**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. 3344 |  | Коняева М.В. |
| Преподаватель |  | Беляев С.А. |

Санкт-Петербург

2025

## 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.

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».

**Основные теоретические положения.**

Nginx (<https://nginx.ru/ru/>) — это веб-сервер, работающий как в Unix-подобных системах, так и в Windows.

JavaScript (<https://developer.mozilla.org/ru/docs/Web/JavaScript>) — язык программирования, поддерживающий как объектно-ориентированный, так и функциональный подход. Он является реализацией стандарта ECMAScript.

Асимметричные алгоритмы шифрования используют пару ключей — открытый и закрытый. Закрытый ключ хранится только у владельца и не подлежит разглашению. Открытый ключ может быть опубликован и применяется, например, для проверки подлинности подписанного сообщения или документа. Он вычисляется на основе закрытого ключа при помощи специальной функции. При этом знание открытого ключа не позволяет восстановить закрытый, тогда как из закрытого ключа открытый получается без труда.

## Выполнение работы

Файл: index.html. Экран входа для игры «Тетрис». Он содержит форму с одним полем для ввода имени пользователя и кнопкой «Начать игру». Имя автоматически подставляется из localStorage при загрузке, сохраняется при изменении и при отправке формы. После отправки происходит переход на game.html.

Файл: game.html. Игровая страница «Тетриса». В ней есть холст (canvas) для отрисовки игры, оверлей для паузы, боковая панель со счётом, уровнем, предпросмотром следующей фигуры, таблицей рекордов и кнопками управления (старт, пауза, рестарт). Внизу подключены все скрипты, реализующие логику игры.

Файл: highscores.html. Страница с таблицей рекордов. Она отображает список лучших результатов в блоке #highscores и содержит кнопку для возврата на экран входа. Внизу подключается скрипт highscores.js, который управляет выводом данных.

Файл: styles.css. Стили, оформление ссылок, полей и кнопок.

* login: центрированный тёмный экран входа с полем ввода и кнопкой.
* main: игровое поле (canvas), боковая панель со счетом, уровнями, рекордами и кнопками управления.
* highscores: таблица рекордов по центру с кнопкой возврата.

Везде используется тёмная палитра (#000–#444), скруглённые панели и hover-эффекты для кнопок.

Файл: auth.js. Загружает сохранённое имя из localStorage, подставляет его в поле ввода, сохраняет изменения при вводе и при отправке формы проверяет, что имя непустое, после чего записывает его в localStorage и перенаправляет на game.html.

Класс: Board. Отвечает за игровое поле.

Поля:

* rows — число строк поля (по умолчанию 20).
* cols — число столбцов поля (по умолчанию 10).
* grid — двумерный массив rows × cols, хранит текущее состояние клеток (null или цвет фигуры).

Методы:

* createEmptyBoard() — создаёт пустое поле.
* isValidMove(tetromino, offsetX, offsetY, newShape) — проверяет, можно ли переместить или повернуть фигуру.
* placeTetromino(tetromino) — размещает фигуру на поле.
* clearLines() — удаляет полностью заполненные строки и возвращает их количество.
* draw(context, blockSize) — отрисовывает поле на canvas.
* clear() — сбрасывает поле в пустое состояние.

Класс: Tetromino. Представляет фигуру в Тетрисе.

Поля**:**

* shape — матрица фигуры,
* color — цвет,
* type — тип (I, O, T…),
* x и y— позиция на поле.

Методы**:** rotate() — возвращает повернутую фигуру, applyRotation() — применяет поворот.

TETROMINO\_DATA — данные всех стандартных фигур с формой и цветом.

Класс Game управляет логикой игры в Тетрис. Поля**:** board — игровое поле, currentTetromino и nextTetromino — текущая и следующая фигуры, score — счёт, gameOver — состояние игры.  
Методы**:**

* randomTetromino() — создаёт случайную фигуру;
* reset() — сбрасывает игру;
* moveDown(), moveLeft(), moveRight() — перемещение фигуры;
* rotate() — поворот фигуры;
* hardDrop() — мгновенное опускание;
* lockTetromino() — фиксирует фигуру на поле, очищает линии, обновляет счёт и проверяет конец игры;
* update() — основной шаг игры (опускает фигуру вниз, если игра не окончена).

Класс GameController управляет процессом игры, соединяя Game, Renderer, UIManager и ScoreManager.

**Поля:**

* game, renderer, ui, scores — объекты для логики, отрисовки, интерфейса и рекордов.
* dropInterval, minInterval, levelUpEvery — параметры скорости падения и повышения уровня.
* level — текущий уровень, running/paused — состояние игры, \_tickTimer — таймер падения, \_gameOverTriggered — флаг конца игры.

**Методы:**

* handleKey(e) — обработка клавиш управления (движение, поворот, падение, пауза).
* start() — запускает игру или продолжает после паузы.
* pause() / togglePause() — ставит игру на паузу и переключает её.
* restart() — перезапускает игру с нуля.
* scheduleTick() — запускает таймер для падения фигур с интервалом, зависящим от уровня.
* getIntervalForLevel() — рассчитывает скорость падения для текущего уровня.
* tick() — один шаг игры: падение фигуры, обновление счёта, проверка уровня и перерисовка.
* moveLeft(), moveRight(), rotate(), softDrop(), hardDrop() — обёртки над методами Game с перерисовкой.
* onGameOver() — фиксирует результат, обновляет рекорды и перенаправляет на страницу рекордов.

Класс Renderer отвечает за визуальное отображение игры на canvas и в блоке предпросмотра.

Поля:

* canvas — элемент <canvas> для основного игрового поля.
* ctx — 2D-контекст canvas для рисования.
* blockSize — размер одного блока в пикселях.

Методы:

* resizeToBoard(board) — подгоняет размеры canvas под размеры игрового поля.
* draw(game) — отрисовывает текущее состояние поля и активную фигуру.
* drawNextPreview(nextTetromino) — рисует мини-превью следующей фигуры в отдельном блоке.
* drawGameOver(score) — отображает затемнение и сообщение о завершении игры с показом счёта.

Класс ScoreManager управляет таблицей рекордов игры.

Поля:

* storageKey — ключ для сохранения в localStorage.
* highscores — массив объектов с именем игрока и очками.

Методы:

* load() — загружает рекорды из localStorage.
* save() — сохраняет текущие рекорды в localStorage.
* addScore(name, score) — добавляет или обновляет рекорд игрока, сортирует топ-10 и сохраняет.
* getHighscores() — возвращает текущие рекорды.

Класс UIManager управляет элементами интерфейса игры и их обновлением.

Поля:

* Ссылки на DOM-элементы: кнопки (startBtn, pauseBtn, restartBtn), счёт (scoreEl), уровень (levelEl), контейнер рекордов (highscoresContainer), оверлей паузы (pauseOverlay).
* Обработчики событий: onStart, onPause, onRestart.

Методы:

* initButtons() — назначает обработчики кликов для кнопок.
* updateScore(score) — обновляет отображение счёта.
* updateLevel(level) — обновляет отображение уровня.
* askPlayerName(score) — спрашивает имя игрока через prompt.
* renderHighscores(highscores) — отображает таблицу рекордов.
* showPause() / hidePause() — показывает или скрывает оверлей паузы.

Изображения с основными страницами игры представлены в приложении A.

## Выводы

В рамках лабораторной работы была создана игра «Tetris» на JavaScript. Для интерфейса и взаимодействия с пользователем подготовлены HTML-страницы. Сгенерирована пара ключей (закрытый и открытый) для шифрования, а веб-сервер nginx настроен на работу по протоколу HTTPS.

# Приложение А

На рисунке 1 представлена стартовая страничка.

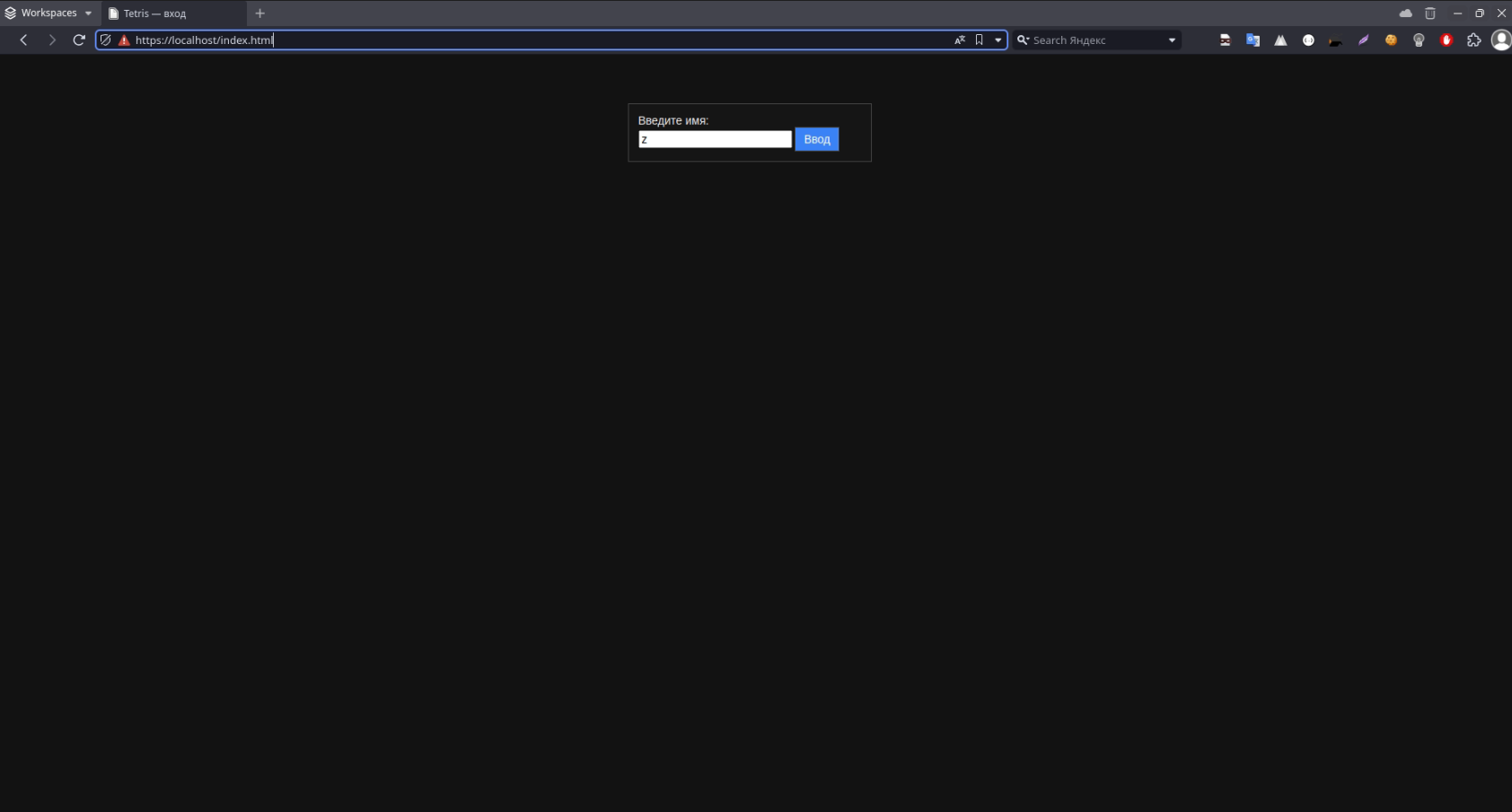


Рисунок 1 – стартовая страница.

На рисунке 2 представлена страничка самой игры.

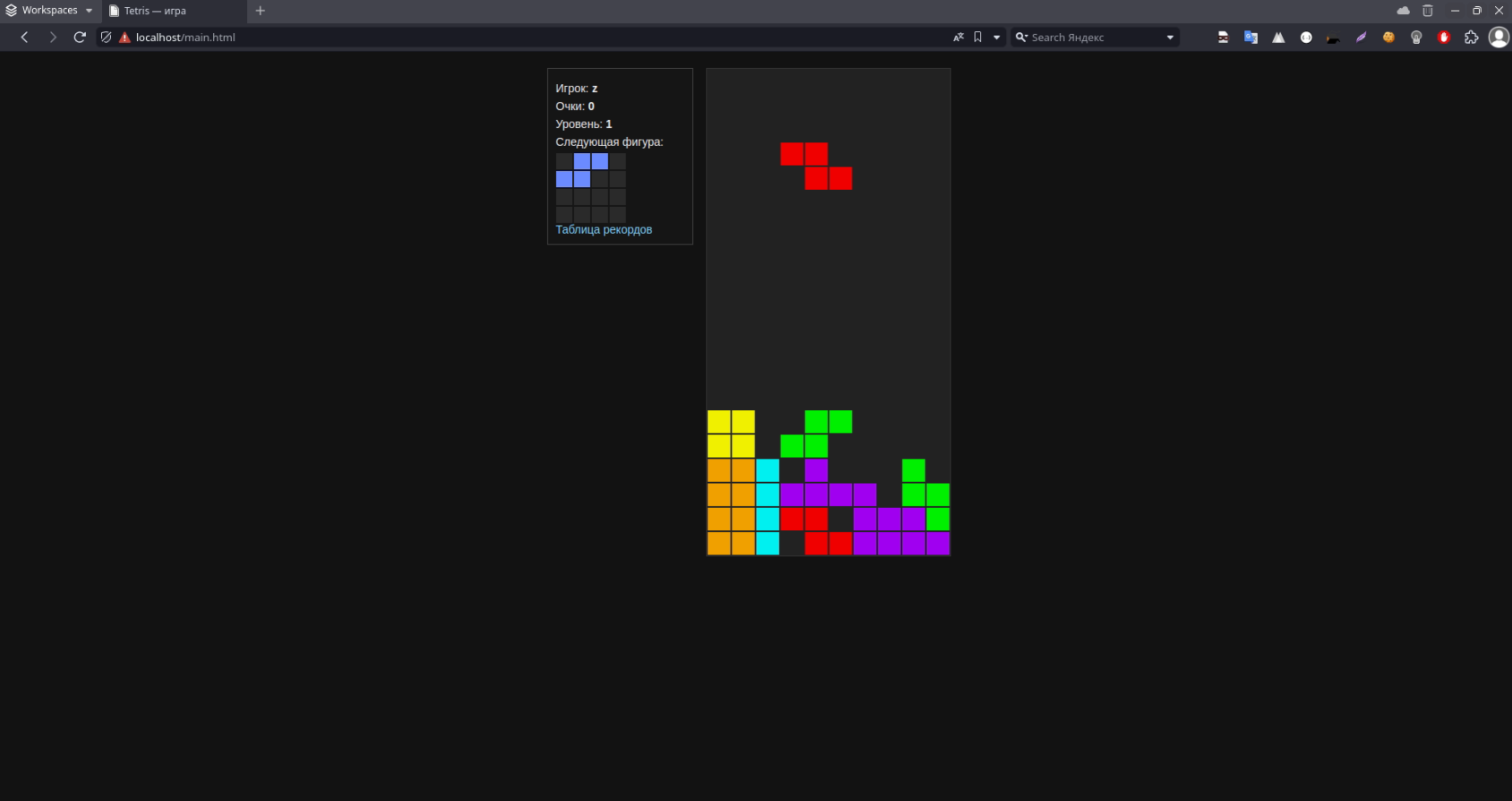


Рисунок 2 – страница игры.

На рисунке 3 представлена страничка после окончания игры с таблицей рекордов.

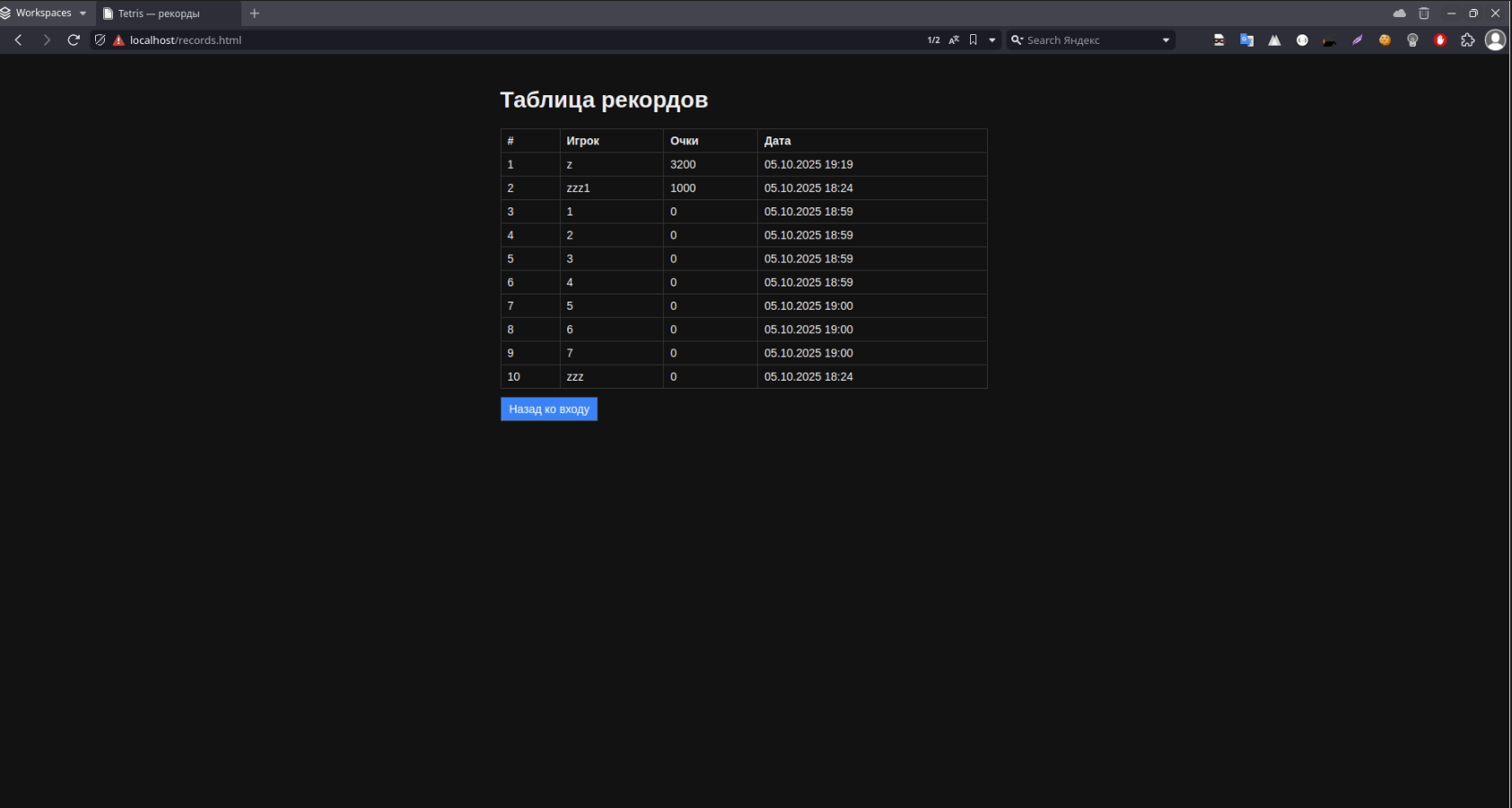


Рисунок 3 – страница конца игры с таблицей рекордов.