Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**РАЗРАБОТКА СЕТЕВОЙ КОМПЬЮТЕРНОЙ ИГРЫ В ЖАНРЕ TOP-DOWN SHOOTER**

БГУИР КП 6-05-0612-01-030 ПЗ

Студент Михович И.С.

Руководитель Болтак С.В.

Минск 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc199159579)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc199159580)

[1.1 Обзор аналогов 5](#_Toc199159581)

[1.2 Постановка задачи 7](#_Toc199159582)

[2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 8](#_Toc199159583)

[2.1 Структура программы 8](#_Toc199159584)

[2.2 Проектирование интерфейса программного средства 9](#_Toc199159585)

[2.3 Проектирование функционала программного средства 9](#_Toc199159586)

[3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 11](#_Toc199159587)

[3.1 Разработка интерфейсов 11](#_Toc199159588)

[3.2 Реализация основного функционала 11](#_Toc199159589)

[3.3 Работа с моделями в игре 12](#_Toc199159590)

[3.4 Работа с сетевым взаимодействием 13](#_Toc199159591)

[4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 15](#_Toc199159592)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 16](#_Toc199159593)

[5.1 Интерфейс программного средства 16](#_Toc199159594)

[5.2 Управление программным средством 17](#_Toc199159595)

[ЗАКЛЮЧЕНИЕ 19](#_Toc199159596)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20](#_Toc199159597)

[ПРИЛОЖЕНИЕ А. Исходный код программы 21](#_Toc199159598)

# ВВЕДЕНИЕ

История компьютерных игр началась с эпохального прорыва в 1961 году, когда Стив Рассел и его коллеги в Массачусетском технологическом институте разработали «Spacewar» для компьютера PDP-1. Эта игра, ставшая первой широко известной и влиятельной, заложила фундамент для интерактивных развлечений, вдохновив разработчиков на эксперименты с геймплеем и графикой. Спустя десятилетие, в 1970-х годах, с развитием сетевых технологий, таких как ARPANET, появились первые сетевые игры.

Одной из пионеров стала «Maze War» (1974), реализовавшая для многопользовательского режима модель «клиент-сервер» и позволившая игрокам сражаться в виртуальном лабиринте от первого лица. Эти ранние сетевые эксперименты стали отправной точкой для стремительного роста жанров и механик, подпитываемого совершенствованием компьютерных систем, интернета и графических технологий.

На волне этой эволюции, от аркадных автоматов до домашних компьютеров, сформировался и получил широкое распространение жанр Top-Down Shooter (шутер с видом сверху). Игры этого жанра, от классических представителей вроде «Gauntlet» до множества современных инди-проектов, ценятся за динамичный геймплей, тактическую глубину и отличную приспособленность для многопользовательских сражений. Вид сверху обеспечивает хороший обзор игровой ситуации, что критически важно для сетевых баталий. Доступность разработки и возможность сосредоточиться на игровом процессе, даже при относительно простой графике, сделали Top-Down шутеры популярным жанром в инди-индустрии.

Современная игровая индустрия предлагает бесчисленное множество решений, охватывающих любые вкусы и предпочтения. Сетевые многопользовательские игры, включая Top-Down шутеры, стали важной частью цифровой культуры, формируя активные сообщества, вдохновляющие на создание пользовательского контент.

Целью данной курсовой работы является разработка сетевой компьютерной игры в жанре Top-Down Shooter. Это включает анализ существующих аналогов в данном жанре, определение ключевых игровых механик и технических требований, а также создание работающей программы с использованием современных веб-технологий (HTML5, JavaScript для клиентской части) и серверной логики на Python (с применением WebSockets для сетевого взаимодействия), который будет отвечать актуальным стандартам геймплея, пользовательского интерфейса и технической реализации.

# **1 АНАЛИЗ** ПРЕДМЕТНОЙ ОБЛАСТИ

## Обзор аналогов

Для успешной разработки курсового проекта в жанре сетевого Top-Down Shooter, важно проанализировать существующие игры-аналоги. Это позволит выявить удачные решения, популярные механики и возможные направления для развития собственного проекта. Рассмотрим несколько ключевых примеров, которые демонстрируют различные аспекты жанра и технической реализации, актуальные для данной работы:

Первой из них станет игра «Enter the Gungeon». Её постер представлен на рисунке 1.1.



Рисунок 1.1 – Постер игры «Enter the Gungeon»

«Enter the Gungeon» — это динамичный roguelike-шутер с видом сверху, в котором игроки исследуют процедурно генерируемые подземелья, сражаясь с многочисленными врагами и боссами. Ключевыми особенностями являются огромное разнообразие оружия с уникальными механиками, система уклонений (перекатов) и кооперативный режим.

«Enter the Gungeon» является отличным примером хорошо проработанной боевой системы в Top-Down шутере: отзывчивое управление, разнообразие врагов и видов атак, а также баланс сложности. Данный проект хоть и сделан в жанре roguelike, анализ механик стрельбы, движения и типов противников может быть полезен.

Далее стоит рассмотреть игру «Brotato», относительно недавний проект в том же жанре roguelike с видом сверху. Постер игры представлен на рисунке 1.2.



Рисунок 1.2 – Постер игры «Brotato»

«Brotato» — аренный шутер с элементами «рогалика», где главная цель игрока – играя за картофелину, дать отпор наступающим ордам пришельцев. В игре представлено несколько видов оружия, десятки предметов на выбор, присутствует кооператив, возможна индивидуальная настройка забега.

Представленные до этого примеры игр в жанре top-down shooter пересекались с жанром roguelike. Игра ниже представляет собой классический top-down shooter, для многих ставший культовой. Это игра «Hotline Miami 2: Wrong Number». Постер игры представлен на рисунке 1.3.



Рисунок 1.3 – Постер игры «Hotline Miami 2: Wrong Number»

«Hotline Miami 2: Wrong Number» — компьютерная игра в жанре 2D top-down shooter, разработанная Dennaton Games и изданная Devolver Digital в 2015 году. В сиквеле у игрока появились 13 управляемых персонажей с уникальным оружием, уровни сложности (классический и сложный), редактор уровней. Игра стала классикой жанра top-down shooter за счёт своего отличного саундтрека и сбалансированного и отзывчивого геймплея.

## Постановка задачи

В рамках данного курсового проекта планируется разработать сетевую многопользовательскую игру в жанре Top-Down Shooter. Игра будет реализована с использованием веб-технологий: клиентская часть на HTML5 и JavaScript, серверная часть на Python. Взаимодействие между клиентом и сервером будет осуществляться посредством протокола WebSocket.

В процессе разработки должны быть реализованы следующие функции:

* Меню для ввода имени игрока и подключения к серверу;
* Отображение статуса подключения к серверу;
* Возможность подключения к существующей комнате;
* Игровой интерфейс, включающий отображение здоровья (HP) и текущего счёта (очков) игрока;
* Область для вывода системных сообщений и сообщений от сервера (например, о подключении/отключении игроков, событиях в игре);
* Таблица лидеров, отображающая лучших игроков.
* Отрисовка игрового мира, включая игроков (текущего игрока и других подключенных), пули, врагов, статические препятствия на карте и подбираемые бонусы.

Для разработки курсового проекта будут использоваться языки разметки и стилей HTML5 и CSS3, языки программирования Python и JavaScript, среда разработки Visual Studio Code.

# 

# 2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

Курсовой проект «Сетевой шутер» проектируется как многопользовательская игра с клиент-серверной архитектурой, где клиентская часть реализована с использованием веб-технологий (HTML5, CSS3, JavaScript), а серверная – на языке Python. Сетевое взаимодействие между клиентов и сервером осуществляется в реальном времени посредством протокола WebSocket. Структура программы состоит из следующих основным модулей:

Модуль пользовательского интерфейса и управления состоянием клиента:

– Отвечает за отображение всех визуальных элементов: стартового меню для ввода имени, игрового интерфейса с информацией о здоровье, счете, системными сообщениями и таблицей лидеров;

– Реализован с использованием HTML для структуры, CSS для стилизации и JavaScript для динамического обновления DOM-элементов.

Модуль отрисовки игрового мира:

– Использует HTML5 Canvas API для динамической отрисовки всех игровых объектов: игроков, пуль, врагов, препятствий и бонусов;

– Отображает объекты и их состояния на основе данных, полученных от сервера;

Модуль клиентского сетевого взаимодействия:

– Устанавливает и поддерживает WebSocket-соединение с сервером;

– Отправляет на сервер команды и данные о действиях пользователя;

– Принимает сообщения от сервера и обрабатывает их, обновляя локальное представление состояния игры и элементы пользовательского интерфейса;

– Реализует логику обработки статуса подключения и попыток переподключения.

Модуль HTTP-сервера:

– Отвечает за раздачу статических файлов клиентской части (HTML, CSS, JavaScript) по протоколу HTTP;

– Обеспечивает доступ пользователей к игре через веб-браузер;

– Обрабатывает запросы и отправку соответствующих файлов или кодов ошибок.

Модуль WebSocket-сервера:

– Основан на стандартной библиотеке socket Python и реализует логику WebSocket-протокола (рукопожатие, обработка фреймов);

– Принимает и управляет подключениями от множества клиентов;

– Отправляет сообщения конкретным клиентам или всем подключенным клиентам;

– Управляет сессиями игроков (хранение информации о клиенте).

Модуль игровой логики:

– Управляет всеми игровыми объектами: их созданием, обновлением свойств (координаты, HP, состояние) и удалением;

– Взаимодействует с модулем WebSocket-сервера через callback-функцию для отправки сообщений о важных игровых событиях.

Клиентская и серверная части являются независимыми приложениями, взаимодействующими по сети.

## 2.2 Проектирование интерфейса программного средства

Интерфейс игры «Сетевой Шутер» проектируется с акцентом на простоту, интуитивную понятность и предоставление необходимой игровой информации в реальном времени. Цель — обеспечить низкий порог входа для игроков и не отвлекать от игрового процесса. Интерфейс состоит из двух основных экранов/состояний:

Меню выбора имени и подключения:

– Элементы: заголовок игры «Сетевой шутер», кнопка «Начать игру»;

– Поле для ввода никнейма игрока;

– Индикатор статуса подключения;

– Экран с темной цветовой схемой и контрастными элементами центрирован.

Игровая зона и пользовательский интерфейс:

– Игровое поле, где отрисовываются все динамические игровые объекты;

– Информация об игроке в левом верхнем угле, где находятся текущее здоровье игрока и его счёт;

– Контейнер сообщений в нижней части экрана, где отображаются системые и игровые сообщения;

– Таблица лидеров в правом верхнем угле, где указан список игроков и их текущие очки.

Дизайн UI разрабатывается с использованием HTML-разметки и CSS-стилей. Основной упор делается на функциональность и четкость представления информации.

## 2.3 Проектирование функционала программного средства

Функционал игры «Сетевой шутер» разрабатывается с целью создания динамичного многопользовательского Top-Down шутера, работающего в веб-браузере. Основные функциональные компоненты:

Клиентская часть:

– Движение игрока: игрок управляет своим персонажем с помощью клавиш. Клиент отслеживает нажатия и отправляет информацию о них на сервер. Перемещение персонажа на экране происходит на основе данных, полученных от сервера;

– Стрельба: игрок осуществляет выстрел кликом левой кнопки мыши. Координаты курсора в момент выстрела отправляются на сервер для определения направления стрельбы;

– Отображение игрового мира: рендеринг всех игровых элементов с помощью HTML5 на основе данных от сервера;

– Взаимодействие с пользовательским интерфейсом: ввод имени игрока и запуск игры, отображение актуальной информации;

– Сетевое взаимодействие со стороны клиента: установление и поддержание WebSocket-соединения, отправка действий и команд игрока на сервер в формате JSON.

Сетевая часть:

– Управление игровым процессом и сессиями: подключение и отключение игроков, синхронизация состояния, инициализация игры;

– Игровая логика: движение игроков, пулей, врагов;

– Система боя и столкновений: расчет урона и попаданий, механики здоровья и смерти;

– Сетевое взаимодействие со стороны сервера: HTTP-сервер и WebSocket-сервер.

Архитектурные особенности:

– Клиент-серверная архитектура с авторитетным сервером: вся ключевая игровая логика и проверка действий выполняется на сервере. Клиент отвечает за отображение;

– Обмен данных в формате JSON;

– Использование стандартных библиотек Python, что минимизирует внешние зависимости.

# 

# 3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 3.1 Разработка интерфейсов

Интерфейс игры «Сетевой Шутер» разработан с акцентом на простоту использования, быструю навигацию и предоставление ключевой информации игроку в реальном времени. Данный проект использует одностраничное веб-приложение с динамическим переключением между двумя основными видами интерфейса. Реализация выполнена с использованием стандартных веб-технологий: HTML для структуры, CSS для стилизации и JavaScript для управления поведением и динамическим контентом. Основные компоненты интерфейса представлены ниже.

**3.1.1** Экран выбора имени и подключения

Это начальный вид, который отображается при загрузке страницы. Представляет собой HTML-блок, стилизованный с помощью CSS.

Элементы и функциональность:

– Заголовок: "Сетевой Шутер";

– Поле для ввода имени: стандартное текстовое поле, позволяющее игроку ввести свой псевдоним;

– Кнопка "Начать Игру": по клику JavaScript-код считывает введенное имя и отправляет запрос на присоединение к игре;

– Индикатор статуса подключения: HTML-элемент, текст которого динамически изменяется для отображения текущего состояния соединения с сервером. CSS-классы используются для цветовой индикации статуса.

Дизайн: Экран имеет центрированное расположение элементов, темный фон и контрастные элементы управления для удобства восприятия.

**3.1.2** Экран "Игровая Зона" и Игровой Интерфейс

Этот вид становится активным после успешного ввода имени и получения от сервера начального состояния игры.

Элементы и функциональность:

– Игровое поле с отрисовкой всех игровых объектов;

– Информационный слой: информация об игроке (очки здоровья и счет), контейнер сообщений (игровые и системные сообщения), таблица лидеров (список игроков, отсортированных по счету).

Дизайн: HUD-элементы имеют полупрозрачный фон для лучшей читаемости на динамическом игровом фоне. Используется простой и четкий шрифт.

## 3.2 Реализация основного функционала

Основной функционал игры «Сетевой Шутер» реализован с использованием веб-технологий для клиента (HTML5, CSS, JavaScript) и Python для сервера, с обменом данными через WebSockets. Упор сделан на создание простого Top-Down шутера с многопользовательским взаимодействием.

**3.2.1** Движение и управление игроком:

– Клиент (script.js): отслеживает нажатия клавиш W, A, S, D для определения движения. При нажатии клавиш отправляет на сервер информацию о их состоянии;

– Сервер (game\_logic.py): получает от клиента информацию о нажатых клавишах и рассчитывает смещение игрока. Перед изменением координат игрока проверяет возможные столкновения с препятствиями. Ограничивает позицию игрока границами игрового поля. Обновленные координаты игрока отправляются всем клиентам.

Движение полностью обрабатывается и авторизуется сервером.

**3.2.2** Система боя:

**–** Клиент (script.js): если игрок жив, при клике левой кнопкой мыши клиент определяет координаты курсора. Отправляет на сервер сообщение, а также текущее состояние клавиш движения;

**–** Сервер (game\_logic.py): создает новый объект "пуля". Идет проверка пули на столкновение с разными типами объектов. При столкновении с врагом последний удаляется, а игроку начисляются очки, также есть шанс выпадения бонуса.

**3.2.3** Сетевое взаимодействие:

**–** Клиент (script.js): использует нативный WebSocket API браузера для подключения и обмена данными с сервером;

– Сервер (server.py): использует socket для создания HTTP-сервера и WebSocket-сервера. WebSocket-сервер обрабатывает рукопожатие, принимает и отправляет фреймы данных вручную. Сообщения между клиентом и сервером передаются в формате JSON.

## 3.3 Работа с моделями в игре

Визуальное представление игровых объектов в проекте «Сетевой Шутер» реализовано с помощью 2D-графики, отрисовываемой на элементе HTML5 Canvas. Объекты представляются простыми геометрическими формами. Такой подход соответствует простоте веб-реализации и обеспечивает хорошую производительность в браузере.

**3.3.1** Игровые персонажи (Игроки и Враги):

**–** Игроки представлены на клиенте как закрашенные прямоугольники с заданными размерами и случайным цветом. Над каждым игроком отображается его имя, а другие пользователи видят полоску здоровья, которая меняет цвет при уменьшении здоровья. После смерти цвет игрока изменяется на серый;

**–** Враги, как и игроки, рисуются как закрашенные прямоугольники. Их размеры немного меньше размеров игрока, а цвет по умолчанию задан красным.

**3.3.2** Снаряды:

**–** Пули отрисовываются как маленькие закрашенные прямоугольники. Движение пуль обеспечивается обновлением их координат на сервере и последующей перерисовкой на клиенте.

**3.3.3** Бонусы:

**–** Бонусы отрисовываются как закрашенные круги с символом или буквой внутри, обозначающей тип бонуса. Цвет круга зависит от типа бонуса. Координаты и размеры поступают от сервера.

**3.3.4** Элементы окружения:

– Препятствия представлены как статичные закрашенные прямоугольники. Их координаты и размеры определяются сервером и передаются клиенту при инициализации, после чего отрисовываются клиентом;

**–** Фон имеет сплошной цвет, который задается на клиенте.

## 3.4 Работа с сетевым взаимодействием

Сетевое взаимодействие в игре «Сетевой Шутер» построено на основе клиент-серверной архитектуры с использованием протокола WebSocket для обмена данными в реальном времени между клиентами (веб-браузерами) и центральным сервером (Python). Такой подход позволяет реализовать многопользовательскую игру, доступную через веб-браузер без необходимости установки дополнительного ПО.

Сетевое взаимодействие в игре «Сетевой Шутер» построено на основе клиент-серверной архитектуры с использованием протокола WebSocket для обмена данными в реальном времени между клиентами (веб-браузерами) и центральным сервером (Python). Такой подход позволяет реализовать многопользовательскую игру, доступную через веб-браузер без необходимости установки дополнительного ПО.

Ключевые компоненты и механизмы сетевого взаимодействия представлены ниже.

**3.4.1** Серверная сторона (Python - server.py):

**–** WebSocket-сервер: Реализован с использованием стандартной библиотеки socket. Сервер вручную обрабатывает WebSocket-рукопожатие с каждым новым клиентом. Для каждого подключенного клиента создается отдельный поток, который обрабатывает входящие и исходящие WebSocket-фреймы. Сервер самостоятельно выполняет парсинг входящих WebSocket-фреймов и формирование исходящих фреймов согласно спецификации протокола WebSocket;

**–** Управление клиентами: ведется учет активных клиентов. После успешного рукопожатия клиент отправляет сообщение join\_game для входа в игру;

**–** Обработка сообщений от клиентов: Сервер принимает сообщения от клиентов в формате JSON. Сервер регистрирует игрока и его имя в модуле игровой логики (join\_game), присваивает ему игрового персонажа и отправляет ему начальное состояние игры (initial\_state), уведомляет всех остальных игроков о подключении нового участника (player\_joined) и передает информацию о действиях игрока (player\_input). Эти данные передаются в game\_logic.py для обновления состояния игры;

**–** Отправка данных клиентам: сервер с заданной частотой отправляет всем подключенным клиентам сообщение game\_update. Это сообщение содержит актуальное состояние всех игровых объектов (игроков, пуль, врагов, бонусов) и текущие очки. Сервер может отправить сообщение конкретному клиенту;

**–** Формат данных: Все структурированные данные передаются в формате JSON.

3.4.2 Клиентская сторона (JavaScript - script.js):

– WebSocket-клиент: используется нативный WebSocket API для установления соединения с WebSocket-сервером по указанному URL;

– Отправка данных на сервер: после ввода имени и нажатия кнопки "Начать Игру" клиент отправляет JSON-сообщение на сервер. В ходе игры клиент регулярно (или по событию) отправляет JSON-сообщения для передачи действий пользователя;

– Прием и обработка данных от сервера: при получении сообщения (onmessage) клиент парсит строку JSON. В зависимости от поля type в полученном объекте, клиент выполняет соответствующие действия: initial\_state (инициализация локального состояния игры), game\_update (обновление локальных объектов и HUD), player\_joined (добавление информации о новом игроке и отображение сообщения о его присоединении), player\_left (удаление информации об ушедшем игроке), message (отображение полученного текстового сообщения);

– Управление состоянием подключения: Клиент отображает статус подключения к серверу и может предпринимать попытки переподключения в случае разрыва соединения.

# 4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В данном разделе описывается процесс тестирования игры «Сетевой Шутер», который включает в себя цели, этапы и методы, применяемые для подтверждения корректности ее функционирования. Основной задачей тестирования является выявление ошибок и подтверждение соответствия проекта заданным требованиям.

Для проверки работоспособности было проведено ручное тестистирование, ориентированное на ключевые аспекты игры.

Цели тестирования:

– Проверить корректность работы игровых механик (движение игрока, механика стрельбы, поведение врагов, система столкновений, механика очков и таблицы лидеров);

– Убедиться в стабильности сетевого взаимодействия через WebSocket;

– Оценить корректность отображения и интуитивность пользовательского интерфейса в браузере;

– Выявить и устранить возможные ошибки синхронизации состояния игры между различными клиентами и сервером.

В ходе нескольких игровых сессий была проверена основная функциональность игры. Значительных ошибок, блокирующих игровой процесс или приводящих к падению сервера, замечено не было.

Тестирование показало, что разработанное программное средство соответствует поставленным задачам курсового проекта и является работоспособной сетевой игрой в жанре Top-Down Shooter.

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Данное руководство пользователя предназначено для игроков, использующих программное средство «Сетевой Шутер». Оно описывает интерфейс игры, управление и основные шаги для начала игры, обеспечивая простое освоение. Игра запускается в веб-браузере и не требует установки на операционную систему.

## **Интерфейс программного средства**

Интерфейс игры «Сетевой Шутер» разработан с акцентом на простоту и интуитивность, предоставляя игроку всю необходимую информацию в удобном виде. Интерфейс состоит из двух основных экранов:

Экран «Выбор имени и Подключение» (рис. 5.1):

– Это первый экран, который видит пользователь при открытии игры в браузере;

– Содержит элементы: Заголовок игры «Сетевый Шутер», кнопка «Начать Игру», индикатор статуса подключения;

– Поле ввода никнейма позволяет задать псевдоним, который используется в сетевой игре.

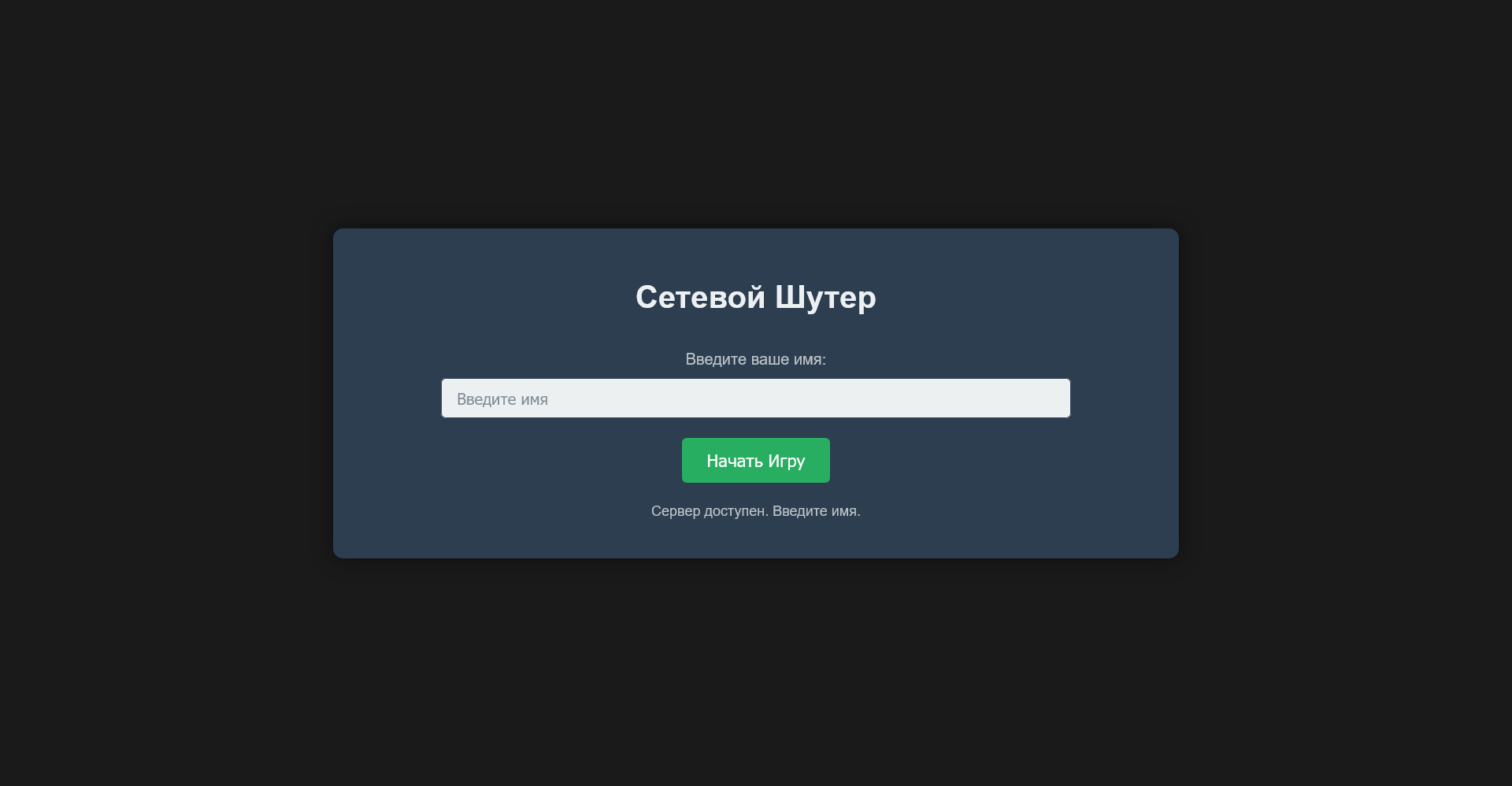


Рисунок 5.1 – Интерфейс экрана «Выбор имени и Подключение»

Экран "Игровая зона" и игровой интерфейс (рис. 5.2):

– Этот экран становится активныс после успешного подключения к игре;

– Основная часть: игровое поле, где отображаются игроки, враги, пули, бонусы и препятствия;

– Элементы игрового интерфейса: информация об игроке (здоровье и очки), сообщения (системные сообщения и уведомления об игровых событиях), таблица лидеров (список игроков, отсортированный по очкам).

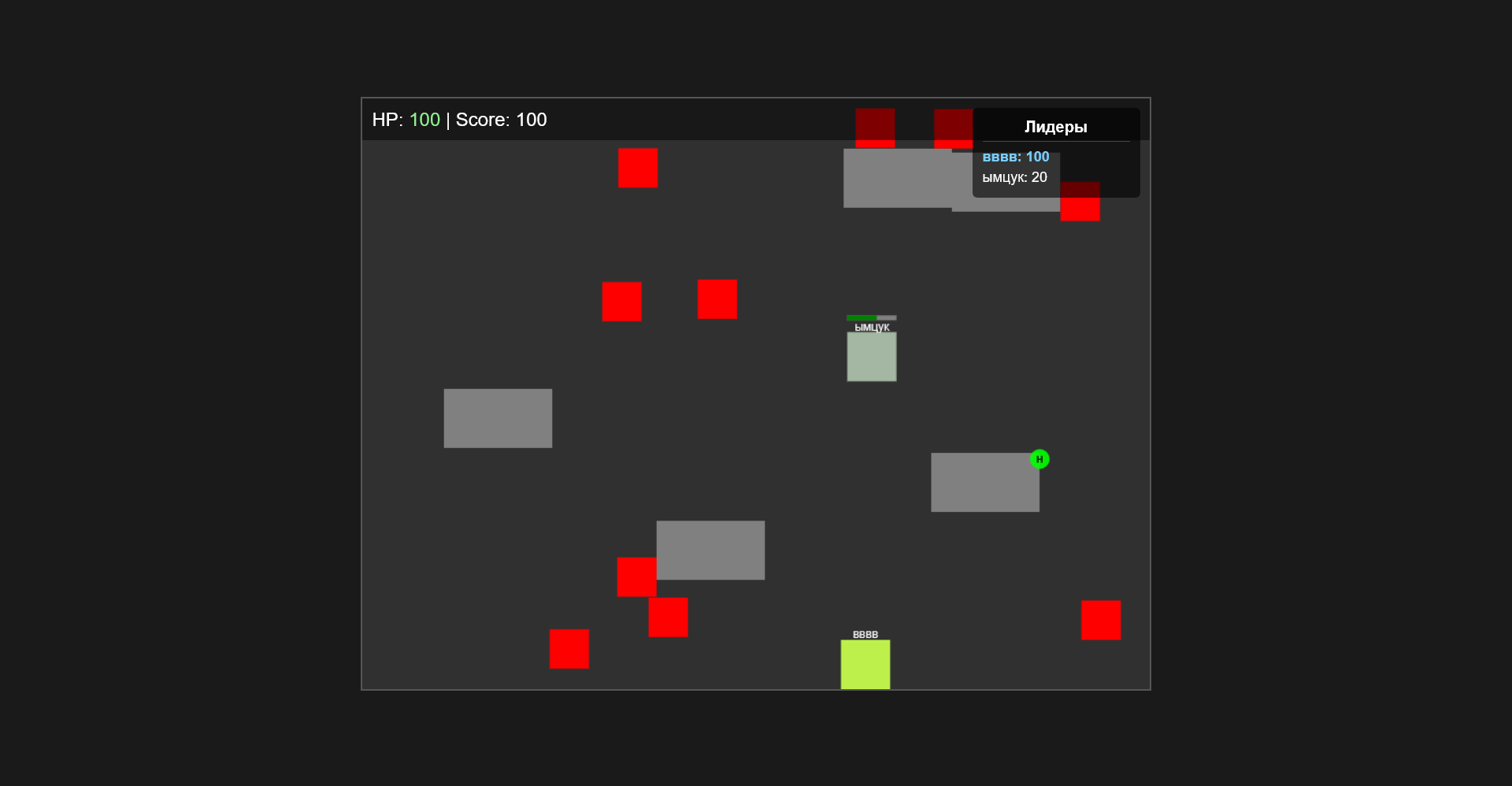


Рисунок 5.2 – Интерфейс игрового поля

## **Управление программным средством**

Требования:

– Современный веб-браузер (например, Chrome, Firefox, Edge);

– Подключение к сети, в которой доступен игровой сервер.

Запуск игры:

1. Запустите сервер (рис. 5.3): Сначала необходимо запустить серверную часть игры (server.py) на компьютере, который будет выполнять роль сервера.

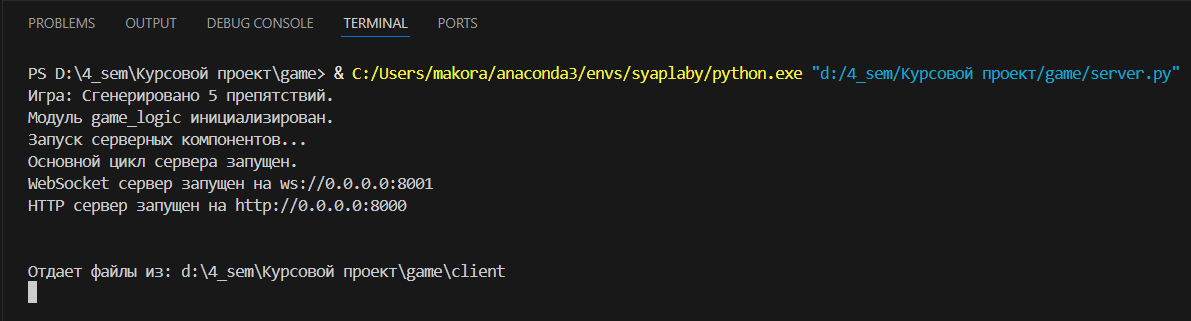


Рисунок 5.3 – Запуск server.py

2. Откройте клиент в браузере: После того как сервер запущен, откройте файл client/index.html в вашем веб-браузере. Это можно сделать, перейдя по адресу, который предоставляет HTTP-сервер (в нашем случае, это <http://0.0.0.0:8000>).

Начало игры:

1. После загрузки страницы вы увидите Экран "Выбор имени и Подключение".

2. Введите желаемый никнейм в текстовое поле.

3. Нажмите кнопку "Начать Игру".

4. Если все пройдет успешно, вы перейдете на Экран "Игровая Зона".

Управление в игре:

– Движение персонажа: используйте клавиши W, A, S, D для перемещения персонажа;

– Стрельба: нажмите левую кнопку мыши для выстрела в направлении курсора мыши.

Навигация по интерфейсу (меню выбора имени):

– Мышь: Для клика по полю ввода имени и кнопке "Начать Игру";

– Клавиатура: Для ввода текста в поле имени.

Для выхода из игры просто закройте вкладку или окно браузера, в котором запущена игра.

# ЗАКЛЮЧЕНИЕ

В рамках курсового проекта была успешно разработана сетевая компьютерная игра в жанре Top-Down Shooter под названием «Сетевой Шутер». Реализация клиентской части выполнена с использованием веб-технологий (HTML5, CSS, JavaScript), а серверная логика и сетевое взаимодействие построены на языке программирования Python с применением протокола WebSocket. Основная цель проекта — создание динамичной и доступной браузерной игры с простым и понятным дизайном, соответствующей современным требованиям к геймплею и интерфейсу для веб-приложений, — была успешно достигнута.

В ходе работы был проведен анализ предметной области, включая обзор аналогов в жанре Top-Down Shooter, что позволило выделить ключевые механики.

Была разработана клиент-серверная архитектура программы. Клиентская часть включает модули пользовательского интерфейса, отрисовки игрового мира на HTML5 Canvas, клиентского сетевого взаимодействия и обработки пользовательского ввода. Серверная часть состоит из HTTP-сервера для раздачи статических файлов, WebSocket-сервера для двусторонней связи и модуля игровой логики, отвечающего за все игровые механики и состояние мира.

Реализованы основные функции, указанные в постановке задачи: выбор псевдонима игрока, подключение к общему игровому серверу, динамичный игровой процесс с механикой стрельбы, движением персонажей, взаимодействием с врагами, статичными препятствиями и подбираемыми бонусами. Также функционирует система начисления очков и отображение таблицы лидеров. Сетевая синхронизация, реализованная на основе протокола WebSocket и авторитетного сервера на Python, обеспечила обмен игровыми данными и обновление состояния игры для всех подключенных клиентов в режиме, близком к реальному времени.

Проведенное ручное тестирование подтвердило работоспособность основного функционала, базовую стабильность многопользовательского режима при участии нескольких клиентов и удобство пользовательского интерфейса в браузере.

Руководство пользователя предоставляет четкие инструкции по запуску серверной части и использованию клиентской части игры через веб-браузер.

Данный курсовой проект подчеркивает значимость доступного дизайна и современных веб-технологий в создании сетевых игр.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Документация по HTML [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/HTML.  
 [2] Документация по CSS [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/CSS

[3] Руководство по JavaScript [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide

[4] Современный учебник JavaScript [Электронный ресурс]. – Режим доступа: https://learn.javascript.ru/

[5] Флэнаган, Дэвид. JavaScript. Полное руководство, 7-е изд.: Пер. с англ. — СПб.: ООО “Диалектика”, 2021. — 720 с.: ил.

[6] Мэтиз Эрик. Изучаем Python: программирование игр, визуализация данных, веб-приложения. 3-е изд. — СПб.: Питер, 2020. — 512 с.: ил.

[7] Нистрем, Роберт. Паттерны программирования игр / Роберт Нистрем; [перевод с английского М. А. Райтмана]. — Москва: Эксмо, 2021. — 432 с.

[8] Роджерс, С. Level Up! Руководство по созданию отличных видеоигр / С. Роджерс. – М.: Манн, Иванов и Фербер, 2020. – 528 с.

[9] Гэддис Т. Начинаем программировать на Python. — 5-е изд.: Пер. с англ. — СПб.: БХВ-Петербург, 2022. — 880 с.: ил.

[10] Роджерс, С. Level Up! Руководство по созданию отличных видеоигр / С. Роджерс. – М.: Манн, Иванов и Фербер, 2020. – 528 с.

[11] The WebSocket Protocol. RFC 6455 [Электронный ресурс]. – Режим доступа: https://tools.ietf.org/html/rfc6455.

# ПРИЛОЖЕНИЕ А. Исходный код программы

**client/index.html**

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<title>Сетевой Шутер</title>

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

</head>

<body>

<div id="name-selection-menu" class="active-view">

<h1>Сетевой Шутер</h1>

<p>Введите ваше имя:</p>

<input type="text" id="playerNameInput" placeholder="Введите имя" maxlength="15">

<button id="startGameButton">Начать Игру</button>

<p id="connection-status" class="status-connecting">Подключение к серверу...</p>

</div>

<div id="game-area">

<div id="game-container">

<canvas id="gameCanvas"></canvas>

<div id="ui-overlay">

<div id="player-info">HP: <span id="playerHp">---</span> | Score: <span id="playerScore">---</span></div>

<div id="messages-container">

<div id="messages"></div>

</div>

<div id="leaderboard">

<h3>Лидеры</h3>

<ul id="leaderboard-list"></ul>

</div>

</div>

</div>

</div>

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

</body>

</html>

**client/script.js**

const nameSelectionMenu = document.getElementById('name-selection-menu');

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

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

const connectionStatusEl = document.getElementById('connection-status');

const gameArea = document.getElementById('game-area');

const canvas = document.getElementById('gameCanvas');

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

const playerHpEl = document.getElementById('playerHp');

const playerScoreEl = document.getElementById('playerScore');

const messagesEl = document.getElementById('messages');

const leaderboardListEl = document.getElementById('leaderboard-list');

let myPlayerId = null;

let players = {};

let bullets = {};

let enemies = {};

let obstacles = [];

let bonuses = {};

let scores = {};

let gameSettings = { width: 800, height: 600, playerSize: 50, bulletSize: 5, enemySize: 40 };

const WS\_PORT = 8001;

const wsUrl = `ws://${window.location.hostname}:${WS\_PORT}`;

let socket = null;

let connectionAttempts = 0;

const MAX\_CONNECTION\_ATTEMPTS = 5;

let gameStarted = false;

// --- Управление UI ---

function showMenu() {

    nameSelectionMenu.classList.add('active-view');

    gameArea.classList.remove('active-view');

    gameStarted = false;

}

function showGameArea() {

    nameSelectionMenu.classList.remove('active-view');

    gameArea.classList.add('active-view');

    gameStarted = true;

    if (gameSettings && gameSettings.width && gameSettings.height) {

        canvas.width = gameSettings.width;

        canvas.height = gameSettings.height;

    }

}

function updateConnectionStatus(message, statusClass) {

    connectionStatusEl.textContent = message;

    connectionStatusEl.className = '';

    connectionStatusEl.classList.add(statusClass);

}

// --- WebSocket соединение ---

function connectWebSocket() {

    if (socket && (socket.readyState === WebSocket.OPEN || socket.readyState === WebSocket.CONNECTING)) {

        console.log("WebSocket уже подключается или подключен.");

        return;

    }

    if (connectionAttempts >= MAX\_CONNECTION\_ATTEMPTS && !gameStarted) {

        updateConnectionStatus(`Не удалось подключиться. Попробуйте перезагрузить страницу.`, 'status-error');

        startGameButton.disabled = true;

        return;

    }

    connectionAttempts++;

    updateConnectionStatus(`Подключение... (попытка ${connectionAttempts})`, 'status-connecting');

    startGameButton.disabled = true;

    socket = new WebSocket(wsUrl);

    socket.onopen = () => {

        console.log("WebSocket подключен!");

        connectionAttempts = 0;

        updateConnectionStatus('Сервер доступен. Введите имя.', 'status-connected');

        startGameButton.disabled = false;

    };

    socket.onmessage = (event) => {

        try {

            const message = JSON.parse(event.data);

            if (!gameStarted && message.type !== 'initial\_state') {

            }

            switch (message.type) {

                case 'initial\_state':

                    handleInitialState(message.data);

                    if (!gameStarted) {

                        showGameArea();

                    }

                    break;

                case 'game\_update':

                    if(gameStarted) handleGameUpdate(message.data);

                    break;

                case 'player\_joined':

                    if(gameStarted) handlePlayerJoined(message.data);

                    break;

                case 'player\_left':

                    if(gameStarted) handlePlayerLeft(message.data);

                    break;

                case 'message':

                    displayMessage(message.data.text, message.data.duration || 3000, message.data.msg\_type || 'info');

                    break;

                default:

                    console.warn("Неизвестный тип сообщения от сервера:", message.type, message.data);

            }

        } catch (error) {

            console.error("Ошибка парсинга JSON или обработки сообщения:", error, event.data);

        }

    };

    socket.onclose = (event) => {

        console.log(`WebSocket соединение закрыто. Код: ${event.code}, причина: ${event.reason}`);

        myPlayerId = null;

        players = {}; bullets = {}; enemies = {}; bonuses = {}; scores = {};

        if (gameStarted) {

            displayMessage(`Соединение с сервером потеряно. Код: ${event.code}.`, 0, 'error');

        } else {

             updateConnectionStatus(`Соединение закрыто. Код: ${event.code}.`, 'status-error');

             if (!event.wasClean && connectionAttempts < MAX\_CONNECTION\_ATTEMPTS) {

                 setTimeout(connectWebSocket, 3000);

             } else if (!event.wasClean) {

                 updateConnectionStatus('Не удалось переподключиться.', 'status-error');

             }

        }

        startGameButton.disabled = true;

    };

    socket.onerror = (error) => {

        console.error("WebSocket ошибка:", error);

        if (!gameStarted) {

            updateConnectionStatus('Ошибка соединения с WebSocket.', 'status-error');

            startGameButton.disabled = true;

        }

    };

}

// --- Обработчики данных от сервера ---

function handleInitialState(data) {

    myPlayerId = data.playerId;

    players = data.players || {};

    bullets = data.bullets || {};

    enemies = data.enemies || {};

    obstacles = data.obstacles || [];

    bonuses = data.bonuses || {};

    scores = data.scores || {};

    if (data.gameSettings) {

        gameSettings = data.gameSettings;

        if (gameStarted && canvas.width !== gameSettings.width) canvas.width = gameSettings.width;

        if (gameStarted && canvas.height !== gameSettings.height) canvas.height = gameSettings.height;

    }

    console.log("Начальное состояние получено, мой ID:", myPlayerId);

    updateUI();

}

function handleGameUpdate(data) {

    if (data.players !== undefined) players = data.players;

    if (data.bullets !== undefined) bullets = data.bullets;

    if (data.enemies !== undefined) enemies = data.enemies;

    if (data.bonuses !== undefined) bonuses = data.bonuses;

    if (data.scores !== undefined) scores = data.scores;

    updateUI();

}

function handlePlayerJoined(playerData) {

    if (playerData && playerData.id) {

        players[playerData.id] = playerData;

        if (!scores[playerData.id]) scores[playerData.id] = 0;

        displayMessage(`${playerData.name || playerData.id} присоединился!`, 3000, 'info');

        updateUI();

    }

}

function handlePlayerLeft(playerId) {

    const PName = players[playerId]?.name || playerId;

    if (players[playerId]) delete players[playerId];

    if (scores[playerId]) delete scores[playerId];

    displayMessage(`${PName} покинул игру.`, 3000, 'info');

    updateUI();

}

// --- Отправка ввода на сервер ---

const keysPressed = {};

window.addEventListener('keydown', (e) => {

    if (!gameStarted) return;

    keysPressed[e.key.toLowerCase()] = true;

});

window.addEventListener('keyup', (e) => {

    if (!gameStarted) return;

    keysPressed[e.key.toLowerCase()] = false;

});

canvas.addEventListener('mousedown', (e) => {

    if (!gameStarted || !socket || socket.readyState !== WebSocket.OPEN) return;

    if (e.button === 0 ) {

        if (myPlayerId && players[myPlayerId] && players[myPlayerId].hp > 0 && !players[myPlayerId].is\_dead) {

            const rect = canvas.getBoundingClientRect();

            const mouseX = e.clientX - rect.left;

            const mouseY = e.clientY - rect.top;

            socket.send(JSON.stringify({

                type: 'player\_input',

                data: { shoot: true, target: { x: mouseX, y: mouseY }, keys: keysPressed }

            }));

        }

    }

});

setInterval(() => {

    if (!gameStarted || !socket || socket.readyState !== WebSocket.OPEN) return;

    if (myPlayerId && players[myPlayerId] && players[myPlayerId].hp > 0 && !players[myPlayerId].is\_dead) {

        if (Object.values(keysPressed).some(pressed => pressed)) {

            socket.send(JSON.stringify({ type: 'player\_input', data: { keys: keysPressed } }));

        }

    }

}, 1000 / 20);

const COLORS = { };

function displayMessage(text, duration = 3000, type = 'info') {

    const messageItem = document.createElement('div');

    messageItem.classList.add('message-item', type);

    messageItem.textContent = text;

    messagesEl.appendChild(messageItem);

    messagesEl.scrollTop = messagesEl.scrollHeight;

    if (duration > 0) {

        setTimeout(() => {

            if (messagesEl.contains(messageItem)) {

                 messagesEl.removeChild(messageItem);

            }

        }, duration);

    }

}

function updateUI() {

    if (!gameStarted) return;

    if (myPlayerId && players[myPlayerId]) {

        const myPlayerData = players[myPlayerId];

        playerHpEl.textContent = players[myPlayerId].hp;

        playerScoreEl.textContent = scores[myPlayerId] || 0;

        if (myPlayerData.is\_dead || (myPlayerData.hp !== undefined && myPlayerData.hp <= 0)) {

            playerHpEl.style.color = "#808080";

            playerHpEl.textContent = "ПОГИБ";

        } else if (myPlayerData.hp !== undefined) {

            playerHpEl.style.color = myPlayerData.hp > 50 ? "lightgreen" :

                                     myPlayerData.hp > 20 ? "orange" : "red";

        } else {

        playerHpEl.textContent = "---";

        playerScoreEl.textContent = "---";

        }

    }

    leaderboardListEl.innerHTML = '';

    const sortedScores = Object.entries(scores || {}).sort(([,a],[,b]) => b-a).slice(0,10);

    for (const [pid, scoreVal] of sortedScores) {

        const playerName = players[pid]?.name || `Игрок ${pid.substring(0,4)}`;

        const listItem = document.createElement('li');

        listItem.textContent = `${playerName}: ${scoreVal}`;

        if (pid === myPlayerId) { listItem.style.fontWeight = 'bold'; listItem.style.color = '#77ccff';}

        leaderboardListEl.appendChild(listItem);

    }

}

function drawGame() {

    requestAnimationFrame(drawGame);

    if (!gameStarted || !ctx) return;

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

    ctx.fillStyle = "#303030";

    ctx.fillRect(0,0, canvas.width, canvas.height);

    // 1. Препятствия

    ctx.fillStyle = COLORS.obstacle || "#808080";

    (obstacles || []).forEach(obs => { ctx.fillRect(obs.x, obs.y, obs.width, obs.height); });

    // 2. Игроки

    for (const id in players) {

        const p = players[id];

        if (!p) continue;

        if (p.is\_dead || p.hp <= 0) {

            ctx.fillStyle = p.color || "#808080";

        } else {

            ctx.fillStyle = p.color || COLORS.player\_default || "#00FF00";

        }

        ctx.fillRect(p.x, p.y, p.width, p.height);

        ctx.fillStyle = "white"; ctx.font = "12px Arial"; ctx.textAlign = "center";

        ctx.fillText(p.name || `P-${id.substring(0,4)}`, p.x + p.width/2, p.y - 5);

        if (id !== myPlayerId && p.hp !== undefined) {

            if (!p.is\_dead) {

                const hpBarWidth = p.width; const hpBarHeight = 5;

                const hpFillWidth = Math.max(0, (p.hp / 100)) \* hpBarWidth;

                ctx.fillStyle = "grey"; ctx.fillRect(p.x, p.y - hpBarHeight - 12, hpBarWidth, hpBarHeight);

                ctx.fillStyle = p.hp > 50 ? "green" : (p.hp > 20 ? "orange" : "red");

                ctx.fillRect(p.x, p.y - hpBarHeight - 12, hpFillWidth, hpBarHeight);

            }

        }

    }

    // 3. Пули

    ctx.fillStyle = COLORS.bullet || "#FFFFFF";

    for (const id in bullets) {

        const b = bullets[id];

        if (!b) continue;

        ctx.fillRect(b.x, b.y, b.width, b.height);

    }

    // 4. Враги

    ctx.fillStyle = COLORS.enemy || "#FF0000";

    for (const id in enemies) {

        const en = enemies[id];

        if (!en) continue;

        ctx.fillRect(en.x, en.y, en.width, en.height);

    }

    // 5. Бонусы

    for (const id in bonuses) {

        const bonus = bonuses[id];

        if (!bonus) continue;

        if (bonus.type === "health") ctx.fillStyle = COLORS.bonus\_health || "rgba(0, 255, 0, 0.9)";

        else if (bonus.type === "score\_boost") ctx.fillStyle = COLORS.bonus\_score\_boost || "rgba(255, 255, 0, 0.9)";

        else ctx.fillStyle = "purple";

        ctx.beginPath();

        ctx.arc(bonus.x + bonus.width/2, bonus.y + bonus.height/2, bonus.width/2, 0, Math.PI \* 2);

        ctx.fill();

        ctx.fillStyle = "black"; ctx.font = "bold 10px Arial"; ctx.textAlign = "center"; ctx.textBaseline = "middle";

        let letter = bonus.type.substring(0,1).toUpperCase();

        if (bonus.type === "score\_boost") letter = "$";

        ctx.fillText(letter, bonus.x + bonus.width/2, bonus.y + bonus.height/2);

    }

}

// --- Инициализация ---

document.addEventListener('DOMContentLoaded', () => {

    nameSelectionMenu.classList.add('active-view');

    gameArea.classList.remove('active-view');

    gameStarted = false;

    connectWebSocket();

    startGameButton.addEventListener('click', () => {

        const playerName = playerNameInput.value.trim() || `Player${Math.floor(Math.random()\*1000)}`;

        if (socket && socket.readyState === WebSocket.OPEN) {

            socket.send(JSON.stringify({

                type: 'join\_game',

                data: { name: playerName }

            }));

            updateConnectionStatus('Ожидание ответа от сервера...', 'status-connecting');

            startGameButton.disabled = true;

        } else {

            updateConnectionStatus('Нет соединения с сервером. Попробуйте позже.', 'status-error');

        }

    });

    drawGame();

});

**client/style.css**

body {

    margin: 0;

    background-color: #1a1a1a;

    font-family: 'Arial', sans-serif;

    color: #f0f0f0;

    overflow: hidden;

    display: flex;

    justify-content: center;

    align-items: center;

    min-height: 100vh;

}

#name-selection-menu,

#game-area {

    display: none;

    flex-direction: column;

    align-items: center;

    justify-content: center;

    width: 100%;

    height: 100%;

    min-width: 100vh;

}

.active-view {

    display: flex !important;

}

#name-selection-menu {

    background-color: #2c3e50;

    padding: 30px;

    border-radius: 10px;

    box-shadow: 0 0 20px rgba(0,0,0,0.5);

    text-align: center;

    max-width: 400px;

}

#name-selection-menu h1 {

    color: #ecf0f1;

    margin-bottom: 20px;

}

#name-selection-menu p {

    color: #bdc3c7;

    margin-bottom: 10px;

}

#playerNameInput {

    padding: 10px 15px;

    margin-bottom: 20px;

    border-radius: 5px;

    border: 1px solid #34495e;

    background-color: #ecf0f1;

    color: #2c3e50;

    font-size: 1em;

    width: 80%;

    box-sizing: border-box;

}

#startGameButton {

    padding: 12px 25px;

    background-color: #27ae60;

    color: white;

    border: none;

    border-radius: 5px;

    font-size: 1.1em;

    cursor: pointer;

    transition: background-color 0.3s ease;

}

#startGameButton:hover {

    background-color: #2ecc71;

}

#startGameButton:disabled {

    background-color: #7f8c8d;

    cursor: not-allowed;

}

#connection-status {

    margin-top: 20px;

    font-size: 0.9em;

}

.status-connecting { color: #f39c12; }

.status-connected { color: #2ecc71; }

.status-error { color: #e74c3c; }

#game-container {

    position: relative;

    border: 2px solid #555;

}

#gameCanvas {

    display: block;

    background-color: #303030;

}

#ui-overlay {

    position: absolute;

    top: 0;

    left: 0;

    width: 100%;

    height: 100%;

    pointer-events: none;

    display: flex;

    flex-direction: column;

    justify-content: space-between;

}

#player-info {

    padding: 10px;

    background-color: rgba(0,0,0,0.5);

    font-size: 1.2em;

    color: #fff;

}

#messages-container {

    position: absolute;

    bottom: 10px;

    left: 50%;

    transform: translateX(-50%);

    width: 80%;

    max-height: 100px;

    overflow-y: auto;

    pointer-events: all;

}

#messages .message-item {

    background-color: rgba(0,0,0,0.7);

    padding: 5px 10px;

    border-radius: 5px;

    margin-bottom: 5px;

    display: block;

    text-align: center;

    color: #ffcc00;

}

#messages .message-item.error { color: #e74c3c; }

#messages .message-item.warning { color: #f39c12; }

#messages .message-item.success { color: #2ecc71; }

#messages .message-item.info { color: #3498db; }

#leaderboard {

    position: absolute;

    top: 10px;

    right: 10px;

    background-color: rgba(0,0,0,0.6);

    padding: 10px;

    border-radius: 5px;

    min-width: 150px;

    color: #fff;

    pointer-events: all;

}

#leaderboard h3 { margin-top: 0; font-size: 1em; text-align: center; border-bottom: 1px solid #444; padding-bottom: 5px; margin-bottom: 5px;}

#leaderboard ul { list-style: none; padding: 0; margin: 0; }

#leaderboard li { font-size: 0.9em; padding: 2px 0; }

**server.py**

import socket

import threading

import time

import hashlib

import base64

import struct

import json

import os

import game\_logic

# --- Конфигурация сервера ---

HTTP\_HOST = '0.0.0.0'

HTTP\_PORT = 8000

WEBSOCKET\_HOST = '0.0.0.0'

WEBSOCKET\_PORT = 8001

WEB\_DIR = os.path.join(os.path.dirname(\_\_file\_\_), 'client')

SERVER\_TICK\_RATE = 1 / 60

# --- WebSocket константы ---

WEBSOCKET\_GUID = "258EAFA5-E914-47DA-95CA-C5AB0DC85B11"

OPCODE\_TEXT = 0x1

OPCODE\_CLOSE = 0x8

OPCODE\_PING = 0x9

OPCODE\_PONG = 0xA

# --- Управление WebSocket клиентами ---

ws\_clients\_lock = threading.Lock()

ws\_clients = {}

ws\_client\_id\_counter = 0

# =========================================================

# HTTP Сервер

# =========================================================

def handle\_http\_request(client\_socket):

    try:

        request\_data = client\_socket.recv(1024).decode('utf-8', errors='ignore')

        if not request\_data:

            client\_socket.close()

            return

        request\_lines = request\_data.split('\r\n')

        if not request\_lines:

            client\_socket.close()

            return

        method, path, \_ = request\_lines[0].split(' ')

        if method == 'GET':

            if path == '/':

                path = '/index.html'

            requested\_path = os.path.normpath(os.path.join(WEB\_DIR, path.lstrip('/')))

            if not requested\_path.startswith(os.path.abspath(WEB\_DIR)):

                response = b"HTTP/1.1 403 Forbidden\r\nContent-Type: text/plain\r\n\r\nForbidden"

                client\_socket.sendall(response)

                return

            file\_path = requested\_path

            if os.path.exists(file\_path) and os.path.isfile(file\_path):

                content\_type = 'text/plain'

                if file\_path.endswith('.html'): content\_type = 'text/html; charset=utf-8'

                elif file\_path.endswith('.css'): content\_type = 'text/css; charset=utf-8'

                elif file\_path.endswith('.js'): content\_type = 'application/javascript; charset=utf-8'

                elif file\_path.endswith('.ico'): content\_type = 'image/x-icon'

                with open(file\_path, 'rb') as f:

                    response\_body = f.read()

                response\_headers = (

                    f"HTTP/1.1 200 OK\r\n"

                    f"Content-Type: {content\_type}\r\n"

                    f"Content-Length: {len(response\_body)}\r\n"

                    f"Connection: close\r\n\r\n"

                )

                client\_socket.sendall(response\_headers.encode('utf-8') + response\_body)

            else:

                response = b"HTTP/1.1 404 Not Found\r\nContent-Type: text/plain\r\n\r\nFile Not Found"

                client\_socket.sendall(response)

        else:

            response = b"HTTP/1.1 405 Method Not Allowed\r\nContent-Type: text/plain\r\n\r\nMethod Not Allowed"

            client\_socket.sendall(response)

    except Exception as e:

        print(f"HTTP Request Error: {e}")

    finally:

        client\_socket.close()

def run\_http\_server():

    server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

    try:

        server\_socket.bind((HTTP\_HOST, HTTP\_PORT))

        server\_socket.listen(5)

        print(f"HTTP сервер запущен на http://{HTTP\_HOST}:{HTTP\_PORT}\n")

        print(f"Отдает файлы из: {os.path.abspath(WEB\_DIR)}")

        while True:

            client\_socket, addr = server\_socket.accept()

            http\_thread = threading.Thread(target=handle\_http\_request, args=(client\_socket,))

            http\_thread.daemon = True

            http\_thread.start()

    except OSError as e:

        print(f"ОШИБКА HTTP СЕРВЕРА: Не удалось запустить сервер на {HTTP\_HOST}:{HTTP\_PORT}. {e}")

    except KeyboardInterrupt:

        print("HTTP сервер останавливается...")

    finally:

        server\_socket.close()

# =========================================================

# WebSocket Сервер: Рукопожатие и Фрейминг

# =========================================================

def parse\_http\_headers(data\_str):

    headers = {}

    lines = data\_str.split("\r\n")

    for line in lines:

        if ":" in line:

            key, value = line.split(":", 1)

            headers[key.strip().lower()] = value.strip()

    return headers

def generate\_websocket\_accept\_key(client\_key):

    return base64.b64encode(hashlib.sha1((client\_key + WEBSOCKET\_GUID).encode()).digest()).decode()

def \_send\_ws\_frame\_to\_conn(conn, payload, opcode=OPCODE\_TEXT):

    if isinstance(payload, str):

        payload = payload.encode('utf-8')

    payload\_len = len(payload)

    frame\_header = bytearray()

    frame\_header.append(0x80 | opcode)

    mask\_bit = 0x00

    if payload\_len <= 125:

        frame\_header.append(mask\_bit | payload\_len)

    elif payload\_len <= 65535:

        frame\_header.append(mask\_bit | 126)

        frame\_header.extend(struct.pack("!H", payload\_len))

    else:

        frame\_header.append(mask\_bit | 127)

        frame\_header.extend(struct.pack("!Q", payload\_len))

    try:

        conn.sendall(bytes(frame\_header) + payload)

        return True

    except (socket.error, BrokenPipeError):

        return False

def \_receive\_ws\_frame\_from\_conn(conn):

    try:

        header\_byte1 = conn.recv(1)

        if not header\_byte1: return None, None

        opcode = header\_byte1[0] & 0b00001111

        header\_byte2 = conn.recv(1)

        if not header\_byte2: return None, None

        mask\_bit = (header\_byte2[0] & 0b10000000) >> 7

        payload\_len\_indicator = header\_byte2[0] & 0b01111111

        payload\_len = 0

        if payload\_len\_indicator <= 125:

            payload\_len = payload\_len\_indicator

        elif payload\_len\_indicator == 126:

            len\_bytes = conn.recv(2)

            if not len\_bytes or len(len\_bytes) < 2: return None, None

            payload\_len = struct.unpack("!H", len\_bytes)[0]

        elif payload\_len\_indicator == 127:

            len\_bytes = conn.recv(8)

            if not len\_bytes or len(len\_bytes) < 8: return None, None

            payload\_len = struct.unpack("!Q", len\_bytes)[0]

        masking\_key = None

        if mask\_bit:

            masking\_key = conn.recv(4)

            if not masking\_key or len(masking\_key) < 4: return None, None

        payload\_data\_bytes = b""

        remaining\_payload\_size = payload\_len

        while remaining\_payload\_size > 0:

            chunk\_size = min(remaining\_payload\_size, 4096)

            chunk = conn.recv(chunk\_size)

            if not chunk: return None, None

            payload\_data\_bytes += chunk

            remaining\_payload\_size -= len(chunk)

        if len(payload\_data\_bytes) != payload\_len: return None, None

        if mask\_bit and masking\_key:

            unmasked\_payload = bytearray(payload\_len)

            for i in range(payload\_len):

                unmasked\_payload[i] = payload\_data\_bytes[i] ^ masking\_key[i % 4]

            payload\_data\_bytes = bytes(unmasked\_payload)

        return opcode, payload\_data\_bytes

    except socket.timeout: return "timeout", None

    except (socket.error, struct.error, IndexError, BrokenPipeError, ConnectionResetError): return None, None

# =========================================================

# WebSocket Сервер: Управление клиентами и сообщениями

# =========================================================

def send\_to\_one\_client\_by\_conn(conn, payload\_obj):

    """Отправляет JSON объект одному клиенту по его сокету."""

    try:

        json\_str = json.dumps(payload\_obj)

        \_send\_ws\_frame\_to\_conn(conn, json\_str)

    except Exception as e:

        print(f"WS Core: Ошибка отправки клиенту: {e}")

def broadcast\_to\_all\_ws\_clients(payload\_obj, exclude\_conn=None):

    """Отправляет JSON объект всем подключенным WebSocket клиентам."""

    if not payload\_obj: return

    try:

        json\_str = json.dumps(payload\_obj)

    except TypeError as e:

        print(f"WS Core: Ошибка сериализации JSON при broadcast: {e}, Payload: {payload\_obj}")

        return

    with ws\_clients\_lock:

        current\_client\_conns = list(ws\_clients.keys())

    for client\_conn in current\_client\_conns:

        if client\_conn != exclude\_conn:

            if not \_send\_ws\_frame\_to\_conn(client\_conn, json\_str):

                pass

def handle\_websocket\_client\_connection(conn, addr):

    global ws\_client\_id\_counter

    client\_session\_data = None

    try:

        request\_data = conn.recv(2048).decode('utf-8', errors='ignore')

        if not request\_data: return

        headers = parse\_http\_headers(request\_data)

        if 'sec-websocket-key' not in headers or \

           headers.get('upgrade', '').lower() != 'websocket' or \

           not headers.get('connection', '').lower().count('upgrade'):

            conn.sendall(b"HTTP/1.1 400 Bad Request\r\n\r\n"); return

        accept\_key = generate\_websocket\_accept\_key(headers['sec-websocket-key'])

        response\_handshake = (

            "HTTP/1.1 101 Switching Protocols\r\nUpgrade: websocket\r\nConnection: Upgrade\r\n"

            f"Sec-WebSocket-Accept: {accept\_key}\r\n\r\n"

        )

        conn.sendall(response\_handshake.encode('utf-8'))

        with ws\_clients\_lock:

            session\_id = f"session\_{ws\_client\_id\_counter}"

            ws\_client\_id\_counter += 1

            client\_session\_data = {'id': session\_id, 'addr': addr, 'status': 'connected', 'name': None, 'game\_id': None}

            ws\_clients[conn] = client\_session\_data

        print(f"WS Core: Сессия {client\_session\_data['id']} ({addr}) подключена, ожидает входа в игру.")

        conn.settimeout(1.0)

        while True:

            opcode, payload\_bytes = \_receive\_ws\_frame\_from\_conn(conn)

            if opcode is None and payload\_bytes is None: break

            if opcode == "timeout": continue

            if opcode == OPCODE\_TEXT:

                try:

                    message\_str = payload\_bytes.decode('utf-8')

                    message = json.loads(message\_str)

                    msg\_type = message.get('type')

                    msg\_data = message.get('data', {})

                    current\_client\_status = client\_session\_data.get('status')

                    if msg\_type == 'join\_game' and current\_client\_status == 'connected':

                        player\_name\_from\_client = msg\_data.get('name', f"Player\_{client\_session\_data['id'][-4:]}")

                        client\_session\_data['name'] = player\_name\_from\_client

                        client\_session\_data['status'] = 'ingame'

                        client\_session\_data['game\_id'] = client\_session\_data['id']

                        print(f"WS Core: Клиент {client\_session\_data['game\_id']} (был {client\_session\_data['id']}) входит в игру как '{player\_name\_from\_client}'.")

                        initial\_state\_data, new\_player\_data = game\_logic.handle\_player\_connect(

                            client\_session\_data['game\_id'],

                            player\_name\_from\_client

                        )

                        if initial\_state\_data:

                            send\_to\_one\_client\_by\_conn(conn, {'type': 'initial\_state', 'data': initial\_state\_data})

                        if new\_player\_data:

                            broadcast\_to\_all\_ws\_clients({'type': 'player\_joined', 'data': new\_player\_data}, exclude\_conn=conn)

                    elif msg\_type == 'player\_input' and current\_client\_status == 'ingame':

                        if client\_session\_data.get('game\_id'):

                            game\_logic.handle\_player\_input(client\_session\_data['game\_id'], msg\_data)

                except (json.JSONDecodeError, UnicodeDecodeError) as e:

                    print(f"WS Core: Некорректные данные от {client\_session\_data['id']}: {e}")

                except Exception as e\_inner:

                     print(f"WS Core: Ошибка обработки сообщения от {client\_session\_data['id']}: {e\_inner}")

            elif opcode == OPCODE\_CLOSE:

                print(f"WS Core: Клиент {client\_session\_data['id']} запросил закрытие.")

                break

            elif opcode == OPCODE\_PING:

                \_send\_ws\_frame\_to\_conn(conn, payload\_bytes, opcode=OPCODE\_PONG)

    except socket.error: pass

    except Exception as e\_outer:

        session\_id\_for\_log = client\_session\_data.get('id', addr) if client\_session\_data else addr

        print(f"WS Core: Общая ошибка с клиентом {session\_id\_for\_log}: {e\_outer}")

    finally:

        if client\_session\_data:

            print(f"WS Core: Сессия {client\_session\_data['id']} отключается.")

            if client\_session\_data.get('status') == 'ingame' and client\_session\_data.get('game\_id'):

                game\_id\_on\_disconnect = client\_session\_data['game\_id']

                \_, disconnected\_player\_name = game\_logic.handle\_player\_disconnect(game\_id\_on\_disconnect)

                if game\_id\_on\_disconnect:

                     broadcast\_to\_all\_ws\_clients({'type': 'player\_left', 'data': game\_id\_on\_disconnect})

                     broadcast\_to\_all\_ws\_clients({'type': 'message', 'data': {'text': f'{disconnected\_player\_name or game\_id\_on\_disconnect} покинул игру.', 'msg\_type': 'info'}})

        else:

            print(f"WS Core: Клиент с {addr} отключается (рукопожатие не завершено или сессия не создана).")

        with ws\_clients\_lock:

            if conn in ws\_clients:

                del ws\_clients[conn]

        conn.close()

def run\_websocket\_server():

    server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

    try:

        server\_socket.bind((WEBSOCKET\_HOST, WEBSOCKET\_PORT))

        server\_socket.listen(5)

        print(f"WebSocket сервер запущен на ws://{WEBSOCKET\_HOST}:{WEBSOCKET\_PORT}\n")

        while True:

            conn, addr = server\_socket.accept()

            ws\_client\_thread = threading.Thread(target=handle\_websocket\_client\_connection, args=(conn, addr))

            ws\_client\_thread.daemon = True

            ws\_client\_thread.start()

    except OSError as e:

        print(f"ОШИБКА WEBSOCKET СЕРВЕРА: Не удалось запустить сервер на {WEBSOCKET\_HOST}:{WEBSOCKET\_PORT}. {e}")

    except KeyboardInterrupt:

        print("WebSocket сервер останавливается...")

    finally:

        server\_socket.close()

# =========================================================

# Основной цикл сервера (интеграция с game\_logic)

# =========================================================

def server\_main\_loop():

    print("Основной цикл сервера запущен.")

    last\_tick\_time = time.perf\_counter()

    game\_logic.set\_broadcast\_callback(broadcast\_to\_all\_ws\_clients)

    while True:

        current\_time = time.perf\_counter()

        delta\_time\_sec = current\_time - last\_tick\_time

        if delta\_time\_sec < SERVER\_TICK\_RATE:

            sleep\_duration = SERVER\_TICK\_RATE - delta\_time\_sec

            time.sleep(sleep\_duration)

            delta\_time\_sec = time.perf\_counter() - last\_tick\_time

        last\_tick\_time = time.perf\_counter()

        game\_snapshot\_data = game\_logic.update\_game\_state(delta\_time\_sec)

        if game\_snapshot\_data:

            payload\_to\_send = {'type': 'game\_update', 'data': game\_snapshot\_data}

            broadcast\_to\_all\_ws\_clients(payload\_to\_send)

# ========================================================

# Запуск Сервера

# =========================================================

if \_\_name\_\_ == "\_\_main\_\_":

    print("Запуск серверных компонентов...")

    http\_server\_thread = threading.Thread(target=run\_http\_server, name="HTTPServerThread", daemon=True)

    http\_server\_thread.start()

    websocket\_server\_thread = threading.Thread(target=run\_websocket\_server, name="WebSocketServerThread", daemon=True)

    websocket\_server\_thread.start()

    try:

        server\_main\_loop()

    except KeyboardInterrupt:

        print("Сервер останавливается по KeyboardInterrupt (в основном потоке)...")

    except Exception as e\_main:

        print(f"Критическая ошибка в server\_main\_loop: {e\_main}")

        import traceback

        traceback.print\_exc()

    finally:

        print("Завершение работы сервера (основной поток)...")

**game\_logic.py**

import random

import math

import uuid

import threading

# --- Игровые константы ---

WIDTH, HEIGHT = 800, 600

PLAYER\_SIZE = 50

BULLET\_SIZE = 5

ENEMY\_SIZE = 40

OBSTACLE\_WIDTH, OBSTACLE\_HEIGHT = 110, 60

DIFFICULTY = {"enemy\_speed": (2.2, 3.0), "spawn\_rate": 1500, "max\_enemies": 20}

GAME\_TICK\_RATE = 1 / 60

# --- Глобальное состояние игры ---

game\_state\_lock = threading.Lock()

game\_players = {}

game\_bullets = {}

game\_enemies = {}

game\_obstacles = []

game\_bonuses = {}

game\_scores = {}

# Таймеры спавна (управляются из game\_loop)

ENEMY\_SPAWN\_TIMER\_MS = 0

BONUS\_SPAWN\_TIMER\_MS = 0

BONUS\_SPAWN\_RATE\_MS = 10000

MAX\_BONUSES = 3

# --- Вспомогательные игровые функции ---

def check\_rect\_collision(rect1, rect2):

    return (rect1['x'] < rect2['x'] + rect2['width'] and

            rect1['x'] + rect1['width'] > rect2['x'] and

            rect1['y'] < rect2['y'] + rect2['height'] and

            rect1['y'] + rect1['height'] > rect2['y'])

def get\_random\_color(): return f"#{random.randint(0, 0xFFFFFF):06x}"

def generate\_initial\_obstacles():

    global game\_obstacles

    with game\_state\_lock:

        game\_obstacles = []

        for \_ in range(5):

            while True:

                x = random.randint(0, WIDTH - OBSTACLE\_WIDTH)

                y = random.randint(0, HEIGHT - OBSTACLE\_HEIGHT)

                new\_obs\_rect = {'x': x, 'y': y, 'width': OBSTACLE\_WIDTH, 'height': OBSTACLE\_HEIGHT}

                player\_spawn\_area = {'x': WIDTH//2 - 100, 'y': HEIGHT//2 - 100, 'width': 200, 'height': 200}

                if check\_rect\_collision(new\_obs\_rect, player\_spawn\_area): continue

                if any(check\_rect\_collision(new\_obs\_rect, obs) for obs in game\_obstacles): continue

                game\_obstacles.append(new\_obs\_rect); break

    print(f"Игра: Сгенерировано {len(game\_obstacles)} препятствий.")

# --- Функции обратного вызова для сервера ---

broadcast\_callback\_func = None

def reset\_simple\_game\_over\_state():

    global game\_players, game\_bullets, game\_enemies, game\_bonuses, game\_scores, is\_game\_over

    global ENEMY\_SPAWN\_TIMER\_MS, BONUS\_SPAWN\_TIMER\_MS

    with game\_state\_lock:

        game\_players.clear()

        game\_bullets.clear()

        game\_enemies.clear()

        game\_bonuses.clear()

        game\_scores.clear()

        is\_game\_over = False

        ENEMY\_SPAWN\_TIMER\_MS = 0

        BONUS\_SPAWN\_TIMER\_MS = 0

    print("Игра: Состояние Game Over сброшено (основные игровые объекты очищены).")

def set\_broadcast\_callback(callback\_func):

    global broadcast\_callback\_func

    broadcast\_callback\_func = callback\_func

def \_broadcast\_message(payload\_obj):

    if broadcast\_callback\_func:

        broadcast\_callback\_func(payload\_obj)

    else:

        print("ПРЕДУПРЕЖДЕНИЕ (Игра): broadcast\_callback не установлен!")

# --- Функции для управления состоянием игры, вызываемые из server\_core ---

def handle\_player\_connect(client\_id, player\_name):

    with game\_state\_lock:

        game\_players[client\_id] = {

            'id': client\_id, 'name': player\_name,

            'x': WIDTH // 2 - PLAYER\_SIZE // 2, 'y': HEIGHT // 2 - PLAYER\_SIZE // 2,

            'width': PLAYER\_SIZE, 'height': PLAYER\_SIZE,

            'hp': 100,

            'color': get\_random\_color(),

            'is\_dead': False

        }

        game\_scores[client\_id] = 0

    initial\_data\_for\_new\_player = {

        'playerId': client\_id, 'players': game\_players, 'bullets': game\_bullets,

        'enemies': game\_enemies, 'obstacles': game\_obstacles, 'bonuses': game\_bonuses,

        'scores': game\_scores,

        'gameSettings': { 'width': WIDTH, 'height': HEIGHT, 'playerSize': PLAYER\_SIZE, 'bulletSize': BULLET\_SIZE, 'enemySize': ENEMY\_SIZE}

    }

    new\_player\_join\_data = game\_players[client\_id].copy()

    return initial\_data\_for\_new\_player, new\_player\_join\_data

def handle\_player\_disconnect(client\_id):

    with game\_state\_lock:

        player\_name = game\_players.get(client\_id, {}).get('name', client\_id)

        if client\_id in game\_players: del game\_players[client\_id]

        if client\_id in game\_scores: del game\_scores[client\_id]

    return client\_id, player\_name

def handle\_player\_input(client\_id, input\_data):

    with game\_state\_lock:

        if client\_id not in game\_players or game\_players[client\_id]['hp'] <= 0: return None

        player = game\_players[client\_id]

        if player.get('is\_dead', False) or player['hp'] <= 0 :

            return None

        speed = 5

        keys = input\_data.get('keys', {})

        dx, dy = 0, 0

        if keys.get('a') or keys.get('ф'): dx -= speed

        if keys.get('d') or keys.get('в'): dx += speed

        if keys.get('w') or keys.get('ц'): dy -= speed

        if keys.get('s') or keys.get('ы'): dy += speed

        if dx != 0 and dy != 0:

            norm = math.sqrt(dx\*dx + dy\*dy); dx = (dx / norm) \* speed; dy = (dy / norm) \* speed

        next\_x\_rect = {'x': player['x'] + dx, 'y': player['y'], 'width': player['width'], 'height': player['height']}

        if not any(check\_rect\_collision(next\_x\_rect, obs) for obs in game\_obstacles):

            player['x'] += dx

        next\_y\_rect = {'x': player['x'], 'y': player['y'] + dy, 'width': player['width'], 'height': player['height']}

        if not any(check\_rect\_collision(next\_y\_rect, obs) for obs in game\_obstacles):

            player['y'] += dy

        player['x'] = max(0, min(player['x'], WIDTH - player['width']))

        player['y'] = max(0, min(player['y'], HEIGHT - player['height']))

        if input\_data.get('shoot') and input\_data.get('target'):

            target = input\_data['target']

            bullet\_id = str(uuid.uuid4())

            start\_x = player['x'] + player['width'] / 2; start\_y = player['y'] + player['height'] / 2

            angle\_dx = target['x'] - start\_x; angle\_dy = target['y'] - start\_y

            dist = math.hypot(angle\_dx, angle\_dy)

            vel\_x, vel\_y = (0, -10) if dist == 0 else ((angle\_dx/dist)\*10, (angle\_dy/dist)\*10)

            game\_bullets[bullet\_id] = {

                'id': bullet\_id, 'owner\_sid': client\_id,

                'x': start\_x - BULLET\_SIZE/2, 'y': start\_y - BULLET\_SIZE/2,

                'width': BULLET\_SIZE, 'height': BULLET\_SIZE, 'vx': vel\_x, 'vy': vel\_y

            }

    return None

def \_spawn\_bonus\_at\_location(x, y):

    bonus\_id = str(uuid.uuid4())

    bonus\_type = random.choice(["health", "score\_boost"])

    game\_bonuses[bonus\_id] = {

        'id': bonus\_id, 'x': x - 10, 'y': y - 10, 'width': 20, 'height': 20, 'type': bonus\_type

    }

def \_apply\_bonus\_effect\_to\_player(player\_id, bonus\_type):

    if player\_id not in game\_players: return

    player = game\_players[player\_id]

    if player.get('is\_dead', False) or player['hp'] <= 0:

        return

    message\_text = ""

    if bonus\_type == "health":

        player['hp'] = min(player['hp'] + 30, 100)

        message\_text = f"{player.get('name', player\_id)} подобрал аптечку!"

    elif bonus\_type == "score\_boost":

        if player\_id in game\_scores: game\_scores[player\_id] += 50

        message\_text = f"{player.get('name', player\_id)} получил бонусные очки!"

    if message\_text:

        \_broadcast\_message({'type': 'message', 'data': {'text': message\_text, 'msg\_type': 'success'}})

def update\_game\_state(delta\_time\_sec):

    global ENEMY\_SPAWN\_TIMER\_MS, BONUS\_SPAWN\_TIMER\_MS

    dt\_ms = delta\_time\_sec \* 1000

    events\_for\_broadcast = []

    with game\_state\_lock:

        if not game\_players:

            ENEMY\_SPAWN\_TIMER\_MS = 0; BONUS\_SPAWN\_TIMER\_MS = 0

            game\_enemies.clear(); game\_bonuses.clear(); game\_bullets.clear()

        # 1. Обновление пуль

        bullets\_to\_remove = []

        for bid, bullet in list(game\_bullets.items()):

            bullet['x'] += bullet['vx']

            bullet['y'] += bullet['vy']

            if not (0 < bullet['x'] < WIDTH and 0 < bullet['y'] < HEIGHT) or \

               any(check\_rect\_collision(bullet, obs) for obs in game\_obstacles):

                bullets\_to\_remove.append(bid)

        for bid in bullets\_to\_remove:

            if bid in game\_bullets: del game\_bullets[bid]

        # 2. Спавн врагов

        ENEMY\_SPAWN\_TIMER\_MS += dt\_ms

        if ENEMY\_SPAWN\_TIMER\_MS >= DIFFICULTY["spawn\_rate"] and len(game\_enemies) < DIFFICULTY["max\_enemies"]:

            ENEMY\_SPAWN\_TIMER\_MS = 0; enemy\_id = str(uuid.uuid4())

            side = random.choice(['top', 'bottom', 'left', 'right'])

            ex, ey = (0,0)

            if side == 'top': ex, ey = random.randint(0, WIDTH-ENEMY\_SIZE), -ENEMY\_SIZE

            elif side == 'bottom': ex, ey = random.randint(0, WIDTH-ENEMY\_SIZE), HEIGHT

            elif side == 'left': ex, ey = -ENEMY\_SIZE, random.randint(0, HEIGHT-ENEMY\_SIZE)

            else: ex, ey = WIDTH, random.randint(0, HEIGHT-ENEMY\_SIZE)

            game\_enemies[enemy\_id] = {

                'id': enemy\_id, 'x': ex, 'y': ey, 'width': ENEMY\_SIZE, 'height': ENEMY\_SIZE,

                'speed': random.uniform(\*DIFFICULTY["enemy\_speed"]), 'hp': 30

            }

        # 3. Движение врагов и коллизии

        enemies\_to\_remove = []

        for eid, enemy in list(game\_enemies.items()):

            target\_player = None; min\_dist = float('inf')

            for pid\_loop, p\_data\_loop in game\_players.items():

                if not p\_data\_loop.get('is\_dead', False) and p\_data\_loop['hp'] > 0:

                    dist = math.hypot(p\_data\_loop['x'] - enemy['x'], p\_data\_loop['y'] - enemy['y'])

                    if dist < min\_dist: min\_dist = dist; target\_player = p\_data\_loop

            if target\_player:

                edx = target\_player['x'] + PLAYER\_SIZE/2 - (enemy['x'] + ENEMY\_SIZE/2)

                edy = target\_player['y'] + PLAYER\_SIZE/2 - (enemy['y'] + ENEMY\_SIZE/2)

                dist\_to\_target = math.hypot(edx, edy)

                if dist\_to\_target > 0:

                    move\_dist = enemy['speed'] \* delta\_time\_sec \* 20

                    e\_vx = (edx / dist\_to\_target) \* move\_dist

                    e\_vy = (edy / dist\_to\_target) \* move\_dist

                    next\_enemy\_rect = {'x': enemy['x'] + e\_vx, 'y': enemy['y'] + e\_vy, 'width': ENEMY\_SIZE, 'height': ENEMY\_SIZE}

                    if not any(check\_rect\_collision(next\_enemy\_rect, obs) for obs in game\_obstacles):

                        enemy['x'] += e\_vx; enemy['y'] += e\_vy

            for pid, player\_data in list(game\_players.items()):

                if player\_data.get('is\_dead', False) or player\_data['hp'] <= 0: continue

                if check\_rect\_collision(enemy, player\_data):

                    player\_data['hp'] = max(0, player\_data['hp'] - 20)

                    if player\_data['hp'] <= 0:

                        player\_data['is\_dead'] = True

                        player\_data['color'] = "#808080"

                        print(f"Игра: Игрок {pid} ({player\_data.get('name', pid)}) погиб.")

                        events\_for\_broadcast.append({'type': 'message', 'data': {'text': f"{player\_data.get('name', pid)} был повержен!", 'msg\_type': 'warning'}})

                    enemies\_to\_remove.append(eid); break

            if eid in enemies\_to\_remove: continue

            for bid, bullet in list(game\_bullets.items()):

                if check\_rect\_collision(enemy, bullet):

                    if bullet['owner\_sid'] in game\_scores: game\_scores[bullet['owner\_sid']] += 10

                    enemies\_to\_remove.append(eid)

                    if bid in game\_bullets: del game\_bullets[bid]

                    if random.random() < 0.20: \_spawn\_bonus\_at\_location(enemy['x'], enemy['y'])

                    break

        for eid in set(enemies\_to\_remove):

            if eid in game\_enemies: del game\_enemies[eid]

        # 5. Коллизия игрока с бонусом

        bonuses\_to\_remove = []

        for bonus\_id, bonus\_data in list(game\_bonuses.items()):

            for pid, player\_data in list(game\_players.items()):

                if player\_data.get('is\_dead', False) or player\_data['hp'] <=0: continue

                player\_rect = {'x':player\_data['x'],'y':player\_data['y'],'width':player\_data['width'],'height':player\_data['height']}

                if check\_rect\_collision(player\_rect, bonus\_data):

                    \_apply\_bonus\_effect\_to\_player(pid, bonus\_data['type'])

                    bonuses\_to\_remove.append(bonus\_id); break

            if bonus\_id in bonuses\_to\_remove: continue

        for b\_id in bonuses\_to\_remove:

            if b\_id in game\_bonuses: del game\_bonuses[b\_id]

        # 6. Формирование текущего состояния для отправки

        current\_snapshot = {

            'players': game\_players, 'bullets': game\_bullets, 'enemies': game\_enemies,

            'bonuses': game\_bonuses, 'scores': game\_scores

        }

    for event\_payload in events\_for\_broadcast:

        \_broadcast\_message(event\_payload)

    return current\_snapshot

generate\_initial\_obstacles()

print("Модуль game\_logic инициализирован.")