**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**ОТЧЕТ**

**по лабораторной работе №1**

**по дисциплине «Web-технологии»**

**Тема: Тетрис на JavaScript**

| Студент гр. 1381 |  | Сагидуллин Э.Р. |
| --- | --- | --- |
| Преподаватель |  | Беляев С.А. |

Санкт-Петербург

2023

**Цель работы.**

Изучение работы web-сервера nginx со статическими файлами и создание клиентских JavaScript web-приложений.

**Задание.**

Для достижения поставленной цели требуется решить следующие задачи:

* Генерация открытого и закрытого ключей для использования шифрования
* 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.

**Ход работы.**

**Настройка сервера.**

Были сгенерированы открытый и закрытый ключ для использования шифрования с помощью следующей команды:

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

Для корректной работы файлы с ключами были скопированы в соответствующие директории с учетом используемой ОС.

Содержимое конфигурационного файла Web-сервера nginx:

server {

listen 443 ssl http2;

listen [::]:443 ssl http2;

server\_name localhost;

ssl\_certificate /etc/ssl/certs/example.csr;

ssl\_certificate\_key /etc/ssl/private/example.key;

location / {

root /home/ernest/WebstormProjects/tetris;

index login.html;

}

}

Чтобы запустить nginx, нужно выполнить исполняемый файл. Когда nginx запущен, им можно управлять, вызывая исполняемый файл с параметром -s.

Используется следующий синтаксис:

nginx -s сигнал

Где сигнал может быть одним из нижеследующих:

* stop — быстрое завершение
* quit — плавное завершение
* reload — перезагрузка конфигурационного файла
* reopen — переоткрытие лог-файлов

Доступ к web-приложению осуществляется по адресу <https://localhost>

**Разработка интерфейса.**

Приложение имеет две основные страницы: index.html и login.html. index.html содержит игровой интерфейс, а login.html предоставляет форму для ввода имени пользователя и отображения таблицы рекордов. Для стилизации игры используется CSS, оформляющий элементы интерфейса, включая шрифты и расположение элементов. Игровой интерфейс состоит из игрового поля, информации о текущем счете, уровне и следующем тетромино. Текущее имя игрока также отображается. Страница login.html предоставляет форму для ввода имени пользователя. После ввода имени и нажатия кнопки "Play" игрок переходит на страницу index.html, где начинается игра. После завершения игры последнее сохраненное имя автоматически вставляется в форму.

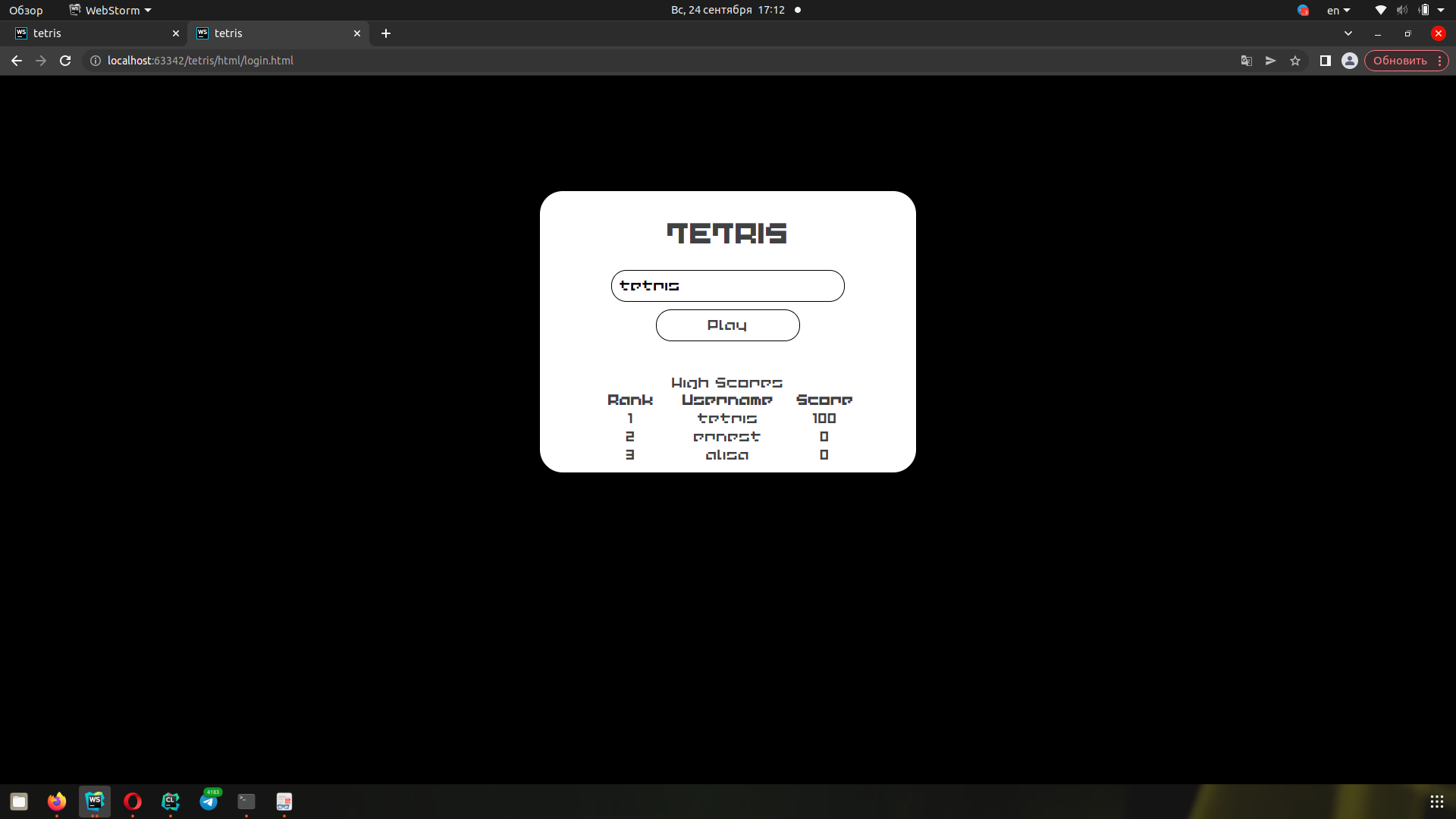


Рисунок 1. Стартовая страница login.html

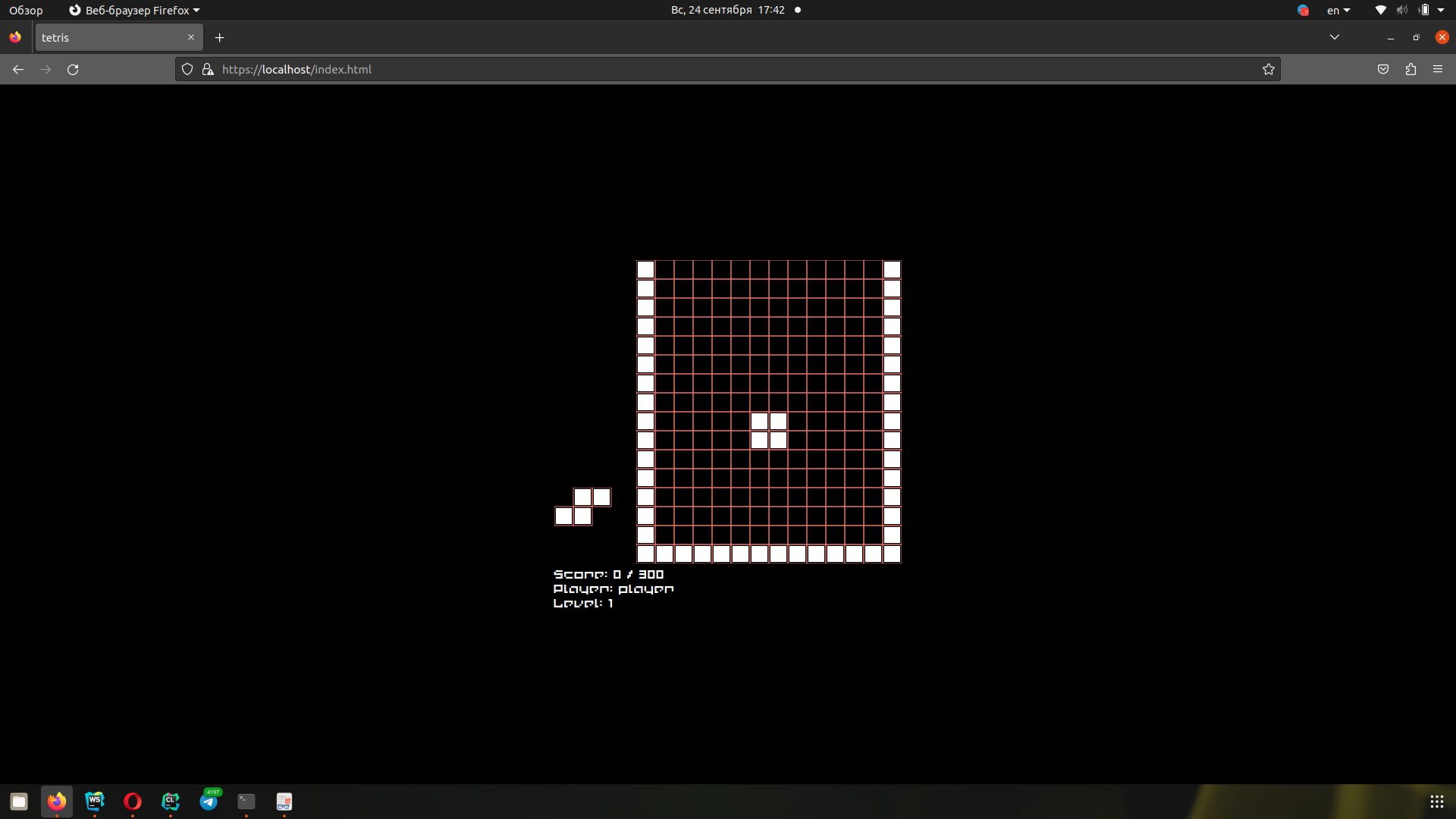


Рисунок 2. Главная страница index.html

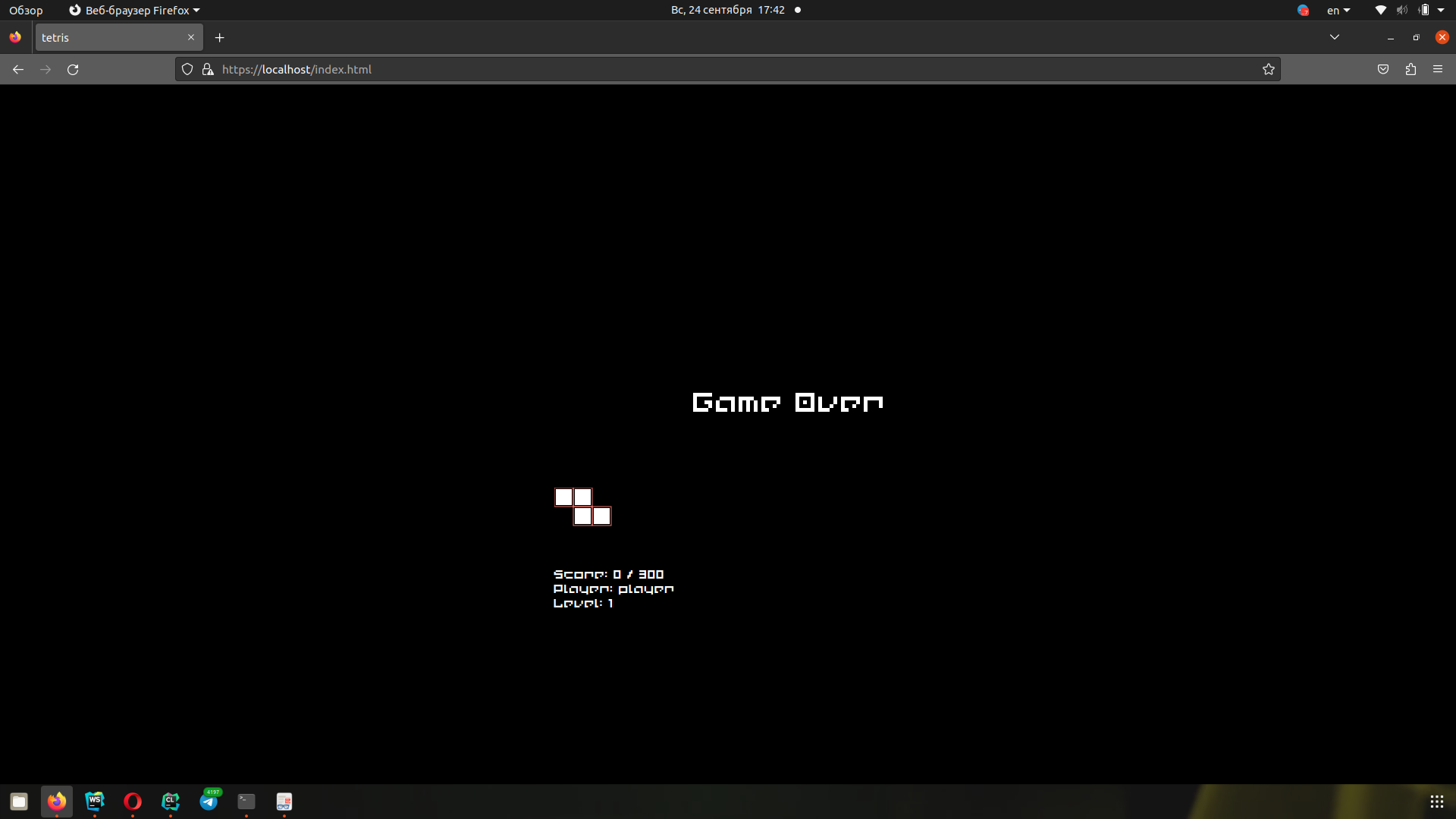


Рисунок 3. Главная страница index.html после проигрыша

**Разработка игровой логики.**

1.1. Класс Tetromino

Функционал:

Данный класс реализует логику тетромино, их поворот и движение.

Методы:

* constructor(shape): конструктор класса Tetromino, который

инициализирует тетромино заданной формой и начальными координатами.

* rotate(board): метод для поворота текущего тетромино на игровом поле.
* canMove(board, dx, dy): проверяет, можно ли переместить текущее

тетромино на заданные координаты на игровом поле.

* move(board, dx, dy): перемещает текущее тетромино на заданные

координаты на игровом поле.

* redrawTetromino(board, oldCoordinates, newCoordinates): перерисовывает

тетромино на игровом поле после его перемещения или поворота.

* getOccupiedCoordinates(x, y, shape): возвращает координаты ячеек,

которые занимает текущее тетромино.

* collidesWith(board, shape, x, y): проверяет, столкнулось ли текущее

тетромино с другими элементами на игровом поле.

1.2. Класс TetrisGame

Функционал:

Данный класс реализует основную логику приложения, включая управление, отображение и взаимодействие с игровым полем.

Методы:

* constructor(): конструктор класса TetrisGame, инициализирующий игровое

поле, начальные параметры и запускающий игру.

* initGrid(): инициализация игрового поля.
* drawTetromino(): отрисовка текущего тетромино на игровом поле.
* drawGrid(): отрисовка сетки игрового поля.
* fillCellsAroundEdges(): заполнение граничных ячеек игрового поля.
* startGame(): начало игры.
* gameOver(): завершение игры и отображение сообщения о конце игры.
* gameLoop(timestamp): основной игровой цикл, обрабатывающий логику

игры и отображение.

* handleKeyPress(event): обработка событий нажатия клавиш для

управления тетромино.

* rotateCurrentTetromino(): поворот текущего тетромино.
* moveCurrentTetrominoDown(): перемещение текущего тетромино вниз.
* moveCurrentTetrominoLeft(): перемещение текущего тетромино влево.
* moveCurrentTetrominoRight(): перемещение текущего тетромино вправо.
* canSpawnTetromino(tetromino): проверка возможности появления нового

тетромино на игровом поле.

* spawnNewTetromino(): создание и инициализация нового тетромино для

появления в игре.

* clearFullRows(): проверка и удаление полных строк на игровом поле.
* drawNextTetromino(): отрисовка следующего тетромино, которое

появится в игре.

**Разработка таблицы рекордов.**

Класс RecordManager:

addRecord(name, score): добавляет новую запись о рекорде в локальное хранилище, сортирует рекорды по убыванию счета и ограничивает список рекордов до 3 записей.

Страница index.html содержит таблицу рекордов с тремя колонками: место, имя игрока и счет. Рекорды сортируются по убыванию счета и отображаются в порядке убывания. Таблица рекордов обновляется автоматически при загрузке страницы и после завершения игры.

**Общие замечания.**

**Хранение данных.**

Для хранения данных о рекордах и имени игрока используется локальное хранилище (localStorage) браузера. Это позволяет сохранять и восстанавливать данные между сеансами игры на одном и том же устройстве.

**Интерфейс для ввода имени.**

Перед началом игры игроку предоставляется возможность ввести свое имя. Это имя сохраняется и отображается в игре, а также используется для сохранения рекордов в таблице.

**Уровни и увеличение сложности.**

В игре реализован механизм уровней, который повышает сложность игры при достижении определенного счета. Увеличение уровня уменьшает интервал между падением тетромино, делая игру более сложной.

**Цель игры.**

Цель игры Tetris - набрать максимальное количество очков, заполняя горизонтальные строки тетромино без пропусков. Игра заканчивается, когда новое тетромино не может появиться на игровом поле.

**Управление.**

Управление тетромино в игре осуществляется с помощью стрелок клавиатуры. Присутствует возможность быстрого опускания тетромино (клавиша Enter) и поворота (клавиша стрелка вверх).

**Выводы.**

В ходе лабораторной работы были изучены основные принципы работы с web-сервером nginx со статическими файлами и создано клиентское JavaScript приложение для игры в тетрис. Был настроен сервер nginx для этого были сгенерированы сертификаты шифрования с помощью библиотеки OpenSSL. Также, приложение имеет функциональность сохранения таблицы рекордов и несколько уровней сложности.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Файл login.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<link href="https://fonts.cdnfonts.com/css/common-pixel" rel="stylesheet">

<title>tetris</title>

</head>

<body>

<div class="welcome">

<h1>

TETRIS

</h1>

<form> <!--action="index.html" method="get" -->

<label>

<input placeholder="Username"><br>

</label>

<input type="submit" value="Play" class="button">

</form>

<table class="table">

<caption>High Scores</caption>

<thead>

<tr>

<th>Rank</th>

<th>Username</th>

<th>Score</th>

</tr>

</thead>

<tbody id="scoreTable">

</tbody>

</table>

<script src="table.js"></script>

</div>

</body>

</html>

Файл index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<link rel="stylesheet" href="style2.css">

<link href="https://fonts.cdnfonts.com/css/common-pixel" rel="stylesheet">

<title>tetris</title>

</head>

<body>

<div>

<canvas id="nextTetrominoCanvas" width="100" height="100"></canvas>

<canvas id="myCanvas" width="350" height="400"></canvas>

<div id = "score"></div>

<div id = "player"></div>

<div id = "level"></div>

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

</div>

</body>

</html>

Файл script.js:

class RecordManager {

static addRecord(name, score) {

const records = JSON.parse(localStorage.getItem('records')) || [];

records.push({ name, score });

records.sort((a, b) => b.score - a.score);

if (records.length > 3) {

records.pop();

}

localStorage.setItem('records', JSON.stringify(records));

}

}

class Tetromino {

constructor(shape) {

this.shape = shape;

this.x = 0;

this.y = 0;

}

rotate(board) {

const oldShape = this.shape;

const newShape = [];

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

const newRow = [];

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

newRow.push(oldShape[row][col]);

}

newShape.push(newRow);

}

if (!this.collidesWith(board, newShape, this.x, this.y)) {

const oldCoordinates = this.getOccupiedCoordinates();

this.shape = newShape;

const newCoordinates = this.getOccupiedCoordinates();

this.redrawTetromino(board, oldCoordinates, newCoordinates);

}

}

canMove(board, dx, dy) {

const nextX = this.x + dx;

const nextY = this.y + dy;

return !this.collidesWith(board, this.shape, nextX, nextY);

}

move(board, dx, dy) {

if (this.canMove(board, dx, dy)) {

const oldCoordinates = this.getOccupiedCoordinates();

this.x += dx;

this.y += dy;

const newCoordinates = this.getOccupiedCoordinates();

this.redrawTetromino(board, oldCoordinates, newCoordinates);

return true;

}

return false;

}

redrawTetromino(board, oldCoordinates, newCoordinates) {

for (const coord of oldCoordinates) {

board[coord.y][coord.x] = 0;

}

for (const coord of newCoordinates) {

board[coord.y][coord.x] = 1;

}

}

getOccupiedCoordinates(x = this.x, y = this.y, shape = this.shape) {

const occupiedCoordinates = [];

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

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

if (shape[row][col] === 1) {

const boardX = x + col;

const boardY = y + row;

occupiedCoordinates.push({ x: boardX, y: boardY });

}

}

}

return occupiedCoordinates;

}

collidesWith(board, shape, x, y) {

const oldCoordinates = this.getOccupiedCoordinates();

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

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

if (shape[row][col] === 1) {

const boardX = x + col;

const boardY = y + row;

if (board[boardY][boardX] === 1 && !oldCoordinates.some(coord => coord.x === boardX && coord.y === boardY)) {

return true;

}

}

}

}

return false;

}

}

function getRandomBrightColor() {

const r = Math.floor(Math.random() \* 256); // Значение от 0 до 255

const g = Math.floor(Math.random() \* 256);

const b = Math.floor(Math.random() \* 256);

return `#${r.toString(16).padStart(2, '0')}${g.toString(16).padStart(2, '0')}${b.toString(16).padStart(2, '0')}`;

}

class TetrisGame {

constructor() {

this.canvas = document.getElementById('myCanvas');

this.ctx = this.canvas.getContext('2d');

this.ctx.strokeStyle = getRandomBrightColor();

this.ctx.fillStyle = 'white';

this.ctx.font = '40px Common Pixel';

this.cellSize = 25;

this.rows = this.canvas.height / this.cellSize;

this.columns = this.canvas.width / this.cellSize;

this.grid = [];

this.score = 0;

this.level = 1

this.scoreThreshold = 300;

this.moveInterval = 500;

this.lastMoveTime = 0;

this.currentTetromino = this.spawnNewTetromino();

this.nextTetromino = this.spawnNewTetromino();

this.nextTetrominoCanvas = document.getElementById('nextTetrominoCanvas');

document.addEventListener('keydown', this.handleKeyPress.bind(this));

this.initGrid();

this.fillCellsAroundEdges();

this.startGame();

}

initGrid() {

for (let i = 0; i < this.rows; i++) {

this.grid[i] = [];

for (let j = 0; j < this.columns; j++) {

this.grid[i][j] = 0;

}

}

}

drawTetromino() {

const occupiedCoordinates = this.currentTetromino.getOccupiedCoordinates();

for (const coord of occupiedCoordinates) {

const cellX = coord.x;

const cellY = coord.y;

this.grid[cellY][cellX] = 1;

}

}

drawGrid() {

const borderSize = 4;

for (let i = 0; i < this.rows; i++) {

for (let j = 0; j < this.columns; j++) {

const cellX = j \* this.cellSize;

const cellY = i \* this.cellSize;

if (this.grid[i][j] === 1) {

this.ctx.fillRect(cellX + borderSize / 2, cellY + borderSize / 2, this.cellSize - borderSize, this.cellSize - borderSize);

}

this.ctx.strokeRect(cellX, cellY, this.cellSize, this.cellSize);

}

}

}

fillCellsAroundEdges() {

for (let i = 0; i < this.rows; i++) {

for (let j = 0; j < this.columns; j++) {

if (i === this.rows - 1 || j === 0 || j === this.columns - 1) {

this.grid[i][j] = 1;

}

}

}

}

startGame() {

this.lastMoveTime = performance.now();

requestAnimationFrame(this.gameLoop.bind(this));

}

gameOver() {

this.ctx.clearRect(0, 0, this.canvas.width, this.canvas.height);

cancelAnimationFrame(this.animationId);

this.ctx.fillText('Game Over', this.canvas.width / 2 - 100, this.canvas.height / 2);

let savedUsername = localStorage.getItem('username');

if (savedUsername !== null) {

if (savedUsername.length > 0) {

RecordManager.addRecord(savedUsername, (this.level - 1) \* this.scoreThreshold + this.score);

}

}

setTimeout(() => {

window.location.href = 'login.html';

}, 3000);

}

gameLoop(timestamp) {

const elapsedTime = timestamp - this.lastMoveTime;

if (elapsedTime >= this.moveInterval) {

this.lastMoveTime = timestamp;

if (!this.moveCurrentTetrominoDown()) {

this.clearFullRows();

this.currentTetromino = this.nextTetromino

this.nextTetromino = this.spawnNewTetromino();

if(!this.canSpawnTetromino(this.nextTetromino)) {

this.gameOver();

return;

}

}

}

this.ctx.clearRect(0, 0, this.canvas.width, this.canvas.height);

this.drawGrid();

this.drawTetromino();

this.drawNextTetromino();

if (this.score >= this.scoreThreshold) {

this.moveInterval = Math.max(150, Math.floor(this.moveInterval \* 0.9));

this.score = 0;

this.level += 1;

}

let savedUsername = localStorage.getItem('username');

if (!savedUsername || savedUsername.length === 0) {

savedUsername = "User";

}

document.getElementById('player').innerText = `Player: ${savedUsername}`;

document.getElementById('score').innerText = `Score: ${this.score} / ${this.scoreThreshold}`;

document.getElementById('level').innerText = `Level: ${this.level}`;

this.animationId = requestAnimationFrame(this.gameLoop.bind(this));

}

handleKeyPress(event) {

const key = event.key;

switch (key) {

case 'Enter':

while(this.moveCurrentTetrominoDown());

break;

case 'ArrowUp':

this.rotateCurrentTetromino();

break;

case 'ArrowDown':

this.moveCurrentTetrominoDown();

break;

case 'ArrowLeft':

this.moveCurrentTetrominoLeft();

break;

case 'ArrowRight':

this.moveCurrentTetrominoRight();

break;

}

}

rotateCurrentTetromino() {

if (this.currentTetromino) {

this.currentTetromino.rotate(this.grid);

}

}

moveCurrentTetrominoDown() {

if (this.currentTetromino) {

return this.currentTetromino.move(this.grid, 0, 1);

}

}

moveCurrentTetrominoLeft() {

if (this.currentTetromino) {

return this.currentTetromino.move(this.grid, -1, 0);

}

}

moveCurrentTetrominoRight() {

if (this.currentTetromino) {

return this.currentTetromino.move(this.grid, 1, 0);

}

}

canSpawnTetromino(tetromino) {

const occupiedCoordinates = tetromino.getOccupiedCoordinates();

for (const coord of occupiedCoordinates) {

if (this.grid[coord.y][coord.x] === 1) {

return false;

}

}

return true;

}

spawnNewTetromino() {

const tetrominoShapes = [

[[1, 1, 1, 1]],

[[1, 1], [1, 1]],

[[1, 1, 0], [0, 1, 1]],

[[0, 1, 1], [1, 1, 0]],

[[0, 1, 0], [1, 1, 1]],

[[1, 0, 0], [1, 1, 1]],

[[0, 0, 1], [1, 1, 1]]

];

const randomShape = tetrominoShapes[Math.floor(Math.random() \* tetrominoShapes.length)];

const newTetromino = new Tetromino(randomShape);

newTetromino.x = Math.floor((this.columns - newTetromino.shape[0].length) / 2);

newTetromino.y = 0;

return newTetromino;

}

clearFullRows() {

const rowsToRemove = [];

for (let row = 0; row < this.rows - 1; row++) {

if (this.grid[row].every(cell => cell === 1)) {

rowsToRemove.push(row);

}

}

for (const rowIndex of rowsToRemove) {

this.grid.splice(rowIndex, 1);

this.score += 100;

}

for (let i = 0; i < rowsToRemove.length; i++) {

const newRow = new Array(this.columns).fill(0);

newRow[0] = 1;

newRow[newRow.length-1] = 1

this.grid.unshift(newRow);

}

}

drawNextTetromino() {

const ctx = this.nextTetrominoCanvas.getContext('2d');

const borderSize = 4;

ctx.clearRect(0, 0, this.nextTetrominoCanvas.width, this.nextTetrominoCanvas.height);

ctx.strokeStyle = this.ctx.strokeStyle;

ctx.fillStyle = this.ctx.fillStyle;

if (this.nextTetromino) {

for (let row = 0; row < this.nextTetromino.shape.length; row++) {

for (let col = 0; col < this.nextTetromino.shape[row].length; col++) {

const cellX = col \* this.cellSize;

const cellY = row \* this.cellSize;

if (this.nextTetromino.shape[row][col] === 1) {

ctx.fillRect(cellX + borderSize / 2, cellY + borderSize / 2, this.cellSize - borderSize, this.cellSize - borderSize);

ctx.strokeRect(cellX, cellY, this.cellSize, this.cellSize);

}

}

}

}

}

}

const tetrisGame = new TetrisGame();

Файл table.js:

const form = document.querySelector('form');

const input = form.querySelector('input');

document.addEventListener('DOMContentLoaded', function () {

form.addEventListener('submit', function (e) {

e.preventDefault();

const username = input.value;

localStorage.setItem('username', username);

window.location.href = 'index.html';

});

});

let savedUsername = localStorage.getItem('username');

if (!savedUsername || savedUsername.length === 0) {

input.value = "";

} else {

input.value = savedUsername;

}

function updateTable() {

const scoreTable = document.getElementById('scoreTable');

scoreTable.innerHTML = '';

const records = JSON.parse(localStorage.getItem('records')) || [];

records.sort((a, b) => b.score - a.score);

records.forEach((record, index) => {

const row = document.createElement('tr');

const rankCell = document.createElement('td');

const usernameCell = document.createElement('td');

const scoreCell = document.createElement('td');

rankCell.textContent = index + 1;

usernameCell.textContent = record.name;

scoreCell.textContent = record.score;

row.appendChild(rankCell);

row.appendChild(usernameCell);

row.appendChild(scoreCell);

scoreTable.appendChild(row);

});

}

window.addEventListener('load', updateTable);

Файл style.css:

@import url('https://fonts.cdnfonts.com/css/common-pixel');

body {

/\*background-image: url(files/background2.jpg);\*/

background-color: black;

background-size: 100%;

}

.welcome{

margin-left: auto;

margin-right: auto;

width: 25%;

text-align: center;

margin-top: 8%;

padding:10px;

border-radius:30px;

font-size: 20px;

color: #404142;

background-color: white;

font-family: 'Common Pixel', sans-serif;

}

.table {

font-family: 'Common Pixel', sans-serif;

margin-left: auto;

margin-right: auto;

margin-top: 35px;

}

.table {

width: 80%;

}

.table th {

width: 30%;

}

input {

width:60%;

padding: 10px;

border-radius:20px;

border: 1px solid black;

margin-bottom: 10px;

font-size: 20px;

background-color: white;

font-family: 'Common Pixel', sans-serif;

}

.button {

width:40%;

padding: 10px;

border-radius:20px;

border: 1px solid black;

margin-bottom: 10px;

font-size: 20px;

color: #404142;

background-color: white;

font-family: 'Common Pixel', sans-serif;

}

Файл style2.css:

@import url('https://fonts.cdnfonts.com/css/common-pixel');

body {

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

background-color: black;

}

canvas {

border: 1px solid black;

}

#score, #player, #level{

color : white;

font-size: 18px;

font-family: 'Common Pixel', sans-serif;

}