 25%, transparent 74%, #ffeec7 75%, #ffeec7);

    background-size: 64px 64px;

    background-position: 0 0, 32px 32px;

}

#back\_to\_menu {

    box-sizing: border-box;

    width: 40%;

    padding: 10px 0;

    background: none;

    outline: none;

    border: none;

    padding: 10px;

    background-color: #736754;

    color:#fff8d3;

}

#back\_to\_menu:hover {

    background-color: #fff8d3;

    color:#736754;

    transition: .2s;

    cursor: pointer;

}

#controls\_description {

    list-style: none;

}

Menu\_interaction.js

// \* DOM-ELEMENTS IMPORTS

// Form elements

const nicknameInput      = document.getElementById("nickname\_input")

const nicknameButton     = document.getElementById("nickname\_btn")

const leaderboardsButton = document.getElementById("leaderboards\_btn")

const leaderboardsCloseButton = document.getElementById("leaderboards\_close\_button")

const backToMenuButton = document.getElementById("back\_to\_menu")

// Containers

const mainMenuWrapper = document.getElementById("main\_menu\_wrapper")

const leaderboardsWrapper = document.getElementById("leaderboards\_wrapper")

const leaderboardsTableBody = document.getElementById("leaderboards\_table\_body")

const gameplayWrapper = document.getElementById("gameplay\_wrapper")

// Text boxes

const playerNickname = document.getElementById("players\_nickname")

const currentLevel = document.getElementById("current\_level")

const currentScore = document.getElementById("current\_score")

// Leaderboards initialize

localStorage["tetris.leaderboards"] = JSON.stringify({})

// \* UTILS

const addLeaderboardsNote = (name, score) => {

    let buf = JSON.parse(localStorage["tetris.leaderboards"])

    buf[name] = score;

    localStorage["tetris.leaderboards"] = JSON.stringify(buf)

}

// \* MAIN MENU SETUP

// Check if there are any already written nickname

if (localStorage["tetris.nickname"]) {

    nicknameInput.value = localStorage["tetris.nickname"]

}

// "Play" button

nicknameButton.addEventListener("click", () => {

    // Clear canvases

    nextShapeContext.clearRect(0, 0, nextShapeCanvas.width, nextShapeCanvas.height);

    playgroundContext.clearRect(0, 0, playgroundCanvas.width, playgroundCanvas.height);

    // Save nickname

    enteredNickname = nicknameInput.value

    processedNickname = enteredNickname.replace(/[.,\/#!$%\^&\\*;:{}=\-\_`~()!?]/g,"")

    let leaderboardsDatabase = JSON.parse(localStorage["tetris.leaderboards"])

    // Check if nickname is empty or incorrect

    if (!enteredNickname || processedNickname != enteredNickname ) {

        alert("Please enter correct name!")

        return

    }

    // Check if nickname is exsists

    if (leaderboardsDatabase[enteredNickname]) {

        alert("Please enter unique name!")

        return

    }

    localStorage["tetris.nickname"] = enteredNickname

    // Hide main menu and open gameplay screen

    mainMenuWrapper.style.display = "none"

    gameplayWrapper.style.display = "flex"

    // Player nickname demo

    playerNickname.innerHTML = "Игрок " + enteredNickname

    currentLevel.innerHTML   = "Текущий уровень: 1"

    currentScore.innerHTML   = "Счет: 0"

    // Start game

    startGame()

})

// "Leaderboards" button

leaderboardsButton.addEventListener("click", () => {

    // Clear previous nodes

    leaderboardsTableBody.innerHTML = ""

    // Load local data

    let leaderboardsDB = JSON.parse(localStorage["tetris.leaderboards"])

    // Sort data

    let leaderboardsDBArray = []

    for (let key in leaderboardsDB) {

        leaderboardsDBArray.push([key, leaderboardsDB[key]])

    }

    leaderboardsDBArray.sort((a, b) => {

        return b[1] - a[1];

    })

    leaderboardsDBArray.forEach( pair => {

        const nameField  = document.createElement("td")

        const scoreField = document.createElement("td")

        const row        = document.createElement("tr")

        let name = pair[0]

        nameField.innerText  = name

        scoreField.innerText = leaderboardsDB[name]

        row.appendChild(nameField)

        row.appendChild(scoreField)

        leaderboardsTableBody.appendChild(row)

    })

    // Open leaderboards

    leaderboardsWrapper.style.display = "flex"

})

// "Close" leaderboards button

leaderboardsCloseButton.addEventListener("click", () => {

    // Close leaderboards

    leaderboardsWrapper.style.display = "none"

})

// "Go to menu" button

backToMenuButton.addEventListener("click", () => {

    // Stop current game, save score

    gameOver = true;

    // Close game

    gameplayWrapper.style.display = "none"

    // Open menu

    mainMenuWrapper.style.display = "flex"

})

Templates.js

// Tetramino patterns

const tetraminos = {

    'I': [

      [0,0,0,0],

      [1,1,1,1],

      [0,0,0,0],

      [0,0,0,0]

    ],

    'J': [

      [1,0,0],

      [1,1,1],

      [0,0,0],

    ],

    'L': [

      [0,0,1],

      [1,1,1],

      [0,0,0],

    ],

    'O': [

      [1,1],

      [1,1],

    ],

    'S': [

      [0,1,1],

      [1,1,0],

      [0,0,0],

    ],

    'Z': [

      [1,1,0],

      [0,1,1],

      [0,0,0],

    ],

    'T': [

      [0,1,0],

      [1,1,1],

      [0,0,0],

    ]

};

// Tetramino coloring

const colors = {

    'I': '#82CCDD',

    'O': '#FAD390',

    'T': '#786FA6',

    'S': '#B8E994',

    'Z': '#E66767',

    'J': '#4A69BD',

    'L': '#F19066',

};

// Tetramino labels

const labels = ['I', 'O', 'T', 'S', 'Z', 'J', 'L']

Tetramino\_interaction.js

// Playground setup

let playground = []

// Initial playground setup

const blankPlayground = () => {

    for (let row = -2; row < 20; row++) {

        playground[row] = []

        for (let col = 0; col < 10; col++) {

            playground[row][col] = 0

        }

    }

}

// Tetramino generator

class Tetramino {

    constructor(name, matrix, row, col) {

        this.name   = name;

        this.matrix = matrix;

        this.row    = row;

        this.col    = col;

    }

}

// Tetramino sequence

let tetraminoSequence = []

// Current and next tetramino

// We need to store next tetramino name to show

// it in the canvas#next-shape DOM object

let tetramino, nextTetraminoName;

// Initial tetramino setup

const blankTetramino = () => {

    tetramino = null;

    nextTetraminoName = null;

    tetraminoSequence = [];

}

// Random integer generator

const getRandomInteger = (min, max) => {

    let rand = min + Math.random() \* (max + 1 - min);

    return Math.floor(rand);

}

// Tetramino sequence generator (randomly)

const generateSequence = () => {

    // Copying tetramino labels from "database"

    let tetraminoLabels = labels.slice();

    // Removing letters from tetraminoLabels array

    // during filling the sequence

    while (tetraminoLabels.length) {

        const randomInteger = getRandomInteger(0, tetraminoLabels.length - 1);

        tetraminoSequence.push(tetraminoLabels.splice(randomInteger, 1)[0])

    }

}

// Get next tetramino

const getNextTetramino = () => {

    // If there is no prepared tetramino sequence

    if (!tetraminoSequence.length) generateSequence()

    nextTetraminoName = tetraminoSequence.splice(0, 1)[0]

}

// Convert tetramino name to playable object

const nameToTetramino = name => {

    // Take pattern of current tetramino

    const matrix = tetraminos[name]

    // I starts from 21 row (offset -1), other - from 22 (offset -2)

    const row = name === "I" ? -1 : -2;

    // I and O starts from the middle

    // Other - to the left

    const col = playground[0].length / 2 - Math.ceil(matrix[0].length / 2);

    // Return object

    return new Tetramino(name, matrix, row, col)

}

// Tetramino rotation function

const rotateTetramino = matrix => {

    const maxIndex = matrix.length - 1;

    const rotated  = matrix.map((row, i) =>

        row.map((value, j) => matrix[maxIndex - j][i])

    );

    return rotated;

}

// Check if we can move tetramino's matrix

// on the current position

const isValidMove = (matrix, startRow, startCol) => {

    for (let row = 0; row < matrix.length; row++) {

        for (let col = 0; col < matrix[0].length; col++) {

            if (

                matrix[row][col] &&

                (

                    startCol + col < 0 ||

                    startCol + col >= playground[0].length ||

                    startRow + row >= playground.length ||

                    playground[startRow + row][startCol + col]

                )

            ) {

                return false;

            }

        }

    }

    return true;

}

// Process of placing tetramino on the board

const placeTetramino = () => {

    // Place tetramino

    for (let row = 0; row < tetramino.matrix.length; row++) {

        for (let col = 0; col < tetramino.matrix[0].length; col++) {

            if (tetramino.matrix[row][col]) {

                // Check if tetramino overcomed the upper border

                if (tetramino.row + row < 0) return alertGameOver();

                playground[tetramino.row + row][tetramino.col + col] = tetramino.name;

            }

        }

    }

    // Check if there any filled rows

    for (let row = playground.length - 1; row >= 0;) {

        if (playground[row].every(cell => !!cell)) {

            for (offsetRow = row; offsetRow >= 0; offsetRow--) {

                for (offsetCol = 0; offsetCol < playground[0].length; offsetCol++) {

                    playground[offsetRow][offsetCol] = playground[offsetRow - 1][offsetCol];

                }

            }

            // Increase score

            score += 50

        } else {

            row--;

        }

    }

    // Switch tetraminos

    tetramino = nameToTetramino(nextTetraminoName);

    getNextTetramino();

    // Increase score

    score += 10;

    // Change score

    currentScore.innerHTML = "Счет: " + score;

}

Controls.js

// Event handlers

const rotateEvent = () => {

    const matrix = rotateTetramino(tetramino.matrix)

    if (isValidMove(matrix, tetramino.row, tetramino.col)) tetramino.matrix = matrix;

}

const moveRightEvent = () => {

    const columnToMove = tetramino.col + 1;

    if (isValidMove(tetramino.matrix, tetramino.row, columnToMove)) tetramino.col = columnToMove;

}

const moveLeftEvent = () => {

    const columnToMove = tetramino.col - 1;

    if (isValidMove(tetramino.matrix, tetramino.row, columnToMove)) tetramino.col = columnToMove;

}

const fallDownEvent = () => {

    let currRow = tetramino.row;

    while (true) {

        currRow++;

        if (!isValidMove(tetramino.matrix, currRow, tetramino.col)) {

            placeTetramino();

            return;

        }

        tetramino.row = currRow;

    }

}

const stepDownEvent = () => {

    const currRow = tetramino.row + 1;

    if (!isValidMove(tetramino.matrix, currRow, tetramino.col)) {

        placeTetramino();

        return;

    }

    tetramino.row = currRow

}

// Controls

document.addEventListener("keydown", (e) => {

    // Check if game is over

    if (gameOver) return;

    // Code of the pressed key (common for upper- and lowercase)

    const keyCode = e.code;

    switch (keyCode){

        case "KeyA": moveLeftEvent();  break;

        case "KeyD": moveRightEvent(); break;

        case "KeyW": rotateEvent();    break;

        case "KeyS": fallDownEvent();  break;

    }

    drawTetris()

});

Graphics.js

// Draw playground

const playgroundCanvas = document.getElementById("playground")

const playgroundContext = playgroundCanvas.getContext('2d')

playgroundCanvas.width  = 320

playgroundCanvas.height = 640

const nextShapeCanvas = document.getElementById("next\_shape")

const nextShapeContext = nextShapeCanvas.getContext('2d')

nextShapeCanvas.width  = 128

nextShapeCanvas.height = 128

// Grid size (px)

const gridSize = 32

// Draw next shape

const drawNext = () => {

    // Clear canvas

    nextShapeContext.clearRect(0, 0, nextShapeCanvas.width, nextShapeCanvas.height);

    // Draw next shape

    if (nextTetraminoName) {

        nextShapeContext.fillStyle = colors[nextTetraminoName]

        const matrix = tetraminos[nextTetraminoName]

        // For centring the next shape

        let increment;

        switch(matrix.length) {

            case 4: increment = 0; break;

            case 3: increment = gridSize/2; break;

            case 2: increment = gridSize; break;

        }

        for (let row = 0; row < matrix.length; row++) {

            for (let col = 0; col < matrix[0].length; col++) {

                if (matrix[row][col]) {

                    nextShapeContext.fillRect(col \* gridSize +  increment, row \* gridSize + increment, gridSize - 1, gridSize - 1);

                }

            }

        }

    }

}

// Draw playground

const drawTetris = () => {

    // Clear canvas

    playgroundContext.clearRect(0, 0, playgroundCanvas.width, playgroundCanvas.height);

    // Draw playground

    for (let row = 0; row < playground.length; row++) {

        for (let col = 0; col < playground[0].length; col++) {

            const name = playground[row][col]

            if (name) {

                playgroundContext.fillStyle = colors[name];

                playgroundContext.fillRect(col \* gridSize, row \* gridSize, gridSize - 1, gridSize - 1);

            }

        }

    }

    // Draw tetramino

    if (tetramino) {

        playgroundContext.fillStyle = colors[tetramino.name]

        for (let row = 0; row < tetramino.matrix.length; row++) {

            for (let col = 0; col < tetramino.matrix[0].length; col++) {

                if (tetramino.matrix[row][col]) {

                    playgroundContext.fillRect((tetramino.col + col) \* gridSize, (tetramino.row + row) \* gridSize, gridSize - 1, gridSize - 1);

                }

            }

        }

    }

    // Draw next

    drawNext()

}

Gameplay\_rules.js

// \* GAMEPLAY RULES

// Game over bool

let gameOver = false;

// Score counter

let score = 0;

// Game over

const alertGameOver = () => {

    // This flag will be used at the conditioning

    // of game loop

    gameOver = true;

    // Draw Game over screen

    alert("Game Over!")

}

// Score and speed system

let currentLevelNum = 1

const levelSpeed = {

    "1": 1000,

    "2": 800,

    "3": 600,

    "4": 500,

    "5": 400,

    "6": 200,

    "7": 100,

}

const levelBorder = {

    "1": 100,

    "2": 200,

    "3": 300,

    "4": 400,

    "5": 500,

    "6": 600,

    "7": 700

}

// Checks and modifies level

// If true, then we'll update speed

const isNeedToUpdateSpeedAndLevel = () => {

    if (score > levelBorder[currentLevelNum + ''] && currentLevelNum != 7) {

        currentLevelNum++;

        currentLevel.innerHTML = "Текущий уровень: " + currentLevelNum;

        return true;

    }

    return false;

}

// Initial gamerules setup

const blankGameRules = () => {

    currentLevelNum = 1;

    score = 0;

    gameOver = false;

}

// Game start setup

const startGame = () => {

    // Preparings

    blankPlayground()

    blankTetramino()

    blankGameRules()

    // Start gameplay process

    getNextTetramino()

    tetramino = nameToTetramino(nextTetraminoName)

    getNextTetramino()

    let gameLoop;

    const gameIteration = () => {

        stepDownEvent();

        drawTetris();

        // If player overcomed special score border

        if (isNeedToUpdateSpeedAndLevel()) {

            clearInterval(gameLoop)

            gameLoop = setInterval(gameIteration, levelSpeed[currentLevelNum + ""])

        }

        // If game is over

        if (gameOver) {

            // Stop game loop

            clearInterval(gameLoop);

            // Make a note about players score

            addLeaderboardsNote(localStorage["tetris.nickname"], score)

        }

    }

    gameLoop = setInterval(gameIteration, 1000)

}