**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. 3343 |  | Коршков А.А. |
| 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.

## Выполнение работы

В ходе выполнения задания было разработано веб-приложение – классическая игра «Тетрис», полностью реализованная на языке TypeScript с использованием HTML и CSS для отображения интерфейса, а также фреймворка Bootstrap 5.

Для реализации логики игры в Tetris в приложении были созданы следующие ключевые классы:

1. *Game* - Основной класс игры, управляющий всем процессом.

Функции:

* *start*() – запускает игру.
* *restart*() – перезапускает игру .
* *loop(time: number = 0)* – основной игровой цикл, обновляющий состояние и отрисовку.
* *update(deltaTime: number)* – обновление позиции текущей фигуры с учётом времени и скорости.
* *lockFigure*() – закрепление текущей фигуры на поле, проверка и очистка заполненных линий.
* *updateLevel*() – повышение уровня игры в зависимости от набранного количества очков.
* hardDrop() – реализация мгновенного падения.
* moveDown() – реализация ускоренного падения.
* *gameOver*() – обработка окончания игры и сохранение рекорда.

1. Cell – класс, представляющий клетку поля. Содержит свойства цвета и состояния (1, если заполнен, иначе 0)
2. Figure - Класс, представляющий тетромино.

Функции:

* *rotate()* – вращение фигуры.
* *static random(): Figure* – генерация случайной фигуры из набора тетромино (не привязан к конкретному объекту).
* clone(): Figure – возвращает новый объект фигуры
* restore(other: Figure) – восстанавливает фигуру
* equals(other: FIgure): Boolean – сравнивает объекты двух фигур
* private matricesEqual(a: number[][], b: number[][]): boolean – поэлементно сравнивает объекты двух фигур

1. *Playfield -* Класс игрового поля.

Функции:

* *clear*() – очистка всего поля.
* *placeFigure(figure: Figure)* – установка фигуры на поле, обновление сетки.
* *clearLines(): number*  – проверка и удаление полностью заполненных линий.
* *isValidPosition(figure: Figure, offsetX: number = 0, offsetY: number = 0): boolean*  – проверка столкновений фигуры с границами и другими блоками.

1. Renderer – абстрактный класс для отрисовки фигуры на <canvas>.

Функции:

* drawCell(x: number, y: number, size: number, fill\_color: string, stroke\_color: string) – рисует элемент клетки на canvas элементе.
* abstract render(arg?: T) – абстрактный метод (наследуется от интерфейса IRenderer)

1. *PlayfieldRenderer* - Класс для отрисовки игрового поля и фигур на <canvas> (наследуется от Renderer).

Функции:

* private renderGrid() – отрисовка сетки поля
* private renderExistingFigures() – отрисовка существующих фигур на доске
* private renderCurrentFigure(currentFigure: Figure) – отрисовка текущей фигуры на поле
* *render(currentFigure: Figure)* – отрисовка всей сетки поля, фигур на ней и текущей фигуры.

1. *NextFigureRenderer* - Класс для отображения следующей фигуры (наследуетcя от Renderer).

Функции:

* *render(nextFigure: Figure)* – отрисовка следующей фигуры на панели интерфейса.

1. *Ui* - Класс для работы с интерфейсом пользователя.

Функции:

* *updateScore(score: number)*– обновление значения очков.
* *updateLevel(level: number)*– обновление текущего уровня.
* *showLeaderboard(recordStorageManager: RecordStorageManager, nickname: string, score: number)* – отображение локальной таблицы рекордов.

Управление кнопками «Start Game», «Restart» и «Change Nickname».

1. KeyboardHandler - Класс для обработки пользовательского ввода с клавиатуры.

Функции:

* *handleKeydown(event: KeyboardEvent)* – обработка нажатий клавиш для движения, вращения и ускоренного падения фигуры.
* *attach*() – подключение слушателя клавиатуры.
* *detach*() – отключение слушателя клавиатуры.

1. LocalStorageManager – абстрактный класс для работы с localStorage.

Функции

* store(key: string, value: string) – сохраняет в localStorage значение по ключу.
* read(key: string): any | null – получает из localStorage элемент по ключу.

1. RecordStorageManager - Класс для работы с локальными рекордами (наследуется от LocalStorageManager).

Функции:

* *saveRecord*(*nickname*, *score*) – сохранение результата игрока с отметкой времени.
* *getSortedPlayers*() – получение всех игроков с сортировкой.

Логика коллизий была продумана так, чтобы фигуры не выходили за пределы игрового поля, а вращение учитывало «стены».

При заполнении игрового поля до верха стакана игра заканчивается, и пользователю отображается экран «Game Over» с локальной таблицей рекордов.

Весь игровой процесс построен на динамическом обновлении canvas через метод requestAnimationFrame, что обеспечивает плавное отображение движения фигур и обновление интерфейса.

## Выводы

В ходе выполнения работы было создано полноценное веб-приложение – игра «Тетрис», полностью реализованная на TypeScript с использованием HTML и CSS, а также с использованием фреймворка Bootstrap.

* пользователь может вводить своё имя, которое отображается на панели с очками и уровнем;
* реализовано игровое поле с отображением текущей и следующей фигуры;
* присутствуют все классические тетромино (I, O, T, S, Z, J, L);
* фигуры случайным образом генерируются и падают с заданной скоростью;
* реализовано управление фигурой с клавиатуры: движение влево/вправо (стрелка влево/право), вращение на 90°(стрелка вверх), ускоренное падение (клавиша пробел);
* при заполнении линии она удаляется, а верхние строки смещаются вниз;
* при наборе определённого количества очков увеличивается уровень, что повышает скорость падения фигуры;
* игра заканчивается при заполнении поля, после чего отображается локальная таблица рекордов с сохранением результатов в localStorage.

Приложение предназначено для работы на веб-сервере Nginx с поддержкой протокола HTTPS версии не ниже 2.0, что обеспечивает безопасное взаимодействие клиента и сервера и позволяет развёртывать игру как полностью статическое веб-приложение.

## ПРИЛОЖЕНИЕ А

## ИЗОБРАЖЕНИЯ ИГРЫ

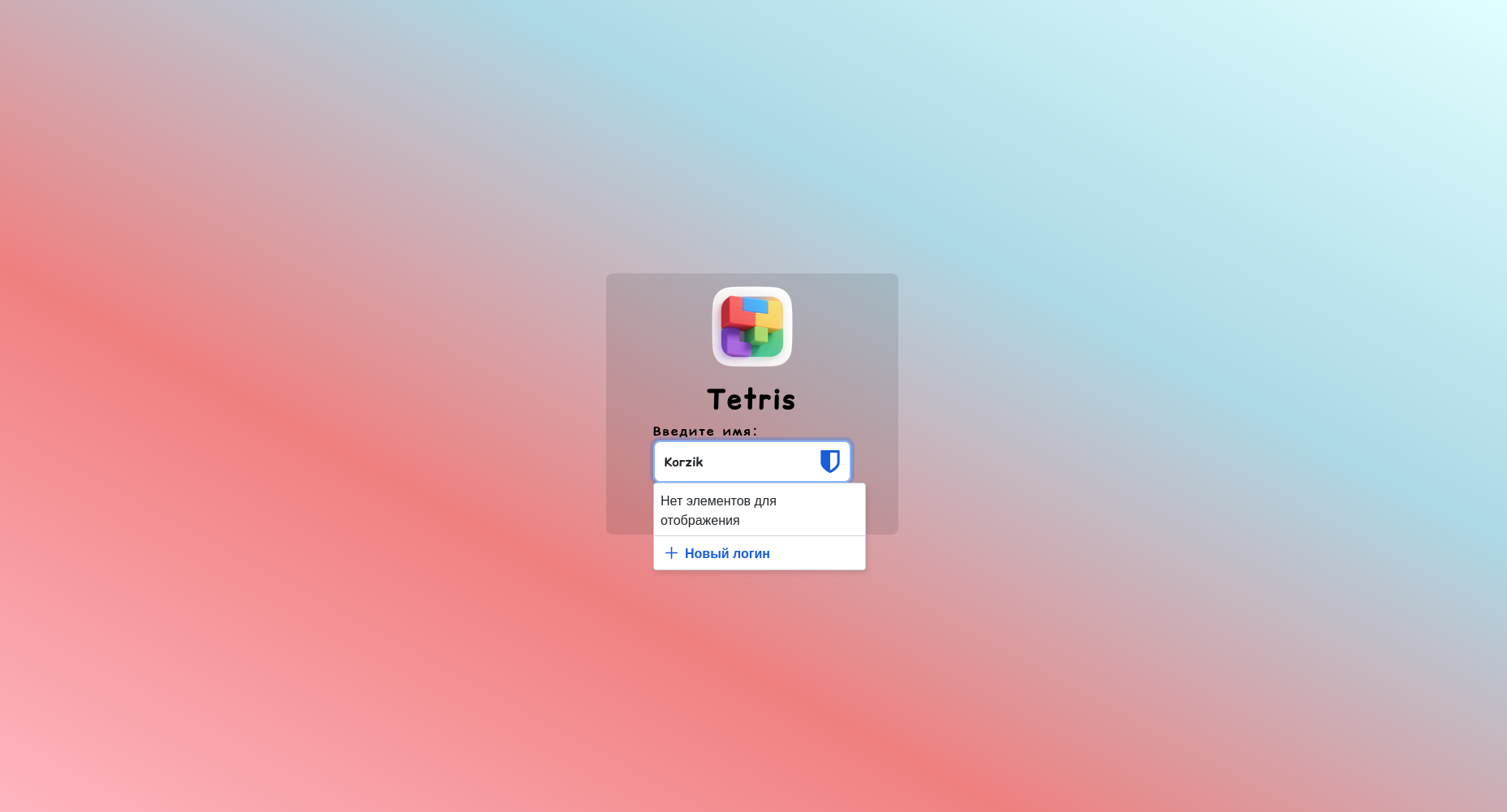


Рисунок 1 – Экран ввода никнейма

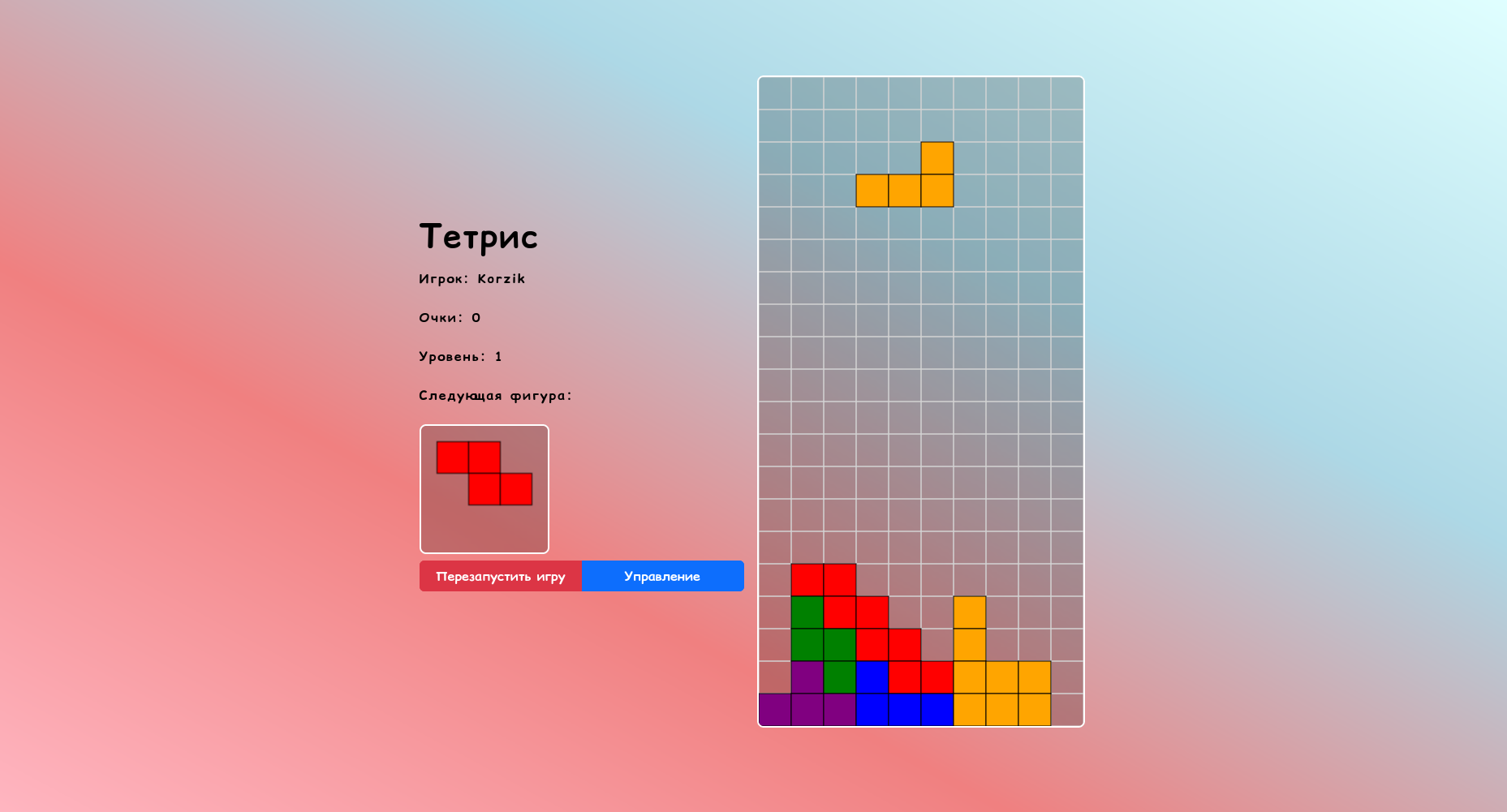


Рисунок 2 – Экран игры