Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**КОМПЬЮТЕРНАЯ ОНЛАЙН-ИГРА «БОИ СТИКМЕНОВ»**

БГУИР КП 6-05-0612-01 018 ПЗ

Студент Карпеченко М.В.

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 8](#_Toc199159582)

[2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 10](#_Toc199159583)

[2.1 Структура программы 10](#_Toc199159584)

[2.2 Проектирование интерфейса программного средства 11](#_Toc199159585)

[2.3 Проектирование функционала программного средства 12](#_Toc199159586)

[3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 13](#_Toc199159587)

[3.1 Разработка интерфейсов 13](#_Toc199159588)

[3.2 Реализация основного функционала 14](#_Toc199159589)

[3.3 Работа с моделями в игре 15](#_Toc199159590)

[3.4 Работа с сетевым взаимодействием 16](#_Toc199159591)

[4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 17](#_Toc199159592)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 18](#_Toc199159593)

[5.1 Интерфейс программного средства 18](#_Toc199159594)

[5.2 Управление программным средством 23](#_Toc199159595)

[ЗАКЛЮЧЕНИЕ 24](#_Toc199159596)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 25](#_Toc199159597)

[ПРИЛОЖЕНИЕ А. Исходный код программы 26](#_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 модель «клиент-сервер» и позволившая игрокам сражаться в виртуальном лабиринте. Эти разработки стали отправной точкой для стремительного роста жанров и механик, подпитываемого совершенствованием компьютерных систем, интернета и графических технологий.

На волне этой эволюции возник жанр «stickman», который выделяется минималистичным дизайном персонажей и акцентом на увлекательные механики. Благодаря простоте, доступности и универсальности, игры вроде «Stickman Supreme Duelist» и «Henry Stickmin» завоевали любовь миллионов игроков. Развитие браузерных технологий, таких как Flash в 2000-х и HTML5 в 2010-х, а также распространение мобильных устройств сделали такие игры массовыми, позволяя запускать их даже на устройствах с низкой производительностью. Жанр «stickman» стал не только игровым феноменом, но и частью поп-культуры: мемы, анимации и мерчендайз с персонажами-стикменами получили широкое распространение в интернете и за его пределами.

Современная игровая индустрия предлагает бесчисленное множество решений, охватывающих любые вкусы и предпочтения. Онлайн-игры стали важной частью культуры, формируя активные сообщества, вдохновляющие на создание фанатского контента, организацию тематических мероприятий и даже развитие киберспорта. Стриминговые платформы, такие как Twitch и YouTube Gaming, усилили популярность казуальных игр, позволяя игрокам делиться своими достижениями и привлекать новую аудиторию.

Кроссплатформенность, обеспечивающая доступ к играм на ПК, смартфонах и консолях, стала ключевым фактором успеха, делая проекты вроде «Бои Стикменов» универсальными и конкурентоспособными.

Целью данной курсовой работы является разработка концепции игры «Бои Стикменов», анализ существующих аналогов и постановка задачи для создания прототипа, который будет отвечать современным требованиям к геймплею, интерфейсу и технической реализации.

# **АНАЛИЗ** ПРЕДМЕТНОЙ ОБЛАСТИ

## Обзор аналогов

Разумно, что стоит рассмотреть, самые актуальные и популярные игры, которые послужат ориентиром для разработки и выполнении задач.

Первой из них станет игра «Stick Fight: The Game».



Рисунок 1.1 – Постер игры «Stick Fight: The Game»

«Stick Fight: The Game» — это динамичная многопользовательская игра в жанре 2D-файтинг, разработанная студией Landfall и выпущенная 28 сентября 2017 года на Steam, а позже, 4 апреля 2021 года, на Nintendo Switch. Игра основана на физике и ориентирована на сражения между культовыми стикменами — минималистичными персонажами, которые стали символом интернет-культуры «золотой эры» 2000-х.

Она рассчитана на 2–4 игрока и поддерживает как локальный, так и онлайн-мультиплеер, но не имеет одиночного режима. Благодаря хаотичному геймплею, простоте управления и юмористическому подходу, игра получила высокие оценки (93% положительных отзывов из 97,848 на Steam) и стала популярной среди игроков и стримеров.

Она является ярким примером аналога для темы курсовой работы и может служить как источник идей и их реализации.

Далее стоит рассмотреть игру «Gang Beasts», которая тоже имеет минималистичный стиль, но отличный от жанра «stickman».  
Игра представлена на рисунке 1.2.



Рисунок 1.2 – Постер игры «Gang Beasts»

«Gang Beasts» (2014, Boneloaf) — многопользовательский beat'em'up с упором на хаотичные сражения и физику ragdoll. Игроки управляют желеобразными персонажами, сражаясь на интерактивных аренах (например, на крыше грузовика или в лифте). Цель — сбросить противников с платформы или вывести их из строя ударами. Игра ориентирована на казуальный мультиплеер и использует физику для зрелищности. Подходит для анализа упрощенных механик и вовлечения аудитории через юмор.

Одной из интересных игр про файтинг и мультиплеер является «Brawlhalla».

Постер игры представлен на рисунке 1.3.



Рисунок 1.3 – Постер игры «Brawlhalla»

«Brawlhalla» (2017, Blue Mammoth Games) — бесплатный 2D-файтинг с акцентом на кроссплатформенный мультиплеер. Игроки выбирают персонажей с уникальными наборами движений и сражаются на небольших аренах, используя оружие, которое появляется на карте (мечи, копья, молоты). Поддерживает до 8 игроков в различных режимах (1v1, 2v2, FFA)

Стоит рассмотреть игру «Super Smash Bros. Ultimate» для анализа возможностей и механик. Постер представлен на рисунке 1.4.



Рисунок 1.4 – Постер «Super Smash Bros. Ultimate»

Super Smash Bros. Ultimate (2018, Bandai Namco/Sora Ltd.) — культовый файтинг, где игроки выбирают персонажей из популярных игровых франшиз (Mario, Zelda, Pokémon и др.) и сражаются на динамичных аренах. Цель — сбить противников за пределы карты, накапливая урон.

## Постановка задачи

В рамках данной курсовой работы планируется разработать игру на движке Unity для операционной системы Windows. В процессе разработки должны быть реализованы следующие функции:

* выбор никнейма (псевдонима в игре);
* создание комнаты (лобби) с параметрами;
* возможность подключения к существующей комнате;
* выбор карты для игры;
* выбор оружия для использования в игре;

Необходимо реализовать основные возможности в игре:

* игровой процесс (раунды или дезматчи);
* вывод результатов в конце игры;
* определение победителей;
* понятная навигация;

Также, при разработке нужно учитывать следующее:

– разработка концепта UI и предварительный анализ и разработка макетов меню

* поиск/создание моделей и префабов;
* поиск/создание материалов и текстур/спрайтов;
* очистка всех закладок;
* просмотр всех закладок;

Планируется реализовать функционал, связанный с разнообразием видов оружия:

* алгоритмы и функции оружия ближнего;
* алгоритмы и функции оружия дальнего боя;
* метательное оружие;

В перспективе

* звуковое сопровождение в игре
* эффекты, частицы
* широкий функционал настроек для удобства игрока и гибкости;

Для разработки курсового проекта будут использоваться язык программирования С#, среда разработки Visual Studio 2022, игровой движок Unity версии 6001.1.0f1, а также Photon PUN 2 как основа для сетевого взаимодействия.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

Курсовой проект «Бои Стикменов» проектируется как многопользовательская игра с клиент-серверной архитектурой, использующей игровой движок Unity и библиотеку Photon PUN 2 для реализации сетевого взаимодействия. Структура программы состоит из следующих основных модулей:

– Модуль управления игрой:

– Отвечает за инициализацию игрового процесса, управление сценами (меню, лобби, игровой уровень) и координацию между клиентом и сервером.

– Реализует состояния игры: главное меню, выбор комнаты, игровой процесс, результаты матча.

– Использует скрипты на C# для обработки событий и переходов между сценами.

Модуль сетевого взаимодействия:

– Основан на Photon PUN 2, обеспечивающем синхронизацию данных между игроками в реальном времени.

– Включает функции создания и подключения к комнатам, синхронизации позиций персонажей, состояний оружия и событий (например, получение урона).

– Реализует обработку сетевых событий, таких как подключение/отключение игроков.

Модуль игровой логики:

– Управляет игровыми механиками: движение персонажей, анимации, физика, взаимодействие с оружием (ближнего и дальнего боя и т.д.).

– Включает алгоритмы расчета урона, определения победителя и завершения раундов.

Модуль пользовательского интерфейса (UI):

– Отвечает за отображение меню, настроек, результатов игры и HUD (интерфейс во время матча).

– Включает элементы управления для выбора никнейма, карты, оружия и параметров комнаты.

– Реализован с использованием Unity UI Canvas.

Модуль ресурсов:

– Хранит и управляет игровыми ассетами: спрайты персонажей, текстуры оружия, карты, звуковые эффекты и частицы.

– Использует префабы для оптимизации загрузки и повторного использования объектов.

Модули взаимодействуют через событийно-ориентированную архитектуру, где некоторые данные передаются между клиентом и хостом через Photon PUN 2, а игровая логика синхронизируется с UI и ресурсами через компоненты Unity.

## 2.2 Проектирование интерфейса программного средства

Интерфейс игры «Бои Стикменов» проектируется с учетом принципов минимализма, и интуитивности, чтобы обеспечить удобство для игроков. Интерфейс включает следующие основные компоненты:

Главное меню:

– Элементы: кнопки «Host», «Connect», «Settings», «Exit».

– Поле для ввода никнейма игрока.

– Фон с минималистичным дизайном, вдохновленным стилем стикменов (например, силуэты персонажей в боевых позах).

Лобби (комната):

– Интерфейс для создания комнаты: поля для ввода названия комнаты, выбора карты, количества игроков (2–4).

– Возможность заходить в комнаты по названию.

– Кнопки готовности для синхронизации игроков.

Игровой интерфейс (HUD):

– Отображение здоровья игрока, текущего оружия и количества боеприпасов.

– Счетчик игроков и таймер раунда (для дезматчей).

– Мини-карта (опционально) для отображения позиций игроков на уровне.

Экран результатов:

– Таблица с результатами (никнеймы, количество убийств, смертей, побед).

– Кнопки для рематча или возврата в главное меню (или же таймер с автоматическим переходом)

Дизайн UI разрабатывается с использованием Unity UI Canvas и адаптивных макетов, чтобы поддерживать любое разрешение экрана. Цветовая палитра минималистична (черный, белый). Для создания макетов используются инструменты Figma и Adobe Photoshop, позволяющие прототипировать интерфейс перед реализацией. Создание кнопки «Exit» представлено на рисунке 2.1:

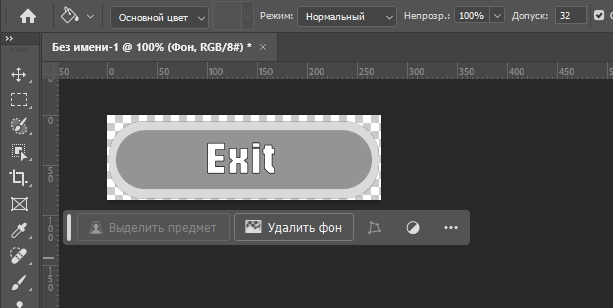


Рисунок 2.1 – Создание кнопки «Exit»

## 2.3 Проектирование функционала программного средства

Функционал игры «Бои Стикменов» разрабатывается с учетом требований к динамичному геймплею, сетевому взаимодействию и разнообразию механик. Основные функциональные компоненты:

Игровой процесс:

– Движение персонажей: реализуется через компонент CharacterController или RigidBody в Unity для физически достоверного перемещения и ragdoll-эффектов при падении или получении урона.

– Система боя: включает два типа оружия:

– Ближний бой: мечи, дубинки (алгоритмы расчета урона при столкновении хитбоксов).

– Дальний бой: пистолеты, винтовки (трассировка лучей для выстрелов, синхронизация через Photon).

Сетевое взаимодействие:

– Реализовано через Photon PUN 2, обеспечивающее низкую задержку и синхронизацию данных.

– Основные функции:

– Создание и подключение к комнатам (PhotonNetwork.CreateRoom, PhotonNetwork.JoinRoom).

– Синхронизация позиций, анимаций и событий (PhotonView для передачи данных).

– Обработка сетевых ошибок (например, отключение игрока).

– Серверная логика минимальна, так как Photon Cloud берет на себя основную нагрузку.

Управление ресурсами:

– Используются префабы для персонажей, оружия и элементов окружения.

– Спрайты и текстуры создаются в минималистичном стиле (например, с использованием Photoshop).

– Звуковое сопровождение включает эффекты для ударов, выстрелов и фоновую музыку (импортируется в формате .mp3).

Возможные дополнительные функции:

– Настройки игры: регулировка громкости, чувствительности управления, выбор языка интерфейса.

– Частицы и эффекты (например, искры при ударе, взрывы гранат) для повышения зрелищности.

Реализация функционала базируется на объектно-ориентированном подходе с использованием C# в Unity

# РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 3.1 Разработка интерфейсов

Интерфейс игры «Бои Стикменов» разработан с акцентом на интуитивность и минимализм, вдохновленный успешным аналогом «Stick Fight: The Game». Использование Unity UI Canvas позволило создать адаптивный и визуально чистый интерфейс, оптимизированный для Windows и потенциально кроссплатформенных решений. Основные элементы интерфейса:

1. Главное меню:

Реализовано как отдельная сцена с Canvas, содержащим кнопки «Host», «Connect», «Settings» и «Exit». Поле ввода никнейма сохраняет данные в PhotonNetwork.NickName для сетевой идентификации. Дизайн включает статичкный фон с изображениями стикменов.

Кнопка «Host» – Открывает меню «HostMenu», в котором присутствуют:

– Поля для ввода имени комнаты, к которой будут подключаться игроки, и максимального количества игроков в комнате;

– Кнопка «Создать комнату» – перенаправляет на сцену «Lobby»;

– Кнопка «Вернуться в главное меню» – скрывает «HostMenu» и показывает главное меню;

Кнопка «Connect» – открывает меню «ConnectMenu»:

– Поля для ввода имени комнаты, к которой хотят подключиться игроки;

– Кнопка «Подключиться» – перенаправляет на сцену «Lobby», в которой уже создана комната;

– Кнопка «Вернуться в главное меню» – скрывает «ConnectMenu» и показывает главное меню;

Кнопка «Settings» – открывает меню «SettingsMenu» для гибкой настройки пользователем игры на свой вкус;

Кнопка «Exit»: закрывает игру.

1. Лобби:

Также реализовано через отдельную сцену. Имеет в себе:

– панель с именами всех подключенных игроков;

– кнопки выхода в главное меню и кнопка «Start» для начала игры, которая пересылает на «WeaponSelectionScene»;

– панель с выбором карты для последующей игры.

1. «WeaponSelectionScene» или же «Выбор оружия» (подраздел лобби):

Слегка повторяет сцену «Lobby», за исключением отсутствия списка игроков (в нем нет необходимости) и вместо выбора карты игроки выбирают себе оружие на бой.  
 Стоит отметить, что игру начинать может только хост, после общего выбора карты. В то же время, оружия игроки выбирают каждый сам себе, а хост ждёт, пока все выберут. Только затем он может запустить игру.

1. Игровой интерфейс (HUD):

Стандартный, слегка упрощенный, игровой интерфейс, содержащий в себе:

– шкалу здоровья (Slider);

– счетчик боеприпасов (TMP\_Text);

– элемент очков здоровья (TMP\_Text);

– cтатистику (ScrollView) и таймер раунда (TMP\_Text);

– предусматривается «KillingFeed», т. е. отчёт об последних (например, 5) убийствах игроков

1. Экран результатов:

Представляется в сцене с игровым процессом по истечению таймера.

Содержит в себе:

– таблицу результатов (GridLayoutGroup), которая показывает статистику игроков: никнеймы, убийства, смерти, очки;

– таймер (15-30 секунд) для автоматического перехода в лобби, реализован через Coroutine с визуальным отсчетом.

## 3.2 Реализация основного функционала

Основной функционал игры «Бои Стикменов» реализован с учетом динамики и хаотичности, характерных для жанра файтингов, таких как «Stick Fight: The Game». Реализация выполнена на C# в Unity, с использованием объектно-ориентированного подхода и оптимизацией для слабых систем, что соответствует требованиям жанра Stickman.

1. Движение персонажей:

– Реализовано через CharacterController и BoxCollider с фиксированным углом вращения камеры для стабильного 3D - движения.

– Управление: W-A-S-D – движение вперед-влево-назад-вправо, SPACE – прыжок, SHIFT – ускорение.

– При смерти игрок телепортируется в одну из точек респавна. Имеет в себе компонент Target и Health, чтобы регистрировать урон.

1. Система боя:

**–** Ближний бой: Мечи и молоты используют BoxCollider для обнаружения попаданий. Урон рассчитывается из введеного параметра на префабе модели.

– Дальний бой: Пистолеты и винтовки используют Physics3D.Raycast для расчета траектории выстрелов. Снаряды (пули) — префабы с Rigidbody3D, синхронизируемые через Photon.

1. Режимы игры:

– Дезматч: подсчет очков пока не кончится время (очки насчитываются за убийство игрока). Статистика хранится в словаре Dictionary<string, PlayerStats>.

– Игра завершается по истечении времени таймера

– Победитель определяется по максимальному количеству очков.

## 3.3 Работа с моделями в игре

Модели и ассеты для «Боёв Стикменов» разработаны и/или взяты из сети Интернет, характерном для жанра Stickman, чтобы снизить системные требования и подчеркнуть культурную связь.

1. Персонажи:

– Модель стикмена разработана в редакторе Unity, занесена в префаб. Схема компонентов изображена на рисунке 3.1:

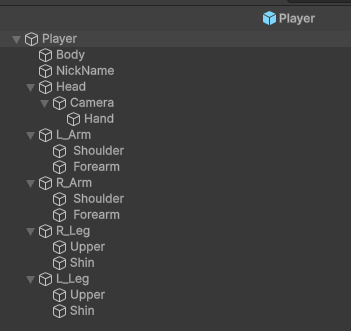


Рисунок 3.1

– Аудиофайлы взяты с просторов Интернета;

– Анимации настроены в Unity Animator с триггерами для переходов (например, isAttacking, isDead), передаются через Photon.

2. Оружие:

– Модели оружия созданы в Unity/скачаны. Префаб винтовки представлен на рисунке 3.2:

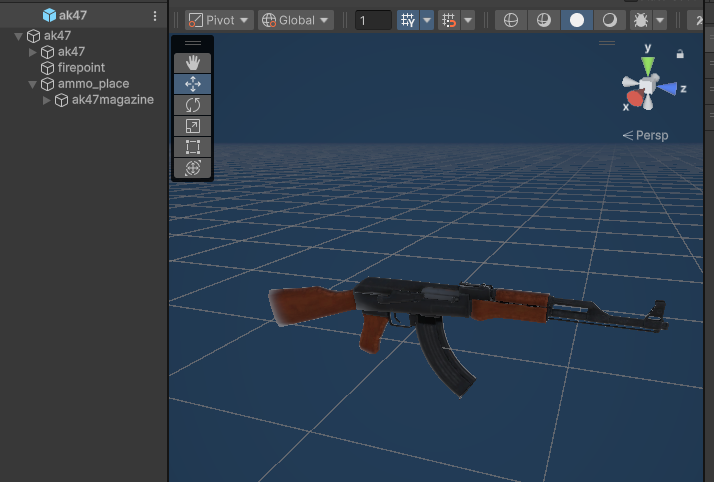


Рисунок 3.2 – Префаб винтовки «AK47»

– Префабы оружия включают скрипты для обработки урона и физики (например, TryDealDamage метод как один из них).

– Также синхронизируемые через Photon.

3. Карты:

– Уровни скачаны из Asset Store Unity, однако тестовые карты были созданы вручную в Unity.

– Карты не имеют слишком большого количества элементов, чтобы соблюдать минимализм.

## 3.4 Работа с сетевым взаимодействием

Сетевое взаимодействие в игре «Бои Стикменов» реализовано с использованием библиотеки Photon PUN 2, обеспечивающей надежную синхронизацию состояний объектов в 3D-пространстве между клиентами в реальном времени. Каждый префаб в сценах (персонажи, оружие, элементы окружения) оснащен компонентом PhotonView, который отвечает за синхронизацию состояний, таких как позиции, повороты, анимации и события (например, нанесение урона).

Главные компоненты системы:

– Изменение никнейма поддерживается через свойство PhotonNetwork.NickName;

– Создание комнаты реализовано через PhotonNetwork.CreateRoom(roomName, new RoomOptions {MaxPlayers = 4, IsVisible = true}), где задаются параметры, такие как название, максимум игроков (2–4) и выбранная карта;

– Присоединение к существующей комнате через поддерживается PhotonNetwork.JoinRoom(roomName);

– Синхронизация данных и процессов реализована через photonView.RPC(methodName, Rpc.Target, params), где photonView – локальный компонент;

– Синхронный переход между сценами игры реализован методом PhotonNetwork.LoadScene(sceneName);

– Передача параметров, необходимых для сетевого взаимодействия и передачи между сценами реализована методами PhotonNetwork.CurrentRoom. SetCustomProperties(properties) и PhotonNetwork.LocalPlayer. SetCustomProperties(properties).

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В этом разделе описывается процесс тестирования игры «Бои стикменов», 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.

Цели:

– Проверить корректность работы игровых механик (движение персонажей, системы боя, выбор оружия).

– Убедиться в стабильности сетевого взаимодействия

– Оценить интуитивность интерфейса.

– Проверить производительность.

– Выявить и устранить возможные ошибки (в том числе ошибки синхронизации), влияющие на игровой процесс.

Было проведено несколько матчей, в которых не было обнаружено значительных ошибок, за исключением некоторых предупреждений. Пример предупреждения представлен на рисунке 4.1.



Рисунок 4.1 – Предупреждение об переопределении лучшего региона

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Руководство пользователя предназначено для игроков, использующих программное средство «Бои Стикменов» на операционной системе Windows. Оно описывает интерфейс игры, управление и основные шаги для начала игры, обеспечивая простое освоение.

## **Интерфейс программного средства**

Интерфейс игры «Бои Стикменов» разработан в минималистичном стиле, соответствующем эстетике жанра Stickman, с использованием черно-белой палитры и ярких акцентов для активных элементов. Основные компоненты интерфейса:

Главное меню:

– Содержит кнопки: «Создать игру» (Host), «Присоединиться» (Connect), «Настройки» (Settings), «Выход» (Exit).

– Поле ввода никнейма позволяет задать псевдоним (до 12 символов), который используется в сетевой игре.

– Фон: силуэты стикменов, созданные в Adobe Photoshop.

Меню представлено на рисунке 5.1:

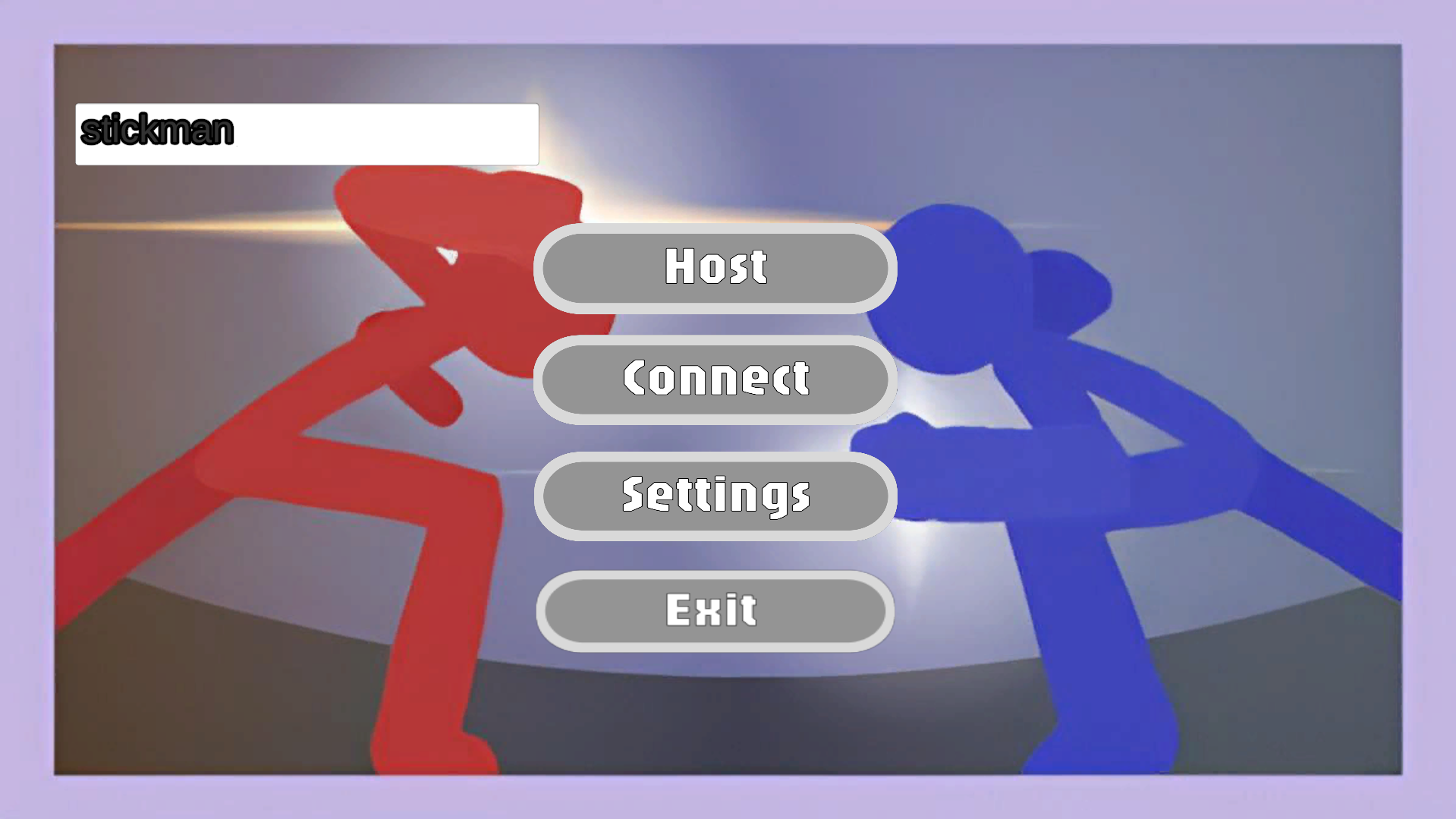


Рисунок 5.1 – Интерфейс главного меню

Меню создания комнаты

Содержит поля для ввода имени комнаты и количества игроков, представлено на рисунке 5.2:



Рисунок 5.2 – Интерфейс создания комнаты

Меню подключения повторяет интерфейс создания комнаты, за исключением отсутствия поля для ввода количества игроков. Представлено на рисунке 5.3:

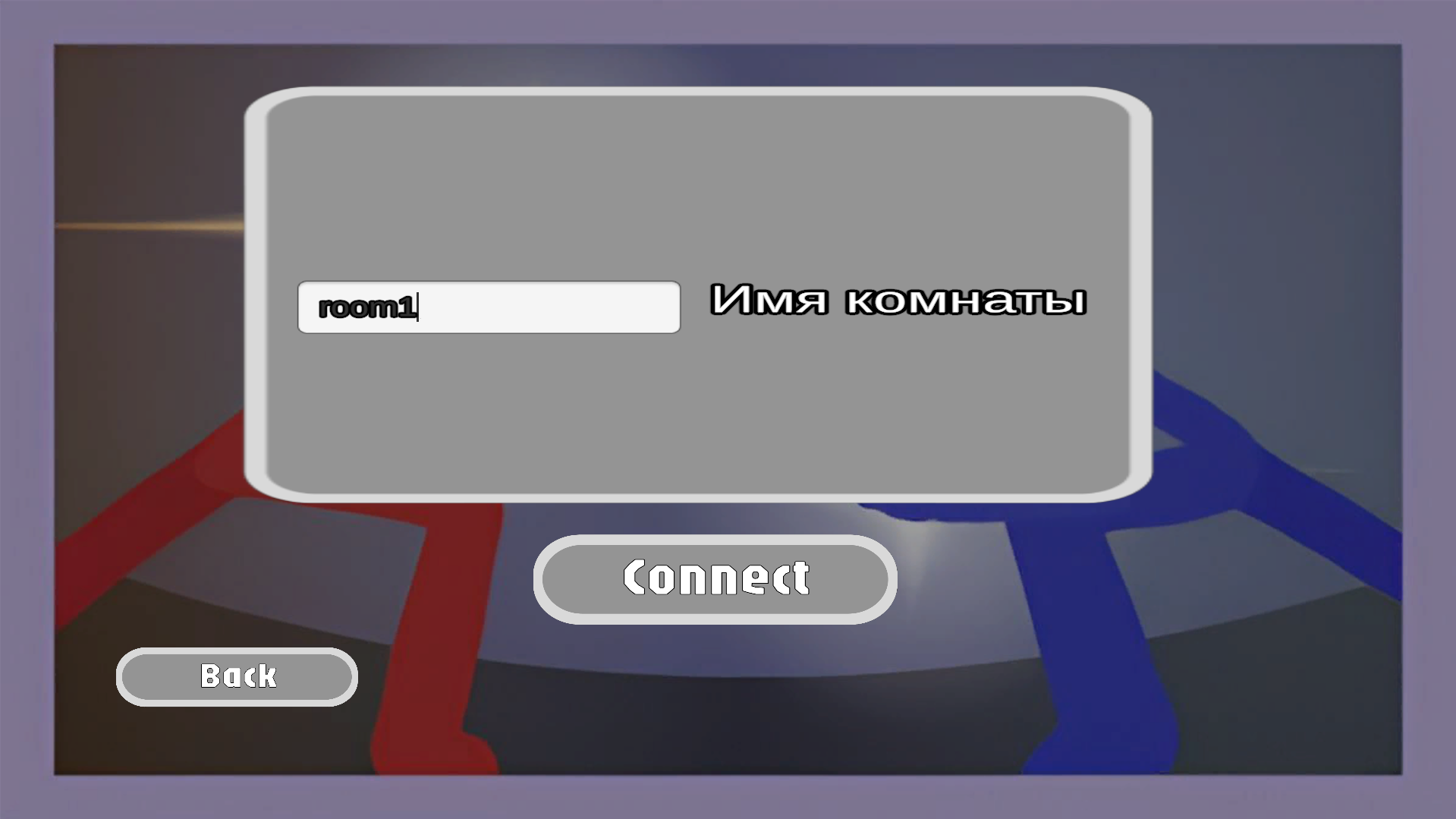


Рисунок 5.3 – Интерфейс подключения к комнате

Меню лобби включает в себя таблицу с выводом имен подключенных игроков, кнопкой для выхода из комнаты и кнопки «Начать», а также панелью с выбором карты на предстоящий матч и текстом, который показывает, какую карту выбрал хост.

Кнопка «Начать» отображается только у хоста, и только он может

выбирать карту.

Представлено на рисунке 5.4:



Рисунок 5.4 – Интерфейс лобби с игроками и выбранной картой

Меню выбора оружия содержит панель выбора оружия игроком, которое он будет использовать на протяжении матча, кнопки выхода и кнопки начала, и плашек с оружием для выбора, на которых изображена картинка оружия и его название. Там же, помимо плашек с оружиями, расположен текст, который показывает, какое оружие выбрал игрок. Представлен на рисунке 5.5:



Рисунок 5.5 – Интерфейс выбора оружия перед матчем

Внутриигровой интерфейс является простым, без большого количества данных или кнопок, здесь только необходимое: панель со статистикой игроков, панелями со значением здоровья и, если у игрока оружие дальнего боя, то и панель с оставшимися патронами в обойме. Представлен на рисунках 5.6 – 5.7:



Рисунок 5.6 – Интерфейс внутри матча с включенной панелью очков



Рисунок 5.7 – Интерфейс внутри матча с выключенной панелью очков

Заключительное окно с выводом результатов матча, в котором игроки расположены по убыванию очков, чтобы было понятнее, кто «победил» в это матче. Состоит из текста с фразой «Конец игры», и панелью, в которой будет распогаться статистика игроков, интерфейс представлен на рисунке 5.8:

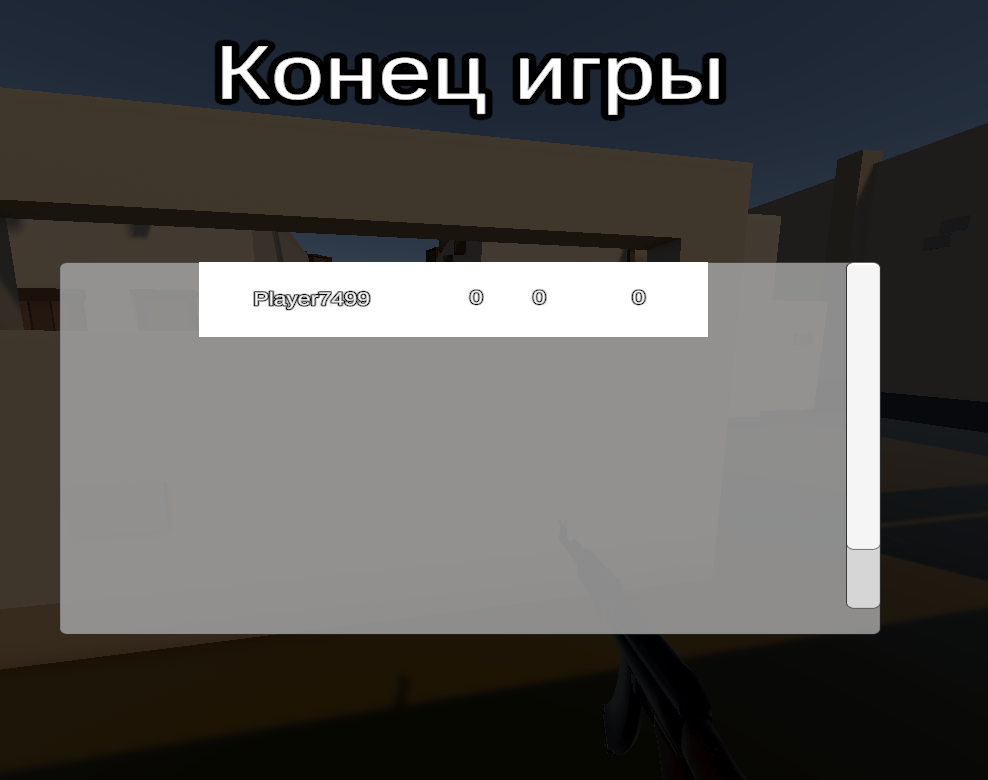


Рисунок 5.8 – Интерфейс в конце матча

## **Управление программным средством**

Перед запуском игры убедитесь, что присутствует интернет-подключение.

Чтобы запустить игру, нужно два раза нажать на kursa4.exe файл (рисунок 5.9). После запуска вы увидете окно загрузки (рисунок 5.10), которое смениться главным меню.



Рисунок 5.9 – Файл «kursa4.exe»



Рисунок 5.10 – Окно загрузки игры

Навигация по интерфейсу происходит с помощью:

– Мыши – выбор кнопок меню, полей для ввода,

– Клавишами Enter (подтверждение ввода) и Esc (вызов меню).

В игровом процессе используются клавиши W-A-S-D для управлением персонажа, клавиша Пробел (Space) для прыжка, левая кнопка мыши — для атаки, перемещение мыши — обзор камерой  
Меню настроек позволяет регулировать:

– Громкость звука (музыка, эффекты).

– Чувствительность мыши (слайдер от 0.5 до 2.0).

# ЗАКЛЮЧЕНИЕ

В рамках курсового проекта была разработана концепция и прототип 3D-мультиплеерной игры «Бои Стикменов» на игровом движке Unity с использованием языка программирования C# и библиотеки Photon PUN 2 для сетевого взаимодействия. Основная цель проекта — создание динамичной и доступной игры в жанре файтинг с минималистичным дизайном, соответствующей современным требованиям к геймплею, интерфейсу и производительности, — была успешно достигнута.

В ходе работы проведен анализ предметной области, включающий обзор аналогов (Stick Fight: The Game, Gang Beasts и другие), что позволило выделить ключевые механики, такие как ragdoll-физика, интерактивное окружение и интуитивный мультиплеер.

Разработана структура программы, включающая модули управления игрой, сетевого взаимодействия, игровой логики, UI и ресурсов, с акцентом на оптимизацию для слабых систем. Реализованы основные функции, указанные в постановке задачи: выбор никнейма, создание и подключение к комнатам, выбор карты и оружия, а также динамичный игровой процесс с поддержкой ближнего и дальнего оружия.

Сетевая синхронизация, реализованная через компоненты PhotonView на всех префабах, обеспечила стабильное взаимодействие между игроками.

Тестирование подтвердило работоспособность функционала, стабильность мультиплеера и удобство интерфейса, а выявленные ошибки (например, рассинхронизация при отключении игроков) были устранены.

Руководство пользователя предоставляет четкие инструкции по использованию игры, делая ее доступной для аудитории.

Разработанный прототип демонстрирует потенциал для дальнейшего развития. Перспективы включают внедрение звукового сопровождения, эффектов частиц, расширенных настроек и поддержки кроссплатформенности (Android, iOS), что усилит конкурентоспособность игры на рынке казуальных файтингов. Проект подчеркивает значимость минималистичного дизайна и сетевых технологий в создании увлекательных игр, способных привлечь широкую аудиторию, и может служить основой для более масштабных разработок в будущем.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Документация по Visual Studio [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/visualstudio/ide/?view=vs-2022.  
 [2] Полное руководство по языку программирования С# – METANIT – [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/tutorial/

[3] Даны, Р. Основы C# и .NET: Полное руководство / Р. Даны. – М.: Диалектика, 2021. – 740 с.

[4] Майерс, Скотт. Эффективное программирование на C#: 50 советов для улучшения кода / Скотт Майерс. – СПб.: Питер, 2021. – 400 с.

[5] Гетц, Джо. C# 8.0: Новые возможности и примеры использования / Джо Гетц. – М.: Издательский дом «Солон-Пресс», 2021. – 536 с.

[6] Официальная документация Unity [Электронный ресурс]. – Режим доступа: https://docs.unity3d.com/Manual/index.html.

[7] Photon PUN 2 Documentation [Электронный ресурс]. – Режим доступа: https://doc.photonengine.com/en-us/pun/current/getting-started/pun-intro.

[8] Шрейнер, Д. Разработка игр на Unity: Полное руководство / Д. Шрейнер, М. Гейг. – М.: ДМК Пресс, 2020. – 672 с.

[9] Окли, Дж. Сетевое программирование для разработчиков игр / Дж. Окли. – СПб.: Питер, 2019. – 432 с.

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

[11] Unity Manual: 3D Physics Reference [Электронный ресурс]. – Режим доступа: https://docs.unity3d.com/Manual/PhysicsSection.html.

[21] Unity Manual: UI System [Электронный ресурс]. – Режим доступа: https://docs.unity3d.com/Manual/UISystem.html.

[22] Торн, А. Unity 2021: Практическое руководство по созданию игр / А. Торн. – М.: ДМК Пресс, 2022. – 592 с.

# ПРИЛОЖЕНИЕ А. Исходный код программы

ConnectToServer.cs:

public class ConnectToServer : MonoBehaviourPunCallbacks

{

void Start()

{

PhotonNetwork.ConnectUsingSettings();

}

public override void OnConnectedToMaster()

{

SceneManager.LoadScene("Menu");

}

}

GameManager.cs:

public class GameManager : MonoBehaviourPunCallbacks

{

[Header("Game Settings")]

public float gameDuration = 300f; // 5 minutes in seconds

public int killPoints = 10;

[Header("UI References")]

public TMP\_Text timerText;

public TMP\_Text killFeedText;

public GameObject scoreboardPanel;

public Transform scoreboardContent;

public GameObject playerScorePrefab;

public GameObject endGamePanel;

public Transform endGameContent;

public GameObject pauseMenuPanel; // Pause menu panel

public Button restartGameButton; // Button for host to restart game

public Button exitGameButton; // Button for all players to exit to main menu

public Button resumeButton; // Resume button in pause menu

public Button pauseExitButton; // Exit button in pause menu

[Header("Scoreboard Settings")]

public float scoreEntryHeight = 50f; // Height of each PlayerScoreEntry

public float scoreEntrySpacing = 5f; // Spacing between entries

public Vector2 scoreEntryStartPos = new Vector2(0, 0); // Starting position (top-center)

private Dictionary<int, PlayerStats> playerStats = new Dictionary<int, PlayerStats>();

private float gameTimeRemaining;

private bool isGameRunning;

private bool isPaused;

private static GameManager instance;

void Awake()

{

// Singleton pattern to prevent duplicate GameManager instances

if (instance != null && instance != this)

{

Debug.LogWarning($"Duplicate GameManager found with PhotonView ID {GetComponent<PhotonView>().ViewID}. Destroying this instance.");

PhotonNetwork.Destroy(gameObject); // Use PhotonNetwork.Destroy for networked objects

return;

}

instance = this;

}

void Start()

{

// Ensure scene synchronization is enabled

PhotonNetwork.AutomaticallySyncScene = true;

InitializeGame();

SetupUI();

}

void OnDestroy()

{

if (instance == this)

{

instance = null;

Debug.Log("GameManager instance cleared on destroy.");

}

}

void InitializeGame()

{

gameTimeRemaining = gameDuration;

isGameRunning = true;

isPaused = false;

playerStats.Clear(); // Clear stats for fresh state

foreach (Player player in PhotonNetwork.PlayerList)

{

playerStats[player.ActorNumber] = new PlayerStats(player);

Debug.Log($"Initialized stats for player {player.NickName} (Actor {player.ActorNumber})");

}

if (timerText != null) UpdateTimerDisplay();

else Debug.LogWarning("timerText is not assigned!");

if (scoreboardPanel != null) scoreboardPanel.SetActive(true);

else Debug.LogWarning("scoreboardPanel is not assigned!");

if (endGamePanel != null) endGamePanel.SetActive(false);

else Debug.LogWarning("endGamePanel is not assigned!");

if (pauseMenuPanel != null) pauseMenuPanel.SetActive(false);

else Debug.LogWarning("pauseMenuPanel is not assigned!");

if (scoreboardContent == null || !scoreboardContent) Debug.LogError("scoreboardContent is not assigned or destroyed!");

if (playerScorePrefab == null) Debug.LogError("playerScorePrefab is not assigned!");

if (endGameContent == null || !endGameContent) Debug.LogError("endGameContent is not assigned or destroyed!");

UpdateScoreboard();

}

void SetupUI()

{

// Show/hide restart button based on whether the player is the host

if (restartGameButton != null)

{

restartGameButton.gameObject.SetActive(PhotonNetwork.IsMasterClient);

restartGameButton.onClick.AddListener(OnRestartGame);

}

else Debug.LogWarning("restartGameButton is not assigned!");

if (exitGameButton != null)

{

exitGameButton.onClick.AddListener(OnExitGame);

}

else Debug.LogWarning("exitGameButton is not assigned!");

if (resumeButton != null)

{

resumeButton.onClick.AddListener(OnResumeGame);

}

else Debug.LogWarning("resumeButton is not assigned!");

if (pauseExitButton != null)

{

pauseExitButton.onClick.AddListener(OnExitGame);

}

else Debug.LogWarning("pauseExitButton is not assigned!");

}

void Update()

{

if (isGameRunning && !isPaused)

{

gameTimeRemaining -= Time.deltaTime;

if (timerText != null) UpdateTimerDisplay();

else Debug.Log($"gameTimeRemaining: {gameTimeRemaining}");

if (gameTimeRemaining <= 0)

{

Debug.Log("Game timer ended, calling EndGame");

EndGame();

}

}

// Check for Escape key to toggle pause menu

if (Input.GetKeyDown(KeyCode.Escape) && isGameRunning)

{

TogglePauseMenu();

}

}

void UpdateTimerDisplay()

{

int minutes = Mathf.FloorToInt(gameTimeRemaining / 60f);

int seconds = Mathf.FloorToInt(gameTimeRemaining % 60f);

timerText.text = $"{minutes:00}:{seconds:00}";

}

void TogglePauseMenu()

{

isPaused = !isPaused;

if (pauseMenuPanel != null)

{

pauseMenuPanel.SetActive(isPaused);

Time.timeScale = isPaused ? 0f : 1f;

Cursor.lockState = isPaused ? CursorLockMode.None : CursorLockMode.Locked;

Debug.Log($"Pause menu {(isPaused ? "shown" : "hidden")}");

}

else

{

Debug.LogWarning("pauseMenuPanel is not assigned!");

}

}

void OnResumeGame()

{

Debug.Log("Resuming game");

TogglePauseMenu();

}

void OnRestartGame()

{

if (PhotonNetwork.IsMasterClient && PhotonNetwork.InRoom)

{

Debug.Log("Host restarting game via RPC");

photonView.RPC("RPC\_RestartGame", RpcTarget.AllBuffered); // Use AllBuffered to ensure RPC persists

}

else

{

Debug.LogWarning("Restart failed: Not master client or not in a room");

}

}

[PunRPC]

void RPC\_RestartGame()

{

Debug.Log($"Restarting game on {PhotonNetwork.LocalPlayer.NickName}");

playerStats.Clear();

InitializeGame();

Time.timeScale = 1f;

if (PhotonNetwork.IsMasterClient)

{

PhotonNetwork.LoadLevel(SceneManager.GetActiveScene().name);

}

}

void OnExitGame()

{

if (!PhotonNetwork.InRoom)

{

Debug.LogWarning("Cannot exit: Not in a room");

LoadMainMenu();

return;

}

if (PhotonNetwork.IsMasterClient)

{

Debug.Log("Host exiting game, destroying room");

StartCoroutine(ExitGameCoroutine());

}

else

{

Debug.Log("Client exiting game");

StartCoroutine(LoadMainMenuCoroutine());

}

}

private IEnumerator ExitGameCoroutine()

{

if (PhotonNetwork.InRoom)

{

photonView.RPC("RPC\_ExitToMainMenu", RpcTarget.AllBuffered);

PhotonNetwork.DestroyPlayerObjects(PhotonNetwork.LocalPlayer);

yield return new WaitForSecondsRealtime(0.5f); // Wait for RPC and object destruction

if (PhotonNetwork.InRoom)

{

PhotonNetwork.LeaveRoom();

}

}

else

{

LoadMainMenu();

}

}

[PunRPC]

void RPC\_ExitToMainMenu()

{

Debug.Log($"Received RPC\_ExitToMainMenu on {PhotonNetwork.LocalPlayer.NickName}");

StartCoroutine(LoadMainMenuCoroutine());

}

private IEnumerator LoadMainMenuCoroutine()

{

if (PhotonNetwork.InRoom)

{

PhotonNetwork.LeaveRoom();

yield return new WaitUntil(() => !PhotonNetwork.InRoom);

}

LoadMainMenu();

}

void LoadMainMenu()

{

Time.timeScale = 1f;

Cursor.lockState = CursorLockMode.None;

if (PhotonNetwork.IsConnected)

{

PhotonNetwork.Disconnect();

}

SceneManager.LoadScene("Menu");

}

public override void OnLeftRoom()

{

base.OnLeftRoom();

Debug.Log("Player left room, loading main menu");

LoadMainMenu();

}

public override void OnDisconnected(DisconnectCause cause)

{

base.OnDisconnected(cause);

Debug.Log($"Disconnected from Photon: {cause}");

if (isGameRunning)

{

Debug.Log("Showing endGamePanel locally due to disconnection");

RPC\_EndGame();

}

LoadMainMenu();

}

public void RegisterKill(int killerActorNumber, int victimActorNumber)

{

if (!isGameRunning) return;

Debug.Log($"RegisterKill called: killer={killerActorNumber}, victim={victimActorNumber}");

if (PhotonNetwork.InRoom)

{

photonView.RPC("RPC\_RegisterKill", RpcTarget.AllBuffered, killerActorNumber, victimActorNumber);

}

else

{

Debug.LogWarning("Cannot register kill: Not in a room");

}

}

[PunRPC]

void RPC\_RegisterKill(int killerActorNumber, int victimActorNumber)

{

if (killerActorNumber != victimActorNumber && playerStats.ContainsKey(killerActorNumber))

{

playerStats[killerActorNumber].kills++;

playerStats[killerActorNumber].score += killPoints;

}

if (playerStats.ContainsKey(victimActorNumber))

{

playerStats[victimActorNumber].deaths++;

}

Player killer = PhotonNetwork.CurrentRoom?.GetPlayer(killerActorNumber);

Player victim = PhotonNetwork.CurrentRoom?.GetPlayer(victimActorNumber);

if (killer != null && victim != null)

{

AddKillFeedMessage($"{killer.NickName} убил {victim.NickName}");

Debug.Log($"Kill feed: {killer.NickName} убил {victim.NickName}");

}

else

{

Debug.LogWarning($"Failed to get player names for killer={killerActorNumber}, victim={victimActorNumber}");

}

UpdateScoreboard();

}

void AddKillFeedMessage(string message)

{

if (killFeedText == null)

{

Debug.LogWarning("killFeedText is not assigned!");

return;

}

killFeedText.text = message + "\n" + killFeedText.text;

string[] lines = killFeedText.text.Split('\n');

if (lines.Length > 5)

{

killFeedText.text = string.Join("\n", lines, 0, 5);

}

}

void UpdateScoreboard()

{

if (scoreboardContent == null || !scoreboardContent)

{

Debug.LogError("Cannot update scoreboard: scoreboardContent is null or destroyed!");

return;

}

if (playerScorePrefab == null)

{

Debug.LogError("Cannot update scoreboard: playerScorePrefab is null!");

return;

}

// Clear existing entries

foreach (Transform child in scoreboardContent)

{

if (child != null) Destroy(child.gameObject);

}

List<PlayerStats> sortedStats = new List<PlayerStats>(playerStats.Values);

sortedStats.Sort((a, b) => b.score.CompareTo(a.score));

Debug.Log($"Updating scoreboard with {sortedStats.Count} entries");

// Position entries manually

for (int i = 0; i < sortedStats.Count; i++)

{

PlayerStats stats = sortedStats[i];

GameObject entry = Instantiate(playerScorePrefab, scoreboardContent);

PlayerScoreEntry scoreEntry = entry.GetComponent<PlayerScoreEntry>();

if (scoreEntry != null)

{

scoreEntry.Initialize(

stats.player.NickName,

stats.kills.ToString(),

stats.deaths.ToString(),

stats.score.ToString()

);

// Set position

RectTransform rect = entry.GetComponent<RectTransform>();

if (rect != null)

{

rect.anchorMin = new Vector2(0.5f, 1f);

rect.anchorMax = new Vector2(0.5f, 1f);

rect.pivot = new Vector2(0.5f, 1f);

float yPos = scoreEntryStartPos.y - (i \* (scoreEntryHeight + scoreEntrySpacing));

rect.anchoredPosition = new Vector2(scoreEntryStartPos.x, yPos);

rect.sizeDelta = new Vector2(rect.sizeDelta.x, scoreEntryHeight);

Debug.Log($"Positioned entry for {stats.player.NickName} at y={yPos}");

}

else

{

Debug.LogWarning($"RectTransform missing on PlayerScoreEntry for {stats.player.NickName}");

}

}

else

{

Debug.LogWarning($"PlayerScoreEntry component missing on instantiated prefab for {stats.player.NickName}");

}

}

}

void EndGame()

{

isGameRunning = false;

Debug.Log("Ending game");

if (PhotonNetwork.InRoom && photonView != null)

{

photonView.RPC("RPC\_EndGame", RpcTarget.AllBuffered);

}

else

{

Debug.LogWarning("Cannot send RPC\_EndGame: Not in a room or photonView is null, showing end game locally");

RPC\_EndGame();

}

}

[PunRPC]

void RPC\_EndGame()

{

Debug.Log($"RPC\_EndGame called on {PhotonNetwork.LocalPlayer.NickName}");

Time.timeScale = 0;

if (timerText != null) timerText.text = string.Empty;

else Debug.LogWarning("timerText is null in RPC\_EndGame");

if (killFeedText != null) killFeedText.text = string.Empty;

else Debug.LogWarning("killFeedText is null in RPC\_EndGame");

if (scoreboardPanel != null && scoreboardPanel) scoreboardPanel.SetActive(false);

else Debug.LogWarning("scoreboardPanel is not assigned or destroyed!");

if (endGamePanel != null && endGamePanel)

{

endGamePanel.SetActive(true);

Debug.Log("endGamePanel activated");

}

else

{

Debug.LogError("endGamePanel is not assigned or destroyed!");

}

if (endGameContent == null || !endGameContent || playerScorePrefab == null)

{

Debug.LogError("Cannot populate endGameContent: endGameContent or playerScorePrefab is null or destroyed!");

return;

}

// Clear existing entries

foreach (Transform child in endGameContent)

{

if (child != null) Destroy(child.gameObject);

}

List<PlayerStats> sortedStats = new List<PlayerStats>(playerStats.Values);

sortedStats.Sort((a, b) => b.score.CompareTo(a.score));

Debug.Log($"Populating endGameContent with {sortedStats.Count} entries");

// Position entries manually

for (int i = 0; i < sortedStats.Count; i++)

{

PlayerStats stats = sortedStats[i];

GameObject entry = Instantiate(playerScorePrefab, endGameContent);

PlayerScoreEntry scoreEntry = entry.GetComponent<PlayerScoreEntry>();

if (scoreEntry != null)

{

scoreEntry.Initialize(

stats.player.NickName,

stats.kills.ToString(),

stats.deaths.ToString(),

stats.score.ToString()

);

// Set position

RectTransform rect = entry.GetComponent<RectTransform>();

if (rect != null)

{

rect.anchorMin = new Vector2(0.5f, 1f);

rect.anchorMax = new Vector2(0.5f, 1f);

rect.pivot = new Vector2(0.5f, 1f);

float yPos = scoreEntryStartPos.y - (i \* (scoreEntryHeight + scoreEntrySpacing));

rect.anchoredPosition = new Vector2(scoreEntryStartPos.x, yPos);

rect.sizeDelta = new Vector2(rect.sizeDelta.x, scoreEntryHeight);

Debug.Log($"Positioned endGame entry for {stats.player.NickName} at y={yPos}");

}

else

{

Debug.LogWarning($"RectTransform missing on PlayerScoreEntry for {stats.player.NickName}");

}

}

else

{

Debug.LogWarning($"PlayerScoreEntry component missing on instantiated prefab for {stats.player.NickName}");

}

}

}

public override void OnPlayerEnteredRoom(Player newPlayer)

{

if (!playerStats.ContainsKey(newPlayer.ActorNumber))

{

playerStats[newPlayer.ActorNumber] = new PlayerStats(newPlayer);

Debug.Log($"Player {newPlayer.NickName} joined, updating scoreboard");

UpdateScoreboard();

}

}

public override void OnPlayerLeftRoom(Player otherPlayer)

{

if (playerStats.ContainsKey(otherPlayer.ActorNumber))

{

playerStats.Remove(otherPlayer.ActorNumber);

Debug.Log($"Player {otherPlayer.NickName} left, updating scoreboard");

UpdateScoreboard();

}

}

private class PlayerStats

{

public Player player;

public int kills;

public int deaths;

public int score;

public PlayerStats(Player player)

{

this.player = player;

kills = 0;

deaths = 0;

score = 0;

}

}

}

SpawnPlayer.cs:

public class SpawnPlayers : MonoBehaviour

{

[SerializeField] private List<Transform> spawnPlaces = new List<Transform>();

[SerializeField] private List<GameObject> weapons = new List<GameObject>();

private string weapon;

public GameObject player;

public float minX, minZ, maxX, maxZ;

void Start()

{

IEnumerable<Transform> randomPlaces = spawnPlaces.OrderBy(x => Random.value).Take(1);

IEnumerable<GameObject> randomWeapons = weapons.OrderBy(x => Random.value).Take(1);

Vector3 randomPosition = randomPlaces.First().position;

if (PhotonNetwork.LocalPlayer.CustomProperties.TryGetValue("selectedWeapon", out object loaded\_weapon))

{

weapon = loaded\_weapon.ToString();

Debug.Log($"Оружие: {weapon}");

}

else

{

weapon = randomWeapons.First().name;

Debug.LogWarning("Сбой, выдача случайного оружия");

}

var c = PhotonNetwork.Instantiate(player.name.Replace("(Clone)", ""), randomPosition, Quaternion.identity);

c.name = c.name.Replace("(Clone)", "");

TMP\_Text nicknameText = c.GetComponentInChildren<TMP\_Text>();

if (nicknameText != null)

{

nicknameText.text = PhotonNetwork.LocalPlayer.NickName;

}

else

{

Debug.LogWarning("Не найден компонент TMP\_Text для отображения ника");

}

c.GetComponent<Target>().positions = spawnPlaces;

GameObject instantiatedWeapon = PhotonNetwork.Instantiate(weapon, Vector3.zero, Quaternion.identity);

instantiatedWeapon.transform.SetParent(c.transform.Find("Head/Camera/Hand"), false);

instantiatedWeapon.GetComponent<WeaponSway>().Picked();

if (instantiatedWeapon.GetComponent<SimpleMeleeWeapon>() != null) {

// Debug.Log("1234567890");

instantiatedWeapon.GetComponent<SimpleMeleeWeapon>().isPicked = true;

instantiatedWeapon.GetComponent<SimpleMeleeWeapon>().model = c;

// Debug.Log(instantiatedWeapon.GetComponent<SimpleMeleeWeapon>().isPicked);

}

else {

//Debug.Log("xc,mnnxc,m,bxxm,n,");

instantiatedWeapon.GetComponent<Gun>().Picked(c.transform.Find("Head/Camera"));

// Debug.Log(instantiatedWeapon.GetComponent<Gun>().isPicked);

}

}

private void OnDrawGizmos()

{

if (spawnPlaces.Count > 0)

{

Gizmos.color = Color.red;

foreach (var place in spawnPlaces)

{

if (place != null)

{

Gizmos.DrawWireSphere(place.position, 0.3f);

Gizmos.DrawLine(place.position, place.position + place.forward \* 0.5f);

}

}

}

}

}

WeaponSelectionManager.cs:

public class WeaponSelectionManager : MonoBehaviourPunCallbacks

{

[Header("Settings")]

public string defaultGameScene = "de\_map1";

public float selectionTime = 60f;

public float startDelay = 5f;

public WeaponData[] availableWeapons;

[Header("UI References")]

public TMP\_Text timerText;

public TMP\_Text statusText;

public RectTransform weaponGrid;

public GameObject weaponButtonPrefab;

public Button startGameButton;

public Button backToLobbyButton;

public TMP\_Text selectedWeaponText;

private float remainingTime;

private bool countingDown;

private int localSelectedWeapon = -1;

private List<WeaponButton> weaponButtons = new List<WeaponButton>();

private string selectedMapScene;

void Start()

{

// Получаем выбранную карту

if (PhotonNetwork.CurrentRoom != null &&

PhotonNetwork.CurrentRoom.CustomProperties.TryGetValue("SelectedMapScene", out object sceneName))

{

selectedMapScene = (string)sceneName;

Debug.Log($"Будет загружена карта: {selectedMapScene}");

}

else

{

selectedMapScene = defaultGameScene;

Debug.LogWarning("Карта не выбрана, будет использована сцена по умолчанию");

}

remainingTime = selectionTime;

countingDown = true;

startGameButton.gameObject.SetActive(PhotonNetwork.IsMasterClient);

startGameButton.onClick.AddListener(StartGame);

backToLobbyButton.onClick.AddListener(ReturnToLobby);

InitializeWeaponGrid();

UpdateUI();

}

void InitializeWeaponGrid()

{

// Очистка предыдущих кнопок

foreach (Transform child in weaponGrid)

{

if (child != null && child.gameObject != null)

Destroy(child.gameObject);

}

weaponButtons.Clear();

if (availableWeapons == null || availableWeapons.Length == 0)

{

Debug.LogError("No weapons available in availableWeapons array!");

return;

}

float tileWidth = 333f;

float spacing = 20f;

float totalWidth = (availableWeapons.Length \* tileWidth) +

((availableWeapons.Length - 1) \* spacing);

weaponGrid.sizeDelta = new Vector2(totalWidth, weaponGrid.sizeDelta.y);

for (int i = 0; i < availableWeapons.Length; i++)

{

CreateWeaponButton(i, tileWidth, spacing);

}

}

void CreateWeaponButton(int weaponIndex, float tileWidth, float spacing)

{

if (weaponButtonPrefab == null)

{

Debug.LogError("Weapon button prefab is not assigned!");

return;

}

GameObject buttonObj = Instantiate(weaponButtonPrefab, weaponGrid);

if (buttonObj == null)

{

Debug.LogError("Failed to instantiate weapon button!");

return;

}

WeaponButton button = buttonObj.GetComponent<WeaponButton>();

if (button == null)

{

Debug.LogError("WeaponButton component not found on prefab!");

Destroy(buttonObj);

return;

}

// Устанавливаем позицию кнопки

RectTransform buttonRect = buttonObj.GetComponent<RectTransform>();

if (buttonRect != null)

{

float xPos = (tileWidth + spacing) \* weaponIndex - tileWidth;

buttonRect.anchoredPosition = new Vector2(xPos, 0);

}

// Инициализируем кнопку

if (weaponIndex >= 0 && weaponIndex < availableWeapons.Length)

{

button.Initialize(

availableWeapons[weaponIndex].weaponName,

availableWeapons[weaponIndex].weaponIcon,

() => SelectWeapon(weaponIndex)

);

}

weaponButtons.Add(button);

}

void SelectWeapon(int weaponIndex)

{

if (weaponIndex < 0 || weaponIndex >= weaponButtons.Count || weaponIndex >= availableWeapons.Length)

{

Debug.LogError($"Invalid weapon index: {weaponIndex}");

return;

}

if (localSelectedWeapon == weaponIndex)

return;

// Снимаем выделение с предыдущей кнопки

if (localSelectedWeapon >= 0 && localSelectedWeapon < weaponButtons.Count)

{

weaponButtons[localSelectedWeapon]?.SetSelected(false);

}

localSelectedWeapon = weaponIndex;

if (weaponButtons[weaponIndex] != null)

{

weaponButtons[weaponIndex].SetSelected(true);

selectedWeaponText.text = $"Выбрано: {availableWeapons[weaponIndex].weaponName}";

}

ExitGames.Client.Photon.Hashtable props = new ExitGames.Client.Photon.Hashtable();

props["selectedWeapon"] = availableWeapons[weaponIndex].weaponName;

Debug.Log(props["selectedWeapon"]);

PhotonNetwork.LocalPlayer.SetCustomProperties(props);

}

void Update()

{

if (countingDown)

{

remainingTime -= Time.deltaTime;

UpdateUI();

if (remainingTime <= 0)

{

countingDown = false;

AutoSelectWeapons();

}

}

}

void UpdateUI()

{

if (timerText != null)

timerText.text = $"Осталось: {Mathf.FloorToInt(remainingTime)} сек";

if (statusText != null)

statusText.text = countingDown ? "Выберите оружие" : "Завершаем выбор...";

}

void AutoSelectWeapons()

{

if (PhotonNetwork.PlayerList == null || availableWeapons == null || availableWeapons.Length == 0)

return;

foreach (Player player in PhotonNetwork.PlayerList)

{

if (player != null && !player.CustomProperties.ContainsKey("selectedWeapon"))

{

int randomWeapon = Random.Range(0, availableWeapons.Length);

if (photonView != null)

{

photonView.RPC("RPC\_ForceSelectWeapon", player, randomWeapon);

}

}

}

}

[PunRPC]

void RPC\_ForceSelectWeapon(int weaponIndex)

{

if (weaponIndex >= 0 && weaponIndex < availableWeapons.Length)

{

SelectWeapon(weaponIndex);

}

}

public void StartGame()

{

if (!PhotonNetwork.IsMasterClient)

return;

bool allSelected = true;

foreach (Player player in PhotonNetwork.PlayerList)

{

if (player == null || !player.CustomProperties.ContainsKey("selectedWeapon"))

{

allSelected = false;

break;

}

}

if (allSelected)

{

if (photonView != null)

{

photonView.RPC("RPC\_StartGameCountdown", RpcTarget.All);

}

}

else if (statusText != null)

{

statusText.text = "Не все игроки выбрали оружие!";

}

}

[PunRPC]

void RPC\_StartGameCountdown()

{

StartCoroutine(GameStartCountdown());

}

IEnumerator GameStartCountdown()

{

countingDown = false;

// Синхронизируем изменение текста для всех игроков

photonView.RPC("RPC\_UpdateStatusText", RpcTarget.All, "Игра начинается...");

yield return new WaitForSeconds(startDelay);

if (PhotonNetwork.IsMasterClient)

{

if (!string.IsNullOrEmpty(selectedMapScene))

{

// Сохраняем выбранную сцену в свойствах комнаты

ExitGames.Client.Photon.Hashtable props = new ExitGames.Client.Photon.Hashtable();

props["SelectedMapScene"] = selectedMapScene;

PhotonNetwork.CurrentRoom.SetCustomProperties(props);

// Загружаем сцену для всех игроков

PhotonNetwork.LoadLevel(selectedMapScene);

}

else

{

Debug.LogError("Selected map scene is null or empty!");

photonView.RPC("RPC\_UpdateStatusText", RpcTarget.All, "Ошибка: карта не выбрана");

}

}

}

[PunRPC]

void RPC\_UpdateStatusText(string newText)

{

if (statusText != null)

{

statusText.text = newText;

}

}

public override void OnRoomPropertiesUpdate(ExitGames.Client.Photon.Hashtable propertiesThatChanged)

{

if (propertiesThatChanged.ContainsKey("SelectedMapScene"))

{

string sceneName = (string)propertiesThatChanged["SelectedMapScene"];

if (!string.IsNullOrEmpty(sceneName))

{

PhotonNetwork.LoadLevel(sceneName);

}

}

}

public void ReturnToLobby()

{

if (PhotonNetwork.IsMasterClient && PhotonNetwork.CurrentRoom != null)

{

PhotonNetwork.CurrentRoom.IsOpen = true;

photonView.RPC("RPC\_ReturnToLobby", RpcTarget.All);

}

else

{

PhotonNetwork.LeaveRoom();

}

}

[PunRPC]

private void RPC\_ReturnToLobby()

{

PhotonNetwork.LoadLevel("Lobby");

}

public override void OnLeftRoom()

{

SceneManager.LoadScene("Menu");

}

}

[System.Serializable]

public class WeaponData

{

public string weaponName;

public Sprite weaponIcon;

public GameObject weaponPrefab;

}

Gun.cs:

public class Gun : MonoBehaviourPunCallbacks

{

[Header("References")]

public GunData gunData;

private Transform cam;

[SerializeField] private AudioClip audioClip;

[SerializeField] private AudioClip reloadSound;

[SerializeField] private KeyCode reloadKey = KeyCode.R;

public bool isPicked = false;

private float timeSinceLastShot;

private PhotonView parentPhotonView; // PhotonView of the parent player

private void Start()

{

parentPhotonView = GetComponentInParent<PhotonView>();

gunData.currentAmmo = gunData.magSize;

}

private void Update()

{

//Debug.Log($"{photonView.IsMine} and {isPicked}");

if (!photonView.IsMine || !isPicked) return;

//Debug.Log($"{!photonView.IsMine} and {!isPicked} 22");

timeSinceLastShot += Time.deltaTime;

if (Input.GetMouseButton(0))

Shoot();

if (Input.GetKeyDown(reloadKey))

StartReload();

Debug.DrawRay(cam.position, cam.forward \* gunData.maxDistance);

}

public void Picked(Transform camera)

{

Debug.Log("asfd");

cam = camera;

isPicked = true;

}

public void Unpicked()

{

cam = null;

isPicked = false;

}

private void OnDisable() => gunData.reloading = false;

public void StartReload()

{

if (!gunData.reloading && gameObject.activeSelf)

{

AudioSource.PlayClipAtPoint(reloadSound, transform.position);

photonView.RPC("RPC\_Reload", RpcTarget.All);

}

}

[PunRPC]

private IEnumerator RPC\_Reload()

{

AudioSource.PlayClipAtPoint(reloadSound, transform.position);

gunData.reloading = true;

yield return new WaitForSeconds(gunData.reloadTime);

gunData.currentAmmo = gunData.magSize;

gunData.reloading = false;

}

private bool CanShoot() => !gunData.reloading && timeSinceLastShot > 1f / (gunData.fireRate / 60f);

private void Shoot()

{

if (gunData.currentAmmo <= 0 || !CanShoot()) return;

AudioSource.PlayClipAtPoint(audioClip, transform.position);

photonView.RPC("RPC\_Shoot", RpcTarget.All);

}

[PunRPC]

private void RPC\_Shoot()

{

if (gunData.currentAmmo <= 0 || !isPicked || cam == null) return;

AudioSource.PlayClipAtPoint(audioClip, transform.position);

gunData.currentAmmo--;

timeSinceLastShot = 0;

if (Physics.Raycast(cam.position, cam.forward, out RaycastHit hitInfo, gunData.maxDistance))

{

IDamageable damageable = hitInfo.transform.GetComponent<IDamageable>();

if (damageable != null)

{

damageable.TakeDamage(gunData.damage, parentPhotonView.Owner);

Debug.Log($"Dealt {gunData.damage} damage to {hitInfo.transform.name}");

}

}

}

}

Target.cs

public class Target : MonoBehaviourPun, IDamageable

{

public float health = 100f;

public List<Transform> positions;

private bool isShielded = false;

[SerializeField] private float shieldDuration = 10f;

[SerializeField] private Material shieldedMaterial;

[SerializeField] private Material defaultMaterial;

[SerializeField] private ParticleSystem shieldParticles;

private void Start()

{

photonView.RPC("RPC\_ActivateShield", RpcTarget.All, false);

if (photonView == null)

{

Debug.LogError("PhotonView is missing on Target!");

}

if (shieldParticles == null)

{

Debug.LogWarning("ShieldParticles not assigned on Target!");

}

}

public void TakeDamage(float damage, Player attacker)

{

if (attacker == null)

{

Debug.LogWarning("TakeDamage called with null attacker!");

return;

}

Debug.Log($"TakeDamage called: damage={damage}, attacker={attacker.NickName}, target={gameObject.name}");

photonView.RPC("RPC\_TakeDamage", RpcTarget.MasterClient, damage, attacker.ActorNumber);

}

[PunRPC]

void RPC\_TakeDamage(float damage, int attackerActorNumber)

{

if (isShielded)

{

Debug.Log($"Damage ignored due to shield on {gameObject.name}");

return;

}

Debug.Log($"RPC\_TakeDamage: damage={damage}, attackerActorNumber={attackerActorNumber}, currentHealth={health}");

health -= damage;

if (health <= 0)

{

photonView.RPC("RPC\_RespawnTarget", RpcTarget.AllBuffered);

GameManager gameManager = FindFirstObjectByType<GameManager>();

if (gameManager != null)

{

Debug.Log($"Registering kill: killer={attackerActorNumber}, victim={photonView.Owner.ActorNumber}");

gameManager.RegisterKill(attackerActorNumber, photonView.Owner.ActorNumber);

}

else

{

Debug.LogWarning("GameManager not found!");

}

photonView.RPC("RPC\_RespawnTarget", RpcTarget.AllBuffered);

}

photonView.RPC("RPC\_UpdateHealth", RpcTarget.AllBuffered, health);

}

[PunRPC]

void RPC\_UpdateHealth(float newHealth)

{

health = newHealth;

Debug.Log($"Health updated to {health} on {gameObject.name}");

}

[PunRPC]

void RPC\_RespawnTarget()

{

if (positions == null || positions.Count == 0)

{

Debug.LogWarning("No positions defined for target respawn!");

return;

}

Transform newPosition = positions[Random.Range(0, positions.Count)];

health = 100f;

isShielded = true;

transform.position = newPosition.position;

transform.rotation = newPosition.rotation;

photonView.RPC("RPC\_UpdateHealth", RpcTarget.All, health);

photonView.RPC("RPC\_ActivateShield", RpcTarget.All, true);

if (photonView.IsMine)

{

StartCoroutine(ShieldTimer());

}

Debug.Log($"Respawned at {newPosition.position}, shield active");

}

[PunRPC]

void RPC\_ActivateShield(bool active)

{

isShielded = active;

if (shieldParticles != null)

{

if (active) shieldParticles.Play();

else shieldParticles.Stop();

}

Debug.Log($"Shield {(active ? "activated" : "deactivated")} on {gameObject.name}");

}

private IEnumerator ShieldTimer()

{

yield return new WaitForSeconds(shieldDuration);

isShielded = false;

photonView.RPC("RPC\_ActivateShield", RpcTarget.All, false);

Debug.Log($"Shield timer ended on {gameObject.name}");

}

}