МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(Наименование кафедры, центра, департамента)

09.03.03 Прикладная информатика
(код и наименование направления подготовки / специальности)

«Разработка программного обеспечения»
(направленность (профиль) / специализация)

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка многопользовательской онлайн игры на платформе Unity»

Обучающийся А.С. Елисеев _____
(Инициалы Фамилия) (личная подпись)

Руководитель Н.Н. Казаченок
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант М.В. Дайнеко
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

## Аннотация

Название выпускной квалификационной работы: «Разработка многопользовательской онлайн игры на платформе Unity».

Выпускная квалификационная работа (ВКР) состоит из введения, трех глав, 21 рисунка, 3 таблиц, заключения, списка литературы из 20 источников, включая иностранные.

Целью данной ВКР является разработка системы онлайн игры, решающей основные архитектурные и инфраструктурные проблемы.

Объектом ВКР является проект многопользовательской онлайн игры.

Предметом ВКР являются средства и принципы разработки многопользовательских онлайн игр.

Ключевым вопросом ВКР является изучение наиболее эффективных методов разработки многопользовательских онлайн игр.

Бакалаврскую работу можно разделить на несколько логически связанных частей: постановку задачи, разработку и оценку эффективности разработанной системы.

В первой части описано значение программного обеспечения, основные функции, аналоги и требования.

Во второй части описаны процессы проектирования системы, выбора технологий и реализации основных функций.

Третья часть состоит из методологии синтеза и биологических испытаний.

В заключение хотелось бы подчеркнуть, что по результатам тестирования была получена система многопользовательской онлайн игры, которая может быть использована для разработки сложных онлайн-игр.

Тем не менее, необходимо получить больше данных о работе системы в реальных условиях.

Работа представляет интерес для широкого круга читателей, интересующихся разработкой многопользовательских онлайн игр.

## Annotation

The title of the graduation work is Development of a multiplayer online game on the Unity platform.

The graduation work consists of an introduction, three parts, 21 figures, 3 tables, a conclusion, and a list of 20 references including foreign sources.

The aim of this graduation work is to develop the basis of a multiplayer online game system that systematically solves various architectural and infrastructural problems.

The object of the graduation work is the project of a multiplayer online game.

The subject of the graduation work is the tools and principles of multiplayer online games development.

The key issue of the graduation work is to study most effective methods of multiplayer online games development.

The graduation work may be divided into several logically connected parts which are problem statement, development and testing.

The first part describes in details the meaning of the software, its main functions, analogues and requirements.

The second part describes the processes of system design, technology selection and implementation of basic functions.

The third part consists of testing and evaluating the functionality of the system.

In conclusion, we'd like to stress that, based on the testing results, a multiplayer online game system was developed, which can be used to create complex online games.

Nevertheless, additional data on the system's real-world operation is required.

The work is of interest for wide circle of readers interested in the development of multiplayer online games.

## Оглавление

# Введение

Игры всегда были неотъемлемой частью жизни людей. Потребность в играх обусловлена особенностями функций человеческого мозга, отвечающих за обучение и моделирование ситуаций, упорядочивание и упрощение информации в хаотичном и сложном мире.

В современном обществе немаловажную роль играет сфера видеоигр. Наибольшую популярность со временем получили многопользовательские онлайн игры, это не трудно объяснить – человек по своей природе существо социальное.

Объектом исследования данной бакалаврской работы является проект многопользовательской онлайн игры.

Предметом исследования бакалаврской работы является использование различных инструментов разработки многопользовательских онлайн игр, способствующих более быстрому, качественному и эффективному развитию проекта разработки.

Целью работы является разработка основы системы сложной многопользовательской онлайн игры, систематически решающей различные архитектурные и инфраструктурные проблемы.

Для достижения поставленной цели требуется решить ряд следующих задач:

- исследование предметной области;
- проектирования общей архитектуры игры;
- разработка основной клиент-серверной части игры;
- разработка инфраструктурного веб-приложения;
- проведение тестирования и оценки полученного результата.

Несмотря на стремительное развитие многопользовательских онлайн игр, в наше время разработчики по-прежнему сталкиваются с большими трудностями в процессе их разработки, особенно если раньше они не имели за плечами соответственного опыта. Данная бакалаврская работа отражает

различные тонкости и базовые аспекты разработки сложных многопользовательских онлайн игр, что представляет её практическую значимость как для начинающих разработчиков, так и для более опытных.

Выпускная квалификационная работа содержит введение, три главы, заключение.

В первой главе проведен анализ процесса разработки многопользовательских онлайн игр.

Во второй главе описан процесс разработки системы многопользовательской онлайн игры.

В третьей главе проведена оценка эффективности разработанной системы многопользовательской онлайн игры.

Выпускная квалификационная работа состоит из 48 страниц и содержит 21 рисунок, 3 таблицы, 20 источников.

# Глава 1 Постановка задачи на разработку программного обеспечения

## 1.1 Значение программного обеспечения

Игра как явление существовала на протяжении всего развития человеческого рода и это несомненно делает её в той или иной степени частью жизни каждого человека. Игры всегда были способом обучения, самопознания и получения удовольствия, что подтверждается высоким уровнем потребности в них в наше время и на любых других исторических промежутках.

В наше время особую популярность получили видеоигры – игры, разработанные для цифровых устройств, таких как персональные компьютеры, смартфоны, консоли и так далее. Высокий спрос на видеоигры логически привёл к развитию сферы их разработки, так появилась профессия разработчика игр.

До появления интернета первые видеоигры зачастую были представлены в однопользовательском формате, реже – в многопользовательском формате с возможностью совместной игры нескольким игрокам на одном устройстве.

Так как человек по своей природе является существом социальным, видеоигры активно начали развиваться в сторону многопользовательского игрового опыта. Так с появлением высокоскоростного межконтинентального интернета стали усложняться технологии проектирования многопользовательских онлайн игры. [12]

В публичном доступе имеется относительно малое количество материалов о разработке сложных многопользовательских онлайн игр, несмотря на их стремительное развитие. Поэтому ответственностью каждого разработчика является обмен знаниями, а также систематизация и документирование шаблонов, методологий, мета-принципов, помогающих

заниматься разработкой более эффективно и безопасно для разработчиков, их клиентов и пользователей разработанного программного продукта.

Наиболее распространенные проблемы, возникающие при разработке многопользовательских онлайн игры:

- проблема повторяющегося кода;
- проблема разделения клиентской и серверной логики;
- проблема нечестных игроков;
- проблема бесшовной авторитарной синхронизации;
- проблема хранения данных игроков;
- проблема сетевой инфраструктуры.

Успешное решение всех вышеуказанных проблем на начальных этапах разработки многопользовательской онлайн игры значительно повышает эффективность разработки игры в дальнейшем, снижает риски технической нестабильности проекта, а также значительно снижает стоимость последующей разработки.

## 1.2 Основные функции

Клиент и сервер имеют общую игровую логику, которая помещается в отдельный модуль, к ней можно отнести следующие составляющие:

- логика перемещения персонажей;
- логика здоровья персонажа;
- логика оружия, используемого персонажами;
- логика игрового времени суток;
- логика игровой локации;
- логика перемещения транспортных средств.

Также и клиент, и сервер одинаково используют инструменты сетевого взаимодействия для общения друг с другом, к ним относятся:

- логика сетевых узлов;
- логика создания сетевого сообщения (пакета);

– логика гарантированной и негарантированной отправки сообщений;
– логика поллинга поступающих сообщений.

Помимо клиентской и серверной части игры, необходимо разработать инфраструктурное веб-приложение для поддержки многопользовательской онлайн игры. Веб-приложение должно поддерживать функции создания аккаунта, авторизации, просмотра доступных игровых серверов.

### 1.3 Обзор существующих решений и их анализ

Одни из самых популярных библиотек для разработки многопользовательских онлайн игр на базе Unity это Photon и UNet.

Photon Unity Networking (PUN) – это мощный инструмент для создания многопользовательских онлайн игр и приложений на платформе Unity. Photon предоставляет разработчикам удобный и эффективный способ синхронизации объектов и взаимодействия между игроками в реальном времени. Photon позволяет разработчикам создавать сложные многопользовательские игровые сценарии, используя простые и интуитивно понятные инструменты. Использование Photon в совокупности с инструментами Unity позволяет создавать онлайн-игры различных типов, от простых казуальных до сложных массовых многопользовательских онлайн игр (MMO).

Unity Networking (UNet) – инструмент сетевой синхронизации Unity объектов, по умолчанию интегрированный в среду разработки Unity. UNet предоставляет разработчикам высокоуровневые методы синхронизации игровых объектов, подходящие под большинство возможных игровых сценариев. Высокоуровневая реализация методов UNet позволяет разработчикам эффективно работать над созданием игровых механик и многопользовательских игровых сценариев, не вдаваясь в подробности низкоуровеной реализации сетевого взаимодействия.

Таблица 1 – Плюсы и минусы аналогов

| | Плюсы | Минусы |
|---|---|---|
| Photon | Простая интеграция с Unity | Высокая стоимость серверов при масштабировании |
| | Бесплатный тариф | Зависимость от серверов Photon |
| | Облачный хостинг | Проблемы с синхронизацией в динамичных играх |
| UNet | Интегрирован в Unity | Устаревший код |
| | Гибкая архитектура | Низкая производительность |
| | Бесплатный доступ | Ограниченный функционал |

Таким образом разработка собственной системы многопользовательской онлайн игры имеет смысл ввиду значительных недостатков рассматриваемых аналогов.

### 1.4 Требования к функционалу и интерфейсу приложения

В таблице 2 отображена классификация требований к функционалу и пользовательскому интерфейсу многопользовательской онлайн игры.

Таблица 2 – Классификация требований

| Требование | Статус | Полезность | Риск |
|---|---|---|---|
| Функциональные требования | | | |
| Возможность получения данных о доступных игровых серверах | Принято | Высокая | Имеется риск медленного выполнения запроса при большом количестве данных о игровых серверах, необходимо использование пагинации |
| Возможность получения данных игрового профиля | Принято | Высокая | Имеется риск медленного выполнения запросов при большом количестве игровых профилей в базе данных, необходима оптимизация частоты отправки запросов |

Продолжение таблицы 2

| | | | |
|---|---|---|---|
| Возможность отправки heartbeat-запроса со стороны сервера для обновления данных о его состоянии | Принято | Высокая | Имеется риск создания большой нагрузки на сервер веб-приложения |
| Удобство использования | | | |
| Возможность гостевого входа в информационную систему | Принято | Средняя | Злоупотребление гостевым входом со стороны нежелательных пользователей для обхода игровой блокировки или получения преимущества |
| Возможность авторизации через сторонние сервисы | Отложено | Средняя | Злоупотребление мульти-сервисной авторизацией для создания нескольких игровых аккаунтов и получения преимущества в игре |
| Возможность использования веб-приложения напрямую из клиента игры | Принято | Высокая | Возможны трудности внедрения методов SDK для работы с веб-приложением в клиент игры |
| Надежность | | | |
| Шифрование паролей при сохранении в базу данных | Принято | Средняя | - |
| Валидация данных полученных от пользователей в результате запроса | Принято | Высокая | - |
| Аутентификации для игровых серверов, с целью исключения постороннего вмешательства в работу сервера | Принято | Средняя | - |
| Производительность | | | |
| Поддержка асинхронного выполнения запросов | Принято | Высокая | Конфликты между параллельными запросами |
| Оптимизация частоты отправки запросов | Принято | Средняя | Сильное ограничение частоты отправки запросов может привести к негативному пользовательскому опыту |
| Поддерживаемость | | | |
| Поддержка развертки в Docker контейнере | Принято | Высокая | Возможна потеря данных при изменении конфигурации контейнера |
| Поддержка развёртки приложения на удаленном сервере с системой Linux | Принято | Высокая | Возможна некорректная работа определенных фреймворков в среде Linux |

Таким образом, были подробно описаны требования, касающиеся функциональности, удобства использования, надежности, производительности и поддерживаемости разрабатываемой системы, а также выявлены возможные ограничения.

Вывод по первой главе

В рамках главы 1 было выявлено значение программного обеспечения, как для разработчиков многопользовательских онлайн игр, так и для конечных пользователей программного продукта.

Был составлен список основных функций системы многопользовательской онлайн игры, описывающий, как функции относящиеся непосредственно к игровым механикам, так и инфраструктурные функции системы.

Также был проведен анализ аналогов среди инструментов организации сетевого взаимодействия, в ходе которого были выявлены их основные плюсы и минусы.

Был составлен полный список требований к системе многопользовательской онлайн игр.

Таким образом, разработка системы многопользовательской онлайн игры является актуальной задачей, направленной на повышение эффективности разработки сложных видеоигр с сетевым взаимодействием.

# Глава 2 Процесс разработки многопользовательской онлайн игры

## 2.1 Проектирование системы многопользовательской онлайн игры

Для повышения уровня эффективности разработки разработчики используют различные принципы проектирования и написания программного кода, которые были получены исторически методом проб и ошибок на протяжении всей истории программирования и системного анализа. Данные принципы называются мета-принципами.

Самые базовые мета-принципы не имеют жесткой спецификации и существуют в формате максимально простых и обобщенных указаний к действию или не действию в определенных ситуациях. Несмотря на свою простоту и прямолинейность мета-принципы остаются важными и обязательными для понимания каждого разработчика. Также простейшие мета-принципы подобно аксиомам лежат в основе более сложных принципов, имеющих более конкретную спецификацию. Так, например, мета-принцип DRY (Don’t repeat yourself) лежит в основе принципа SRP (Single-responsibility principle), который в свою очередь является одним из принципов спецификации SOLID.

Основные мета-принципы:

- DRY (Don’t repeat yourself),
- KISS (Keep it simple stupid),
- YAGNI (You ain’t gonna need it),
- SOLID.

Принцип DRY (Don’t repeat yourself) – это принцип, согласно которому ранее написанный код не должен повторяться в системе повторно. Повторяющийся код – большая проблема при разработке больших и сложных программных систем. Человеческий фактор играют главную роль при разработке программного обеспечения. Большие системы очень сложно проектировать, в случае если они находятся в состоянии хаоса. Хаос в системе

создаётся также и за счёт дублирования кода. При дублировании одного кода в нескольких местах у разработчика появляется необходимость в отслеживании и актуализации состояния кода во всех местах, в которых код расположен. В такой ситуации как раз-таки и вмешивается человеческий фактор, очень легко в процессе изменения повторяющегося кода пропустить одну важную деталь в одном из мест где этот код находиться, в таком случае это может привести к критическим ошибкам, а также может создать сложность в поиске причины ошибки и устранении её последствий. Для минимизации повторяющегося кода современные языки программирования предлагают множество инструментов, например, таких как методы классов и сами классы, функции, различные модификаторы доступа и так далее.

Принцип KISS (Keep it simple stupid) – это принцип, призывающий разработчиков к максимальному упрощению системы. Идеальный код будет максимально прямолинеен и понятен для каждого разработчика, который будет его читать, вне зависимости от уровня подготовки разработчика. Код не должен быть извилистым и хитросплетенным, не должен делать ничего лишнего. Очень часто разработчикам приходиться работать в команде, разбираться в чужом коде или в своём старом коде. Легко придумать сложное хитросплетенное решение и с умным видом реализовать его в системе, но как показывает практика в дальнейшем при разборе кода другими разработчиками, или даже самостоятельном разборе своего кода, бывает тяжело понять ход мысли автора и полученное решение только замедляет разработку.

Принцип YAGNI (You ain’t gonna need it) – это принцип, призывающий разработчиков не разрабатывать лишнюю функциональность, которая не нужна на текущий момент. Очень часто у разработчиков есть желание сыграть на опережение и попытаться спрогнозировать и реализовать следующую логическую функциональность системы, новая функциональность же накладывает определенные неоправданные ограничения на систему и замедляет и усложняет её разработку.

Помимо таких мета-принципов как DRY, KISS и YAGNI существуют так называемые принципы объектно-ориентированного программирования, также известные как принципы ООП.

На принципах объектно-ориентированного программирования основывается большинство широко используемых, популярных в наше время языков программирования. Это не удивительно, ведь объектное представление программного кода остаётся самым простым для понимания любому человеку и разработчику. Помимо этого, объектно-ориентированная методология плотно перекликается с системным анализом и заимствует из него многие полезные практики системного проектирования, появившиеся ещё задолго до появления программирования и компьютеров. [8]

Всего выделяют четыре принципа объектно-ориентированного программирования:

- инкапсуляция,
- наследование,
- полиморфизм,
- абстракция.

Инкапсуляция – это принцип объектно-ориентированного программирования, подразумевающий сокрытие определенного механизма внутри закрытой от других частей системы среды, с одновременным созданием четкой границы взаимодействия с этим объектом в виде так называемого интерфейса. Суть принципа отражена в его названии: логика определенного системного механизма «инкапсулируется», помещается в некую «капсулу», обособленную от других частей системы. Этот принцип широко используется не только в программировании, но и во всех систематизированных сферах человеческой жизни. Подобное сокрытие механизмов в некие «капсулы» облегчает разработку и поддержку системы, помогает сокрыть неважную для других частей системы сложную реализацию и оставить на поверхности простой интерфейс взаимодействия, за счёт чего

выстраивается понятная логическая иерархия системы, с последующим разделением на подсистемы.

Наследование – это принцип объектно-ориентированного программирования, суть которого в избегании повторяющегося кода за счёт расширения уже имеющейся реализованной логики. Очень часто системные объекты могут иметь схожие черты, механизм наследования помогает выделять общее поведение и данные в родительские классы и образовывать от них соответствующие дочерние подтипы, включающие дополнительную логику и при этом полностью наследующие родительские особенности.

Полиморфизм – это принцип объектно-ориентированного программирования, суть которого в возможности повторного использования одной логической цепочки для разных типов объектов. Данный принцип также подразумевает борьбу с повторяющимся кодом, о вреде которого уже было написано ранее в абзаце про принцип проектирования DRY. Выделяют следующие типы полиморфизма:

- полиморфизм подтипов,
- параметрический полиморфизм,
- ad-hoc полиморфизм.

Абстракция – это принцип объектно-ориентированного программирования, суть которого в представлении системы в виде разных уровней абстрагирования её деталей. Уровни абстракции тесно связаны с инкапсуляцией и также способствуют построению иерархии системы, что в свою очередь способствует повышению эффективности проектирования и разработки системы.

Также к принципам объектно-ориентированного программирования относятся принципы SOLID. SOLID – это аббревиатура, каждая буква которой подразумевает определенный принцип. Принципы SOLID впервые были выведены Робертом Мартином в его книге о чистом коде, на данный момент существуют разные интерпретации этих принципов, но их основная суть осталась неизменной.

Всего SOLID содержит пять принципов:

– SRP, Single Responsibility Principle, принцип единственной ответственности;
– OCP, Open Closed Principle, принцип открытости-закрытости;
– LSP, Liskov Substitution Principle, принцип подстановки Барбары Лисков;
– ISP, Interface Segregation Principle, принцип разделения интерфейсов;
– DIP, Dependency Inversion Principle, принцип инверсии зависимостей.

Из-за большого количества разночтений принципы остаются сложными для понимания многим разработчикам, помимо этого принципы SOLID в своём изначальном издании в книге Роберта Мартина были представлены в очень обобщенном и отдаленном от реального программирования виде. Это повлекло за собой шквал критики в адрес изначального прародителя SOLID и появление новых интерпретаций и версий ранее разработанных принципов. [1], [6]

Стоит отметить что прямолинейное, непродуманное использование принципов разработки на практике очень часто приводит к усложнению системы и понижению уровня эффективности процесса её разработки. При проектировании программной системы в первую очередь нужно основываться на практический результат, полученный в результате применения каких-либо принципов, если применение принципа приводит к отрицательному результату в определенной ситуации – лучше отказаться от использования данного принципа проектирования.

Одна из проблем при реализации многопользовательской онлайн игры с авторитарностью на стороне сервера – это проблема разделения проектов клиента и сервера, а также дублирования общего между сервером и клиентом кода.

Дублирующийся код является плохой практикой при разработке любого программного обеспечения с большой кодовой базой. Повторяющийся код требует особого внимания, при изменении такого кода в одном месте надо

также следить чтобы изменения были применены в других дублирующих местах. В больших проектах при росте количества повторяющегося кода разработчикам требуется больше внимания уделять поддержке такого рода кода, что нередко приводит к созданию в участках повторяющегося кода ошибок и снижению скорости разработки. [1], [6]

Существует два способа разделения проектов разработки клиента и сервера:

- разделение на уровне архитектуры в одном проекте;
- создание отдельных проектов для клиента и сервера.

При разделении логики клиента и логики сервера на уровне архитектуры в одном проекте разработчикам удаётся до минимума сократить количество повторяющегося кода, логику достаточно реализовать один раз и после этого её можно использовать и в клиентской логике игры, и в серверной. К плюсам такого подхода можно отнести удобство и скорость разработки, малая вероятность возникновения багов. К минусам же можно отнести сложность реализации, необходимость постоянно менеджмента ресурсов/кода, попадающих в проект на этапе сборки.

При создании двух отдельных проектов для клиента и сервера, разработчики создают два отдельных проекта для клиента и сервера, что можно сравнить с разделением при разработке веб-приложений (Frontend и Backend). К плюсам такого подхода можно отнести возможность использования разных платформ и языков программирования для двух проектов, а также при использовании такого способа разработчикам не обязательно уделять много внимания менеджменту ресурсов на этапе сборки каждого из проекта, так как изначально каждый проект содержит только необходимые ему ресурсы. [8]. [11], [13]

Чтобы выбрать тип разделения для проекта надо исходить из требования к проекту. Например, если игра не предполагает авторитарности на основе физики и физические состояния никак не контролируются сервером, в таком случае выгоднее разделить игру на два проекта, для сервера игры можно

использовать консольное приложение с подключением к базе данных, а для клиента игровой движок. В таком случае повторяющегося кода практически не должно быть, и выбор такого способа будет обоснован своей эффективностью.

Если же игра предполагает авторитарность на уровне физического состояния игроков и полный контроль этого состояния на стороне сервера, тогда следует выбрать способ разделения на уровне архитектуры одного проекта. При авторитарности сервера для качественной и комфортной для игрока синхронизации физическая логика должна быть детерменированной, любой ввод со стороны пользователя/игрока должен приводить к определенному точно установленному состоянию. Так при осуществлении ввода игроком на клиенте и изменении физического состояния, ввод отправляется на сервер, и приводит к такому же изменению состояния на сервере.

Если физическая логика не дублируется, храниться в одном проекте и равноправно используется в серверном игровом цикле и в клиентской игровом цикле, в таком случае получается избежать ошибок, приводящих к недетерменированности, значительно упростить разработку и повысить качество сетевой синхронизации. [15], [16], [17]

Исходя из требований проекта в полной авторитарности, а также авторитарности физических состояний, необходимо использовать способ разделения клиент-серверной логики на уровне архитектуры проекта.

Учтём данные особенности и составим соответствующие UML диаграммы, представляющие архитектуру проекта. На рисунке 1 представлена диаграмма вариантов использования, на рисунке 2 представлена диаграмма ключевых классов.


Рисунок 1 – Диаграмма вариантов использования

На диаграмме изображены два ключевых актора основной части игры – сервер и клиент. К клиенту относятся прецеденты подключения к серверу, считывания ввода игрока и отправки его на сервер, прогнозирования и отката на основе серверного состояния, отображения удаленных игроков на основе полученной от сервера информации. К серверу относятся прецеденты проверки подключения игроков, применения ввода игроков, отправки данных об удаленных игроках.

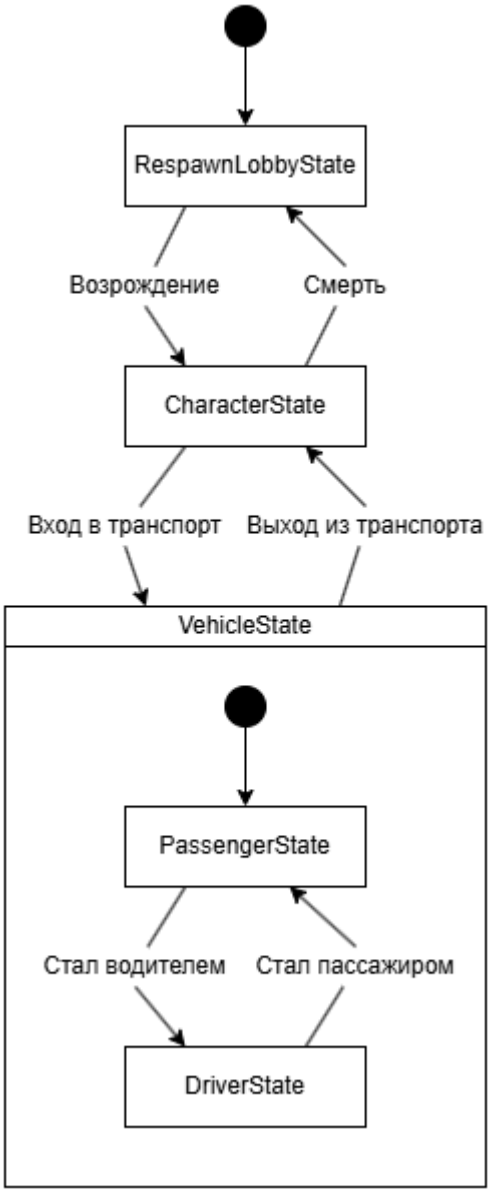



Рисунок 2 – Диаграмма состояний игрока

На рисунке 2 изображен диаграмма возможных состояний игрока. Для полной детерменированности состояния игрока должны дублироваться, как на сервере, так и на клиенте. На диаграмме изображены взаимосвязи состояний возрождения, управления персонажем и управления транспортом. [6]

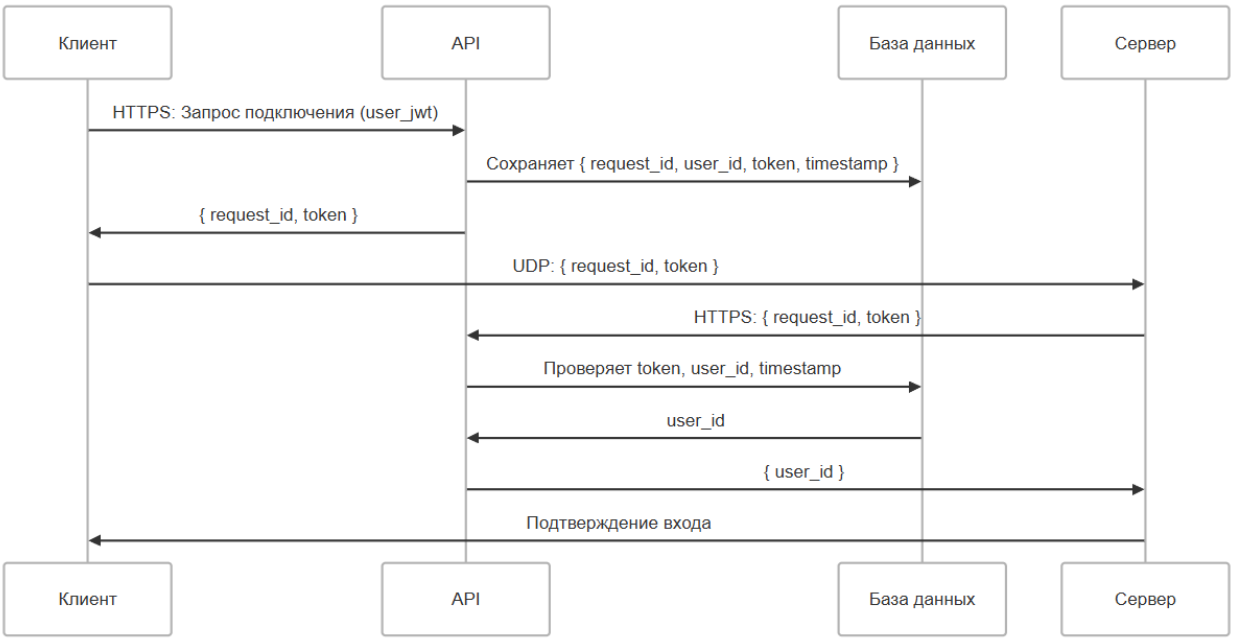



Рисунок 3 – Диаграмма последовательности

На рисунке 3 изображена диаграмма последовательности, отображающая взаимодействие всех компонентов сложной системы многопользовательской онлайн игры на примере механизма безопасного подключения игрока к игровому серверу. Так можно видеть, что клиент и сервер взаимодействуют друг с другом путем небезопасного соединения, а для безопасного обмена используют инфраструктурное веб-приложение (API). Путём генерации уникального токена подключения осуществляется проверка и аутентификация пользователя при подключении к игровому серверу, после чего токен инвалидизируется. [3], [4], [5]

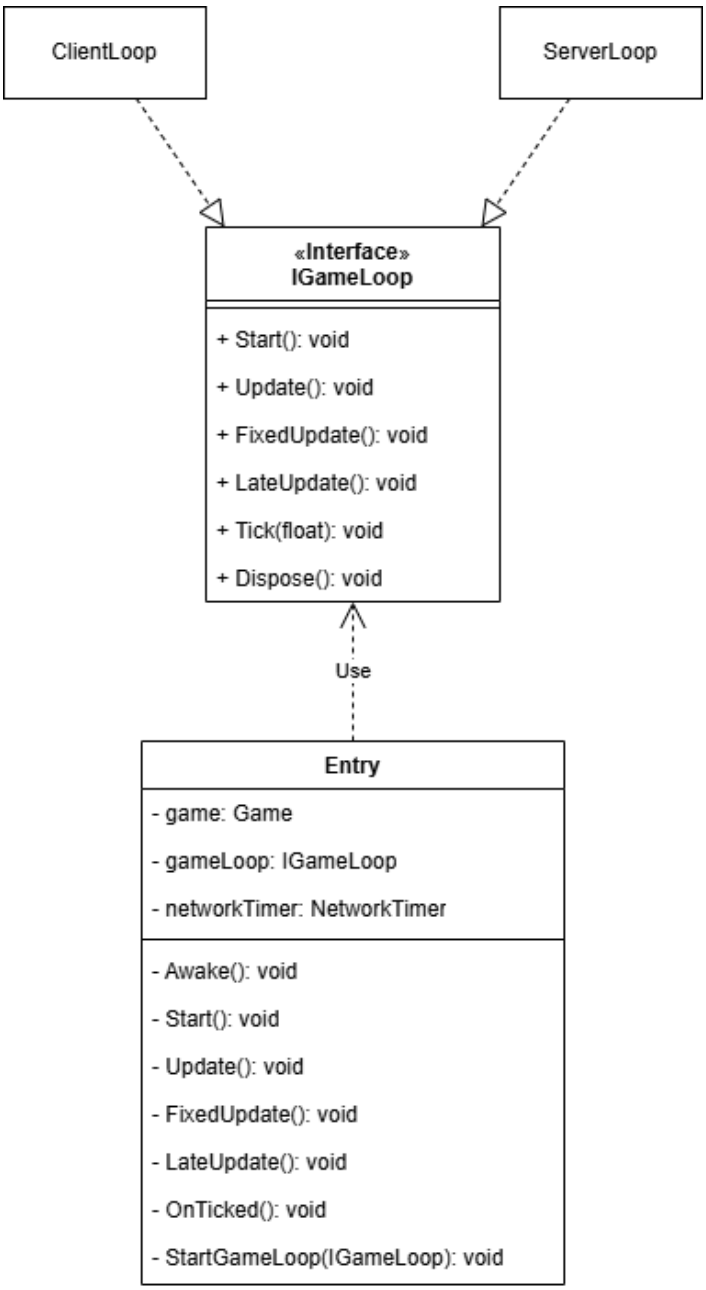



Рисунок 4 – Диаграмма ключевых классов проекта

Ключевым моментом на диаграмме является интерфейс IGameLoop, выделяющий общие методы, которые есть и у клиентского и у серверного циклов. Метод Update, FixedUpdate, LateUpdate являются встроенными методами Unity, метод Start вызывается при запуске цикла, метод Tick вызывается при сетевом тике и нужен для детерминированных вычислений на клиенте и сервере. Класс Entry наследуется от MonoBehaviour и является входом в программу, его ответственность – осуществления выбора цикла и его

запуск, инъекция необходимых зависимостей, обеспечение взаимозаменяемости игровых циклов. Из интерфейса IGameLoop реализуются классы игровых циклов ClientLoop и ServerLoop. [1], [2]

## 2.2 Выбор технологий и инструментов для разработки

В современной индустрии разработки видеоигр ключевую роль играют специализированные программные инструменты, включающие игровые движки, разнообразные языки программирования и базовые платформы, определяющие архитектуру создаваемых проектов.

При реализации данного исследовательского проекта в рамках бакалаврской работы, направленного на создание прототипа многопользовательской онлайн-игры, основным инструментарием была выбрана платформа Unity в сочетании с языком программирования C#, что обусловлено их широкими функциональными возможностями и технологической зрелостью.

Unity представляет собой комплексную среду разработки игровых приложений, созданную компанией Unity Technologies. Данная платформа обеспечивает полноценную поддержку основных операционных систем: Microsoft Windows, Mac OS X и Linux. Существенно расширяя традиционные подходы к игровому программированию, Unity выступает как многофункциональный инструмент, позволяющий создавать не только игровые проекты, но и разнообразные графические приложения и веб-решения. Особое внимание разработчики платформы уделили созданию интуитивно понятного интерфейса, что значительно упрощает и ускоряет процесс разработки интерактивных приложений, особенно для мобильных устройств.

Платформа Unity обладает рядом существенных преимуществ:

– инновационная архитектура игрового движка, обеспечивающая оптимальное сочетание производительности и функциональности;

- использование современного и широко распространенного языка программирования C# в качестве основы среды разработки;
- крупнейшее в индустрии сообщество разработчиков, что обеспечивает широкие возможности для обмена опытом и получения профессиональной поддержки;
- обширная база учебных материалов и документации, способствующая эффективному освоению платформы как начинающими, так и опытными разработчиками;
- постоянное совершенствование платформы через систему регулярных обновлений, включающих как исправление выявленных ошибок, так и внедрение новых функциональных возможностей.

Выбор Unity в качестве основной платформы разработки обусловлен не только перечисленными преимуществами, но и возможностью создания кроссплатформенных приложений, что существенно расширяет потенциальную аудиторию разрабатываемого продукта. Интеграция с языком C# обеспечивает доступ к мощным инструментам программирования и богатой экосистеме .NET, что позволяет реализовывать сложную игровую логику и создавать высокопроизводительные приложения. [19], [20]

На представленной схеме (рисунок 5) отображена комплексная структурная организация игровой платформы Unity, демонстрирующая взаимосвязи между ключевыми компонентами среды разработки. Схематическое изображение наглядно показывает, как различные модули движка взаимодействуют друг с другом, образуя целостную систему для создания игровых приложений. Данная архитектурная диаграмма позволяет понять основные принципы работы платформы и оценить масштаб взаимосвязей между ее составными частями.

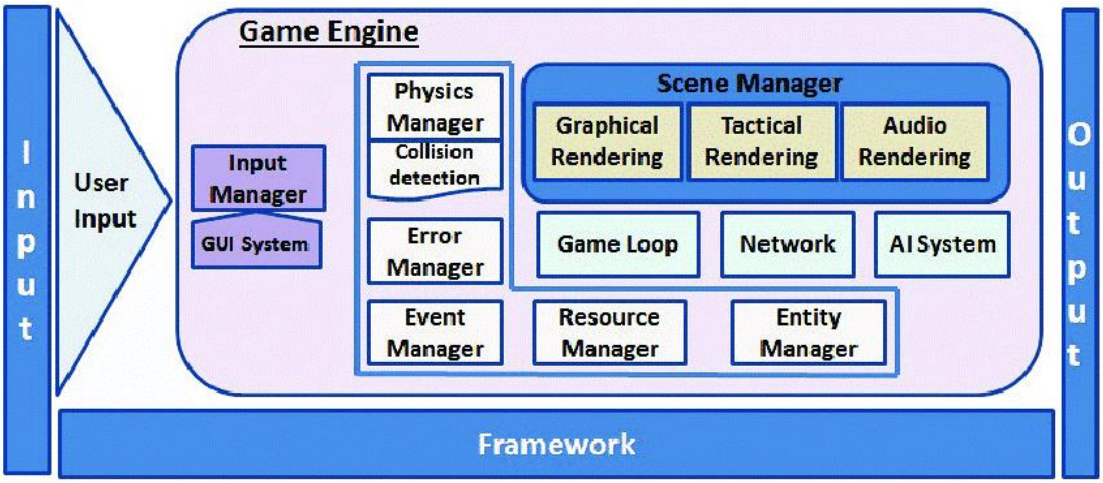



Рисунок 5 – Архитектура Unity

В качестве фундаментального решения для организации сетевого взаимодействия была использована открытая библиотека Riptide Networking, которая подверглась существенной модификации и настройке в соответствии со спецификой проекта и современными стандартами разработки сетевых многопользовательских игр. Важным преимуществом данной библиотеки является поддержка основных протоколов передачи данных – UDP и TCP, причем особого внимания заслуживает реализованная система подтверждений доставки сообщений (Acknowledgements, ACKS) при работе через UDP. [14]

Такой подход обеспечивает надежную передачу критически важных данных, одновременно позволяя существенно снизить нагрузку на каналы связи между игровыми клиентами и сервером. Дополнительными достоинствами Riptide Networking являются высокое качество программного кода, соответствующее современным стандартам разработки, а также полная доступность исходного кода для внесения необходимых изменений и адаптации под конкретные задачи проекта.

Для разработки веб-приложений существует множество технологий и языков программирования. К самым популярным языкам программирования можно отнести такие языки как Python, C# и Java. Сами по себе языки программирования не содержат функционала для разработки веб-приложений,

чтобы разработать веб-приложение разработчики используют специализированные библиотеки и фреймворки, содержащие функционал для разработки веб-приложений. Для языка программирования Java к таким можно отнести Spring Framework, Spark, JUnit и другие. Для разработки веб-приложений на C# широко используется платформа ASP.NET в совокупности с Entity Framework для работы с базой данной, Minimal API для декларации конечных точек веб-приложения, FluentValidation для валидации данных полученных от пользователей приложения, Swagger для отладки и тестирования работоспособности конечных точек (эндпоинтов) веб-приложения, JwtBearer для создания аутентификационных и авторизационных JTW-токенов, BCrypt для шифрования пользовательских паролей.

Платформа ASP.NET является активно развивающейся платформой на данных момент. Помимо этого, ASP.NET является одной из самых надежных платформ для разработки веб-приложений на данный момент и подходит как для быстрой разработки простых веб-приложений с небольшим количеством эндпоинтов, так и для разработки более сложных систем.

Язык программирования C# также является активно развивающимся и надежным языком программирования. Также C# предоставляет много возможностей для сокращения количества boilerplate-кода, написания чистого кода и создания чистой архитектуры приложения. [7], [18]

Исходя из вышеуказанных плюсов для разработки инфраструктурного веб-приложения был выбран язык программирования C# и платформа ASP.NET.

Для поддержки многопользовательской онлайн игры веб-приложение должно предоставлять функции для получения списка доступных серверов, аутентификации и авторизации пользователей, методы получения данных профиля игрока, методы для управления профилем игрока, а также конечные точки для администрирования.

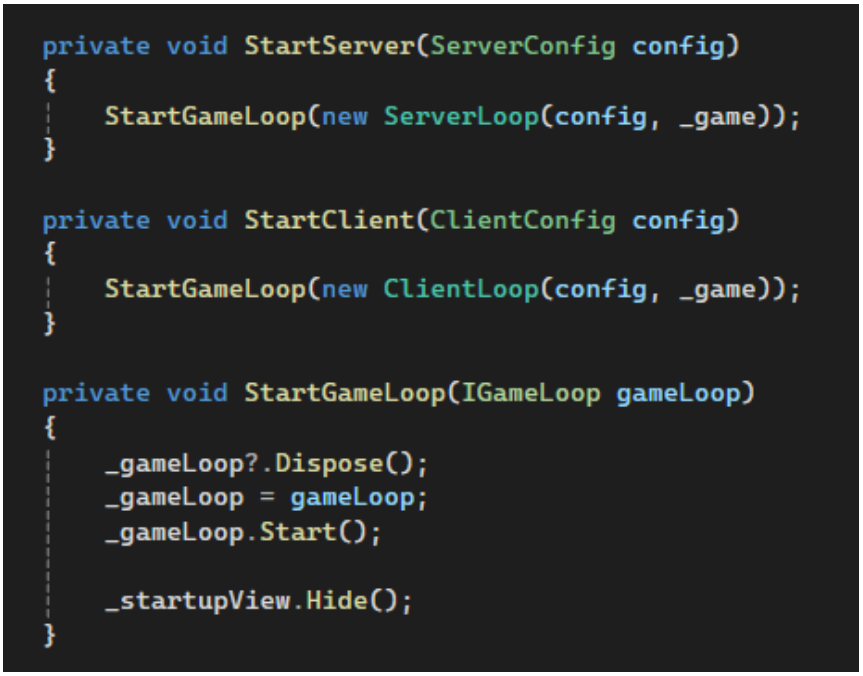
### 2.3 Реализация функциональных требований

Согласно описанной ранее архитектуре реализуем модуль Game с основными игровыми механиками. Модуль Game должен содержать конфигурацию игровых механик, фабрику предметов, фабрику персонажей, фабрику выпадающих предметов (фабрика лута), фабрику наземного транспорта, загрузчик мира, модуль считывания ввода игрока, модуль управления камерой игрока и модуль управления игровым интерфейсом. На рисунке 6 изображен получившийся код модуля Game.

```
public class Game : MonoBehaviour
{
    // Конфигурация основных игровых механик
    public GameConfig Config => _config;

    // Фабрика предметов
    public ItemFactory ItemFactory => _itemFactory;

    // Фабрика персонажей
    public CharacterFactory CharacterFactory => _characterFactory;

    // Фабрика выпадающих предметов (фабрика лута)
    public LootItemFactory LootFactory => _lootFactory;

    // Фабрика назменого транспорта
    public VehicleFactory VehicleFactory => _vehicleFactory;

    // Загрузчик мира
    public WorldLoader WorldLoader => _worldLoader;

    // Модуль считывания ввода игрока
    public PlayerInputReader PlayerInputReader => _playerInputReader;

    // Модуль камеры игрока
    public PlayerCamera PlayerCamera => _playerCamera;

    // Модуль управления игровым интерфейсом
    public GameUI UI => _uI;
```

Рисунок 6 – Модуль Game

Далее реализуем точку входа в программу – модуль Entry. Модуль Entry осуществляет базовую инициализацию и начинает выполнения цикла многопользовательской онлайн игры. На рисунке 7 изображен код модуля Entry.



```
private void StartServer(ServerConfig config)
{
    StartGameLoop(new ServerLoop(config, _game));
}

private void StartClient(ClientConfig config)
{
    StartGameLoop(new ClientLoop(config, _game));
}

private void StartGameLoop(IGameLoop gameLoop)
{
    _gameLoop?.Dispose();
    _gameLoop = gameLoop;
    _gameLoop.Start();

    _startupView.Hide();
}
```

Рисунок 7 – Модуль Entry

Далее реализуем модуль ClientLoop, содержащий обработчики сообщений от сервера и основную логику клиентского игрового цикла – логику управления персонажем с авторитарной синхронизацией с серверов, логику синхронизации удаленных персонажей других игроков, логику синхронизации наземного транспорта и логику синхронизации выпадающих предметов. На рисунке 8 изображен код инициализации класса модуля ClientLoop.

```
public ClientLoop(ClientConfig config, Game game)
{
    _config = config;
    _client = new Client();
    _game = game;
    _worldLoader = _game.WorldLoader;
    _gameUI = _game.UI;
    _mapView = _gameUI.MapView;
    _respawnView = _gameUI.RespawnView;

    // Словарь удаленных персонажей других игроков
    _remotePlayers = new Dictionary<int, RemotePlayer>();

    // Словарь наземного транспорта
    _vehicles = new Dictionary<int, IRemoteVehicle>();

    // Модуль управления персонажем игрока
    _player = new Player(_client, _game, _vehicles);

    // Модуль синхронизации удаленных персонажей других игроков
    _remotePlayerLoop = new RemotePlayerLoop(_game, _remotePlayers, _vehicles);

    // Модуль синхронизации наземного транспорта
    _vehicleLoop = new VehicleLoop(_game.VehicleFactory, _vehicles, _remotePlayers);

    // Модуль синхронизации выпадающих предметов
    _lootLoop = new LootLoop(_game.LootFactory);

    // Обработчики сообщений от сервера
    _messageHandlers = CreateMessageHandlers();
}
```

Рисунок 8 – Инициализация класса модуля ClientLoop

На рисунке 9 изображен код инициализации обработчиков сообщений от сервера.

```
private Dictionary<ClientMessageId, Action<Message>> CreateMessageHandlers()
{
    return new Dictionary<ClientMessageId, Action<Message>>()
    {
        { ClientMessageId.SpawnPlayer, _player.Spawn },
        { ClientMessageId.HandlePlayerState, _player.HandleState },
        { ClientMessageId.SpawnRemotePlayer, _remotePlayerLoop.Spawn },
        { ClientMessageId.DespawnRemotePlayer, _remotePlayerLoop.Despawn },
        { ClientMessageId.UpdateRemotePlayerState, _remotePlayerLoop.UpdateState },
        { ClientMessageId.UpdateRemotePlayerItem, _remotePlayerLoop.UpdateItem },
        { ClientMessageId.TeleportPlayer, _player.Teleport },
        { ClientMessageId.TeleportRemotePlayer, _remotePlayerLoop.Teleport },
        { ClientMessageId.UpdateHealth, _player.UpdateHealth },
        { ClientMessageId.HandleDamageResult, _player.HandleDamageResult },
        { ClientMessageId.UpdateInventory, _player.UpdateInventory },
        { ClientMessageId.SpawnLoot, _lootLoop.Spawn },
        { ClientMessageId.UpdateLoot, _lootLoop.Update },
        { ClientMessageId.DespawnLoot, _lootLoop.Despawn },
        { ClientMessageId.SpawnVehicle, _vehicleLoop.Spawn },
        { ClientMessageId.UpdateVehicleState, _vehicleLoop.UpdateState },
        { ClientMessageId.DespawnVehicle, _vehicleLoop.Despawn },
        { ClientMessageId.EnterVehicle, _player.EnterVehicle },
        { ClientMessageId.GetUp, _player.GetUp },
        { ClientMessageId.RemotePlayerEnterVehicle, _remotePlayerLoop.EnterVehicle },
        { ClientMessageId.RemotePlayerGetUp, _remotePlayerLoop.GetUp },
        { ClientMessageId.HandleVehicleState, _player.HandleVehicleState },
        { ClientMessageId.EnterRespawnLobby, _player.EnterRespawnLobby },
        { ClientMessageId.UpdateRemotePlayerHealth, _remotePlayerLoop.UpdateHealth },
        { ClientMessageId.OpenStorage, _player.OpenStorage },
        { ClientMessageId.CloseStorage, _player.CloseStorage },
        { ClientMessageId.UpdateStorage, _player.UpdateStorage },
        { ClientMessageId.UpdateStats, _player.UpdateStats },
    };
}
```

Рисунок 9 – Инициализация обработчиков сообщений от сервера

Далее реализуем модуль ServerLoop, содержащий обработчики сообщений от клиентов и основную логику серверного игрового цикла – цикл обработки игроков, цикл обработки выпадающих предметов, цикл обработки точек появления выпадающих предметов, цикл обработки наземного транспорта. На рисунке 10 изображен код инициализации класса модуля ServerLoop.

```
public ServerLoop(ServerConfig config, Game game)
{
    Dictionary<Connection, Player> players = new Dictionary<Connection, Player>();

    _config = config;
    _server = new Server();
    _game = game;
    _worldLoader = _game.WorldLoader;

    // Цикл обработки игроков
    _playerLoop = new PlayerLoop(players, _game);

    // Цикл обработки выпадающих предметов
    _lootLoop = new LootLoop(players, _game.LootFactory);

    // Цикл обработки точек появления выпадающих предметов
    _lootSpawnLoop = new LootSpawnLoop(_game.Config.LootSpawnConfig, _lootLoop);

    // Цикл обработки наземного транспорта
    _vehicleLoop = new VehicleLoop(_game.VehicleFactory, _game.ItemFactory);

    // Обработчики сообщений от клиентов
    _messageHandlers = CreateMessageHandlers();
}
```

Рисунок 10 – Инициализация класса модуля ServerLoop

На рисунке 11 изображен код инициализации обработчиков сообщений от клиентов.

```
private Dictionary<ServerMessageId, Action<Connection, Message>> CreateMessageHandlers()
{
    return new Dictionary<ServerMessageId, Action<Connection, Message>>()
    {
        { ServerMessageId.HandlePlayerInput, _playerLoop.HandleInput },
        { ServerMessageId.PickupItem, _lootLoop.Pickup },
        { ServerMessageId.EnterVehicle, _vehicleLoop.Enter },
        { ServerMessageId.GetUp, _playerLoop.GetUp },
        { ServerMessageId.HandleVehicleInput, _playerLoop.HandleVehicleInput },
        { ServerMessageId.Respawn, _playerLoop.Respawn },
        { ServerMessageId.TransferItem, _playerLoop.TransferItem },
        { ServerMessageId.OpenVehicleStorage, _vehicleLoop.OpenStorage },
        { ServerMessageId.CloseStorage, _playerLoop.CloseStorage },
    };
}
```

Рисунок 11 – Инициализация обработчиков сообщений от клиентов

В соответствии с заданными функциональными требованиями создадим программную реализацию на языке C#, обеспечивающую взаимодействие с серверной частью приложения. Центральным элементом разработанного решения выступает класс Program, представленный на рисунке 12. Данный класс является отправной точкой работы всего приложения и содержит ключевые компоненты для его корректного функционирования. В рамках этого класса выполняется последовательная настройка и запуск необходимых программных служб, определяются основные параметры конфигурации системы, а также производится активация всех компонентов приложения в требуемом порядке. Такая структурная организация обеспечивает правильную последовательность инициализации и гарантирует стабильную работу программного комплекса.

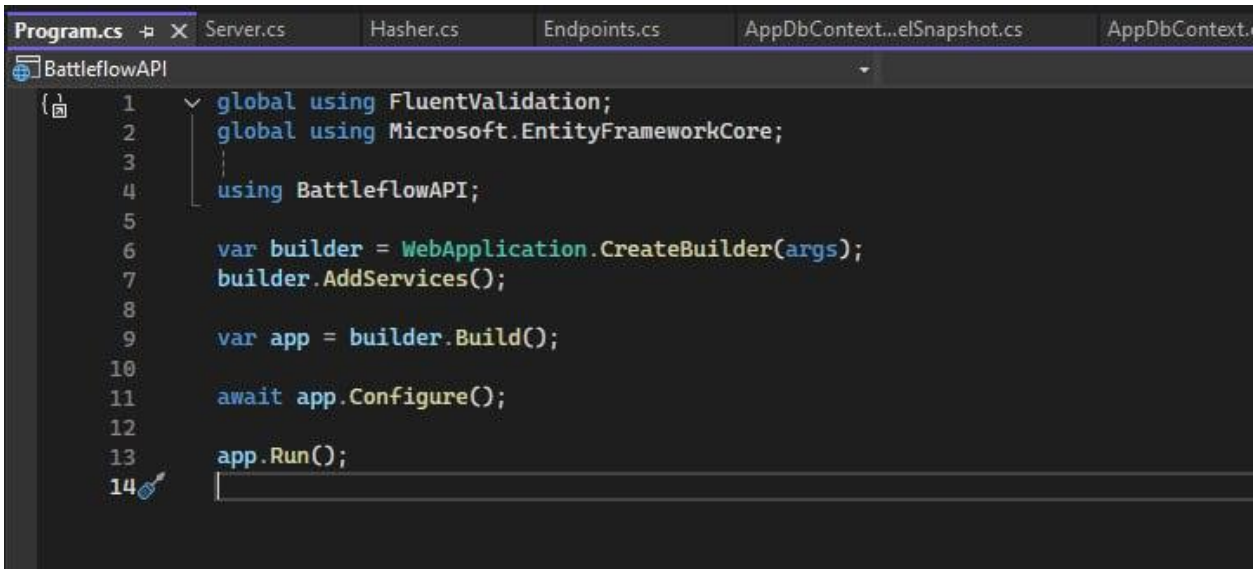



Рисунок 12 – Класс Program

Детальное рассмотрение программного кода, представленного на рисунке 13, позволяет увидеть реализацию класса ConfigureApp, отвечающего за первоначальную настройку веб-приложения. В рамках данного класса выполняется комплекс важнейших подготовительных операций, необходимых для корректного запуска системы. Среди ключевых задач можно выделить

настройку безопасного перенаправления через протокол HTTPS, определение и регистрацию всех доступных сетевых маршрутов приложения, а также работу с базой данных. Особое внимание уделяется процессу подготовки базы данных, включающему её начальную инициализацию и, при необходимости, выполнение автоматического обновления структуры через механизм миграций. Такой подход обеспечивает надежную основу для дальнейшей работы всех компонентов системы.

```
public static async Task Configure(this WebApplication app)
{
    if (app.Environment.IsDevelopment())
    {
        app.UseSwagger();
        app.UseSwaggerUI();
    }

    app.UseHttpsRedirection();

    app.UseCors((builder) =>
    {
        builder.AllowAnyOrigin();
        builder.AllowAnyHeader();
        builder.AllowAnyMethod();
    });

    app.MapEndpoints();

    await app.EnsureDatabaseCreated();
}

private static async Task EnsureDatabaseCreated(this WebApplication app)
{
    using var scope = app.Services.CreateScope();
    var db = scope.ServiceProvider.GetRequiredService<AppDbContext>();

    await db.Database.EnsureCreatedAsync();
    await db.Database.MigrateAsync();
}
```

Рисунок 13 – Класс ConfigureApp

Программный код класса ConfigureServices, представленный на рисунке 14, демонстрирует процесс настройки и подключения основных программных компонентов, обеспечивающих работоспособность приложения. В данном классе производится первоначальная настройка набора служебных модулей, каждый из которых отвечает за определенный аспект функционирования

системы. Особое внимание уделяется настройке службы, обеспечивающей взаимодействие с базой данных через специализированную библиотеку объектно-реляционного отображения. Также осуществляется конфигурация системы безопасности, включающей в себя службу создания и обработки защищенных токенов доступа. Помимо этого, выполняется инициализация и настройка дополнительных вспомогательных служб, необходимых для полноценной работы всех компонентов приложения. [9], [10]

```
public static void AddServices(this WebApplicationBuilder builder)
{
    builder.AddSwagger();
    builder.AddDatabase();
    builder.AddPasswordHasher();
    builder.AddAuthentication();

    builder.Services.AddCors();
    builder.Services.AddValidatorsFromAssembly(typeof(ConfigureServices).Assembly);
}

private static void AddSwagger(this WebApplicationBuilder builder)
{
    builder.Services.AddEndpointsApiExplorer();
    builder.Services.AddSwaggerGen(options =>
    {
        options.CustomSchemaIds(type => type.FullName?.Replace('+', '.'));
        options.InferSecuritySchemes();
    });
}

private static void AddDatabase(this WebApplicationBuilder builder)
{
    builder.Services.AddDbContext<AppDbContext>(options =>...);
}

private static void AddPasswordHasher(this WebApplicationBuilder builder)
{
    builder.Services.AddTransient<Hasher>();
}

private static void AddAuthentication(this WebApplicationBuilder builder)
{
    builder.Services.AddAuthentication().AddJwtBearer(options =>...);

    builder.Services.Configure<JwtOptions>(builder.Configuration.GetSection("Jwt"));
    builder.Services.AddTransient<Jwt>();

    builder.Services.AddAuthorization();
}
```

Рисунок 14 – Код класса ConfigureServices

В коде класса Endpoints, который можно наблюдать на рисунке 15, реализована система распределения и настройки сетевых маршрутов веб-

приложения. Данный класс выступает центральным элементом маршрутизации, определяя все доступные точки взаимодействия с системой. В его структуре выделяются три основных функциональных блока: первый обеспечивает процессы проверки подлинности пользователей, второй отвечает за все операции, связанные с управлением и обработкой данных игровых профилей, а третий предоставляет функционал для работы со списком активных игровых серверов. Такая организация позволяет структурировать и систематизировать все возможные пути обращения к приложению, обеспечивая четкое разграничение различных аспектов взаимодействия пользователей с системой.

```
public static void MapEndpoints(this WebApplication app)
{
    var endpoints = app.MapGroup("");

    endpoints.MapAuthenticationEndpoints();
    endpoints.MapPlayersEndpoints();
    endpoints.MapServersEndpoints();
}

private static void MapAuthenticationEndpoints(this IEndpointRouteBuilder app)
{
    var endpoints = app.MapGroup("/auth");

    endpoints.MapPublicGroup()
        .MapEndpoint<Signup>()
        .MapEndpoint<Login>();
}

private static void MapPlayersEndpoints(this IEndpointRouteBuilder app)
{
    var endpoints = app.MapGroup("/players");

    endpoints.MapAuthorizedGroup()
        .MapEndpoint<GetPlayerProfile>();
}

private static void MapServersEndpoints(this IEndpointRouteBuilder app)
{
    var endpoints = app.MapGroup("/servers");

    endpoints.MapPublicGroup()
        .MapEndpoint<GetServers>()
        .MapEndpoint<UpdateServer>();
}
```

Рисунок 15 – Код класса Endpoints

Разработанное веб-приложение обеспечивает полноценную поддержку сетевой многопользовательской игры, предоставляя широкий спектр необходимых функциональных возможностей. В состав реализованного функционала входит система отображения активных игровых серверов, комплексный механизм проверки подлинности и предоставления прав доступа пользователям, а также набор методов для работы с информацией об игроках. Важной составляющей приложения является возможность просмотра и изменения пользовательских профилей, включая различные игровые показатели и достижения. Дополнительно реализован административный интерфейс, позволяющий осуществлять управление системой и мониторинг игрового процесса на серверах. Все эти компоненты тесно взаимодействуют между собой, образуя целостную систему обслуживания многопользовательской игровой платформы.

Вывод по второй главе

В главе 2 были рассмотрены различные аспекты разработки многопользовательских онлайн игр. На начальном этапе было проведено проектирование архитектуры системы, что позволило выделить её основные составляющие и компоненты, которые необходимо реализовать.

Далее была разработана основная часть игровой системы, включающая серверную и клиентскую составляющие. Ключевым инструментом при создании основного функционала выступила современная среда разработки Unity в сочетании с языком программирования C#. Использование единой проектной среды Unity для обеих частей приложения существенно оптимизировало процесс разработки, исключив необходимость дублирования программного кода.

Таким образом, вторая глава охватывает весь цикл разработки системы многопользовательской онлайн игры, начиная с проектирования и заканчивая написанием программного кода.

# Глава 3 Оценка эффективности разработанной системы

## 3.1 Тестирование и отладка приложения

Для проверки работоспособности игры необходимо запустить сервер и клиент игры. Сервер и клиент запускаются через стартовое отладочное меню, изображенное на рисунке 16.

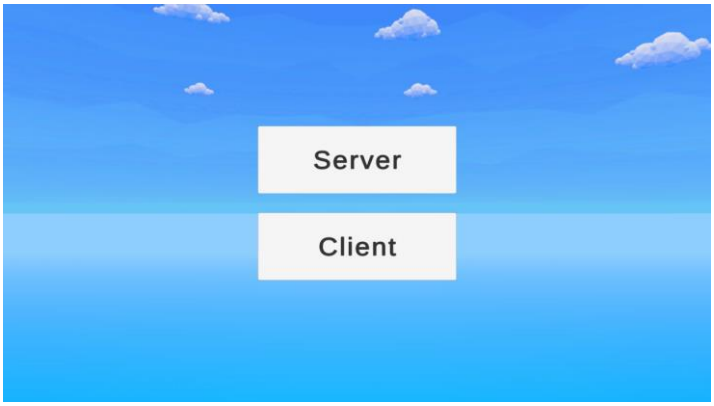



Рисунок 16 – Отладочное меню запуска

На рисунке 17 изображено окно игры запущенной в режиме сервера, с отображением логов игры.


Рисунок 17 – Сервер игры

При успешном запуске серверной части необходимо провести тестирование клиентского приложения. Для этого открывается дополнительное игровое окно в клиентском режиме, через которое выполняется соединение с уже работающим сервером. После установления соединения пользователю становится доступен весь набор реализованных игровых возможностей и функций. Как показано на рисунке 18, в клиентском окне игры можно наблюдать не только собственного персонажа, но и видеть модель игрока, управляемого другим участником, что подтверждает корректную работу сетевого взаимодействия и синхронизации данных между участниками игрового процесса.


Рисунок 18 – Клиент игры

При проведении тестирования созданной системы требуется последовательное выполнение двух основных этапов: активация серверной составляющей и запуск клиентской части с набором специализированных интерфейсов взаимодействия. Особое внимание следует уделить проверке взаимосвязи между этими компонентами, убедившись в их корректном

взаимодействии. Наглядным примером такого взаимодействия служит пользовательский интерфейс, представленный на рисунке 19, где отображается диалоговое окно создания новой учетной записи в системе. Данный интерфейс позволяет пользователям вводить необходимые данные для регистрации в веб-приложении и наглядно демонстрирует работоспособность механизма обмена информацией между клиентской и серверной частями системы.

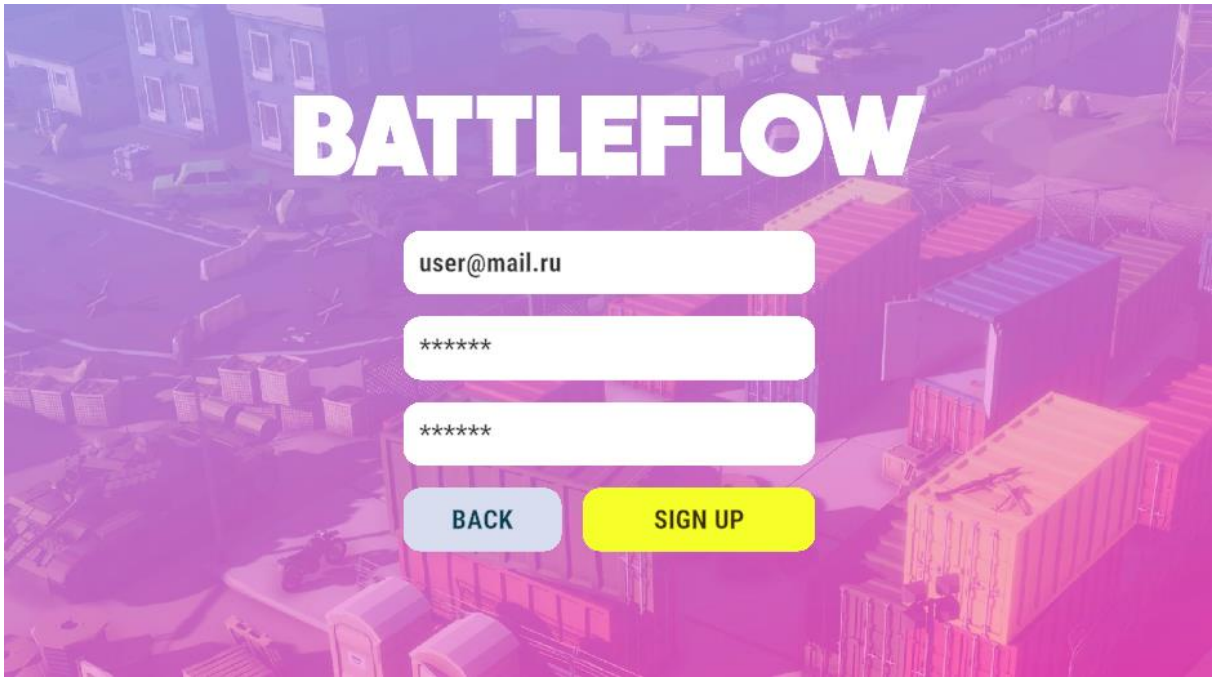



Рисунок 19 – Форма регистрации

После успешного завершения процедуры создания учетной записи система автоматически направляет пользователя в центральный раздел приложения, который представлен на рисунке 20. Этот основной экран содержит все ключевые элементы управления, необходимые для полноценного взаимодействия с игровой средой. В верхней части интерфейса пользователь может видеть информацию о своем текущем финансовом состоянии в игре, отображаемую в виде количества имеющихся игровых денежных единиц. Центральная часть экрана содержит основные элементы

управления: здесь располагается кнопка для быстрого старта игрового процесса, а также элемент доступа к развернутому списку действующих игровых серверов, позволяющий пользователю выбрать наиболее подходящий вариант для начала игры.

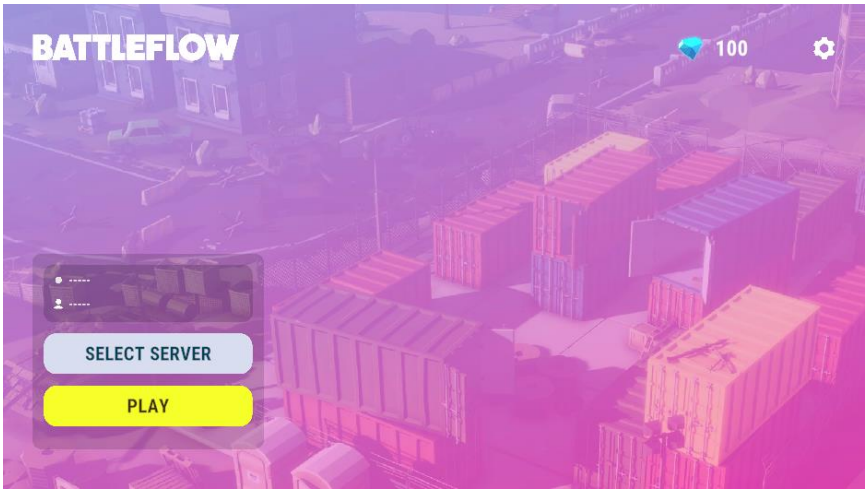



Рисунок 20 – Главное меню

На рисунке 21 изображен браузер серверов, содержащий информацию о доступных серверах, полученную через веб-приложение.

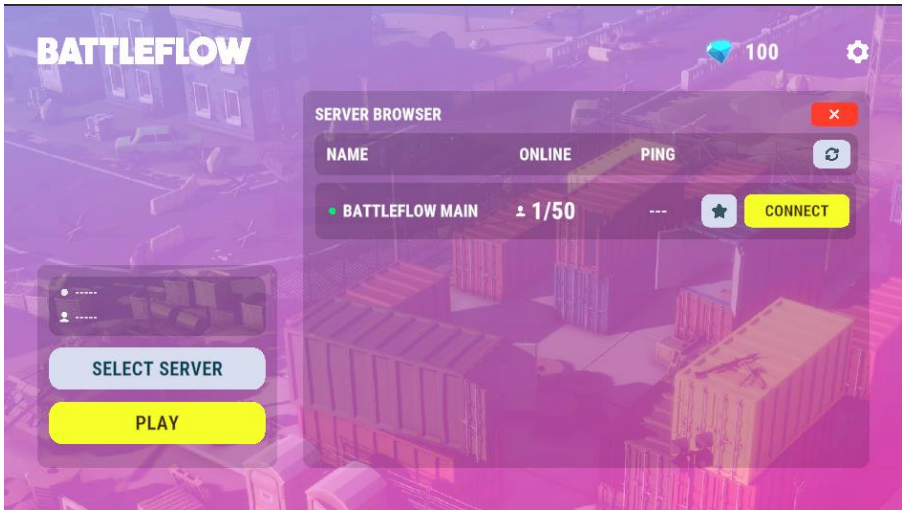



Рисунок 21 – Браузер серверов

Таким образом, была проведена первоначальная проверка работоспособности клиента и сервера разработанной многопользовательской онлайн игры. В текущей реализации критических проблем выявлено не было.

### 3.2 Оценка удобства и функциональности приложения

В результате тестирования и отладки была проверена работа системы многопользовательской онлайн игры и инфраструктурного веб-приложения. В таблице 3 отображены тестовые сценарии, помогающие выяснить ключевые проблемы текущей реализации.

Таблица 3 – Тестовые сценарии

| ID сценария | Название сценария | Шаги тестирования | Ожидаемый результат | Фактический результат | Статус |
|---|---|---|---|---|---|
| 001 | Проверка боевой системы при нестабильном соединении | 1.Подключиться к серверу с нестабильным соединением<br>2.Выбрать быстрый слот с оружием<br>3. Атаковать противника | Урон применяется корректно | Урон применяется корректно | Пройден |
| 002 | Нагрузочное тестирование сервера | 1.Запустить сервер<br>2.Запустить на сервер более 25 клиентов | Сервер стабилен | Сервер периодически зависает | Не пройден |
| 003 | Проверка коллизий карты | 1.Подключиться к серверу<br>2.Перемещаться по карте | Персонаж перемещается свободно по всей карте | Персонаж перемещается свободно по всей карте | Пройден |
| 004 | Проверка вычисления серверной задержки в браузере серверов | 1.Авторизоваться в игре<br>2. Перейти в браузер серверов | Отображается корректная задержка | Задержка не отображается | Не пройден |

Продолжение таблицы 3

| ID сценария | Название сценария | Шаги тестирования | Ожидаемый результат | Фактический результат | Статус |
|---|---|---|---|---|---|
| 005 | Проверка синхронизации движения игроков | 1.Подключиться в игру с нескольких аккаунтов<br>2.Наблюдать за передвижением друг друга | Передвижение отображается корректно | Передвижение отображается корректно | Пройден |
| 006 | Проверка безопасного подключения к серверу | 1.Авторизоваться в игре<br>2.Подключиться к серверу<br>3.Авторизоваться в игре с другого устройства, используя тот же аккаунт<br>4.Подключиться к серверу | Сервер отклонит повторное подключение ранее подключенного аккаунта | Сервер одобрил повторное подключение | Не пройден |

Согласно полученным результатам прохождения тестовых сценариев, были выявлены проблемы с производительностью сервера при большом количестве подключений, возможное решение – оптимизация серверного кода, вынесение высоконагруженной логики в отдельные потоки. Помимо этого, необходимо добавить отображение задержки между сервером и клиентом в браузер серверов, а также реализовать безопасное подключение к серверу с верификацией через API.

### 3.3 Анализ результатов использования программного обеспечения

В ходе тестирования программного обеспечения были проведены нагрузочные тесты, которые показали, что игровой сервер плохо справляется с большим количеством игроков. В связи с этим были сделаны выводы о необходимости многопоточной организации серверной логики. Самыми затратными по производительности оказались запросы к базе данных –

выполнение одного запроса может занимать до 10 секунд, в зависимости от нагруженности системы управления базой данных. Такая задержка может отразиться на всех игроках, подключенных к серверу, что значительно нарушит игровой процесс и создаст негативные впечатления у игроков. Возможное решение проблемы – вынесение логики взаимодействия с базой данных в отдельные потоки. Такое решение разгрузит основной игровой цикл, игроки в свою очередь перестанут чуствовать зависания при взаимодействии сервера с базой данных.

Помимо оптимизации работы с данными, необходимо провести оптимизацию синхроинизации игровых данных между всеми игроками. В текущей реализации не предусмотрена выборочная синхронизация только тех объектов, которые находятся близко к персонажу игрока, что приводит к экспоненциальному росту нагрузки на сервер при большом количестве подключенных клиентов.

Несмотря на проблемы с оптимизацией, синхронизация состояния игроков получилось довольно точной и авторитарной за счёт механизма прогнозирования и отката серверного состояния игрока, что соответствует изначальным требованиям.

Вывод по третьей главе

В рамках главы 3 была проведена оценка эффективности разработанной системы многопользовательской онлайн игры. Было проведено тестирование основных систем, согласно которому была подтверждена точность синхронизации состояния игровых объектов и персонажей. Также были выявлены узкие места касающиеся оптимизации синхронизации игровых объектов и логики взаимодействия с базой данных, в связи с этим были намечены рекомендации по оптимизации.

# Заключение

В ходе бакалаврской работы было проведено углубленное изучение фундаментальных принципов и передовых методологий создания программных продуктов. Основной целью являлась разработка оптимизированной, легко масштабируемой архитектурной структуры для сетевой многопользовательской игры.

Центральным инструментом при создании основного функционала, включающего как клиентскую, так и серверную составляющие, выступила современная среда разработки Unity в сочетании с языком программирования C#. Примечательно, что использование единой проектной среды Unity для обеих частей приложения существенно оптимизировало процесс разработки, исключив необходимость дублирования программного кода.

В процессе создания вспомогательного веб-приложения был задействован комплексный технологический стек на базе C# и платформы ASP.NET. Взаимодействие с базами данных обеспечивалось посредством библиотеки Entity Framework Core, в то время как безопасность пользовательских данных гарантировалась системой хеширования BCrypt. Достоверность получаемой информации контролировалась механизмами FluentValidation, а маршрутизация запросов осуществлялась через MinimalAPI. Благодаря продуманной архитектуре и гибкости используемых инструментов, веб-приложение демонстрирует высокий потенциал к дальнейшему расширению функционала в соответствии с эволюцией проекта.

Полученные в ходе выполнения бакалаврской работы результаты позволяют сформулировать ценные практические рекомендации относительно применения конкретных принципов проектирования и паттернов разработки при создании комплексных сетевых многопользовательских игровых систем.

## Список используемой литературы


1. Мартин, Р. К. Чистый код : создание, анализ и рефакторинг / Р. К. Мартин ; Роберт Мартин ; [пер. с англ. Е. Матвеев]. – Москва [и др.] : Питер, 2010. – 464 с. – (Библиотека программиста). – ISBN 978-5-498-07381-1. – EDN OVXZCB.

2. Рихтер, Д. CLR via C# : программирование на платформе Microsoft®.NET FRAMEWORK 2.0 на языке C# : [пер. с англ.] / Д. Рихтер ; Джеффри Рихтер. – Москва [и др.] : Русская ред., 2007. – (Мастер-класс). – ISBN 5-7502-0285-2. – EDN QMQLFT.

3. Кулинча, П. В. Разработка игр на C#: исследование возможностей разработки игр с использованием игровых движков и фреймворков на основе C#, таких как Unity / П. В. Кулинча // Дневник науки. – 2023. – № 6(78). – EDN OVPLXV.

4. Шандыбин, Е. А. Разработка игр в межплатформенной среде разработки Unity / Е. А. Шандыбин, О. С. Слепова // Актуальные социально-экономические проблемы развития общества в России и за рубежом : Материалы IV Всероссийской научно-практической конференции с международным участием, Волгоград, 30 ноября 2022 года. Том 1. – Волгоград: ООО Амирит, 2022. – С. 542-545. – EDN UHCZFM.

5. Турпалова, М. С. А. Разработка приложений в среде разработки Unity / М. С. А. Турпалова, И. И. Хайдаров // Актуальные вопросы физико-математического образования : Материалы межрегиональной студенческой научно-практической конференции, Грозный, 21 апреля 2022 года. – Махачкала: АЛЕФ, 2022. – С. 361-365. – EDN PYYQSA.

6. Искусство чистого кода / В. В. Фиоктистова, А. Д. Кузнецов, М. Д. Порохня, М. И. Никанорова // Тенденции развития науки и образования. – 2024. – № 108-12. – С. 164-169. – DOI 10.18411/trnio-04-2024-683. – EDN IPAMXC.

7. Программирование на языке C# : учебное пособие / Р. Г. Гильванов, Л. М. Божко, А. Д. Хомоненко [и др.]. – Санкт-Петербург : Петербургский государственный университет путей сообщения Императора Александра I, 2024. – 89 с. – ISBN 978-5-7641-1977-9. – EDN BMFIXC.

8. Биллиг, В. А. Основы объектного программирования на C# (C# 3.0, Visual Studio 2008) : Учебное пособие / В. А. Биллиг. – Москва, Саратов : Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2017. – 583 с. – ISBN 978-5-4487-0145-0. – EDN ZVDJYH.

9. Волков, М. Р. Идентификация и аутентификация в ASP.NET Core с использованием ASP.NET Core Identity / М. Р. Волков // Развитие науки и технологий в современной России (шифр - ВКРН) : Сборник материалов II Всероссийской научно-практической конференции, Москва, 29 апреля 2024 года. – Москва: ООО "Издательство Академическая среда", 2024. – С. 41-56. – EDN AYUBUV.

10. Березовский, М. С. Разработка веб-приложения на платформе ASP.net / М. С. Березовский, М. И. Жадан // Молодежная наука в XXI веке: традиции, инновации, векторы развития : материалы Международной научно-исследовательской конференции молодых ученых, аспирантов, студентов и старшеклассников: в 3 частях, Самара-Оренбург, 05 апреля 2017 года. Том Часть 1. – Самара-Оренбург: Общество с ограниченной ответственностью "Аэтерна", 2017. – С. 177-179. – EDN ZHHQZJ.

11. Санин, Н. А. Использование универсальной платформы Unity 3d для разработки информационных систем / Н. А. Санин // Научный вестник государственного образовательного учреждения Луганской Народной Республики "Луганский национальный аграрный университет". – 2020. – № 8-3. – С. 413-416. – EDN YDIZPG.

12. Бальцев, Д. И. Современная игровая индустрия: геймдев (Gamedev) / Д. И. Бальцев // XII Международный молодежный форум "Образование. Наука. Производство" : Материалы форума, Белгород, 01–20 октября 2020 года. – Белгород: Белгородский государственный

технологический университет им. В.Г. Шухова, 2020. – С. 1871-1875. – EDN NLYFZT.

13. Какаулин, Д. В. Разработка архитектуры логики скриптов при создании кроссплатформенных видеоигровых проектов в Unity / Д. В. Какаулин, Н. М. Сеидова // Прикладная математика: современные проблемы математики, информатики и моделирования : материалы V Всероссийской научно-практической конференции, молодых ученых, Краснодар, 11–15 апреля 2023 года. – Краснодар: ФГБУ "Российское энергетическое агентство" Минэнерго России Краснодарский ЦНТИ- филиал ФГБУ "РЭА" Минэнерго России, 2023. – С. 108-113. – EDN DWKBCE.

14. Felk, V. S. TCP and UDP protocols / V. S. Felk, E. V. Tyubaeva // Молодежь. Общество. Современная наука, техника и инновации. – 2022. – No. 21. – P. 33-34. – EDN TLJDNX.

15. McLaughlin B., Pollice G., West D. Head First Object-Oriented Analysis and Design: A Brain-Friendly Guide to OOA&D / B. McLaughlin, G. Pollice, D. West. – 1st ed. – O'Reilly Media, 2007. – 634 p. – ISBN 978-0-596-00867-3.

16. Gandhi R., Richards M., Ford N. Head First Software Architecture: A Learner's Guide to Architectural / R. Gandhi, M. Richards, N. Ford – 1st ed. – O'Reilly Media, 2024. – 634 p. – ISBN 978-1-098-13435-8.

17. Freeman E., Robson E. Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software / E. Freeman, E. Robson – 2nd ed. – O’Reilly Media, 2021. – 669 p. – 978-1-492-07800-5.

18. Stellman A., Greene J. Head First C#: A Learner's Guide to Real-World Programming with C# and .NET / A. Stellman, J. Greene – 5th ed. – O’Reilly Media, 2024. – 835 p. – 978-1-098-14178-3.

19. Buttfield-Addison P., Manning J., Nugent T. Unity Game Development Cookbook: Essentials for Every Game / P. Buttfield-Addison, J. Manning, T. Nugent – 1st ed. – O’Reilly Media, 2019. – 405 p. – 978-1-491-99915-8.

20. Hocking J. Unity in Action, Third Edition: Multiplatform game development in C# / J. Hocking – 3rd ed. – Manning, 2022. – 416 p. – 978-1-617-29933-9.