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:

**СЕТЕВАЯ КОМПЬЮТЕРНАЯ ИГРА В ЖАНРЕ CТРАТЕГИИ**

БГУИР КП 6-05-0612-01 038 ПЗ

Студент Поливкин В.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 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 13](#_Toc199159586)

[3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 15](#_Toc199159587)

[3.1 Разработка интерфейсов 15](#_Toc199159588)

[3.2 Реализация основного функционала 17](#_Toc199159589)

[3.3 Работа с моделями в игре 18](#_Toc199159590)

[4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 19](#_Toc199159592)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 20](#_Toc199159593)

[5.1 Интерфейс программного средства 20](#_Toc199159594)

[5.2 Управление программным средством 23](#_Toc199159595)

[ЗАКЛЮЧЕНИЕ 25](#_Toc199159596)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 26](#_Toc199159597)

[ПРИЛОЖЕНИЕ А. Исходный код программы 27](#_Toc199159598)

# ВВЕДЕНИЕ

История компьютерных игр берет начало в 1950–1960-х годах, когда первые интерактивные программы, такие как «Tennis for Two» (1958) и «Spacewar!» (1962), заложили основы для развития индустрии. С появлением персональных компьютеров и игровых консолей в 1970–1980-х годах начался стремительный рост жанров, среди которых выделился жанр Tower Defense. Его корни уходят в стратегии реального времени, такие как «Dune II» (1992), где игрокам впервые пришлось защищать базы от волн врагов. В 2000-х годах жанр Tower Defense стал самостоятельным благодаря популярности браузерных игр, таких как «Desktop Tower Defense» (2007), которые привлекли миллионы игроков простотой и стратегической глубиной.

Жанр Tower Defense характеризуется необходимостью защищать базу или территорию, размещая оборонительные сооружения (башни) для уничтожения наступающих врагов. Простота механик в сочетании с тактической сложностью делает этот жанр привлекательным для широкой аудитории. Современные технологии, такие как игровой движок Unity, упростили создание таких игр, позволяя реализовать сложные алгоритмы поведения врагов, разнообразные типы башен и динамичные визуальные эффекты. Развитие мобильных платформ и сетевых технологий также способствовало популяризации жанра, позволяя игрокам соревноваться или сотрудничать в реальном времени.

Актуальность разработки игр в жанре Tower Defense обусловлена их универсальностью: они подходят как для казуальных игроков, так и для тех, кто ищет глубокий стратегический опыт. Такие проекты, как «Kingdom Rush» и «Bloons TD», демонстрируют коммерческий успех и устойчивый интерес аудитории. Кроме того, жанр предоставляет широкие возможности для экспериментов с механиками, такими как улучшение башен, управление ресурсами или внедрение героев с уникальными способностями.

Целью данного курсового проекта является разработка концепции и прототипа игры в жанре Tower Defense на игровом движке Unity, предназначенной для операционной системы Windows. Задачи включают анализ предметной области, проектирование игровых механик, создание интуитивного интерфейса и реализацию базового функционала, соответствующего требованиям жанра. Проект ориентирован на создание увлекательной игры с акцентом на стратегическое планирование и динамичный геймплей, способной привлечь внимание современной аудитории.

# **АНАЛИЗ** ПРЕДМЕТНОЙ ОБЛАСТИ

## Обзор аналогов

Для разработки игры в жанре Tower Defense необходимо изучить успешные проекты, которые задают стандарты качества в жанре, предлагают проверенные механики и демонстрируют подходы к вовлечению аудитории. Анализ аналогов позволяет выявить ключевые элементы успешных игр, включая геймплей, визуальный стиль, техническую реализацию и особенности интерфейса. Ниже представлен обзор трёх популярных игр в жанре Tower Defense, которые послужат ориентирами для проектирования разрабатываемого прототипа.



Рисунок 1.1 – Постер игры «Kingdom Rush»

«Kingdom Rush» (2011, Ironhide Game Studio) — эталон жанра Tower Defense, доступный на ПК, мобильных устройствах и в браузерах. Игрок защищает средневековое королевство от орд врагов (орки, тролли, драконы), размещая башни четырех типов: лучники, маги, артиллерия и казармы. Каждая башня имеет ветку улучшений, позволяющую адаптировать стратегию к различным типам врагов (например, летающим или бронированным). Уникальной особенностью является система героев — управляемых персонажей с уникальными способностями, которые добавляют динамики в геймплей. Уровни имеют разветвленные пути, что требует от игрока тактического планирования.

Игра получила высокие оценки (93% положительных отзывов на Steam из 14,672) благодаря сбалансированному сочетанию простоты управления и стратегической глубины. Визуальный стиль — мультяшная 2D-графика с яркими цветами — делает игру доступной для широкой аудитории. Однако, в отличие от Kingdom Rush, разрабатываемая игра будет ориентирована на минималистичный визуальный стиль, чтобы снизить системные требования.



Рисунок 1.2 – Постер игры «Bloons TD 6»

«Bloons TD 6» (2018, Ninja Kiwi) — одна из самых популярных современных игр жанра, доступная на ПК, iOS и Android. Игрок защищает территорию от волн разноцветных воздушных шаров, используя башни в виде обезьян, каждая из которых обладает уникальными способностями (например, метание дротиков, магические заклинания, взрывы). Ключевая особенность — глубокая система улучшений: каждая башня имеет три ветки апгрейдов, позволяющих создавать специализированные стратегии.

Игра поддерживает кооперативный мультиплеер, что повышает «реиграбельность», а также предлагает сотни уровней с различными режимами сложности. «Bloons TD 6» имеет свыше 250,000 отзывов на Steam (97% положительных), что подтверждает её коммерческий успех. Технически игра использует 2D-графику с оптимизацией для мобильных устройств, а её интерфейс интуитивно понятен даже новичкам. Игра представлена на рисунке 1.2.

«Plants vs. Zombies» (2009, PopCap Games) — культовая игра, которая популяризировала жанр Tower Defense среди казуальных игроков. Игрок защищает дом от зомби, размещая растения с различными способностями (например, стреляющие горохом, замедляющие или взрывающиеся).

Игра выделяется ярким мультяшным стилем, юмористическим подходом и простотой механик, что делает её доступной для всех возрастов. Уровни представляют собой линейные дорожки, а ресурсы (солнце) собираются во время игры, что добавляет элемент управления экономикой. «Plants vs. Zombies» получила 93% положительных отзывов на Steam и остается популярной благодаря своей универсальности. С технической точки зрения игра использует 2D-графику и минимальные системные требования, что делает её примером оптимизации. Для разрабатываемого проекта «Plants vs. Zombies» полезна как пример создания интуитивного интерфейса и баланса между простотой и глубиной. Постер игры представлен на рисунке 1.3.



Рисунок 1.3 – Постер игры «Plants vs. Zombies»

## Постановка задачи

В рамках данного курсового проекта планируется разработка прототипа многопользовательской игры в жанре Tower Defense на игровом движке Unity для операционной системы Windows. Игра предназначена для двух игроков, которые соревнуются друг против друга по локальной сети, защищая свои базы с помощью трех типов башен и выпуская юнитов для атаки базы противника. Цель проекта — создать динамичную и сбалансированную игру, сочетающую элементы стратегии и тактического планирования, с акцентом на «мультиплеерное» взаимодействие, минималистичный дизайн и простоту управления.

В процессе разработки должны быть реализованы следующие функции:

* Поддержка «мультиплеерного» режима для двух игроков, позволяющего соревноваться в реальном времени через локальную сеть;
* Реализация трех типов башен для каждого игрока, различающихся по функциональности, для защиты базы от юнитов противника;
* Введение экономической системы, в которой игроки начинают с равного количества очков и зарабатывают дополнительные очки за уничтожение юнитов противника башнями;
* Возможность выпуска юнитов, выступающих в роли врагов для базы соперника, с одинаковыми характеристиками для обоих игроков;
* Организация игрового процесса, направленного на защиту своей базы и уничтожение базы противника, с завершением матча при разрушении одной из баз;
* Интуитивного понятный интерфейс.

Дополнительные функции, которые должны быть реализованы:

* Обеспечение простого управления размещением башен и выпуском юнитов с помощью мыши;
* Добавление настроек игры,;
* Определение победителей;
* Система «ачивок» (достижений).

Проект должен быть реализован с использованием игрового движка Unity, который обеспечит поддержку 3D-графики и базовой игровой логики, необходимой для создания «мультиплеерной» игры. Разработка программного кода будет осуществляться на языке программирования C# в среде Visual Studio 2022, что позволит эффективно реализовать игровые механики и логику взаимодействия. Для обеспечения «мультиплеерного» режима игра должна поддерживать сетевое взаимодействие через локальную сеть, гарантируя синхронизацию действий игроков, включая размещение башен, выпуск юнитов и обновление состояния баз. Визуальное оформление игры будет основано на «3D-префобов», используемых для представления башен, юнитов, баз и карты, выполненных в минималистичном стиле, чтобы снизить требования к производительности.

Звуковое сопровождение должно включать эффекты для игровых действий, таких как выстрелы и уничтожение юнитов, а также фоновую музыку в формате .mp3. Проект должен быть оптимизирован для работы на персональных компьютерах с операционной системой Windows 10 или 11, обеспечивая минимальные системные требования для доступности широкой аудитории.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

Проектирование программного средства для многопользовательской игры в жанре Tower Defense предусматривает модульную структуру, обеспечивающую гибкость, удобство разработки и поддержку сетевого взаимодействия. Программа разрабатывается на игровом движке Unity с использованием языка программирования C# и состоит из нескольких ключевых модулей, взаимодействующих через событийно-ориентированную архитектуру. Каждый модуль отвечает за определенные аспекты игрового процесса, что упрощает тестирование и потенциальное расширение функционала.

Модуль управления игрой выполняет центральную роль в координации игрового процесса. Он отвечает за инициализацию матча. Этот модуль обеспечивает завершение игры при уничтожении одной из баз, фиксируя победу одного из игроков, и координирует взаимодействие между другими модулями, передавая события, такие как изменение состояния базы. Кроме того, он управляет игровой логикой, обеспечивая корректное выполнение правил игры, таких как начисление очков и определение условий победы.

Модуль сетевого взаимодействия управляет подключением двух игроков через локальную сеть, обеспечивая стабильное и синхронизированное взаимодействие. Он отвечает за передачу данных о действиях игроков, таких как размещение башен, выпуск юнитов, а также за синхронизацию игрового состояния, включая позиции, цели и здоровье юнитов, состояние здоровья баз. Это гарантирует, что оба игрока видят одинаковую картину матча без рассинхронизации.

Модуль башен реализует логику трех типов башен, доступных каждому игроку, обеспечивая их корректное размещение на карте. Этот модуль управляет функциональностью башен, включая их автоматическую атаку юнитов противника в радиусе действия, а также расчет урона. Он отслеживает состояние каждой башни и обеспечивает взаимодействие с другими модулями, например, с модулем экономики для списания очков при установке башни. Логика башен проектируется с учетом баланса, чтобы каждый тип вносил уникальный вклад в стратегию.

Модуль юнитов отвечает за создание, движение и поведение юнитов, выпущенных игроками для атаки базы противника. Он управляет их перемещением по заданным путям на карте, взаимодействием с башнями и достижением базы соперника. Поскольку юниты обоих игроков имеют одинаковые характеристики (здоровье, скорость), модуль обеспечивает равные условия, поддерживая справедливый геймплей. Этот модуль также передает данные о состоянии юнитов модулю сетевого взаимодействия для синхронизации их позиций и здоровья между игроками.

Модуль экономики управляет системой очков, которая лежит в основе стратегического планирования. Он отслеживает начальное количество очков, одинаковое для обоих игроков, и начисляет дополнительные очки за уничтожение юнитов противника башнями. Модуль контролирует расходование очков на строительство башен или выпуск юнитов, обеспечивая корректное обновление баланса после каждого действия. Он взаимодействует с модулями башен и юнитов, чтобы проверять наличие достаточного количества очков перед выполнением действий, и с модулем интерфейса для отображения текущего состояния экономики.

Модуль интерфейса обрабатывает визуальное представление игровых элементов и взаимодействие игрока с игрой. Он отвечает за отображение ключевой информации, такой как здоровье базы, количество очков, а также за предоставление элементов управления для размещения башен, выпуска юнитов и доступа к настройкам. Модуль обеспечивает интуитивное взаимодействие через мышь, минимизируя время, необходимое для освоения управления. Он также обновляет визуальные индикаторы в реальном времени, отражая изменения в игровом состоянии.

Все модули интегрируются через компоненты Unity (MonoBehaviour), что позволяет эффективно обрабатывать события, такие как уничтожение юнита или установка башни, и обновлять игровое состояние. Событийно-ориентированная архитектура обеспечивает гибкость, позволяя модулям обмениваться данными через сообщения и реагировать на действия игроков или изменения в игре.

**2.2 Проектирование интерфейса программного средства**

Пользовательский интерфейс многопользовательской игры в жанре Tower Defense проектируется с учетом принципов минимализма, интуитивности и функциональности, чтобы обеспечить удобное взаимодействие для двух игроков, соревнующихся по локальной сети. Интерфейс разрабатывается с использованием встроенных инструментов Unity UI и включает несколько ключевых экранов, каждый из которых выполняет определенные функции, поддерживая динамичный игровой процесс и предоставляя игрокам необходимую информацию в реальном времени. Дизайн интерфейса ориентирован на простоту, четкость и оптимизацию, чтобы минимизировать отвлекающие элементы и обеспечить быструю адаптацию игроков к управлению.

Главное меню служит отправной точкой для взаимодействия с игрой. Оно содержит кнопки для создания нового матча, подключения к мачу, доступа к настройкам, достижениям и выхода из игры, а также поля для ввода IP-адреса для подключения к локальной сети. Дизайн меню выполнен в минималистичном стиле с использованием нейтральной цветовой палитры и четких шрифтов, чтобы обеспечить читаемость.

На рисунке 2.1 представлен экран главного меню.

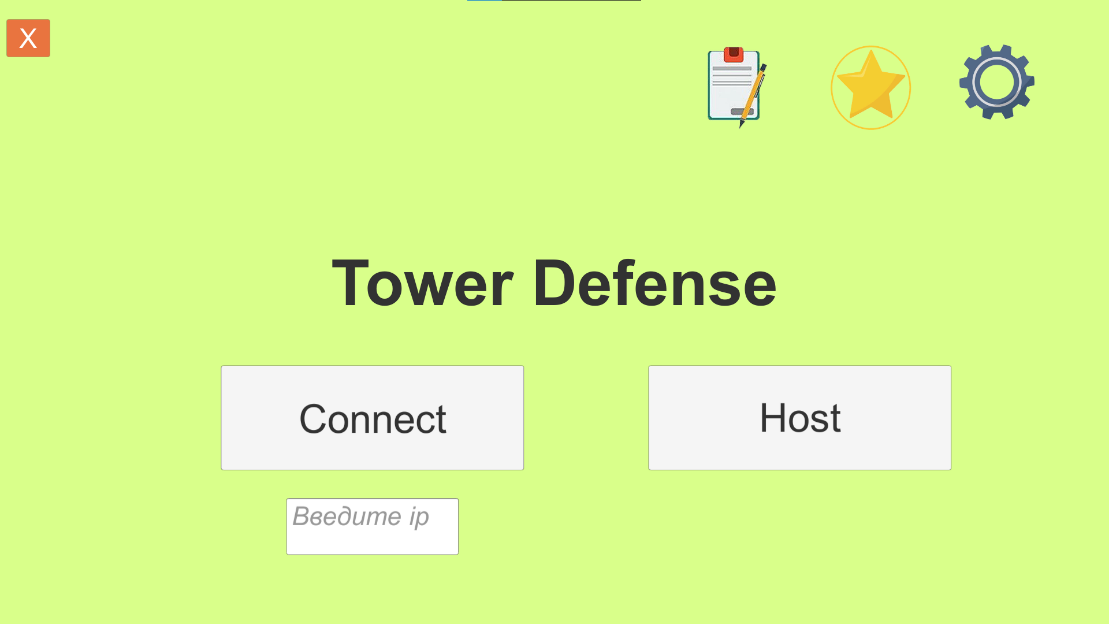


Рисунок 2.1 – Главное меню.

Игровой экран является центральным элементом интерфейса, обеспечивая игроков всей необходимой информацией во время матча. Этот экран отображает ключевые данные: здоровье собственной базы и базы противника, текущее количество очков. Панель управления, расположенная в нижней части экрана, содержит иконки для размещения башен, выпуска юнитов и просмотра их характеристик, таких как стоимость в очках. Визуальные индикаторы, такие как анимации атак башен или мигание шкалы здоровья при получении урона, проектируются для быстрого восприятия, что критично в динамичном «мультиплеерном» режиме. Элементы игрового используют контрастные цвета и простые формы. Игровой экран представлен на рисунке 2.2.

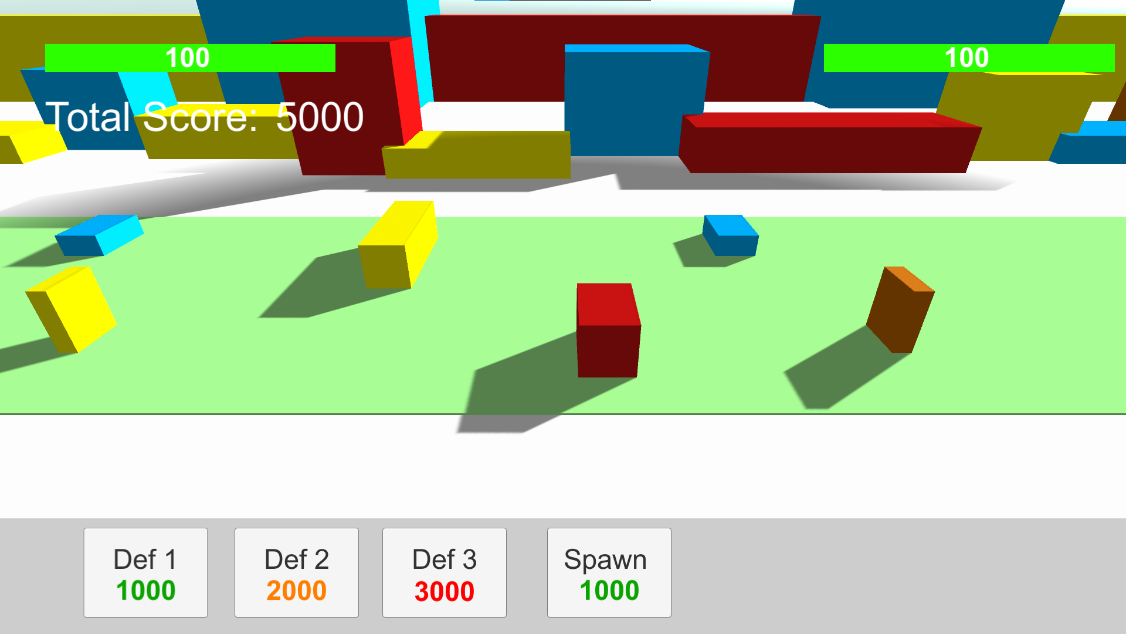


Рисунок 2.2 – Игровой экран.

Экран результатов появляется после завершения матча. Этот экран включает кнопки для начала нового матча, возврата в главное меню, а также возможность просмотра статистики в виде таблицы для анализа стратегии. Дизайн экрана результатов сохраняет минималистичную эстетику, фокусируясь на четком представлении данных. Экран настроек предоставляет игрокам возможность отключать или включать звук. Настройки доступны из главного меню и игрового экрана, чтобы игроки могли вносить изменения без прерывания матча.



Рисунок 2.3 – Экран статистики.

Все элементы интерфейса проектируются с учетом 2D-графики, используя спрайты для иконок, кнопок и индикаторов, что соответствует минималистичному стилю игры. Шрифты выбираются с высокой читаемостью, а цветовая схема избегает избыточной яркости, чтобы не отвлекать от игрового процесса. Интерфейс оптимизирован для управления мышью, что обеспечивает удобство в мультиплеере. Адаптивность интерфейса гарантирует корректное отображение на различных разрешениях экрана, поддерживаемых Windows, а интеграция с Unity UI обеспечивает отзывчивость и минимальную нагрузку на производительность.

## 2.3 Проектирование функционала программного средства

Функционал многопользовательской игры в жанре Tower Defense проектируется для обеспечения динамичного и сбалансированного геймплея, сочетающего стратегическое планирование и тактическое взаимодействие двух игроков, соревнующихся по локальной сети. Основное внимание уделяется реализации механик башен, юнитов, экономической системы и сетевой синхронизации, что позволяет создать соревновательный процесс, соответствующий требованиям жанра.

Механика башен предусматривает наличие трех типов башен для каждого игрока, отличающихся функциональностью, чтобы обеспечить разнообразие стратегий. Например, у башен разный урон и скорость атаки. Башни размещаются на заранее определенных позициях на карте, выбранной игроками, и автоматически атакуют юнитов в пределах своего радиуса действия. Логика башен включает расчет урона, выбор цели и обработку эффектов выстрела, что обеспечивает их взаимодействие с юнитами противника. Размещение башен требует затрат очков, что связывает эту механику с экономической системой.

Механика юнитов позволяет каждому игроку выпускать юниты, которые выступают в роли врагов для базы соперника. Юниты обоих игроков обладают одинаковыми характеристиками, такими как здоровье и скорость, чтобы гарантировать равные условия. Юниты движутся по заранее заданным путям на карте, взаимодействуя с башнями противника, которые наносят им урон. При достижении базы юниты наносят урон её здоровью, а их уничтожение приносит очки игроку, чьи башни их ликвидировали. Выпуск юнитов также требует очков, что заставляет игроков балансировать между атакой и защитой. Логика юнитов включает управление их движением, обработку получения и нанесения урона.

Экономическая система является ключевым элементом стратегического планирования. Каждый игрок начинает матч с одинаковым количеством очков, которое пополняется за счет уничтожения юнитов противника башнями. Очки расходуются на размещение башен или выпуск юнитов, что создает необходимость выбора между усилением обороны и атакой базы соперника. Система проектируется так, чтобы поддерживать баланс, предотвращая доминирование одного игрока за счет равных стартовых условий и фиксированных затрат на действия. Экономическая механика интегрируется с интерфейсом для отображения текущего количества очков.

Сетевая синхронизация обеспечивает согласованность игрового состояния между двумя игроками в реальном времени. Все действия, такие как размещение башни, выпуск юнита или уничтожение объекта, передаются через локальную сеть, чтобы оба игрока видели одинаковую картину матча. Синхронизация включает позиции и здоровье юнитов, позиции башен, здоровье баз. Событийно-ориентированная архитектура, реализованная через компоненты Unity (MonoBehaviour), позволяет эффективно обрабатывать сетевые события и обновлять игровое состояние.

Функционал оптимизирован для минимальных системных требований, чтобы обеспечить плавную работу на Windows, с учетом нагрузки от сетевого взаимодействия. Все механики интегрируются через модули, описанные в структуре программы, обеспечивая согласованность и гибкость для возможности дальнейшего расширения.

# 3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 3.1 Работа с сетевым взаимодействием

Одной из ключевых составляющих программного средства является организация сетевого взаимодействия между клиентами. Для обеспечения совместной работы пользователей в режиме реального времени реализована собственная система обмена данными, включающая TCP и UDP протоколы. Это позволяет синхронизировать действия, такие как создание юнитов, их движение и боевые взаимодействия.

Сервер реализован как одноранговая архитектура (peer-to-peer), где один из игроков выступает в роли хоста, а второй подключается к нему по IP-адресу. При запуске игры хост запускает TCP и UDP прослушивание, а клиент инициализирует соединение и начинает приём сообщений.

…

public void StartHost()

{

isHost = true;

string localIP = GetLocalIPAddress();

ipDisplayText.text = $"Your IP: {localIP}";

Debug.Log("Set IP text to UI: " + ipDisplayText.text);

tcpListener = new TcpListener(IPAddress.Any, tcpPort);

tcpListener.Start();

tcpThread = new Thread(() =>

{

tcpClient = tcpListener.AcceptTcpClient();

Debug.Log(" Client connected!");

stream = tcpClient.GetStream();

StartTCPListening();

remoteIP = ((IPEndPoint)tcpClient.Client.RemoteEndPoint).Address.ToString();

StartUDP(remoteIP);

SceneManager.LoadScene(gameplaySceneName);

});

tcpThread.IsBackground = true;

tcpThread.Start();

}

public void ConnectToHost()

{

isHost = false;

remoteIP = ipInputField.text.Trim();

tcpClient = new TcpClient();

try

{

tcpClient.Connect(remoteIP, tcpPort);

Debug.Log(" Connected to host!");

stream = tcpClient.GetStream();

StartTCPListening();

StartUDP(remoteIP);

SceneManager.LoadScene(gameplaySceneName);

}

catch (Exception ex)

{

Debug.LogError(" Connection failed: " + ex.Message);

}

}

…

Для критически важных данных (например, спавн юнита) используется TCP-соединение. Это обеспечивает надёжную доставку сообщений. Например, при создании юнита отправляется строка вида:

«SPAWN\_ENEMY\_GLOBAL;5;3;Enemy\_5\_3\_1234»

Это сообщение содержит индекс префаба, индекс точки спавна и уникальный идентификатор объекта. Получатель, получив сообщение, создаёт локальную копию юнита с аналогичными параметрами.

Для непрерывной синхронизации (например, положения юнитов) используется UDP. Владелец объекта каждые 0.1 секунды отправляет его текущие координаты:

«ENEMY\_POS;Enemy\_5\_3\_1234;123.45;1.04;210.67»

Сторона-получатель при получении такого сообщения плавно корректирует позицию соответствующего объекта, обеспечивая визуальную синхронность.

Обработка сообщений происходит в SynReceiver.cs. Система подписывается на события при подключении(пример для синхронизации позиции врагов):  
void OnEnable()

{

NetworkConnector.OnUDPMessageReceived += HandleUDP;

}

void OnDisable()

{

NetworkConnector.OnUDPMessageReceived -= HandleUDP;

}

void HandleUDP(string msg)

{

if (!msg.StartsWith("ENEMY\_POS;")) return;

string[] parts = msg.Split(';');

if (parts.Length < 5) return;

string id = parts[1];

if (!EnemySyncTracker.All.TryGetValue(id, out var tracker))

return;

if (tracker.isOwner) return;

float x = float.Parse(parts[2].Replace(',', '.'), CultureInfo.InvariantCulture);

float y = float.Parse(parts[3].Replace(',', '.'), CultureInfo.InvariantCulture);

float z = float.Parse(parts[4].Replace(',', '.'), CultureInfo.InvariantCulture);

tracker.SetRemotePosition(new Vector3(x, y, z));

}  
 EnemySynReciver подписываеться на событие HandleUDP и затем расшифровывает это сообщение и отправляет в модуль управления юнитов.

Каждый игрок ведёт расчёт своих юнитов локально, а данные синхронизируются периодически, что позволяет избежать полной зависимости от сетевой задержки. Дополнительно у каждой сущности указано, кто её владелец, чтобы исключить конфликты при взаимодействии (например, стрельбе по своей базе).

## 3.2 Реализация основного функционала

Каждый игрок может инициировать создание вражеских юнитов, которые спавнятся в одной из заранее определённых точек. Все юниты делятся на два направления: от базы A к базе B и наоборот. Сторона, от которой исходит атака, определяется индексом спавна и типом юнита.

public void SpawnEnemyByIndex(int globalIndex, int spawnIndex, string enemyId)

{

GameObject prefab = EnemyRegistry.Instance.GetEnemy(globalIndex);

if (prefab == null)

{

Debug.LogError($" Не найден враг с индексом {globalIndex}");

return;

}

Transform spawnPoint = spawnPoints[Mathf.Clamp(spawnIndex, 0, spawnPoints.Length - 1)];

GameObject enemy = Instantiate(prefab, spawnPoint.position, Quaternion.identity);

enemy.name = enemyId;

bool isOwner = (net.isHost && globalIndex >= 6) || (!net.isHost && globalIndex < 6);

string side = (globalIndex < 6) ? "B" : "A";

if (enemy.TryGetComponent(out Enemy\_AI ai))

{

ai.mySide = side;

ai.enabled = true;

}

if (enemy.TryGetComponent(out NavMover mover))

{

mover.enabled = true;

}

if (enemy.TryGetComponent(out EnemySyncTracker tracker))

{

tracker.enemyId = enemyId;

tracker.isOwner = isOwner;

}

Debug.Log($" Враг создан: {enemyId} на точке #{spawnIndex} | isOwner = {isOwner}");

}

Спавн осуществляется с помощью метода SpawnEnemyByIndex(), в который передаются индекс префаба, индекс точки и уникальный идентификатор юнита.

Локально каждый игрок создаёт экземпляр юнита, даже если он не владелец. У владельца активируется логика движения и атаки, а у второго игрока юнит получает периодические обновления позиции по сети.  
Все юниты, башни и защитники имеют компонент Health, который хранит текущее здоровье, реагирует на входящий урон и, при необходимости, запускает анимацию смерти и удаление объекта. Юниты стреляют с помощью компонента Weapon, который создаёт снаряд, летящий в направлении противника.

Снаряды при столкновении с целью (базой или врагом) проверяют, является ли цель допустимой (по тегу и стороне), и наносят урон только вражеской стороне. Это предотвращает "дружественный огонь" и обеспечивает корректную боевую логику.

Здоровье баз (Tower A и Tower B) отображается с помощью UI-элементов — слайдеров и числовых индикаторов. Уменьшение здоровья на стороне A или B происходит только в случае попадания по соответствующей башне. При снижении здоровья до 0 активируется окно поражения

У каждого игрока своя независимая экономика. Начальный баланс составляет 5000 монет. Монеты тратятся на размещение защитников и спавн врагов. Уничтожение вражеских юнитов приносит награду. Баланс постоянно обновляется и отображается на экране в UI.

## 3.3 Работа с моделями в игре

Каждый враг в игре представлен отдельным префабом. Всего используется 12 префабов атакующих юнитов: 6 для направления от базы B к базе A и 6 — от базы A к базе B. Префабы различаются визуально и маршрутом движения, но используют одни и те же управляющие компоненты (NavMover, Enemy\_AI, Health, Weapon и др.).

Для обеспечения различимости юнитов и сторон, все юниты визуально оформлены в соответствии с цветом своей базы: Юниты стороны A (база A) — синие, Юниты стороны B (база B) — красные.

Базы (Tower A и Tower B) также представлены 3D-моделями. Каждая база окрашена в свой цвет (синий или красный), соответствующий стороне. Это усиливает ассоциацию и помогает избежать путаницы при наблюдении за боем. На каждую из баз навешивается компонент Health, который контролирует отображение слайдера здоровья и обработку урона.

Защитные башни (defenders) имеют свои собственные модели. Все они размещаются игроками вручную и автоматически поворачиваются к ближайшей цели. При атаке из моделей стреляют снаряды, что добавляет динамичности.

Некоторые модели содержат встроенные анимации, проигрываемые при стрельбе, смерти или разрушении. Также используются визуальные эффекты (частицы, вспышки), чтобы подчеркнуть действия (например, попадание пули или уничтожение врага).

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В этом разделе описывается процесс тестирования игры «Tower Defense Game», 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 юнитов, взаимодействие башен и врагов, получение урона базой).

– Убедиться в стабильности сетевого взаимодействия

– Оценить интуитивность интерфейса.

– Проверить производительность.

– Выявить и устранить возможные ошибки влияющие на игровой процесс.

Было проведено несколько матчей, в которых были обнаружены незначительные ошибки синхронизации позиции врагов ,которые заключались в неправильной расшифровке полученных сообщений , но они были быстро исправлены и остались лишь некоторые предупреждения, связанные с реорганизацией структуры проекта. Пример предупреждения представлен на рисунке 4.1.



Рисунок 4.1 – Предупреждение об отсутствии сцены.

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

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

## **5.1 Интерфейс программного средства**

Дизайн меню выполнен в минималистичном стиле с использованием нейтральной цветовой палитры и четких шрифтов, чтобы обеспечить читаемость.. Главное меню содержит кнопки для создания нового матча(Host), подключения к матчу(Connect), доступа к настройкам(Шестерёнка), достижениям(Звезда),статистике(Блокнот с ручкой ) и выхода из игры(белый крестик в красном прямоугольнике), а также поля для ввода IP-адреса для подключения к локальной сети. Фон представляет собой монотонный экран.

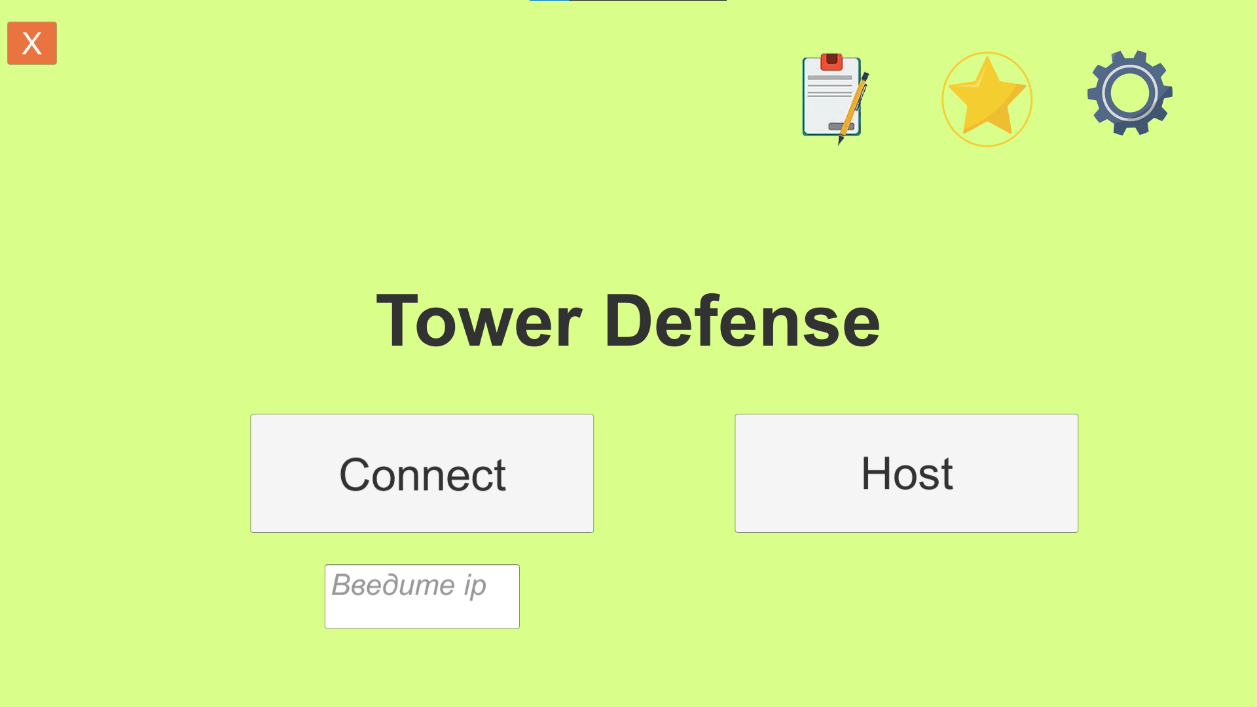


Рисунок 5.1 – Главное меню.

Меню настроек представляет из себя экран ,который появляется поверх основного и имеет два элемента типа «checkbox» для постобработки и включения/отключения звука, а также кнопку для закрытия этого экрана.

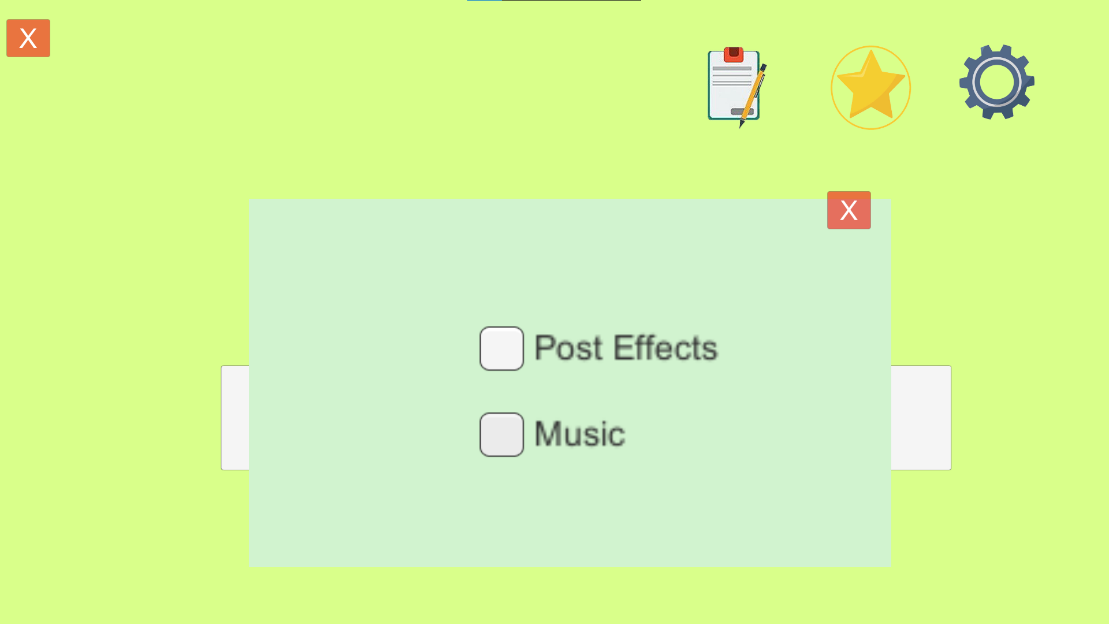


Рисунок 5.2 – Меню настроек.

Меню достижений представляет из себя экран ,который появляется поверх основного и имеет множества достижений которые достиг игрок, а также кнопку для закрытия этого экрана.



Рисунок 5.3 – Меню достижений.

Меню статистики представляет из себя экран ,который появляется поверх основного и имеет статистику по общим убийствам и очкам игрока, а также кнопку для закрытия этого экрана.



Рисунок 5.4 – Меню статистики.

Игровой экран является центральным элементом интерфейса. На этом экран отображает ключевые данные: здоровье собственной базы и базы противника, текущее количество очков(Total Score). Панель управления, расположенная в нижней части экрана, содержит кнопки для размещения башен(Def 1, Def 2, Def 3), выпуска юнитов(Spawn) и просмотра их характеристик,таких как стоимость в очках. Визуальные индикаторы, такие как анимации атак башен или уменьшение шкалы здоровья при получении урона, проектируются для быстрого восприятия, что критично в динамичном «мультиплеерном» режиме.

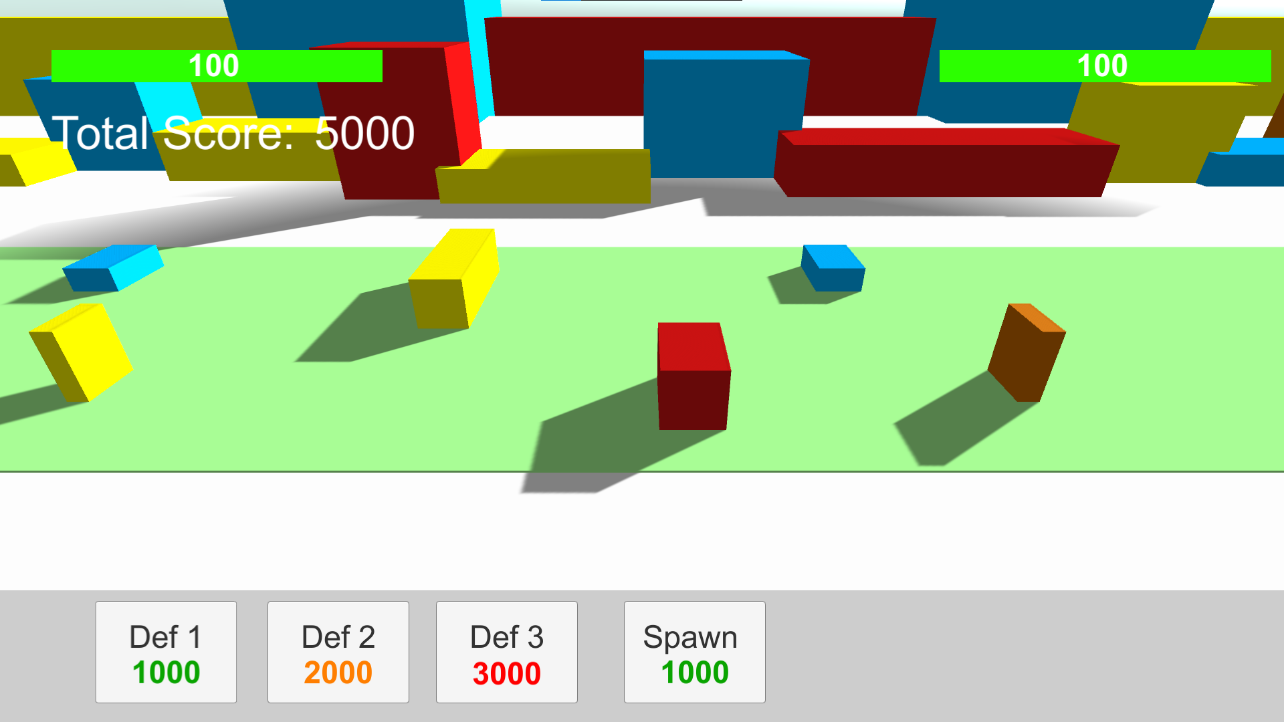


Рисунок 5.5 – Игровой экран.

Экран результатов появляется после завершения матча. Этот экран включает кнопки для начала нового матча, возврата в главное меню.



Рисунок 5.6 – Экран поражения.

## **5.2 Управление программным средством**

Для запуска игры убедитесь, что присутствует интернет-подключение и два игрока находятся в локальной сети. Чтобы запустить игру, нужно открыть TowerDefenseGame.exe файл. После запуска вы увидите меню (рисунок 5.1).

Для создания матча нужно нажать на кнопку «Host» и вам покажут ваш IP-адрес (Рисунок 5.7). Если же вы хотите подключиться, то вы вводите IP-адрес хоста в поле где написано «введите ip» и жмёте на кнопку «Connect».

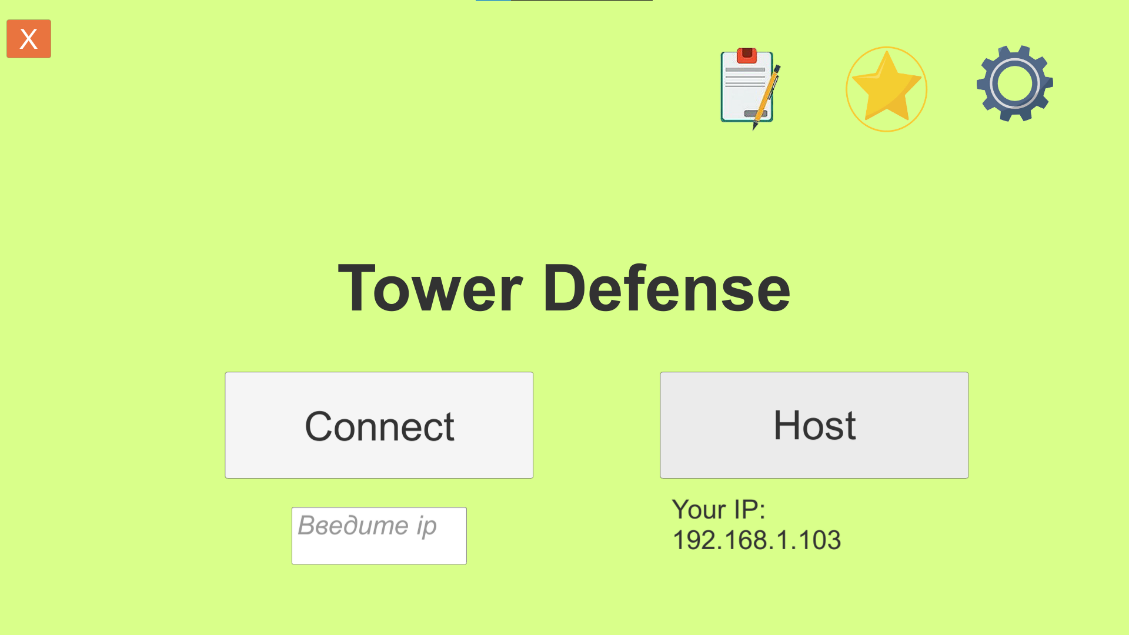


Рисунок 5.7 – Главное меню при создании матча.

Навигация по интерфейсу происходит с помощью:

– Мыши – выбор кнопок меню, полей для ввода,чекбоксов.

– Клавишой Esc (вызов меню).

В игровом процессе используются мышь для создания юнитов, упрвление камерой путём перетягивания, управление позицией башни путём перетаскивания её на доступную позицию.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была разработана 3D-мультиплеерная игра в жанре Tower Defense на движке Unity с использованием языка программирования C#. Главной задачей стало создание сетевого взаимодействия между двумя игроками без применения сторонних библиотек, что позволило глубже погрузиться в принципы передачи и обработки данных через протоколы TCP и UDP. Разработка включала реализацию спавна юнитов, передачи их позиции, синхронизации стрельбы, обработки урона и состояния здоровья баз. Были реализованы две стороны конфликта — база A и база B, каждая из которых имеет собственную команду, экономику, защиту и систему управления. Игроки могут строить защитные башни, накапливать и тратить ресурсы, а также отправлять в бой атакующих юнитов, взаимодействующих с элементами другой стороны.

Игровой процесс синхронизируется между клиентами: один из игроков отвечает за контроль юнита, передавая его позицию другому, в то время как на стороне второго игрока юнит двигается автономно, получая корректирующие данные. Это позволило достичь устойчивой передачи информации и воспроизведения боевых действий без необходимости в централизованном сервере. Были решены ключевые задачи оптимизации сетевого взаимодействия, обеспечения плавности движения юнитов и независимой работы логики защиты и нападения на каждой стороне. Игра сопровождается минималистичным визуальным оформлением, четко различающим стороны конфликта с помощью цветовой маркировки, а также предоставляет простой и интуитивно понятный пользовательский интерфейс.

Разработанный прототип подтвердил свою работоспособность при тестировании, показал стабильную синхронизацию действий игроков и продемонстрировал потенциал для расширения. В дальнейшем проект может быть дополнен поддержкой нескольких игроков, расширенной системой прокачки, новыми типами юнитов и картами, а также возможностью игры на мобильных устройствах. Полученный опыт стал ценным практическим результатом изучения сетевых технологий в игровой разработке и позволил на практике реализовать базовые принципы построения мультиплеерных игр.

# 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] Holzner S. Unity 2022 Game Development Bible, 2022. — 1000 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] Nystrom R. Game Development with Unity and C#, 2021. — 512 с.

[8]Стандарт предприятия [Электронный ресурс]. – Режим доступа: СТП\_2024.pdf.

[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.

# ПРИЛОЖЕНИЕ А. Исходный код программы

NetworkConnector.cs:

using System;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

public class NetworkConnector : MonoBehaviour

{

[Header("UI")]

public Text ipDisplayText;

public InputField ipInputField;

[Header("Scene")]

public string gameplaySceneName = "Main\_Gameplay\_Multi";

private TcpClient tcpClient;

private TcpListener tcpListener;

private NetworkStream stream;

private Thread tcpThread;

private Thread listenThread;

public static Action<string> OnMessageReceived;

private UdpClient udpSender;

private UdpClient udpReceiver;

private Thread udpListenThread;

public static Action<string> OnUDPMessageReceived;

private const int tcpPort = 5555;

private const int udpPort = 5556;

public bool isHost = false;

private string remoteIP;

public void StartHost()

{

isHost = true;

string localIP = GetLocalIPAddress();

ipDisplayText.text = $"Your IP: {localIP}";

Debug.Log("Set IP text to UI: " + ipDisplayText.text);

tcpListener = new TcpListener(IPAddress.Any, tcpPort);

tcpListener.Start();

tcpThread = new Thread(() =>

{

tcpClient = tcpListener.AcceptTcpClient();

Debug.Log(" Client connected!");

stream = tcpClient.GetStream();

StartTCPListening();

remoteIP = ((IPEndPoint)tcpClient.Client.RemoteEndPoint).Address.ToString();

StartUDP(remoteIP);

SceneManager.LoadScene(gameplaySceneName);

});

tcpThread.IsBackground = true;

tcpThread.Start();

}

public void ConnectToHost()

{

isHost = false;

remoteIP = ipInputField.text.Trim();

tcpClient = new TcpClient();

try

{

tcpClient.Connect(remoteIP, tcpPort);

Debug.Log(" Connected to host!");

stream = tcpClient.GetStream();

StartTCPListening();

StartUDP(remoteIP);

SceneManager.LoadScene(gameplaySceneName);

}

catch (Exception ex)

{

Debug.LogError(" Connection failed: " + ex.Message);

}

}

void StartTCPListening()

{

listenThread = new Thread(() =>

{

byte[] buffer = new byte[1024];

while (tcpClient.Connected)

{

try

{

int bytesRead = stream.Read(buffer, 0, buffer.Length);

if (bytesRead > 0)

{

string message = Encoding.UTF8.GetString(buffer, 0, bytesRead);

Debug.Log(" TCP Received: " + message);

OnMessageReceived?.Invoke(message);

}

}

catch (Exception ex)

{

Debug.LogError("TCP listening error: " + ex.Message);

break;

}

}

});

listenThread.IsBackground = true;

listenThread.Start();

}

public void SendMessageToPeer(string message)

{

if (tcpClient != null && tcpClient.Connected)

{

byte[] data = Encoding.UTF8.GetBytes(message);

stream.Write(data, 0, data.Length);

stream.Flush();

Debug.Log(" TCP Sent: " + message);

}

}

void StartUDP(string ip)

{

try

{

udpSender = new UdpClient();

udpSender.Connect(ip, udpPort);

udpReceiver = new UdpClient(udpPort);

udpListenThread = new Thread(() =>

{

IPEndPoint remoteEP = new IPEndPoint(IPAddress.Any, udpPort);

while (true)

{

try

{

byte[] data = udpReceiver.Receive(ref remoteEP);

string msg = Encoding.UTF8.GetString(data);

Debug.Log(" UDP Received: " + msg);

OnUDPMessageReceived?.Invoke(msg);

}

catch (Exception ex)

{

Debug.LogError("UDP listen error: " + ex.Message);

break;

}

}

});

udpListenThread.IsBackground = true;

udpListenThread.Start();

}

catch (Exception ex)

{

Debug.LogError("UDP init error: " + ex.Message);

}

}

public void SendUDP(string message)

{

if (udpSender != null)

{

byte[] data = Encoding.UTF8.GetBytes(message);

udpSender.Send(data, data.Length);

Debug.Log(" UDP Sent: " + message);

}

}

private string GetLocalIPAddress()

{

string localIP = "Not found";

var host = Dns.GetHostEntry(Dns.GetHostName());

foreach (var ip in host.AddressList)

{

if (ip.AddressFamily == AddressFamily.InterNetwork)

{

localIP = ip.ToString();

break;

}

}

return localIP;

}

void Awake()

{

if (FindObjectsOfType<NetworkConnector>().Length > 1)

{

Destroy(gameObject);

return;

}

DontDestroyOnLoad(gameObject);

}

void OnApplicationQuit()

{

try

{

listenThread?.Abort();

tcpThread?.Abort();

stream?.Close();

tcpClient?.Close();

tcpListener?.Stop();

udpListenThread?.Abort();

udpSender?.Close();

udpReceiver?.Close();

}

catch (Exception e)

{

Debug.LogWarning("Cleanup failed: " + e.Message);

}

}

}

EnemySpawnReciver.cs:

using System.Globalization;

using UnityEngine;

public class EnemySyncReceiver : MonoBehaviour

{

void OnEnable()

{

NetworkConnector.OnUDPMessageReceived += HandleUDP;

}

void OnDisable()

{

NetworkConnector.OnUDPMessageReceived -= HandleUDP;

}

void HandleUDP(string msg)

{

if (!msg.StartsWith("ENEMY\_POS;")) return;

string[] parts = msg.Split(';');

if (parts.Length < 5) return;

string id = parts[1];

if (!EnemySyncTracker.All.TryGetValue(id, out var tracker))

return;

if (tracker.isOwner) return;

float x = float.Parse(parts[2].Replace(',', '.'), CultureInfo.InvariantCulture);

float y = float.Parse(parts[3].Replace(',', '.'), CultureInfo.InvariantCulture);

float z = float.Parse(parts[4].Replace(',', '.'), CultureInfo.InvariantCulture);

tracker.SetRemotePosition(new Vector3(x, y, z));

}

}

EnemySyncTracker:  
  
using System.Globalization;

using UnityEngine;

public class EnemySyncReceiver : MonoBehaviour

{

void OnEnable()

{

NetworkConnector.OnUDPMessageReceived += HandleUDP;

}

void OnDisable()

{

NetworkConnector.OnUDPMessageReceived -= HandleUDP;

}

void HandleUDP(string msg)

{

if (!msg.StartsWith("ENEMY\_POS;")) return;

string[] parts = msg.Split(';');

if (parts.Length < 5) return;

string id = parts[1];

if (!EnemySyncTracker.All.TryGetValue(id, out var tracker))

return;

if (tracker.isOwner) return;

float x = float.Parse(parts[2].Replace(',', '.'), CultureInfo.InvariantCulture);

float y = float.Parse(parts[3].Replace(',', '.'), CultureInfo.InvariantCulture);

float z = float.Parse(parts[4].Replace(',', '.'), CultureInfo.InvariantCulture);

tracker.SetRemotePosition(new Vector3(x, y, z));

}

}

EnemyRegistry.cs:  
  
using UnityEngine;

public class EnemyRegistry : MonoBehaviour

{

public static EnemyRegistry Instance;

[Header("Общий список из 12 врагов")]

public GameObject[] allEnemies; // 0–5 = B→A, 6–11 = A→B

void Awake()

{

if (Instance == null)

{

Instance = this;

DontDestroyOnLoad(gameObject);

}

else

{

Destroy(gameObject);

}

}

public GameObject GetEnemy(int index)

{

if (index < 0 || index >= allEnemies.Length)

{

Debug.LogError($" EnemyRegistry: индекс вне диапазона ({index})");

return null;

}

return allEnemies[index];

}

}

GameManager.cs:  
  
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class GameManagerMulty : MonoBehaviour

{

[Header("Defenders")]

public GameObject[] defenders;

public int[] defendersPrice;

private List<GameObject> createdDefenders = new();

private int createdDefCounts = 0;

[Header("UI")]

public Text coinsText;

public Slider towerHealthSliderA;

public Slider towerHealthSliderB;

public Text towerHealthTextA;

public Text towerHealthTextB;

[Header("Tower Settings")]

public int towerHealthA = 100;

public int towerHealthB = 100;

public int towerDamage = 1;

[Header("UI Windows")]

public GameObject gameLostWindow;

public GameObject gameWinWindow;

[HideInInspector] public int currentDefender;

[HideInInspector] public bool canInstantiate = true;

[HideInInspector] public bool isDraging = false;

[HideInInspector] public bool dragOnViewSpace = true;

private bool purchasedCurrentItem = true;

private int localCoins = 5000;

private string mySide;

public string MySide => mySide;

void Start()

{

mySide = FindObjectOfType<NetworkConnector>().isHost ? "A" : "B";

currentDefender = 1;

purchasedCurrentItem = true;

localCoins = 5000;

coinsText.text = localCoins.ToString();

createdDefenders = new List<GameObject>();

canInstantiate = true;

UpdateTowerUI();

}

void Update()

{

CreateDefender();

EndDraging();

}

public void SetDefenderID(int id)

{

currentDefender = id;

}

public void SetDraging(bool dragState)

{

dragOnViewSpace = false;

if (localCoins >= defendersPrice[currentDefender - 1])

{

isDraging = dragState;

canInstantiate = true;

purchasedCurrentItem = false;

}

}

public void EndDraging()

{

if (Input.GetKeyUp(KeyCode.Mouse0))

{

dragOnViewSpace = true;

isDraging = false;

if (!purchasedCurrentItem)

{

localCoins -= defendersPrice[currentDefender - 1];

purchasedCurrentItem = true;

coinsText.text = localCoins.ToString();

}

}

}

void CreateDefender()

{

if (isDraging)

{

if (canInstantiate)

{

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

if (Physics.Raycast(ray, out RaycastHit hit, 1000))

{

if (hit.transform.CompareTag("Ground"))

{

canInstantiate = false;

GameObject newDef = Instantiate(defenders[currentDefender - 1], hit.point, Quaternion.identity);

createdDefenders.Add(newDef);

createdDefCounts++;

}

}

}

else

{

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

if (Physics.Raycast(ray, out RaycastHit hit, 1000))

{

if (hit.transform.CompareTag("Ground"))

{

GameObject current = createdDefenders[createdDefCounts - 1];

current.transform.position = hit.point;

Vector3 lookPos = FindClosestPoints("Center Point").position - current.transform.position;

lookPos.y = 0;

current.transform.rotation = Quaternion.Slerp(current.transform.rotation, Quaternion.LookRotation(lookPos), Time.deltaTime \* 1000);

}

}

}

}

}

public void ReduceCoins(int value)

{

localCoins -= value;

coinsText.text = localCoins.ToString();

}

public void AddCoins(int value)

{

localCoins += value;

coinsText.text = localCoins.ToString();

}

public void Game\_Lost()

{

Time.timeScale = 0;

gameLostWindow.SetActive(true);

}

public void You\_Win()

{

Time.timeScale = 0;

gameWinWindow.SetActive(true);

}

public void Reduce\_Tower\_Health(string side, int value)

{

if (side == "A")

{

towerHealthA -= value;

towerHealthSliderA.value = towerHealthA;

towerHealthTextA.text = towerHealthA.ToString();

if (towerHealthA <= 0) Game\_Lost();

}

else

{

towerHealthB -= value;

towerHealthSliderB.value = towerHealthB;

towerHealthTextB.text = towerHealthB.ToString();

if (towerHealthB <= 0) Game\_Lost();

}

}

void UpdateTowerUI()

{

towerHealthSliderA.value = towerHealthA;

towerHealthSliderB.value = towerHealthB;

towerHealthTextA.text = towerHealthA.ToString();

towerHealthTextB.text = towerHealthB.ToString();

}

Transform FindClosestPoints(string tag)

{

GameObject[] points = GameObject.FindGameObjectsWithTag(tag);

float closestDist = Mathf.Infinity;

GameObject closest = null;

Vector3 currentPos = createdDefenders[createdDefCounts - 1].transform.position;

foreach (GameObject go in points)

{

float dist = (go.transform.position - currentPos).sqrMagnitude;

if (dist < closestDist)

{

closestDist = dist;

closest = go;

}

}

return closest?.transform;

}

}

Health.cs:  
  
using System.Collections;

using UnityEngine;

public enum TargetType { Enemy, Tower, Defender }

public class Health : MonoBehaviour

{

public TargetType targetType;

public string side = "A"; // A или B

public int healthValue = 100;

public int destroyAwardedCoins = 100;

public GameObject damageParticle;

public float destroyDelay = 0;

public MeshRenderer healthColor;

private GameManagerMulty gManager;

private bool isDead;

void Start()

{

gManager = GameObject.FindObjectOfType<GameManagerMulty>();

if (healthColor)

healthColor.material.color = Color.green;

}

public void ApplyDamage(int damage)

{

healthValue -= damage;

if (healthColor)

{

if (healthValue > 0)

healthColor.transform.localScale = new Vector3(healthColor.transform.localScale.x, healthColor.transform.localScale.y, healthValue);

else

healthColor.transform.localScale = Vector3.zero;

}

if (targetType == TargetType.Enemy && healthValue <= 0)

{

if (GetComponent<CapsuleCollider>()) GetComponent<CapsuleCollider>().enabled = false;

if (GetComponent<Weapon>()) GetComponent<Weapon>().canShoot = false;

gManager.AddCoins(destroyAwardedCoins);

if (damageParticle)

Instantiate(damageParticle, transform.position, transform.rotation);

if (healthColor && healthColor.transform.parent) Destroy(healthColor.transform.parent.gameObject);

if (GetComponent<Weapon>()) GetComponent<Weapon>().shootingDelay = 999f;

if (!isDead)

{

isDead = true;

StartCoroutine(Destroy\_Delay());

}

}

if (targetType == TargetType.Defender && healthValue <= 0)

{

if (!isDead)

{

isDead = true;

StartCoroutine(Destroy\_Delay());

}

}

}

IEnumerator Destroy\_Delay()

{

yield return new WaitForSeconds(destroyDelay);

Destroy(gameObject);

}

}

Weapon.cs:  
  
using System.Collections;

using UnityEngine;

public enum ShootingMode

{

Gun, Sword

}

public class Weapon : MonoBehaviour

{

[Header("General Settings")]

public ShootingMode shootingMode = ShootingMode.Gun;

public GameObject projectile;

public Transform shootPoint;

public float force = 100f;

public float shootingDelay = 1f;

[Header("Sword Settings")]

public int SwordDamage = 1;

[Header("Additional Options")]

public GameObject secondProjectile;

public Transform secondShootPoint;

[Header("Sound Settings")]

public AudioClip fireClip;

AudioSource audioSource;

[HideInInspector] public bool canShoot = false;

GameManagerMulty gameManager;

IEnumerator Start()

{

audioSource = GetComponent<AudioSource>();

gameManager = FindObjectOfType<GameManagerMulty>();

while (true)

{

yield return new WaitForSeconds(shootingDelay);

if (canShoot)

{

if (shootingMode == ShootingMode.Gun)

{

GameObject bullet = Instantiate(projectile, shootPoint.position, shootPoint.rotation);

bullet.GetComponent<Rigidbody>().AddForce(shootPoint.forward \* force);

if (secondProjectile)

{

GameObject bullet2 = Instantiate(secondProjectile, secondShootPoint.position, secondShootPoint.rotation);

bullet2.GetComponent<Rigidbody>().AddForce(secondShootPoint.forward \* force);

}

if (audioSource && fireClip)

audioSource.PlayOneShot(fireClip);

if (TryGetComponent(out AnimationList animList) && animList.actor)

animList.actor.CrossFade(animList.fireClip);

}

if (shootingMode == ShootingMode.Sword)

{

if (TryGetComponent(out AnimationList animList) && animList.actor)

animList.actor.CrossFade(animList.fireClip);

gameManager.Reduce\_Tower\_Health(gameManager.MySide, SwordDamage);

}

}

}

}

}

Projectile.cs:  
  
using System.Collections;

using UnityEngine;

public enum ShootingMode

{

Gun, Sword

}

public class Weapon : MonoBehaviour

{

[Header("General Settings")]

public ShootingMode shootingMode = ShootingMode.Gun;

public GameObject projectile;

public Transform shootPoint;

public float force = 100f;

public float shootingDelay = 1f;

[Header("Sword Settings")]

public int SwordDamage = 1;

[Header("Additional Options")]

public GameObject secondProjectile;

public Transform secondShootPoint;

[Header("Sound Settings")]

public AudioClip fireClip;

AudioSource audioSource;

[HideInInspector] public bool canShoot = false;

GameManagerMulty gameManager;

IEnumerator Start()

{

audioSource = GetComponent<AudioSource>();

gameManager = FindObjectOfType<GameManagerMulty>();

while (true)

{

yield return new WaitForSeconds(shootingDelay);

if (canShoot)

{

if (shootingMode == ShootingMode.Gun)

{

GameObject bullet = Instantiate(projectile, shootPoint.position, shootPoint.rotation);

bullet.GetComponent<Rigidbody>().AddForce(shootPoint.forward \* force);

if (secondProjectile)

{

GameObject bullet2 = Instantiate(secondProjectile, secondShootPoint.position, secondShootPoint.rotation);

bullet2.GetComponent<Rigidbody>().AddForce(secondShootPoint.forward \* force);

}

if (audioSource && fireClip)

audioSource.PlayOneShot(fireClip);

if (TryGetComponent(out AnimationList animList) && animList.actor)

animList.actor.CrossFade(animList.fireClip);

}

if (shootingMode == ShootingMode.Sword)

{

if (TryGetComponent(out AnimationList animList) && animList.actor)

animList.actor.CrossFade(animList.fireClip);

gameManager.Reduce\_Tower\_Health(gameManager.MySide, SwordDamage);

}

}

}

}

}

CameraPan.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CameraPanMulty : MonoBehaviour

{

[Space(5)]

[Header("Camera Pan System")]

public float mouseSensitivity = 1.0f;

Vector3 lastPosition;

public float limitLeft, limitRight;

GameManagerMulty gameManager;

private void Start()

{

gameManager = GameObject.FindObjectOfType<GameManagerMulty>();

}

void Update()

{

if (Input.GetKeyDown(KeyCode.Mouse0))

{

lastPosition = Input.mousePosition;

}

if (Input.GetKey(KeyCode.Mouse0))

{

if (gameManager.dragOnViewSpace)

{

Vector3 delta = Input.mousePosition - lastPosition;

Vector3 CameralastLocation = gameObject.transform.position;

transform.Translate(delta.x \* mouseSensitivity, 0, 0);

if (transform.position.x > limitRight)

transform.position = new Vector3(CameralastLocation.x, transform.position.y, transform.position.z);

if (transform.position.x < limitLeft)

transform.position = new Vector3(CameralastLocation.x, transform.position.y, transform.position.z);

lastPosition = Input.mousePosition;

}

}

}

}

NavMover.cs:  
  
using UnityEngine;

using System.Collections.Generic;

public class NavMover : MonoBehaviour

{

public string waypointName;

public float moveSpeed = 2.5f;

public float reachDistance = 0.1f;

private List<Transform> points = new List<Transform>();

private int currentPoint = 0;

public bool reachedToEnd = false;

void Start()

{

var pathObject = GameObject.Find(waypointName);

if (pathObject == null)

{

Debug.LogError($"[NavMover] ❌ Не найден путь: {waypointName}");

return;

}

var path = pathObject.GetComponent<WaypointSystem>();

if (path == null)

{

Debug.LogError($"[NavMover] ❌ У объекта {waypointName} нет компонента WaypointSystem");

return;

}

points = path.waypoints;

if (points.Count == 0)

{

Debug.LogError("[NavMover] ❌ Нет точек пути!");

return;

}

transform.position = points[0].position;

currentPoint = 1;

}

void Update()

{

if (reachedToEnd || points.Count == 0 || currentPoint >= points.Count)

return;

Vector3 target = points[currentPoint].position;

Vector3 direction = (target - transform.position).normalized;

float distance = Vector3.Distance(transform.position, target);

transform.position += direction \* moveSpeed \* Time.deltaTime;

if (distance < reachDistance)

{

currentPoint++;

if (currentPoint >= points.Count)

{

reachedToEnd = true;

OnReachEnd();

}

}

}

void OnReachEnd()

{

Debug.Log($"[NavMover] {gameObject.name} достиг конца пути");

}

}

Enemy\_Ai:

using System.Collections;

using UnityEngine;

public class Enemy\_AI : MonoBehaviour

{

public Transform gunHead;

public float dampingSpeed = 1f;

public string mySide = "A";

public float shootingDistance = 30f;

public bool shootOnlyTower = true;

Quaternion originalRotation;

bool isActive;

Transform target;

void Start()

{

originalRotation = gunHead.rotation;

StartCoroutine(AIUpdate());

}

IEnumerator AIUpdate()

{

while (true)

{

target = FindClosestEnemy();

if (target != null && Vector3.Distance(transform.position, target.position) <= shootingDistance)

{

if (shootOnlyTower)

{

if (GetComponent<NavMover>().reachedToEnd)

{

GetComponent<Weapon>().canShoot = true;

isActive = true;

}

}

else

{

GetComponent<Weapon>().canShoot = true;

isActive = true;

}

}

else

{

GetComponent<Weapon>().canShoot = false;

isActive = false;

}

yield return new WaitForSeconds(0.3f);

}

}

void Update()

{

if (isActive && target != null)

{

Vector3 lookPos = target.position - gunHead.position;

lookPos.y = 0;

Quaternion rotation = Quaternion.LookRotation(lookPos);

gunHead.rotation = Quaternion.Slerp(gunHead.rotation, rotation, Time.deltaTime \* dampingSpeed);

}

}

GameObject closest;

Transform FindClosestEnemy()

{

string tagToSearch = mySide == "A" ? "Tower\_B" : "Tower\_A";

GameObject[] gos = GameObject.FindGameObjectsWithTag(tagToSearch);

float distance = Mathf.Infinity;

Vector3 position = transform.position;

foreach (GameObject go in gos)

{

float curDistance = (go.transform.position - position).sqrMagnitude;

if (curDistance < distance)

{

closest = go;

distance = curDistance;

}

}

return closest ? closest.transform : null;

}

}

Defender\_Ai:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Defender\_AI : MonoBehaviour

{

public Transform gunHead;

public float dampingSpeed = 10f;

public string targetTag = "Enemy";

public float shootingDistance = 30f;

[Header("Seek Animation")]

public bool playAnimationClip;

public float seekSpeed = 50f;

public float rotateAngle = 70f;

Vector3 originalRotation;

bool isActive;

Transform target;

IEnumerator Start()

{

originalRotation = gunHead.localRotation.eulerAngles;

while (true)

{

target = FindClosestEnemy();

if(target)

{

if (Vector3.Distance(transform.position, target.position) <= shootingDistance)

{

GetComponent<Weapon>().canShoot = true;

isActive = true;

}

else

{

GetComponent<Weapon>().canShoot = false;

isActive = false;

}

}

else

{

GetComponent<Weapon>().canShoot = false;

isActive = false;

}

yield return new WaitForSeconds(0.3f);

}

}

void Update()

{

if (isActive)

{

if (target)

{

Vector3 lookPos = target.position - gunHead.position;

lookPos.y = 0;

Quaternion rotation = Quaternion.LookRotation(lookPos);

gunHead.rotation = Quaternion.Slerp(gunHead.rotation, rotation, Time.deltaTime \* dampingSpeed);

}

}

else

{

if(playAnimationClip)

{

if (GetComponent<AnimationList>().actor)

{

GetComponent<AnimationList>().actor.CrossFade(GetComponent<AnimationList>().seekClip);

}

}

else

{

gunHead.localRotation = Quaternion.Euler(originalRotation.x, Mathf.PingPong(Time.time \* seekSpeed, rotateAngle \* 2) - rotateAngle, 1f);

}

}

}

GameObject closest;

Transform FindClosestEnemy()

{

GameObject[] gos;

gos = GameObject.FindGameObjectsWithTag(targetTag);

if (gos.Length == 0)

return null;

float distance = Mathf.Infinity;

Vector3 position = transform.position;

foreach (GameObject go in gos)

{

Vector3 diff = go.transform.position - position;

float curDistance = diff.sqrMagnitude;

if (curDistance < distance)

{

closest = go;

distance = curDistance;

}

}

return closest.transform;

}

}