

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. 3343

Гребнев Е.Д.

Преподаватель

Беляев С. А.

Санкт-Петербург

2025

Задание

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.

Выполнение работы

Файл index.html

Описывает структуру игрового интерфейса, включая модальные окна для старта игры, паузы, завершения игры и таблицы рекордов. Содержит основные элементы управления: игровое поле, панель статистики, информацию о следующей фигуре и управлении.

Файл style.scss

Задает современный неоновый стиль для игры с использованием CSS-переменных для цветовой схемы. Реализованы:

- Градиентные фоны с эффектом стекла (glassmorphism)
- Неоновая подсветка элементов интерфейса
- Адаптивная верстка для различных разрешений экрана
- Анимации появления элементов и пульсации счетчика
- Стилизация модальных окон с размытым фоном

Файл main.ts

Является основным модулем приложения, координирующим все компоненты игры:

Класс Particle реализует частицы для визуальных эффектов:

- Содержит свойства позиции, скорости, времени жизни и цвета
- Метод update() обновляет позицию с учетом гравитации
- Метод draw() отрисовывает частицу с учетом прозрачности

Класс ParticleSystem управляет системами частиц:

- Создает множество частиц в заданной позиции
- Обновляет и отрисовывает все частицы системы
- Удаляет "мертвые" частицы

Функции управления таблицей рекордов:

- getLeaderboard() - получает рекорды из localStorage
- saveToLeaderboard() - сохраняет новый результат с проверкой дубликатов

- renderLeaderboard() - отображает таблицу рекордов в указанном контейнере
- getRecentScores() - возвращает последние 5 результатов

Функции управления игровым процессом:

- setupCanvasScale() - настраивает масштабирование canvas для Retina-дисплеев
- gravityInterval() - вычисляет интервал падения фигур в зависимости от уровня
- drawGhostPiece() - отрисовывает тень-предview места падения текущей фигуры
- checkCollision() - проверяет столкновения фигуры с границами и другими блоками

Игровой цикл gameLoop():

- Обрабатывает автоматическое падение фигур с учетом уровня сложности
- Управляет системами частиц
- Обрабатывает повторное нажатие клавиш для плавного управления
- Обновляет интерфейс и проверяет условия завершения игры

Система управления:

- Обработчики клавиатуры для перемещения, вращения иброса фигур
- Поддержка как английской, так и русской раскладки
- Автоповтор при длительном нажатии клавиш

Файл types.ts

Определяет основные типы данных:

- Piece - описывает тетрамино с формой, позицией и цветом
- GameState - представляет полное состояние игры

Файл tetris.ts (игровой движок)

Класс Tetris реализует основную игровую логику:

Конструктор инициализирует игровое поле, создает первую и следующую фигуры.

Основные методы:

- createBoard() - создает пустое игровое поле
- spawnPiece() - размещает новую фигуру, проверяя условие завершения игры
- move() - перемещает фигуру горизонтально с проверкой столкновений
- rotate() - вращает фигуру в указанном направлении
- softDrop() - ускоренное падение с начислением очков
- hardDrop() - мгновенный сброс фигуры с бонусными очками
- lockPiece() - фиксирует фигуру на поле и проверяет заполненные линии
- clearLines() - удаляет заполненные линии и сдвигает содержимое выше
- updateScore() - рассчитывает очки на основе количества очищенных линий и уровня
- tick() - основной игровой такт, обрабатывающий автоматическое падение

Файл utils.ts

Содержит вспомогательные функции и константы:

Константы:

- COLORS - палитра неоновых цветов для фигур
- SHAPES - матричные представления всех семи типов тетрамино

Функции:

- createPiece() - создает случайную фигуру с случайным цветом
- checkCollision() - проверяет столкновение фигуры с границами и другими блоками
- rotatePiece() - выполняет поворот фигуры на 90 градусов по или против часовой стрелки

Файл pieces.ts

Содержит закомментированные исходные определения фигур тетриса с их цветами, которые были заменены на динамическую генерацию в utils.ts.

Архитектурные особенности

Модульная структура

Приложение разделено на логические модули:

- **main.ts** - координация и управление интерфейсом
- **tetris.ts** - игровая логика
- **utils.ts** - вспомогательные функции
- **types.ts** - определения типов TypeScript

Система состояний

Игра управляется через четко определенное состояние (`GameState`), что упрощает отладку и возможное добавление функции сохранения игры.

Визуальные эффекты

Реализована система частиц для:

- Эффектов при сбросе фигур
- Анимации при запуске и рестарте игры
- Визуального усиления игровых событий

Локализация

Поддержка как английской, так и русской раскладки клавиатуры делает игру доступной для разных пользователей.

Persistence

Использование `localStorage` обеспечивает сохранение:

- Имени пользователя между сессиями
- Таблицы рекордов
- Последних результатов

Адаптивность

Реализована responsive-верстка с медиа-запросами для корректного отображения на различных устройствах.

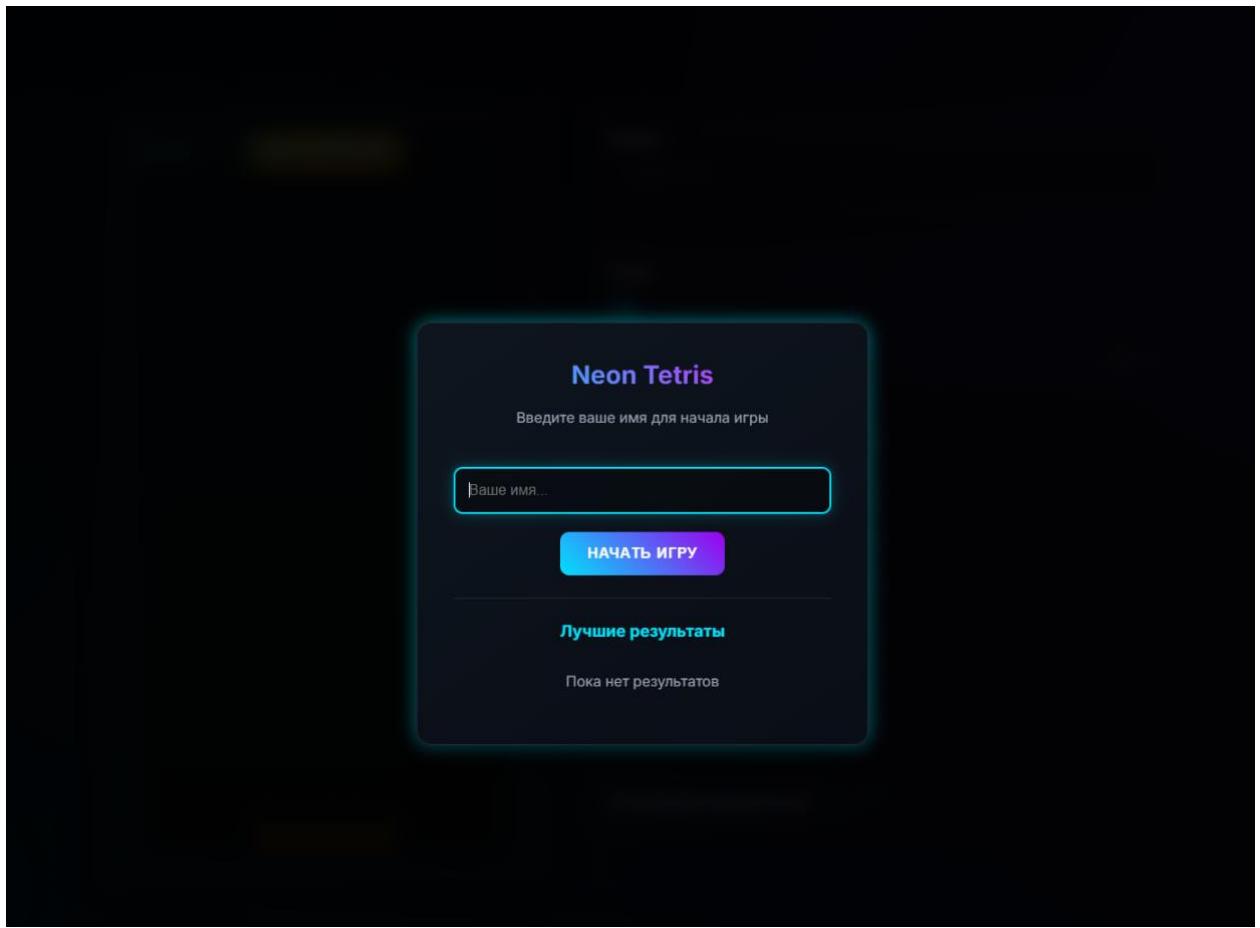


Рисунок 1 – Интерфейс формы входа и таблица рекордов

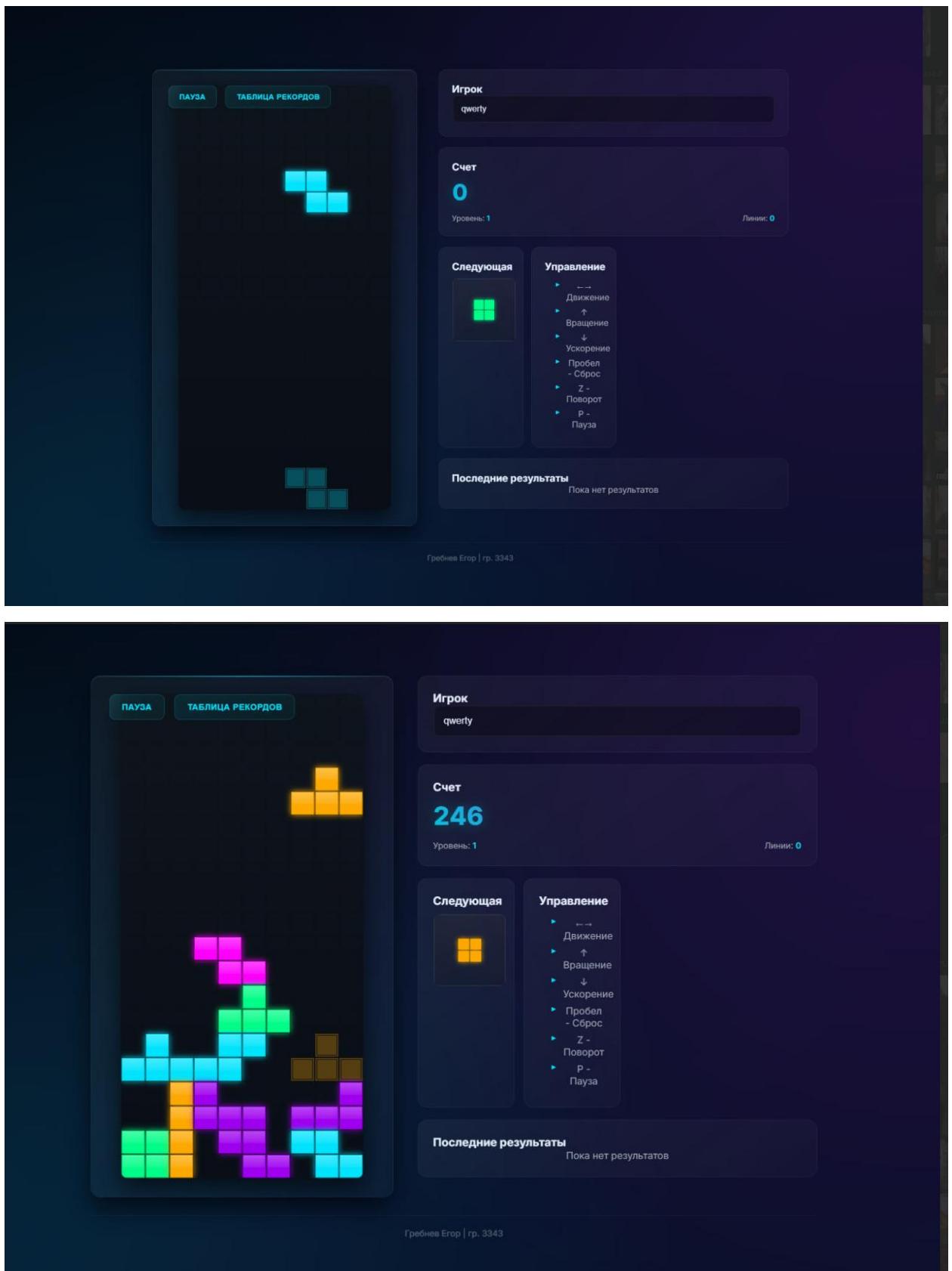


Рисунок 2 – Основная форма игры

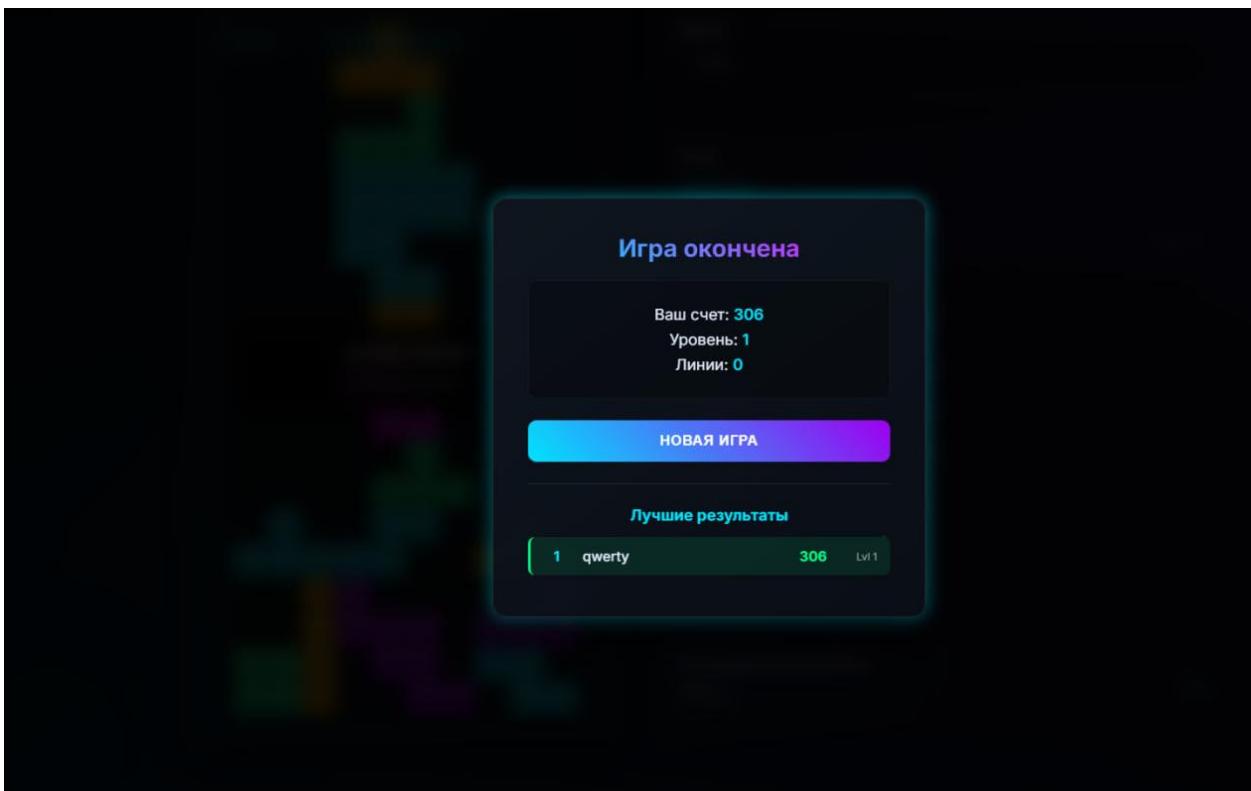


Рисунок 3 – Завершение игры

Вывод

Успешно создано приложение на языке JavaScript. Была реализована безопасная инфраструктура с настройкой nginx для работы по протоколу HTTPS с использованием сгенерированных асимметричных ключей. Разработан пользовательский интерфейс с системой ввода имени и сохранением данных в локальном хранилище браузера. Создан полнофункциональный игровой движок тетриса с генерацией фигур, управлением их движением и обработкой заполненных линий. Реализована система таблицы рекордов, которая отображает лучшие результаты игроков по окончании игры. Все компоненты приложения интегрированы в единую систему, обеспечивающую корректную работу игрового процесса.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: index.html

```
<!DOCTYPE html>
<html lang="ru">

    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Neon Tetris - Web (lb1)</title>
        <link rel="stylesheet" href="styles/style.css" />
        <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;700;800&display=swap" rel="stylesheet" />
    </head>

    <body>
        <div id="start-modal" class="modal-overlay active">
            <div class="modal">
                <h2>Neon Tetris</h2>
                <p>Введите ваше имя для начала игры</p>
                <input type="text" id="modal-input" placeholder="Ваше имя..." maxlength="15" autofocus />
                <button id="modal-button">Начать игру</button>
                <div class="leaderboard-preview">
                    <h3>Лучшие результаты</h3>
                    <div id="leaderboard-preview-list"></div>
                </div>
            </div>
        </div>

        <div id="pause-modal" class="modal-overlay">
            <div class="modal">
                <h2>Пауза</h2>
                <p>Игра приостановлена</p>
                <div class="modal-buttons">
                    <button id="resume-button">Продолжить</button>
                    <button id="restart-modal-button">Рестарт</button>
                </div>
            </div>
        </div>

        <div id="gameover-modal" class="modal-overlay">
            <div class="modal">
                <h2>Игра окончена</h2>
                <div class="score-display">
                    <div>Ваш счет: <span id="final-score">0</span></div>
                    <div>Уровень: <span id="final-level">1</span></div>
                    <div>Линии: <span id="final-lines">0</span></div>
                </div>
                <div class="modal-buttons">
                    <button id="gameover-restart">Новая игра</button>
                </div>
            </div>
        </div>
    </body>

```

```

<div class="leaderboard-preview">
    <h3>Лучшие результаты</h3>
    <div id="gameover-leaderboard"></div>
</div>
</div>
</div>

<div id="leaderboard-modal" class="modal-overlay">
    <div class="modal">
        <h2>Таблица рекордов</h2>
        <div class="leaderboard-full">
            <div id="leaderboard-list"></div>
        </div>
        <div class="modal-buttons">
            <button id="close-leaderboard">Закрыть</button>
            <button id="clear-leaderboard">Очистить таблицу</button>
        </div>
    </div>
</div>

<div class="wrapper">
    <div class="game-area">
        <div class="canvas-wrap">
            <div class="overlay">
                <button class="control" id="btn-pause">Пausа</button>
                <button class="control" id="btn-leaderboard">Таблица
рекордов</button>
            </div>
            <canvas id="tetris"></canvas>
        </div>
    </div>

    <div class="panel">
        <div class="card player">
            <h3>Игрок</h3>
            <input type="text" id="username" placeholder="Введите
имя..." maxlength="15" />
        </div>

        <div class="card">
            <h3>Счет</h3>
            <div class="big" id="score">0</div>
            <div class="meta">
                <div>Уровень: <span id="level">1</span></div>
                <div>Линии: <span id="lines">0</span></div>
            </div>
        </div>

        <div class="mini-panels">
            <div class="card small">
                <h3>Следующая</h3>
                <canvas id="next" width="120" height="120"></canvas>
            </div>
        </div>

        <div class="card small">
            <h3>Управление</h3>
            <div class="legend">
                <ul>
                    <li>→ Движение</li>

```

```

        <li>↑ Вращение</li>
        <li>↓ Ускорение</li>
        <li>Пробел - Сброс</li>
        <li>Z - Поворот</li>
        <li>P - Пауза</li>
    </ul>
</div>
</div>
</div>

<div class="card">
    <h3>Последние результаты</h3>
    <div id="recent-scores"></div>
</div>
</div>
</div>

<div class="footer">Гребнев Егор | гр. 3343</div>
</div>

<script type="module" src="main.js"></script>
</body>

</html>

```

Название файла: style.scss

```

// style.scss
:root {
    --bg: #0b0f17;
    --card: #0f1720;
    --accent: #00e5ff;
    --accent-glow: 0 0 20px rgba(0, 229, 255, 0.6);
    --muted: #9aa4b2;
    --glass: rgba(255, 255, 255, 0.03);
    --radius: 16px;
    --gap: 20px;
    --neon-pink: #ff00ff;
    --neon-blue: #00e5ff;
    --neon-green: #00ff88;
    --neon-purple: #a000f0;
    --neon-orange: #ffaa00;
    --success: #00ff88;
    --warning: #ffaa00;
    --error: #ff4444;
}

* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

html,
body,
#root {
    height: 100%;
}

```

```

body {
    margin: 0;
    font-family: 'Inter', 'Segoe UI', Roboto, Arial, sans-serif;
    background: linear-gradient(135deg, #02040a 0%, #071229 50%, #0a0f2a 100%);
    color: #e6f0ff;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
    min-height: 100vh;
    display: flex;
    align-items: center;
    justify-content: center;
    padding: 24px;
    overflow-x: hidden;

    &::before {
        content: '';
        position: fixed;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        background:
            radial-gradient(circle at 20% 80%, rgba(0, 229, 255, 0.1) 0%, transparent 50%),
            radial-gradient(circle at 80% 20%, rgba(255, 0, 255, 0.1) 0%, transparent 50%);
        pointer-events: none;
        z-index: -1;
    }
}

.modal-overlay {
    position: fixed;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background: rgba(0, 0, 0, 0.9);
    display: none;
    align-items: center;
    justify-content: center;
    z-index: 1000;
    backdrop-filter: blur(10px);
    animation: fadeIn 0.5s ease-out;

    &.active {
        display: flex;
    }
}

.modal {
    background: linear-gradient(145deg, rgba(15, 23, 32, 0.95), rgba(11, 15, 23, 0.95));
    padding: 40px;
    border-radius: var(--radius);
    border: 1px solid rgba(255, 255, 255, 0.1);
}

```

```

        text-align: center;
        max-width: 500px;
        width: 90%;
        max-height: 90vh;
        overflow-y: auto;
        box-shadow: 0 20px 40px rgba(0, 0, 0, 0.5), var(--accent-glow);
        animation: slideUp 0.5s ease-out;

    h2 {
        margin-bottom: 20px;
        background: linear-gradient(45deg, var(--neon-blue), var(--neon-pink));
        -webkit-background-clip: text;
        -webkit-text-fill-color: transparent;
        font-size: 28px;
        font-weight: 700;
    }

    p {
        color: var(--muted);
        margin-bottom: 25px;
        font-size: 16px;
    }
}

.modal input {
    width: 100%;
    padding: 15px;
    margin: 20px 0;
    border-radius: 10px;
    border: 2px solid rgba(255, 255, 255, 0.1);
    background: rgba(0, 0, 0, 0.3);
    color: white;
    font-size: 16px;
    transition: all 0.3s ease;

    &:focus {
        outline: none;
        border-color: var(--neon-blue);
        box-shadow: 0 0 15px rgba(0, 229, 255, 0.3);
    }
}

.modal-buttons {
    display: flex;
    flex-direction: column;
    gap: 12px;
    margin: 25px 0;
}

.modal button {
    background: linear-gradient(45deg, var(--neon-blue), var(--neon-purple));
    color: white;
    border: none;
    padding: 15px 30px;
    border-radius: 10px;
    font-weight: 600;
}

```

```

        cursor: pointer;
        font-size: 16px;
        transition: all 0.3s ease;
        text-transform: uppercase;
        letter-spacing: 1px;

        &:hover {
            transform: translateY(-2px);
            box-shadow: 0 10px 20px rgba(0, 229, 255, 0.3);
        }

        &.secondary {
            background: transparent;
            border: 2px solid var(--neon-blue);

            &:hover {
                background: rgba(0, 229, 255, 0.1);
            }
        }
    }

    .score-display {
        background: rgba(0, 0, 0, 0.3);
        padding: 20px;
        border-radius: 10px;
        margin: 20px 0;
        border: 1px solid rgba(255, 255, 255, 0.05);

        div {
            margin: 8px 0;
            font-size: 18px;

            span {
                color: var(--neon-blue);
                font-weight: 600;
            }
        }
    }

    .leaderboard-preview {
        margin-top: 25px;
        padding-top: 25px;
        border-top: 1px solid rgba(255, 255, 255, 0.1);

        h3 {
            color: var(--neon-blue);
            margin-bottom: 15px;
            font-size: 18px;
        }
    }

    .leaderboard-full {
        max-height: 400px;
        overflow-y: auto;
        overflow-x: hidden;
        margin: 20px 0;
    }
}

```

```

.leaderboard-item {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 12px 15px;
    margin: 8px 0;
    background: rgba(0, 0, 0, 0.2);
    border-radius: 8px;
    border-left: 3px solid var(--neon-blue);
    transition: all 0.3s ease;

    &:hover {
        background: rgba(0, 229, 255, 0.1);
        transform: translateX(5px);
    }

    &.current {
        border-left-color: var(--neon-green);
        background: rgba(0, 255, 136, 0.1);
    }

    .rank {
        font-weight: 700;
        color: var(--neon-blue);
        min-width: 30px;
    }

    .name {
        flex: 1;
        text-align: left;
        margin: 0 15px;
        font-weight: 500;
    }

    .score {
        font-weight: 700;
        color: var(--neon-green);
        min-width: 80px;
    }

    .details {
        font-size: 12px;
        color: var(--muted);
        margin-left: 10px;
    }
}

.recent-score-item {
    display: flex;
    justify-content: space-between;
    padding: 8px 0;
    border-bottom: 1px solid rgba(255, 255, 255, 0.05);
    font-size: 14px;

    &:last-child {
        border-bottom: none;
    }
}

```

```

.score-value {
    color: var(--neon-green);
    font-weight: 600;
}

.date {
    color: var(--muted);
    font-size: 12px;
}
}

.wrapper {
    width: 100%;
    max-width: 1200px;
    animation: fadeIn 1s ease-out;
}

.game-area {
    display: grid;
    grid-template-columns: 500px 1fr;
    gap: 40px;
    align-items: start;
}

.canvas-wrap {
    position: relative;
    width: 500px;
    padding: 30px;
    background: linear-gradient(145deg, rgba(255, 255, 255, 0.05),
    rgba(255, 255, 255, 0.02));
    border-radius: var(--radius);
    box-shadow:
        0 20px 40px rgba(3, 8, 20, 0.8),
        inset 0 1px 0 rgba(255, 255, 255, 0.1);
    border: 1px solid rgba(255, 255, 255, 0.05);
    backdrop-filter: blur(10px);

    &::before {
        content: '';
        position: absolute;
        top: 0;
        left: 0;
        right: 0;
        height: 1px;
        background: linear-gradient(90deg, transparent, var(--neon-blue), transparent);
    }
}

canvas#tetris {
    display: block;
    width: 400px;
    height: 800px;
    margin: 0 auto;
    background: linear-gradient(160deg, #0a1018 0%, #050a12 100%);
    border-radius: 12px;
    box-shadow:
        inset 0 0 0 2px rgba(255, 255, 255, 0.03),

```

```

        0 10px 30px rgba(0, 0, 0, 0.5);
image-rendering: crisp-edges;
transition: all 0.3s ease;

    &:hover {
        box-shadow:
            inset 0 0 0 2px rgba(255, 255, 255, 0.05),
            0 15px 40px rgba(0, 0, 0, 0.6);
    }
}

.overlay {
    position: absolute;
    left: 30px;
    top: 30px;
    display: flex;
    gap: 15px;
    z-index: 10;
}

.control {
    background: linear-gradient(145deg, rgba(0, 229, 255, 0.1),
    rgba(0, 229, 255, 0.05));
    color: var(--neon-blue);
    border: 1px solid rgba(0, 229, 255, 0.2);
    padding: 12px 20px;
    font-weight: 600;
    border-radius: 10px;
    cursor: pointer;
    transition: all 0.3s ease;
    backdrop-filter: blur(10px);
    text-transform: uppercase;
    letter-spacing: 0.5px;
    font-size: 14px;
}

    &:hover {
        transform: translateY(-2px);
        box-shadow: 0 8px 20px rgba(0, 229, 255, 0.3);
        border-color: rgba(0, 229, 255, 0.4);
    }

    &:active {
        transform: translateY(0);
    }
}

.panel {
    display: flex;
    flex-direction: column;
    gap: 20px;
}

.card {
    background: linear-gradient(145deg, rgba(255, 255, 255, 0.05),
    rgba(255, 255, 255, 0.02));
    padding: 25px;
    border-radius: var(--radius);
    border: 1px solid rgba(255, 255, 255, 0.05);
}

```

```

        backdrop-filter: blur(10px);
        transition: all 0.3s ease;
        position: relative;
        overflow: hidden;

        &::before {
            content: '';
            position: absolute;
            top: 0;
            left: -100%;
            width: 100%;
            height: 100%;
            background: linear-gradient(90deg, transparent, rgba(255, 255, 255, 0.05), transparent);
            transition: left 0.5s ease;
        }

        &:hover::before {
            left: 100%;
        }

        &:hover {
            transform: translateY(-5px);
            box-shadow: 0 15px 30px rgba(0, 0, 0, 0.4);
        }
    }

    .card.small {
        width: 160px;
        text-align: center;
    }

    .player input {
        width: 100%;
        padding: 15px;
        border-radius: 10px;
        border: 2px solid rgba(255, 255, 255, 0.05);
        background: rgba(0, 0, 0, 0.3);
        color: inherit;
        font-size: 16px;
        transition: all 0.3s ease;

        &:focus {
            outline: none;
            border-color: var(--neon-blue);
            box-shadow: 0 0 15px rgba(0, 229, 255, 0.2);
        }
    }

    .big {
        font-size: 42px;
        font-weight: 800;
        background: linear-gradient(45deg, var(--neon-blue), var(--neon-pink));
        -webkit-background-clip: text;
        -webkit-text-fill-color: transparent;
        margin-top: 10px;
        text-shadow: 0 0 20px rgba(0, 229, 255, 0.3);
    }

```

```

        animation: pulse 2s ease-in-out infinite;
    }

.meta {
    display: flex;
    justify-content: space-between;
    margin-top: 15px;
    color: var(--muted);
    font-size: 14px;

    span {
        color: var(--neon-blue);
        font-weight: 600;
    }
}

.mini-panels {
    display: flex;
    gap: 15px;
}

#next {
    width: 120px;
    height: 120px;
    margin: 10px auto;
    background: rgba(0, 0, 0, 0.2);
    border-radius: 10px;
    border: 1px solid rgba(255, 255, 255, 0.05);
}

.legend ul {
    margin: 15px 0 0 20px;
    color: var(--muted);
    list-style: none;

    li {
        margin-bottom: 8px;
        padding-left: 20px;
        position: relative;

        &::before {
            content: '▶';
            position: absolute;
            left: 0;
            color: var(--neon-blue);
            font-size: 10px;
        }
    }
}

.footer {
    margin-top: 30px;
    text-align: center;
    color: rgba(255, 255, 255, 0.3);
    font-size: 14px;
    padding: 20px;
    border-top: 1px solid rgba(255, 255, 255, 0.05);
}

```

```

// Анимации
@keyframes fadeIn {
    from {
        opacity: 0;
    }

    to {
        opacity: 1;
    }
}

@keyframes slideUp {
    from {
        opacity: 0;
        transform: translateY(30px);
    }

    to {
        opacity: 1;
        transform: translateY(0);
    }
}

@keyframes pulse {
    0%,
    100% {
        opacity: 1;
    }

    50% {
        opacity: 0.8;
    }
}

@keyframes glow {
    0%,
    100% {
        box-shadow: 0 0 5px var(--neon-blue);
    }

    50% {
        box-shadow: 0 0 20px var(--neon-blue), 0 0 30px var(--neon-blue);
    }
}

// Эффекты для игрового поля
.line-clear {
    animation: lineClear 0.3s ease-out;
}

@keyframes lineClear {
    0% {
        background-color: white;
    }
}

```

```

        100% {
            background-color: transparent;
        }
    }

    // Адаптивность
    @media (max-width: 1100px) {
        .game-area {
            grid-template-columns: 1fr;
            justify-items: center;
            gap: 30px;
        }

        .panel {
            width: 100%;
            max-width: 500px;
        }

        .mini-panels {
            justify-content: center;
        }
    }

    @media (max-width: 600px) {
        .canvas-wrap {
            width: 100%;
            padding: 20px;
        }

        canvas#tetris {
            width: 100%;
            height: auto;
            max-width: 400px;
            max-height: 800px;
        }

        .overlay {
            position: relative;
            left: 0;
            top: 0;
            justify-content: center;
            margin-bottom: 15px;
        }

        .modal {
            padding: 25px;
        }

        .modal-buttons {
            flex-direction: column;
        }
    }
}

```

Название файла: main.ts

```
// main.ts
import { Tetris } from "./engine/tetris";
```

```

import type { Piece } from "./engine/types";

const canvas = document.getElementById("tetris") as HTMLCanvasElement;
const ctx = canvas.getContext("2d", { alpha: false })!;
const nextCanvas = document.getElementById("next") as HTMLCanvasElement;
const nextCtx = nextCanvas.getContext("2d")!;

const scoreEl = document.getElementById("score")!;
const levelEl = document.getElementById("level")!;
const linesEl = document.getElementById("lines")!;
const usernameInput = document.getElementById("username") as HTMLInputElement;
const btnPause = document.getElementById("btn-pause") as HTMLButtonElement;
const btnLeaderboard = document.getElementById("btn-leaderboard") as HTMLButtonElement;
const recentScoresEl = document.getElementById("recent-scores") as HTMLDivElement;

// Модальные окна
const startModal = document.getElementById("start-modal") as HTMLDivElement;
const pauseModal = document.getElementById("pause-modal") as HTMLDivElement;
const gameoverModal = document.getElementById("gameover-modal") as HTMLDivElement;
const leaderboardModal = document.getElementById("leaderboard-modal") as HTMLDivElement;

// Кнопки модальных окон
const modalInput = document.getElementById("modal-input") as HTMLInputElement;
const modalButton = document.getElementById("modal-button") as HTMLButtonElement;
const resumeButton = document.getElementById("resume-button") as HTMLButtonElement;
const restartModalButton = document.getElementById("restart-modal-button") as HTMLButtonElement;
const gameoverRestart = document.getElementById("gameover-restart") as HTMLButtonElement;
const closeLeaderboard = document.getElementById("close-leaderboard") as HTMLButtonElement;
const clearLeaderboard = document.getElementById("clear-leaderboard") as HTMLButtonElement;

// Элементы отображения результатов
const finalScoreEl = document.getElementById("final-score") as HTMLSpanElement;
const finalLevelEl = document.getElementById("final-level") as HTMLSpanElement;
const finalLinesEl = document.getElementById("final-lines") as HTMLSpanElement;
const leaderboardPreviewList = document.getElementById("leaderboard-preview-list") as HTMLDivElement;
const gameoverLeaderboard = document.getElementById("gameover-leaderboard") as HTMLDivElement;

```

```

const leaderboardList = document.getElementById("leaderboard-list")
as HTMLDivElement;

const TILE = 40;
const C_W = 10;
const C_H = 20;
const LOGICAL_W = C_W * TILE;
const LOGICAL_H = C_H * TILE;

let game = new Tetris(C_W, C_H);

let username = "";
let animationFrameId: number;
let particleSystems: ParticleSystem[] = [];
let paused = false;

interface ScoreEntry {
    name: string;
    score: number;
    level: number;
    lines: number;
    date: number;
}

class Particle {
    x: number;
    y: number;
    vx: number;
    vy: number;
    life: number;
    maxLife: number;
    color: string;
    size: number;

    constructor(x: number, y: number, color: string) {
        this.x = x;
        this.y = y;
        this.vx = (Math.random() - 0.5) * 8;
        this.vy = (Math.random() - 0.5) * 8;
        this.life = 1;
        this.maxLife = 1;
        this.color = color;
        this.size = Math.random() * 3 + 1;
    }

    update(dt: number) {
        this.x += this.vx * dt;
        this.y += this.vy * dt;
        this.vy += 0.1 * dt;
        this.life -= 0.02 * dt;
    }

    draw(ctx: CanvasRenderingContext2D) {
        const alpha = this.life / this.maxLife;
        ctx.save();
        ctx.globalAlpha = alpha;
        ctx.fillStyle = this.color;
    }
}

```

```

        ctx.fillRect(this.x - this.size / 2, this.y - this.size / 2, this.size, this.size);
        ctx.restore();
    }
}

class ParticleSystem {
    particles: Particle[] = [];
    x: number;
    y: number;

    constructor(x: number, y: number, color: string, count: number = 20) {
        this.x = x;
        this.y = y;
        for (let i = 0; i < count; i++) {
            this.particles.push(new Particle(x, y, color));
        }
    }

    update(dt: number) {
        this.particles = this.particles.filter(particle => {
            particle.update(dt);
            return particle.life > 0;
        });
    }

    draw(ctx: CanvasRenderingContext2D) {
        this.particles.forEach(particle => particle.draw(ctx));
    }

    isAlive() {
        return this.particles.length > 0;
    }
}

// Leaderboard functions
function getLeaderboard(): ScoreEntry[] {
    const stored = localStorage.getItem("tetris.leaderboard");
    return stored ? JSON.parse(stored) : [];
}

function saveToLeaderboard(entry: ScoreEntry) {
    const leaderboard = getLeaderboard();

    // Проверяем, не является ли это дубликатом (по имени и счету)
    const isDuplicate = leaderboard.some(item =>
        item.name === entry.name && item.score === entry.score
    );

    // Если это не дубликат, добавляем в таблицу
    if (!isDuplicate) {
        leaderboard.push(entry);
    }

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

    const topScores = leaderboard.slice(0, 20);
}

```

```

        localStorage.setItem("tetris.leaderboard",
JSON.stringify(topScores));

    return topScores.findIndex(score => score.date === entry.date
&& score.name === entry.name) + 1;
}

function getRecentScores(): ScoreEntry[] {
    const leaderboard = getLeaderboard();
    return leaderboard
        .sort((a, b) => b.date - a.date)
        .slice(0, 5);
}

function renderLeaderboard(list: ScoreEntry[], container:
HTMLElement, showCurrent: boolean = false) {
    container.innerHTML = '';

    list.forEach((entry, index) => {
        const item = document.createElement('div');
        item.className = 'leaderboard-item';
        if (showCurrent && entry.name === username) {
            item.classList.add('current');
        }

        const date = new Date(entry.date).toLocaleDateString();
        item.innerHTML =
            `<div class="rank">${index + 1}</div>
             <div class="name">${entry.name}</div>
             <div class="score">${entry.score.toLocaleString()}</div>
             <div class="details">Lvl ${entry.level}</div>
            `;
        container.appendChild(item);
    });

    if (list.length === 0) {
        container.innerHTML = '<div style="text-align: center; color: var(--muted); padding: 20px;">Пока нет результатов</div>';
    }
}

function renderRecentScores() {
    const recentScores = getRecentScores();
    recentScoresEl.innerHTML = '';

    if (recentScores.length === 0) {
        recentScoresEl.innerHTML = '<div style="text-align: center; color: var(--muted);">Пока нет результатов</div>';
        return;
    }

    recentScores.forEach(entry => {
        const item = document.createElement('div');
        item.className = 'recent-score-item';
        const date = new Date(entry.date).toLocaleDateString();
        item.innerHTML =
            `<div>
             <div>${entry.name}</div>
            `;
    });
}

```

```

        <div class="date">${date}</div>
    </div>
    <div class="score-
value">${entry.score.toLocaleString()}</div>
    ;
    recentScoresEl.appendChild(item);
}
}

function showModal(modal: HTMLElement) {
    document.querySelectorAll('.modal-overlay').forEach(m => {
        (m as HTMLElement).classList.remove('active');
    });
    modal.classList.add('active');
}

function hideAllModals() {
    document.querySelectorAll('.modal-overlay').forEach(m => {
        (m as HTMLElement).classList.remove('active');
    });
}

function handleGameOver() {
    if (!username) {
        username = "Игрок";
        usernameInput.value = username;
        localStorage.setItem("tetris.username", username);
    }

    if (game.gameOver) {
        const entry: ScoreEntry = {
            name: username,
            score: game.score,
            level: game.level,
            lines: game.lines,
            date: Date.now()
        };

        const position = saveToLeaderboard(entry);

        finalScoreEl.textContent = game.score.toLocaleString();
        finalLevelEl.textContent = String(game.level);
        finalLinesEl.textContent = String(game.lines);

        const leaderboard = getLeaderboard();
        renderLeaderboard(leaderboard.slice(0, 5),
gameoverLeaderboard, true);
        renderRecentScores();

        renderLeaderboard(leaderboard.slice(0, 3),
leaderboardPreviewList);
    }
}

showModal(gameoverModal);
}

// Event Listeners
modalButton.addEventListener("click", () => {

```

```

        if (modalInput.value.trim()) {
            username = modalInput.value.trim();
            localStorage.setItem("tetris.username", username);
            hideAllModals();
            usernameInput.value = username;
            createParticleSystem(LOGICAL_W / 2, LOGICAL_H / 2,
"#00e5ff", 50);
            renderRecentScores();

            // Перезапускаем игру при первом входе
            game = new Tetris(C_W, C_H);
            paused = false;
        }
    });

    modalInput.addEventListener("keypress", (e) => {
        if (e.key === "Enter") {
            modalButton.click();
        }
    });

    resumeButton.addEventListener("click", () => {
        hideAllModals();
        paused = false;
        btnPause.textContent = "Пауза";
    });

    restartModalButton.addEventListener("click", () => {
        hideAllModals();
        game = new Tetris(C_W, C_H);
        paused = false;
        btnPause.textContent = "Пауза";
        particleSystems = [];
        createParticleSystem(LOGICAL_W / 2, LOGICAL_H / 2, "#00ff88",
30);
    });

    gameoverRestart.addEventListener("click", () => {
        hideAllModals();
        game = new Tetris(C_W, C_H);
        paused = false;
        btnPause.textContent = "Пауза";
        particleSystems = [];
        createParticleSystem(LOGICAL_W / 2, LOGICAL_H / 2, "#00ff88",
30);
    });

    btnLeaderboard.addEventListener("click", () => {
        const leaderboard = getLeaderboard();
        renderLeaderboard(leaderboard, leaderboardList, true);
        showModal(leaderboardModal);
    });

    closeLeaderboard.addEventListener("click", () => {
        hideAllModals();
    });

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

```

```

        if (confirm("Вы уверены, что хотите очистить таблицу
рекордов?")) {
            localStorage.removeItem("tetris.leaderboard");
            renderLeaderboard([], leaderboardList);
            renderRecentScores();
            const leaderboard = getLeaderboard();
                renderLeaderboard(leaderboard.slice(0,      3),
leaderboardPreviewList);
                renderLeaderboard(leaderboard.slice(0,      5),
gameoverLeaderboard, true);
            }
        });

// Инициализация
if (localStorage.getItem("tetris.username")) {
    username = localStorage.getItem("tetris.username")!;
    usernameInput.value = username;
    hideAllModals();
    renderRecentScores();
} else {
    showModal(startModal);
}

// Показ превью таблицы рекордов при старте
const initialLeaderboard = getLeaderboard();
renderLeaderboard(initialLeaderboard.slice(0,
leaderboardPreviewList),      3),

function setupCanvasScale() {
    const dpr = window.devicePixelRatio || 1;

    canvas.width = Math.round(LOGICAL_W * dpr);
    canvas.height = Math.round(LOGICAL_H * dpr);
    canvas.style.width = `${LOGICAL_W}px`;
    canvas.style.height = `${LOGICAL_H}px`;
    ctx.setTransform(dpr, 0, 0, dpr, 0, 0);

    const nextDpr = dpr;
        nextCanvas.width = Math.round(nextCanvas.clientWidth *
nextDpr);
        nextCanvas.height = Math.round(nextCanvas.clientHeight *
nextDpr);
        nextCtx.setTransform(nextDpr, 0, 0, nextDpr, 0, 0);
    }

    setupCanvasScale();
    window.addEventListener("resize", setupCanvasScale);

    function gravityInterval(level: number) {
        return Math.max(100, 100 * Math.pow(0.8, Math.max(1, level) -
1));
    }

    let last = performance.now();
    let accumulator = 0;
    let gravity = gravityInterval(game.level);

    function drawGhostPiece(ctx: CanvasRenderingContext2D) {

```

```

        if (!game.currentPiece) return;

        let dropDistance = 0;
        let testPiece = { ...game.currentPiece };

            while (!checkCollision(game.board, { ...testPiece, y:
testPiece.y + 1 })) {
                testPiece.y++;
                dropDistance++;
            }

        if (dropDistance === 0) return;

            const ghostPiece = { ...game.currentPiece, y:
game.currentPiece.y + dropDistance };
            const m = ghostPiece.shape;

            ctx.save();
            ctx.globalAlpha = 0.3;

            for (let y = 0; y < m.length; y++) {
                for (let x = 0; x < m[y].length; x++) {
                    if (m[y][x]) {
                        const px = ghostPiece.x + x;
                        const py = ghostPiece.y + y;
                        if (py >= 0) {
                            ctx.strokeStyle = ghostPiece.color;
                            ctx.lineWidth = 2;
                            ctx.strokeRect(px * TILE + 2, py * TILE + 2,
TILE - 4, TILE - 4);

                            ctx.fillStyle = ghostPiece.color;
                            ctx.fillRect(px * TILE + 4, py * TILE + 4, TILE
- 8, TILE - 8);
                        }
                    }
                }
            }

            ctx.restore();
        }

    function checkCollision(board: (string | null)[][], piece: Piece):
boolean {
    // Проверяем, что board существует и имеет правильные размеры
    if (!board || board.length === 0 || board[0].length === 0) {
        return false;
    }

    for (let y = 0; y < piece.shape.length; y++) {
        for (let x = 0; x < piece.shape[y].length; x++) {
            if (piece.shape[y][x]) {
                const boardX = piece.x + x;
                const boardY = piece.y + y;

                // Проверка границ
                if (boardX < 0 || boardX >= board[0].length ||
boardY >= board.length) {

```

```

        return true;
    }

    // Проверка столкновения с другими фигурами (только
если boardY >= 0)
        if (boardY >= 0 && board[boardY] &&
board[boardY][boardX] !== null) {
            return true;
        }
    }
}

return false;
}

function updateParticles(dt: number) {
    particleSystems = particleSystems.filter(system => {
        system.update(dt);
        return system.isAlive();
    });
}

function createParticleSystem(x: number, y: number, color: string,
count: number = 20) {
    particleSystems.push(new ParticleSystem(x, y, color, count));
}

function render() {
    // Очистка canvas с градиентом
    const gradient = ctx.createLinearGradient(0, 0, 0, LOGICAL_H);
    gradient.addColorStop(0, "#0a1018");
    gradient.addColorStop(1, "#050a12");
    ctx.fillStyle = gradient;
    ctx.fillRect(0, 0, LOGICAL_W, LOGICAL_H);

    // Сетка
    ctx.strokeStyle = "rgba(255, 255, 255, 0.03)";
    ctx.lineWidth = 1;
    for (let x = 0; x <= C_W; x++) {
        ctx.beginPath();
        ctx.moveTo(x * TILE, 0);
        ctx.lineTo(x * TILE, LOGICAL_H);
        ctx.stroke();
    }
    for (let y = 0; y <= C_H; y++) {
        ctx.beginPath();
        ctx.moveTo(0, y * TILE);
        ctx.lineTo(LOGICAL_W, y * TILE);
        ctx.stroke();
    }

    // Отрисовка стакана
    for (let y = 0; y < C_H; y++) {
        for (let x = 0; x < C_W; x++) {
            const cell = game.board[y][x];
            const px = x * TILE;
            const py = y * TILE;

```

```

        if (cell) {
            drawTile(ctx, x, y, cell);
        } else {
            ctx.fillStyle = "rgba(255, 255, 255, 0.02)";
            ctx.fillRect(px, py, TILE, TILE);
        }

        ctx.strokeStyle = "rgba(0, 0, 0, 0.25)";
        ctx.strokeRect(px + 0.5, py + 0.5, TILE - 1, TILE - 1);
    }
}

drawGhostPiece(ctx);

if (game.currentPiece) {
    drawPiece(ctx, game.currentPiece);
}

particleSystems.forEach(system => system.draw(ctx));

renderPreview(nextCtx, game.nextPiece);

scoreEl.textContent = game.score.toLocaleString();
levelEl.textContent = String(game.level);
linesEl.textContent = String(game.lines);

if (game.gameOver) {
    ctx.fillStyle = "rgba(2, 2, 8, 0.85)";
    const boxH = 100;
    ctx.fillRect(0, (LOGICAL_H / 2) - (boxH / 2), LOGICAL_W, boxH);

    ctx.fillStyle = "#fff";
    ctx.font = "bold 24px Inter, Arial";
    ctx.textAlign = "center";
    ctx.fillText("GAME OVER", LOGICAL_W / 2, LOGICAL_H / 2 - 10);

    ctx.font = "16px Inter, Arial";
    ctx.fillText("Нажми Рестарт", LOGICAL_W / 2, LOGICAL_H / 2 + 20);
}

handleGameOver();
}
}

function drawTile(ctx: CanvasRenderingContext2D, x: number, y: number, color: string) {
    const px = x * TILE;
    const py = y * TILE;

    ctx.fillStyle = color;
    ctx.fillRect(px + 1, py + 1, TILE - 2, TILE - 2);

    ctx.shadowColor = color;
    ctx.shadowBlur = 10;
    ctx.fillRect(px + 1, py + 1, TILE - 2, TILE - 2);
}

```

```

        ctx.shadowBlur = 0;

        const gradient = ctx.createLinearGradient(px, py, px, py +
TILE);
        gradient.addColorStop(0, "rgba(255, 255, 255, 0.3)");
        gradient.addColorStop(0.5, "rgba(255, 255, 255, 0.1)");
        gradient.addColorStop(1, "transparent");
        ctx.fillStyle = gradient;
        ctx.fillRect(px + 2, py + 2, TILE - 4, TILE / 2);

        ctx.strokeStyle = "rgba(0, 0, 0, 0.4)";
        ctx.lineWidth = 1;
        ctx.strokeRect(px + 1, py + 1, TILE - 2, TILE - 2);
    }

function drawPiece(ctx: CanvasRenderingContext2D, piece: Piece) {
    const m = piece.shape;
    for (let y = 0; y < m.length; y++) {
        for (let x = 0; x < m[y].length; x++) {
            if (m[y][x]) {
                const px = piece.x + x;
                const py = piece.y + y;
                if (py >= 0) {
                    drawTile(ctx, px, py, piece.color);
                }
            }
        }
    }
}

function renderPreview(ctx: CanvasRenderingContext2D, piece: Piece
| null) {
    ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);

    const gradient = ctx.createLinearGradient(0, 0, 0,
ctx.canvas.height);
    gradient.addColorStop(0, "rgba(255, 255, 255, 0.05)");
    gradient.addColorStop(1, "rgba(255, 255, 255, 0.02)");
    ctx.fillStyle = gradient;
    ctx.fillRect(0, 0, ctx.canvas.width, ctx.canvas.height);

    ctx.strokeStyle = "rgba(255, 255, 255, 0.1)";
    ctx.lineWidth = 1;
    ctx.strokeRect(0.5, 0.5, ctx.canvas.width - 1,
ctx.canvas.height - 1);

    if (!piece) return;

    const m = piece.shape;
    const scale = 20;
    const offsetX = (ctx.canvas.width - m[0].length * scale) / 2;
    const offsetY = (ctx.canvas.height - m.length * scale) / 2;

    for (let y = 0; y < m.length; y++) {
        for (let x = 0; x < m[y].length; x++) {
            if (m[y][x]) {
                ctx.fillStyle = piece.color;

```

```

        ctx.fillRect(offsetX + x * scale + 1, offsetY + y
* scale + 1, scale - 2, scale - 2);

        ctx.shadowColor = piece.color;
        ctx.shadowBlur = 8;
        ctx.fillRect(offsetX + x * scale + 1, offsetY + y
* scale + 1, scale - 2, scale - 2);
        ctx.shadowBlur = 0;
    }
}
}

// Управление
const keyState: { [key: string]: boolean } = {};
const keyRepeat: { [key: string]: number } = {};
const REPEAT_DELAY = 200;
const REPEAT_INTERVAL = 50;

document.addEventListener("keydown", (e) => {
    if (e.key === "p" || e.key === "P" || e.key === "з" || e.key
==== "З") {
        e.preventDefault();
        paused = !paused;
        btnPause.textContent = paused ? "Продолжить" : "Пауза";
        if (paused) {
            showModal(pauseModal);
        } else {
            hideAllModals();
        }
        return;
    }

    if (paused || game.gameOver || !username) return;

    if (!keyState[e.key]) {
        keyState[e.key] = true;
        keyRepeat[e.key] = performance.now();
        handleKeyPress(e.key);
    }
});

document.addEventListener("keyup", (e) => {
    keyState[e.key] = false;
});

function handleKeyPress(key: string) {
    switch (key) {
        case "ArrowLeft":
        case "a":
        case "A":
        case "ϕ":
        case "Φ":
            game.move(-1);
            break;
        case "ArrowRight":
        case "d":
        case "D":
    }
}

```

```

        case "B":
        case "B":
            game.move(1);
            break;
        case "ArrowDown":
        case "S":
        case "S":
        case "Ы":
        case "Ы":
            game.softDrop();
            break;
        case "ArrowUp":
        case "W":
        case "W":
        case "Ц":
        case "Ц":
            game.rotate();
            break;
        case " ":
            game.hardDrop();
            if (game.currentPiece) {
                createParticleSystem(
                    game.currentPiece.x * TILE + TILE / 2,
                    game.currentPiece.y * TILE + TILE / 2,
                    game.currentPiece.color,
                    30
                );
            }
            break;
        case "z":
        case "Z":
        case "Я":
        case "Я":
            game.rotate(-1);
            break;
    }
}

function processKeyRepeat(now: number) {
    if (paused || game.gameOver || !username) return;

    Object.keys(keyState).forEach(key => {
        if (keyState[key] && now - keyRepeat[key] > REPEAT_DELAY) {
            if ((now - keyRepeat[key] - REPEAT_DELAY) % REPEAT_INTERVAL < 16) {
                handleKeyPress(key);
            }
        }
    });
}

function gameLoop(now = performance.now()) {
    const dt = Math.min(now - last, 1000 / 60);
    last = now;

    processKeyRepeat(now);
}

```

```

        if (!paused && !game.gameOver && username) {
            accumulator += dt;
            while (accumulator >= gravity) {

                game.tick();

                if (game.gameOver) {
                    break;
                }
                accumulator -= gravity;
            }
            gravity = gravityInterval(game.level);
        }

        updateParticles(dt);
        render();

        if (game.gameOver
&& !gameoverModal.classList.contains('active')) {
            handleGameOver();
        }

        animationFrameId = requestAnimationFrame(gameLoop);
    }

    animationFrameId = requestAnimationFrame(gameLoop);

    btnPause.addEventListener("click", () => {
        paused = !paused;
        btnPause.textContent = paused ? "Продолжить" : "Пауза";
        if (paused) {
            showModal(pauseModal);
        } else {
            hideAllModals();
        }
    });

    usernameInput.addEventListener("change", () => {
        username = usernameInput.value.trim();
        localStorage.setItem("tetris.username", username);
    });

    window.addEventListener("beforeunload", () => {
        cancelAnimationFrame(animationFrameId);
    });

```

Название файла: engine/tetris.ts

```

import { Piece, GameState } from "./types";
import { createPiece, checkCollision, rotatePiece } from "./utils";

export class Tetris implements GameState{
    public board: (string | null)[][];
    public currentPiece: Piece | null = null;
    public nextPiece: Piece | null = null;
    public score: number = 0;
    public level: number = 1;
    public lines: number = 0;
    public gameOver: boolean = false;

```

```

constructor(public width: number, public height: number) {
    this.board = this.createBoard();
    this.nextPiece = createPiece();
    this.spawnPiece();
}

private createBoard(): (string | null)[][] {
    return Array.from({ length: this.height }, () =>
        Array(this.width).fill(null));
}

public spawnPiece(): void {
    this.currentPiece = this.nextPiece || createPiece();
    this.nextPiece = createPiece();

    // Проверяем, можно ли разместить новую фигуру
    if (this.currentPiece && checkCollision(this.board,
this.currentPiece)) {
        this.gameOver = true;
    }
}

public move(dx: number): void {
    if (!this.currentPiece || this.gameOver) return;

    const newPiece = { ...this.currentPiece, x:
this.currentPiece.x + dx };
    if (!checkCollision(this.board, newPiece)) {
        this.currentPiece = newPiece;
    }
}

public rotate(direction: number = 1): void {
    if (!this.currentPiece || this.gameOver) return;

    const newPiece = rotatePiece(this.currentPiece, direction);
    if (!checkCollision(this.board, newPiece)) {
        this.currentPiece = newPiece;
    }
}

public softDrop(): void {
    if (!this.currentPiece || this.gameOver) return;

    const newPiece = { ...this.currentPiece, y:
this.currentPiece.y + 1 };
    if (!checkCollision(this.board, newPiece)) {
        this.currentPiece = newPiece;
        this.score += 1;
    } else {
        this.lockPiece();
    }
}

public hardDrop(): void {
    if (!this.currentPiece || this.gameOver) return;
}

```

```

        let dropDistance = 0;
        let testPiece = { ...this.currentPiece };

            while (!checkCollision(this.board, { ...testPiece, y:
testPiece.y + 1 })) {
                testPiece.y++;
                dropDistance++;
            }

        this.currentPiece = testPiece;
        this.score += dropDistance * 2;
        this.lockPiece();
    }

private lockPiece(): void {
    if (!this.currentPiece) return;

    // Помещаем фигуру на доску
    for (let y = 0; y < this.currentPiece.shape.length; y++) {
        for (let x = 0; x < this.currentPiece.shape[y].length;
x++) {
            if (this.currentPiece.shape[y][x]) {
                const boardY = this.currentPiece.y + y;
                const boardX = this.currentPiece.x + x;

                if (boardY >= 0 && boardX >= 0 && boardY <
this.height && boardX < this.width) {
                    this.board[boardY][boardX] =
this.currentPiece.color;
                }
            }
        }
    }

    const linesCleared = this.clearLines();
    this.updateScore(linesCleared);

    this.spawnPiece();
}

private clearLines(): number {
    let linesCleared = 0;
    let y = this.height - 1;

    while (y >= 0) {
        if (this.board[y].every(cell => cell !== null)) {
            this.board.splice(y, 1);
            this.board.unshift(Array(this.width).fill(null));
            linesCleared++;
            // Не уменьшаем y, так как нужно проверить новую
линию на этой позиции
        } else {
            y--;
        }
    }

    return linesCleared;
}

```

```

    private updateScore(linesCleared: number): void {
        const linePoints = [0, 40, 100, 300, 1200];
        this.score += linePoints[linesCleared] * this.level;
        this.lines += linesCleared;
        this.level = Math.floor(this.lines / 1) + 1;
    }

    public tick(): void {
        if (this.gameOver) return;

        if (!this.currentPiece) {
            this.spawnPiece();
            return;
        }

        const newPiece = { ...this.currentPiece, y:
this.currentPiece.y + 1 };
        if (!checkCollision(this.board, newPiece)) {
            this.currentPiece = newPiece;
        } else {
            this.lockPiece();
        }
    }

    public reset(): void {
        this.board = this.createBoard();
        this.currentPiece = null;
        this.nextPiece = createPiece();
        this.score = 0;
        this.level = 1;
        this.lines = 0;
        this.gameOver = false;
        this.spawnPiece();
    }
}

```

Название файла: utils.ts

```

import { Piece } from "./types";

export const COLORS = [
    "#00e5ff", // neon blue
    "#ff00ff", // neon pink
    "#00ff88", // neon green
    "#a000f0", // neon purple
    "#ffaa00", // neon orange
];

export const SHAPES = [
    [
        [1, 1, 1, 1] // I

```

```

] ,
[
  [1, 1],
  [1, 1] // O
] ,
[
  [
    [0, 1, 0],
    [1, 1, 1] // T
] ,
[
  [
    [1, 0, 0],
    [1, 1, 1] // L
] ,
[
  [
    [0, 0, 1],
    [1, 1, 1] // J
] ,
[
  [
    [0, 1, 1],
    [1, 1, 0] // S
] ,
[
  [
    [1, 1, 0],
    [0, 1, 1] // Z
]
];
}

export function createPiece(): Piece {
  const shapeIndex = Math.floor(Math.random() * SHAPES.length);
  const colorIndex = Math.floor(Math.random() * COLORS.length);

  return {
    shape: SHAPES[shapeIndex],
    x: Math.floor((10 - SHAPES[shapeIndex][0].length) / 2),
    y: 0,
    color: COLORS[colorIndex]
  };
}

```

```

export function checkCollision(board: (string | null)[][], piece: Piece): boolean { // Обновлен тип

    for (let y = 0; y < piece.shape.length; y++) {
        for (let x = 0; x < piece.shape[y].length; x++) {
            if (piece.shape[y][x]) {
                const boardX = piece.x + x;
                const boardY = piece.y + y;

                // Проверяем выход за границы
                if (boardX < 0 || boardX >= board[0].length || boardY >= board.length) {
                    return true;
                }
            }
        }
    }

    // Проверяем столкновение с другими фигурами (только если внутри игрового поля)
    if (boardY >= 0 && board[boardY][boardX] !== null) {
        return true;
    }
}

return false;
}

export function rotatePiece(piece: Piece, direction: number = 1): Piece {
    const shape = piece.shape;
    const rows = shape.length;
    const cols = shape[0].length;

    const newShape: number[][] = [];

    if (direction === 1) { // По часовой стрелке
        for (let x = 0; x < cols; x++) {
            newShape[x] = [];
            for (let y = rows - 1; y >= 0; y--) {
                newShape[x][rows - 1 - y] = shape[y][x];
            }
        }
    }
}

```

```

        }
    }

} else { // Против часовой стрелки
    for (let x = cols - 1; x >= 0; x--) {
        newShape[cols - 1 - x] = [];
        for (let y = 0; y < rows; y++) {
            newShape[cols - 1 - x][y] = shape[y][x];
        }
    }
}

return {
    ...piece,
    shape: newShape
};
}

```

Название файла types.ts

```

export interface Piece {
    shape: number[][];
    x: number;
    y: number;
    color: string;
}

export interface GameState {
    board: (string | null)[][][];
    currentPiece: Piece | null;
    nextPiece: Piece | null;
    score: number;
    level: number;
    lines: number;
    gameOver: boolean;
}

```