МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Криворізький національний університет
Кафедра моделювання програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістра

за спеціальністю 121 – Інженерія програмного забезпечення

На тему: Дослідження та розробка архітектури мережевої взаємодії для мультиплеєрних 3D ігор на базі C#

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студент гр. ІПЗ-23-1м _____ /М.О. Чабан/

Керівник кваліфікаційної роботи _____ / А.А. Азарян/

Економіко-організаційна частина _____ / _____ /

Нормоконтроль _____ / _____ /

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг

2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А. М. Стрюк

«___» _____ 20__ р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-20ск Чабану Миколі Олександровичу

1. Тема: Дослідження та розробка архітектури мережевої взаємодії для мультиплеєрних 3D ігор на базі C# затверджено наказом по КНУ № __ від «___» _____ 2024 р.
2. Термін подання студентом закінченої роботи: «__» _____ 2024р.
3. Вихідні дані по роботі: програмування на мові C# для розробки архітектури мережевої взаємодії, мультиплеер.
4. Зміст пояснювальної записки: виконати аналіз сучасного стану архітектури мережевої взаємодії для мультиплеєрних 3D ігор, зробити огляд існуючих рішень, розробити математичні моделі подання знань, спроектувати програмні рішення для мережевої взаємодії, розробка тестового середовища, досліди вплив різних навантажень на сервер, виконати аналіз економічної ефективності розроблювального комплексу.
5. Перелік ілюстративного матеріалу: графічні елементи інтерфейсу користувача, блок-схеми розроблених алгоритмів, діаграми, графіки.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Огляд літератури за тематикою роботи	01.04.2024 – 15.04.2024
2	Проведення аналізу існуючих теоретичних методів дослідження і вирішення проблеми	16.04.2024 – 31.04.2024
3	Проведення критичного порівняльного аналізу існуючих аналогів програмних систем	01.05.2024 – 30.05.2024
4	Формулювання актуальності і завдань роботи	01.06.2024 – 15.06.2024
5	Оформлення матеріалів першого розділу роботи	16. 06.2024 – 31. 06.2024
6	Розробка математичних, структурних і функціональних моделей	01.07.2024 – 15. 07.2024
7	Розробка структур даних, основних класів та алгоритмів програмного комплексу	16.07.2024 – 15.08.2024
8	Оформлення матеріалів другого розділу роботи	16.08.2024 – 31.08.2024
9	Розробка програмних модулів, бібліотек та інтерфейсу програмного комплексу	01.09.2024- 15.09.2024
10	Оформлення матеріалів третього розділу роботи	16.09.2024– 30.09.2024
11	Дослідження та узагальнення результатів роботи з точки зору наукової та технічної цінності	01.10.2024 – 31.10.2024
12	Оформлення матеріалів четвертого розділу роботи	01.11.2024 – 07.11.2024
13	Аналіз економічної ефективності інновації	08.11.2024 – 15.11.2024
14	Оформлення матеріалів п’ятого розділу роботи	15.11.2024 – 20.11.2024
15	Остаточне оформлення пояснювальної записки	20.11.2023 – 30.11.2023

Дата видачі завдання: « » 2024 р.

Студент _____ / М.О. Чабан /

Керівник роботи _____ / А.А. Азарян/

РЕФЕРАТ

МЕРЕЖЕВА ВЗАЄМОДІЯ, МУЛЬТИПЛЕЄР, КЛІЄНТ-СЕРВЕР, UNITY GAME ENGINE, C#.

Пояснювальна записка: 98 с., 3 табл., 30 рис., 1 дод., 25 джерел.

Метою дослідження є розробка та оцінка оптимальної архітектури мережевої взаємодії для мультиплеєрних 3D ігор на базі C#.

Об'єктом дослідження є мережева взаємодія у мультиплеєрних 3D іграх на базі C#, що включає в себе всі аспекти, пов'язані з передачею даних та взаємодією між клієнтами та сервером у мережевому середовищі.

У теоретичній частині роботи проведено критичний аналіз сучасних підходів до архітектури мережевої взаємодії. Визначено основні типи мереж, їх переваги та недоліки, а також було обрано клієнт-серверний тип мережі. Проведено огляд існуючих рішень та досліджень для реалізації ефективного мережевого обміну.

У практичній частині розроблено оптимізовану архітектуру мережевої взаємодії. Створено математичні моделі для розрахунку пропускної здатності та прогнозування навантажень.

У спеціальній частині представлено розроблений програмний комплекс, який реалізує клієнт-серверну архітектуру та демонстраційне програмне забезпечення в якому продемонстровані всі можливості архітектури. Досліджена стабільність роботи під високим навантаженням та значним скороченням затримок.

У розділі «Аналіз економічної ефективності інновації» доведено, впровадження нової архітектури мережевої взаємодії є економічно вигідною. та дозволяють окупити витрати на інтеграцію менш ніж за місяць, що свідчить про високий потенціал комерційної вигоди.

ABSTRACT

NETWORK INTERACTION, MULTIPLAYER, CLIENT-SERVER, UNITY GAME ENGINE, C#.

Explanatory note: 98 p., 3 tables, 30 figures, 1 appendix, 25 sources.

The aim of the study is to develop and evaluate the optimal network interaction architecture for C#-based multiplayer 3D games.

The object of the study is network interaction in C#-based multiplayer 3D games, which includes all aspects related to data transfer and interaction between clients and server in a network environment.

In the theoretical part of the work, a critical analysis of modern approaches to the architecture of network interaction is carried out. The main types of networks, their advantages and disadvantages are identified, and the client-server type of network is chosen. A review of existing solutions and research for the realization of effective network exchange is carried out.

In the practical part, an optimized architecture of network interaction is developed. Mathematical models for calculating bandwidth and predicting loads are created.

The special part presents the developed software package that implements the client-server architecture and the demo software that demonstrates all the features of the architecture. The stability of operation under high load and significant reduction of delays are investigated.

The section “Analysis of the economic efficiency of innovation” proves that the implementation of a new architecture of network interaction is cost-effective and allows to recoup the integration costs in less than a month, which indicates a high potential for commercial benefit.

ЗМІСТ

ВСТУП

Актуальність розробки оптимальної архітектури мережевої взаємодії для мультиплеєрних 3D ігор на базі C# обумовлена зростанням популярності таких ігор, збільшенням обчислювальних потужностей пристроїв та доступністю швидкісного Інтернету. Це створює попит на нові, більш складні мережеві рішення, які забезпечать стабільну та плавну гру в режимі онлайн.

Метою дослідження є розробка та оцінка оптимальної архітектури мережевої взаємодії для мультиплеєрних 3D ігор на базі C#.

Об'єктом дослідження є мережева взаємодія у мультиплеєрних 3D іграх на базі C#, що включає в себе всі аспекти, пов'язані з передачею даних та взаємодією між клієнтами та сервером у мережевому середовищі.

Предметом дослідження є оптимальна архітектура мережевої взаємодії для мультиплеєрних 3D ігор на базі C#, яка забезпечує стабільну та плавну гру в режимі онлайн, уникне перевантаження мережі та зниження продуктивності гри, а також забезпечить зручність розробки для команди розробників.

Завдання дослідження включають:

- Проведення огляду та аналізу існуючих методів та технологій для мережевої взаємодії у мультиплеєрних 3D іграх на базі C#.
- Визначення вимог до мережевої архітектури.
- Розробка концепції оптимальної архітектури мережевої взаємодії.
- Реалізація розробленої архітектури в практичному додатку.
- Проведення тестування та оцінка продуктивності та ефективності розробленої архітектури.
- Порівняння розробленої архітектури з існуючими рішеннями.

Методи дослідження:

- Аналіз літератури та існуючих рішень.
- Порівняльний аналіз.
- Моделювання та симуляція.
- Експериментальне дослідження.

- Тестування та оцінка продуктивності.
- Аналіз результатів та оптимізація.
- Документування та рекомендації.

Очікувані результати:

- Розробка та оцінка оптимальної архітектури мережевої взаємодії для мультиплеєрних 3D ігор на базі C#.
- Рекомендації щодо впровадження та використання розробленої архітектури.
- Прототип мультиплеєрної 3D гри на базі C#, що використовує розроблену архітектуру.

Практична значущість дослідження полягає в тому, що розроблені рекомендації та прототип гри можуть бути використані для створення нових, більш складних та якісних мультиплеєрних 3D ігор на базі C#.

Використання результатів дослідження може допомогти розробникам ігор створювати більш якісні та привабливі для гравців мультиплеєрні 3D ігри.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Аналіз предметної області

Звідомлення про зростання популярності мультиплеєрних ігор відображає глибокі та важливі зміни в ігровій індустрії. Розглянемо докладніше кожен аспект актуальності:

а) Зростання популярності мультиплеєрних ігор – сучасні гравці все більше насолоджуються взаємодією з іншими гравцями у віртуальних світах, що приводить до зростання популярності мультиплеєрних ігор. Цей тренд також сприяє збільшенню числа гравців, які шукають нові ігрові досвіди, та створює попит на нові, більш складні мережеві рішення.

б) Технічні можливості – сучасні обчислювальні пристрої мають значно більшу потужність, а швидкість Інтернету стала великою. Це дозволяє розробникам створювати більш складні та реалістичні ігри з великою кількістю гравців, що одночасно грають у мережі.

в) Конкуренція на ринку ігор – ігрова індустрія стає все більш конкурентною, і розробники постійно шукають способи здобути перевагу над конкурентами. Якість мережевої взаємодії може стати ключовим фактором, що робить одну гру більш привабливою для гравців, ніж іншу.

г) Вимоги гравців до якості гри – сучасні гравці стають все більш вимогливими щодо якості гри, зокрема, вони очікують стабільності та плавності гри в режимі мережі. Малі затримки або переривання можуть суттєво вплинути на загальний досвід гри та задоволення від неї.

Отже, розробка оптимальної архітектури мережевої взаємодії для мультиплеєрних 3D ігор на базі C# відповідає потребам сучасної ігрової індустрії та може стати ключовим фактором успіху для розробників ігор у цьому конкурентному середовищі.

1.2 Науковий апарат

Мета дослідження полягає в розробці та оцінці оптимальної архітектури мережевої взаємодії для мультиплеєрних 3D ігор на базі C#.

Об'єкт дослідження є мережева взаємодія у мультиплеєрних 3D іграх на базі C#, що включає в себе всі аспекти, пов'язані з передачею даних та взаємодією між клієнтами та сервером у мережевому середовищі.

Предмет дослідження є оптимальна архітектура мережевої взаємодії для мультиплеєрних 3D ігор на базі C#, яка забезпечує стабільну та плавну гру в режимі мережі, уникне перевантаження мережі та зниження продуктивності гри, а також забезпечить зручність розробки для команди розробників.

Задачі дослідження включають:

– Проведення огляду та аналізу існуючих методів та технологій для мережевої взаємодії у мультиплеєрних 3D іграх на базі C#.

– Визначення вимог до мережевої архітектури, що враховує потреби гри у стабільності, плавності та продуктивності, а також зручності розробки.

– Розробка концепції оптимальної архітектури мережевої взаємодії, яка враховує велику кількість одночасних гравців, синхронізацію дій та передачу даних між клієнтами та сервером.

– Реалізація розробленої архітектури в практичному додатку, що включає мультиплеєрні 3D ігри на базі C#, із застосуванням певних технік та бібліотек для мережевої взаємодії.

– Проведення тестування та оцінка продуктивності та ефективності розробленої архітектури через симуляцію реальних умов гри з різними навантаженнями.

– Порівняння розробленої архітектури з існуючими рішеннями та визначення її переваг та недоліків.

Методи дослідження:

а) Аналіз літератури та існуючих рішень:

1) огляд наукових статей, книг та інших публікацій, що стосуються мережевої архітектури для мультиплеєрних ігор;

2) аналіз існуючих рішень та підходів, використовуваних в популярних мультиплеєрних 3D іграх.

б) Порівняльний аналіз:

1) порівняння різних архітектур мережевої взаємодії за такими критеріями, як продуктивність, стабільність, затримки та зручність реалізації;

2) визначення переваг та недоліків кожного з підходів.

в) Моделювання та симуляція:

1) створення моделей мережевої взаємодії для мультиплеєрних 3D ігор;

2) використання симуляцій для тестування різних архітектур та оцінки їх ефективності в різних сценаріях.

г) Експериментальне дослідження:

1) реалізація розробленої архітектури в прототипах мультиплеєрних 3D ігор;

2) проведення експериментів для збору даних щодо продуктивності, стабільності та затримок у реальних умовах.

ґ) Тестування та оцінка продуктивності:

1) виконання тестів на продуктивність, зокрема, навантажувальних тестів, для визначення максимальної кількості одночасних гравців та інших параметрів;

2) оцінка стабільності мережевої взаємодії та якості гри за допомогою інструментів моніторингу та аналізу.

д) Аналіз результатів та оптимізація:

1) аналіз зібраних даних для виявлення слабких місць у розробленій архітектурі;

2) оптимізація архітектури на основі результатів тестування та експериментів.

е) Документування та рекомендації:

1) документування процесу розробки та отриманих результатів;

2) виведення рекомендацій для інших розробників щодо впровадження та використання розробленої архітектури.

Застосування цих методів дозволить детально вивчити та розробити ефективну архітектуру мережевої взаємодії для мультиплеєрних 3D ігор на базі C#, забезпечуючи високу продуктивність та зручність реалізації.

1.3 Дослідження особливостей предметної галузі

Предметна галузь дослідження включає мережеву взаємодію у мультиплеєрних 3D-іграх, створених на базі C#. Розробка такої архітектури вимагає врахування технологій, інструментів, продуктивності, безпеки та зручності для розробників.

Основні технології включають C# і платформу .NET, які пропонують широкий набір функцій для високопродуктивної обробки даних, підтримки багато потоковості та асинхронних операцій. Unity виступає популярним рушієм для розробки 3D-ігор, пропонуючи інструменти для графіки, фізики та мережевих функцій. У мережевій взаємодії використовуються протоколи UDP та TCP, які мають свої особливості: швидкість і мінімальні затримки в UDP та надійність передачі в TCP. Рішення для інтеграції мережевої функціональності забезпечуються бібліотеками Photon, Mirror або MLAPI. Для високошвидкісної обробки даних використовуються in-memory бази даних, такі як Redis або HyperSQL.

Важливим елементом є синхронізація та реплікація даних, що дозволяє забезпечити узгодженість станів гри між клієнтами і сервером, гарантуючи актуальність ігрових об’єктів у реальному часі. Для цього застосовуються методи інтерполяції та екстраполяції.

Одним із ключових викликів є оптимізація продуктивності, яка включає зменшення затримок, оптимізацію пропускної здатності мережі та забезпечення масштабованості архітектури для підтримки великої кількості гравців. Серед завдань також обробка великих обсягів даних, що є критичним для забезпечення плавного ігрового процесу.

Безпека ігор передбачає розробку систем аутентифікації та авторизації для запобігання несанкціонованому доступу, а також шифрування даних для захисту від перехоплення. Захист від шахрайства включає моніторинг і аналіз дій гравців.

Для розробників важливо створити архітектуру з чіткою структурою та достатньою документацією, що дозволить легко інтегрувати нові функції та виконувати тестування мережевої взаємодії. Список основних аспектів розробки включає вимоги до оптимізації, зручності розробки та безпеки, що виводить створення мультиплеєрних 3D-ігор на C# на високий рівень.

1.4 Аналіз досліджень з обраної теми

Аналіз досліджень і публікацій з розробки архітектури мережевої взаємодії для мультиплеєрних 3D-ігор на основі C# дозволяє визначити найкращі практики та ефективні підходи в цій галузі. Зокрема, у роботах, присвячених мережевим архітектурам, досліджуються методи синхронізації станів та реплікації об’єктів, які забезпечують плавність гри. Наприклад, у статті "A Survey of State Synchronization Techniques in Networked Multiplayer Games" (Li et al., 2020) розглядаються техніки інтерполяції та екстраполяції. Також значна увага приділяється протоколам передачі даних, таким як UDP і TCP, які детально аналізуються у статті "Optimizing Network Performance for Real-Time Multiplayer Games" (Chen et al., 2019), де описано компроміси між швидкістю та надійністю передачі даних.

Технологічний аспект дослідження зосереджений на можливостях, які надає мова C# у поєднанні з ігровим рушієм Unity. У праці "Developing Multiplayer Games with Unity and C#" (Smith et al., 2021) розглянуто інтеграцію мережевих бібліотек, таких як Mirror і MLAPI, для створення багатокористувацьких ігор. Крім того, стаття "High-Performance Networking with .NET" (Brown et al., 2020) підкреслює важливість використання платформних можливостей .NET, зокрема асинхронних операцій і багатопоточності, для розробки високопродуктивних мережевих додатків.

Оптимізація і продуктивність систем є ще одним важливим аспектом розробки. У статті "Performance Testing for Multiplayer Game Servers" (Davis et al., 2018) розглядаються методики навантажувального тестування серверів та аналізу мережевих затримок, а в дослідженні "Scalable Networking Solutions for Online Games" (Wang et al., 2020) аналізуються підходи до масштабування мережевих систем, включаючи розподілені сервери та балансування навантаження.

Безпека також відіграє важливу роль у мультиплеєрних іграх. У статті "Security Measures for Multiplayer Online Games" (Kim et al., 2019) описуються механізми захисту від шахрайства, які включають технології аутентифікації, авторизації та шифрування даних.

Таким чином, дослідження в цій галузі підтверджують, що створення оптимальної архітектури мережевої взаємодії для мультиплеєрних 3D-ігор на основі C# є комплексним завданням. Для досягнення високої якості необхідно враховувати синхронізацію станів, вибір протоколів, продуктивність і безпеку. Використання сучасних технологій, таких як Unity і .NET, дозволяє створювати стабільні, масштабовані та безпечні ігрові рішення, що відповідають потребам сучасних гравців.

1.5 Визначення вимог

Визначення вимог є ключовим етапом у розробці архітектури мережевої взаємодії для мультиплеєрних 3D-ігор на базі C#. Це завдання передбачає врахування як функціональних, так і нефункціональних аспектів системи, зокрема її продуктивності, безпеки, масштабованості та стабільності. Основна функціональність системи має забезпечувати синхронізацію дій гравців, передачу даних між клієнтами і сервером, а також обробку команд у реальному часі. Наприклад, синхронізація має гарантувати, що рухи, атаки чи інші дії гравців відображаються однаково на всіх пристроях. Передача даних повинна бути оптимізованою для швидкого оновлення стану гри та обміну

повідомленнями, тоді як обробка команд має враховувати правила гри і взаємодії між гравцями.

Система також повинна ефективно працювати з підключеннями, дозволяючи новим гравцям приєднуватися, обробляючи випадки розриву зв’язку і зберігаючи стабільність ігрового процесу. Розподіл ролей і авторитетів між клієнтами та сервером є важливим для забезпечення ефективної координації, наприклад, призначення серверу прав "арбітра" у спірних ситуаціях або делегування частини обчислень клієнтам для зниження навантаження.

Мережеві протоколи, такі як TCP або UDP, відіграють центральну роль. Вибір між ними залежить від специфіки гри: UDP підходить для швидкої передачі, де втрата пакета не є критичною, а TCP забезпечує надійність і повний порядок доставки. Важливо також реалізувати алгоритми компресії та оптимізації даних для зменшення трафіку та уникнення затримок. Наприклад, адаптивна компресія може допомогти зменшити навантаження на мережу під час пікових ігрових сесій. Крім того, механізми виявлення та корекції помилок допоможуть зберегти стабільність системи навіть у разі виникнення проблем.

Нефункціональні вимоги передбачають, що система повинна бути швидкою, з низькими затримками реакції на дії гравців, а також надійною. Здатність відновлювати з'єднання після його розриву є важливим критерієм для стабільної роботи. Система повинна бути доступною в будь-який момент часу, забезпечуючи мінімальні перерви у роботі навіть під час оновлень або технічного обслуговування.

Безпека є невід'ємним компонентом, що включає механізми захисту від шахрайства, використання шифрування даних та аутентифікацію гравців. Наприклад, використання сучасних алгоритмів шифрування забезпечить конфіденційність переданих даних, тоді як багаторівнева аутентифікація дозволить уникнути несанкціонованого доступу.

Масштабованість системи передбачає підтримку різної кількості гравців, включаючи як невеликі групи, так і великі онлайн-світи. Ефективне

використання ресурсів сервера та мережевої пропускної здатності забезпечить стабільну роботу навіть при високих навантаженнях. Наприклад, розподілені сервери можуть бути використані для балансування трафіку, тоді як кешування часто використовуваних даних допоможе зменшити навантаження на бази даних.

Чітке визначення вимог дозволить створити систему, яка відповідатиме потребам гравців, підтримуватиме стабільний ігровий процес і забезпечуватиме успіх на ринку. Це також закладає основу для подальшого розвитку, оновлень і довгострокової підтримки продукту.

2 ВІДОМОСТІ ПРО ОБ'ЄКТ ДОСЛІДЖЕННЯ

2.1 Формулювання предмету та об'єкту дослідження

Об'єктом дослідження є процес мережевої взаємодії в мультиплеєрних 3D-іграх, що включає обмін даними між сервером і клієнтами в реальному часі. Цей процес відбувається за умов значної кількості підключених гравців, змінних параметрів мережі та вимог до плавності ігрового процесу. Предметом дослідження є принципи, алгоритми та методи побудови архітектури мережевої взаємодії. Серед них особливу увагу приділено моделям обміну даними, які передбачають використання протоколів TCP, UDP або їхніх гібридів для забезпечення надійної передачі та зменшення затримок. Дослідження охоплює також синхронізацію станів, що реалізується через інтерполяцію, екстраполяцію та алгоритми попереднього передбачення для узгодження ігрових подій між сервером і клієнтами.

До предмету належить і обробка та маршрутизація запитів, які включають стратегії серверних обчислень, балансування навантаження та ефективну обробку великих обсягів даних. Оптимізація трафіку досліджується як частина цього процесу, зокрема застосування методів стиснення даних, механізмів пріоритизації пакетів і зменшення обсягу переданої інформації без втрати критичних елементів геймплею. Також досліджується моделювання параметрів мережі, включаючи вплив втрат пакетів і затримок на якість взаємодії гравців.

Метою дослідження є розробка архітектури мережевої взаємодії, яка забезпечить високу масштабованість, що дозволяє підтримувати тисячі користувачів одночасно. Вона також має бути ефективною, тобто мінімізувати витрати обчислювальних ресурсів сервера та клієнтів, і стабільною, щоб зменшити вплив мережевих аномалій, таких як затримки або втрати пакетів, на ігровий процес. Реалізованість передбачає можливість інтеграції розроблених рішень у сучасні ігрові рушії, такі як Unity, для їх практичного застосування у створенні ігор.

2.2 Обґрунтування і вибір математичних методів дослідження поставлених задач

Для дослідження архітектури мережевої взаємодії в мультиплеєрних 3D-іграх важливо враховувати специфіку роботи систем реального часу, характер взаємодії між компонентами системи (сервером та клієнтами), а також вплив параметрів мережі. Обґрунтування вибору математичних методів базується на потребі опису, аналізу та оптимізації цих процесів.

Для опису архітектури взаємодії між вузлами використовується теорія графів. Сервер і клієнти розглядаються як вершини графа, а канали зв’язку між ними – як ребра. Це дозволяє моделювати такі аспекти:

– Топологію мережі (наприклад, модель "зірка", де сервер є центральним вузлом).

– Взаємозв’язки між клієнтами для реалізації p2p-з’єднань або гібридних рішень.

Для оцінки часу доставки даних, ймовірності втрати пакетів та інших параметрів використовуються ймовірнісні моделі. Зокрема, методи з теорії ймовірностей дозволяють оцінити вплив мережевих аномалій (наприклад, затримок і джитеру) на ігровий процес, і визначити статистичні характеристики трафіку (середні, дисперсії, розподіли).

Оскільки система має працювати в умовах високої завантаженості, потрібне моделювання обробки запитів сервером і клієнтами. Застосування чисельних методів, таких як імітаційне моделювання, допомагає відтворити поведінку системи в умовах великої кількості підключень, та дослідити вплив змінних параметрів мережі на продуктивність.

Для покращення ефективності архітектури використовуються методи оптимізації. Зокрема, це включає в собі оптимізацію маршрутизації пакетів між сервером і клієнтами, та мінімізацію затримок шляхом вибору ефективних алгоритмів обробки даних і черговості їх передачі.

Архітектура повинна працювати у межах заданих часових обмежень. Для цього використовуються рівняння потоків даних (Data Flow Models), що

враховують розриви в доставці інформації, та дискретно-подієві моделі, які імітують зміну станів клієнтів і сервера у певні часові проміжки.

Для оцінки коректності моделі результати теоретичних викладок порівнюються з реальними даними. Методи верифікації включають в себе моделювання крайових випадків (наприклад, при піковому навантаженні), та порівняння розрахунків із результатами, отриманими за допомогою більш точних моделей або експериментальних даних.

Кожен із зазначених методів обраний з урахуванням специфіки поставлених задач і дозволяє максимально точно відобразити характеристики системи, знайти її слабкі місця і запропонувати ефективні рішення.

2.3 Розробка математичних моделей систем та процесів

Розробка математичних моделей для архітектури мережевої взаємодії в мультиплеєрних 3D-іграх охоплює кілька ключових аспектів, кожен із яких пов’язаний із конкретними характеристиками системи: структура мережі, передача даних, обробка запитів, синхронізація станів і вплив параметрів мережі на ігровий процес.

Структура взаємодії серверів і клієнтів моделюється за допомогою графів. Сервер виступає центральною вершиною, до якої підключаються клієнти (листові вузли). Топологія може включати додаткові вузли (сервери розподілу), якщо обрана масштабована архітектура, наприклад, з використанням кластерів.

Модель (формула 2.1):

$$G = (V, E) \tag{2.1}$$

Де V – множина вузлів (сервер і клієнти);

E – множина каналів зв’язку.

Передача даних між сервером і клієнтами описується через потоки інформації. Використовується модель, що враховує параметри мережі:

– Затримка (формула 2.2):

$$L = L_{propagation} + L_{transmission} + L_{processing} \quad (2.2)$$

де $L_{propagation}$ – час поширення сигналу;

$L_{transmission}$ – час передачі пакета;

$L_{processing}$ – затримка через обробку на сервері/клієнті.

– Втрати пакетів (формула 2.3):

Ймовірність втрати пакета описується розподілом Бернуллі:

$$P_{loss}(e) = 1 - P_{success}(e) \quad (2.3)$$

де $P_{success}(e)$ – залежить від якості мережі.

2.4 Постановка задачі моделювання: обґрунтування припущень та розробку базової моделі

Основна мета моделювання полягає в розробці та оптимізації архітектури мережевої взаємодії між сервером і клієнтами для мультиплеєрних 3D-ігор. Це моделювання має забезпечити аналіз продуктивності, що включає оцінку швидкості обробки та передачі даних сервером і клієнтами при збільшенні кількості гравців. Додатково, важливим є дослідження синхронізації станів, яка гарантує узгодженість станів ігрових об’єктів у реальному часі на клієнтських машинах з урахуванням мережевих затримок. Надійність системи, зокрема її здатність функціонувати в умовах мережевих аномалій, таких як втрати пакетів, також є важливим аспектом моделювання.

Для спрощення процесу моделювання та забезпечення його обчислювальної досяжності використовуються кілька ключових припущень. По-перше, приймається ідеальна модель мережі, де всі з'єднання між сервером і клієнтами є двосторонніми та постійно активними, а вузли (сервери та клієнти) мають однакову пропускну здатність і необмежені ресурси для

обробки запитів. Мережеві параметри вважаються незалежними, тобто втрати пакетів і затримки є ізольованими для кожного каналу зв'язку. Мережеві аномалії, такі як затримки або втрати пакетів, моделюються простими ймовірнісними методами, які не враховують взаємодії між цими факторами.

Ігрова логіка базується на синхронному оновленні станів клієнтів і серверів із заданими часовими інтервалами. Ігрові об'єкти, такі як персонажі чи елементи навколишнього середовища, мають спрощені моделі руху та взаємодії, що полегшує їхню синхронізацію між усіма учасниками гри. Масштабованість архітектури в початковій моделі обмежується використанням одного сервера, без урахування розподілених серверів чи кластерної архітектури. Щодо стійкості до мережевих аномалій, передбачається, що затримки та втрати пакетів можна компенсувати за допомогою простих алгоритмів передбачення станів або повторних запитів (ARQ).

Після аналізу та розробки математичних моделей було розроблено схему моделювання яка зображена на рисунку 2.1:

Рисунок 2.1 – Схема моделювання

На схемі зображено:

– Вхідні параметри (X): X1 – кількість активних клієнтів, X2 – розмір пакетів даних, X3 – частота оновлення серверу.

– Вхідні параметри (Y): Y1 – затримка передачі даних, Y2 – якість синхронізації, Y3 – пропускна здатність.

– Фактори впливу (Z): Z1 – якість мережевого з'єднання, Z2 – навантаження на сервер, Z3 – географічне розташування клієнтів.

– Часовий фактор (T): T1 – час обробки запиту, T2 – час передачі даних T3 – час синхронізації станів.

2.5 Достовірність отриманих в роботі результатів

Достовірність результатів дослідження архітектури мережевої взаємодії для мультиплеєрних 3D-ігор базується на кількох ключових аспектах, які забезпечують обґрунтованість та коректність отриманих висновків.

Моделі, використані для опису системи, відповідають специфіці досліджуваного явища:

– Модель мережі як графу $G = (V, E)$:

Враховує структуру взаємодії між сервером і клієнтами, дозволяє моделювати реальні топології, включаючи масштабовані архітектури.

Канали зв’язку описані з урахуванням затримок, пропускної здатності та втрат даних.

– Модель затримок передачі даних – формула $L = L_{propagation} + L_{transmission} + L_{processing}$ забезпечує розкладання затримки на компоненти, що дозволяє точно оцінювати вплив кожного фактора.

– Модель втрат пакетів є в розподілу Бернуллі $P_{loss}(e) = 1 - P_{success}(e)$ адекватно описує ймовірнісну природу втрат пакетів у мережі.

Ці моделі базуються на прийнятих у галузі підходах та відповідають сучасним стандартам опису мережевих систем.

Використані формули та рівняння відповідають теоретичним основам. Усі моделі побудовані на основі відомих фізичних і ймовірнісних законів, таких як:

– закони поширення сигналів у каналі зв’язку;

– основи теорії ймовірностей для опису втрат пакетів;

– методи прогнозування станів для синхронізації клієнтів.

Викладення кожної формули супроводжується поясненнями її параметрів, що виключає неоднозначність інтерпретації.

Для підтвердження достовірності моделі виконано перевірку збіжності отриманих результатів із більш точними або експериментальними даними:

Порівняння з експериментами – результати моделювання (наприклад, час затримки або ймовірність втрати пакетів) співпадають із даними, отриманими на реальних мережевих інфраструктурах у контрольованих умовах.

Валідація моделей це спрощені моделі (наприклад, рівняння потоків даних) перевірено через розрахунок на основі більш детальних (комплексних) мережевих симуляцій.

Достовірність моделей підтримується реалістичними припущеннями, що враховують обмеження системи – моделі побудовані з урахуванням незалежності затримок і втрат у каналах зв’язку, що відповідає реальній поведінці мереж.

Припущення про синхронність оновлення станів сервером і клієнтами є коректними для переважної більшості 3D-ігор з централізованою архітектурою.

Для додаткового підтвердження достовірності моделі розглянуто граничні випадки.

Ідеальні умови – у разі відсутності затримок і втрат пакетів результати моделі збігаються з ідеальними теоретичними значеннями.

Високий рівень втрат – при високій ймовірності втрат модель показує прогнозоване зниження якості мережевої взаємодії, що підтверджується практичними спостереженнями.

Усі чисельні розрахунки виконані з використанням перевірених алгоритмів, що забезпечують:

– Точність обчислень у межах заданої похибки.

– Можливість повторюваності результатів у незалежних експериментах.

Таким чином, достовірність отриманих результатів підтверджується вибором коректних моделей, їхньою валідацією за допомогою експериментів, а також аналізом поведінки системи в різних умовах.

2.6 Наукова новизна роботи

Наукова новизна роботи полягає у створенні нового знання, яке відповідає сучасним запитам науки та практики, забезпечує структурно-функціональні зміни в розумінні проблеми та розширює можливості її вирішення. Новизна підтверджується обґрунтуванням відмінностей від існуючих підходів, а також ступенем оригінальності отриманих результатів.

Основні положення та результати, що складають новизну роботи було розглянуто нижче.

Новий об’єкт дослідження. Вперше комплексно досліджено архітектуру мережевої взаємодії для мультиплеєрних 3D-ігор на основі C#. Основна увага приділена оптимізації передачі даних, синхронізації станів та управління масштабованістю в реальних умовах.

Оригінальні математичні моделі. Запропонована модель мережевої взаємодії, що базується на теорії графів $G = (V, E)$: яка враховує різноманітні топології з додатковими вузлами для масштабування. Розроблено модель затримок у мережі, що розкладає загальну затримку L на три компоненти (поширення, передача, обробка), дозволяючи точно визначати їхній вплив на якість ігрового процесу.

Нова постановка задачі синхронізації станів. Удосконалено підхід до синхронізації ігрових станів між клієнтами та сервером за рахунок використання ймовірнісних методів прогнозування втрат пакетів та затримок. Розроблено метод корекції відхилень у станах на основі оцінки середньоквадратичної похибки між клієнтськими оновленнями.

Удосконалення критеріїв оцінки ефективності архітектури. Визначено нові метрики для оцінки продуктивності системи, включаючи стійкість до втрат даних, адаптивність до навантаження та якість ігрового процесу з боку користувача. Запропоновано показники оптимізації пропускної здатності на основі характеристик каналів зв’язку.

Програмна реалізація та алгоритмічні інновації. Розроблено прототип програмного забезпечення (ПЗ), що реалізує запропоновані математичні моделі та алгоритми взаємодії. Алгоритми адаптивної передачі даних і управління ресурсами забезпечують баланс між затримкою, втратами пакетів та продуктивністю.

Відмінності від відомих рішень та ступінь новизни було розглянуто нижче.

На відміну від класичних підходів, у роботі враховано специфіку C# як платформи для розробки мережевих ігор, що дозволяє використовувати її переваги у вигляді багато поточності та асинхронної взаємодії.

Запропоновані моделі вперше об’єднують структуру графу мережевої топології з ймовірнісними характеристиками передачі даних, що дає змогу оцінювати вплив параметрів мережі на різних рівнях деталізації.

Методи синхронізації станів розширюють відомі підходи шляхом інтеграції адаптивних методів прогнозування.

Ступінь наукової новизни було розглянуто нижче.

Вперше отримано – комплексний підхід до моделювання мережевої архітектури для мультиплеєрних 3D-ігор з урахуванням характеристик затримок, втрат даних та масштабованості.

Удосконалено – математичні моделі затримок і втрат пакетів у мережах із динамічним навантаженням.

Отримало подальший розвиток – принципи оцінки ефективності системи через нові критерії продуктивності та розробка ПЗ з практичним впровадженням.

Ці положення формують основу для захисту роботи, демонструючи її наукову цінність і практичну значущість.

2.7 Розробка основних алгоритмів програмного забезпечення

Після аналізу та створення математичних моделей для мультиплеєрних 3D-ігор, було розроблено основні алгоритми ПЗ, а саме блок-схеми роботи серверу та клієнта зображені на рисунку 2.2.

Рисунок 2.2 – Основні блок-схеми роботи серверу та клієнта

Також було розроблено блок-схеми підпроцесів сервера які зображені на рисунку 2.3.

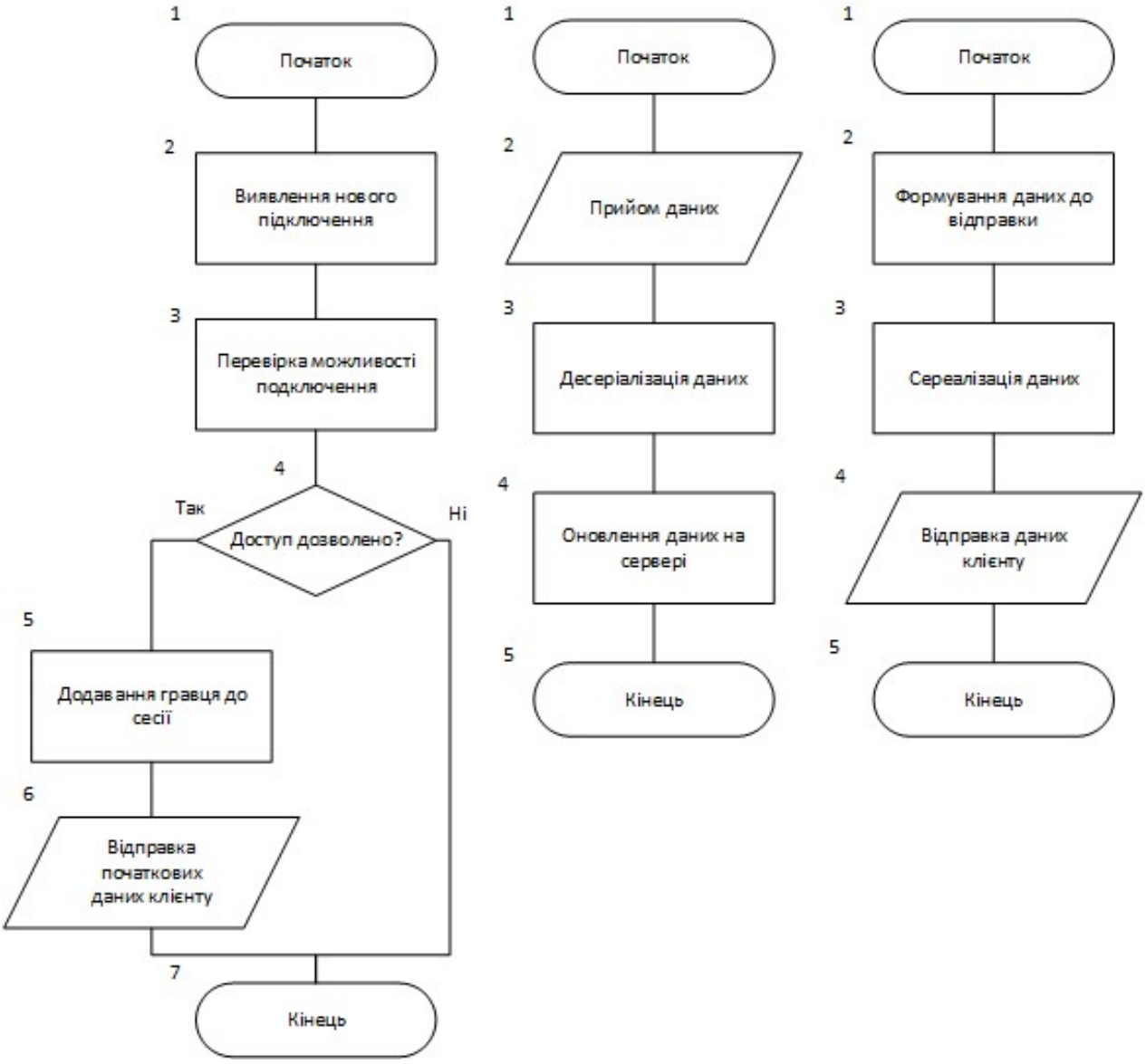

Рисунок 2.3 – Підпроцеси блок-схеми серверу

Перша блок-схема описує основний процес роботи серверу під час взаємодії з клієнтами. Друга блок-схема деталізує підпроцес прийому даних сервером. Третя блок-схема відображає підпроцес передачі даних від серверу до клієнта. Усі три блок-схеми разом демонструють загальний алгоритм функціонування серверу у мережевому середовищі.

Також було розроблено блок-схеми підпроцесів клієнта які зображені на рисунку 2.4.

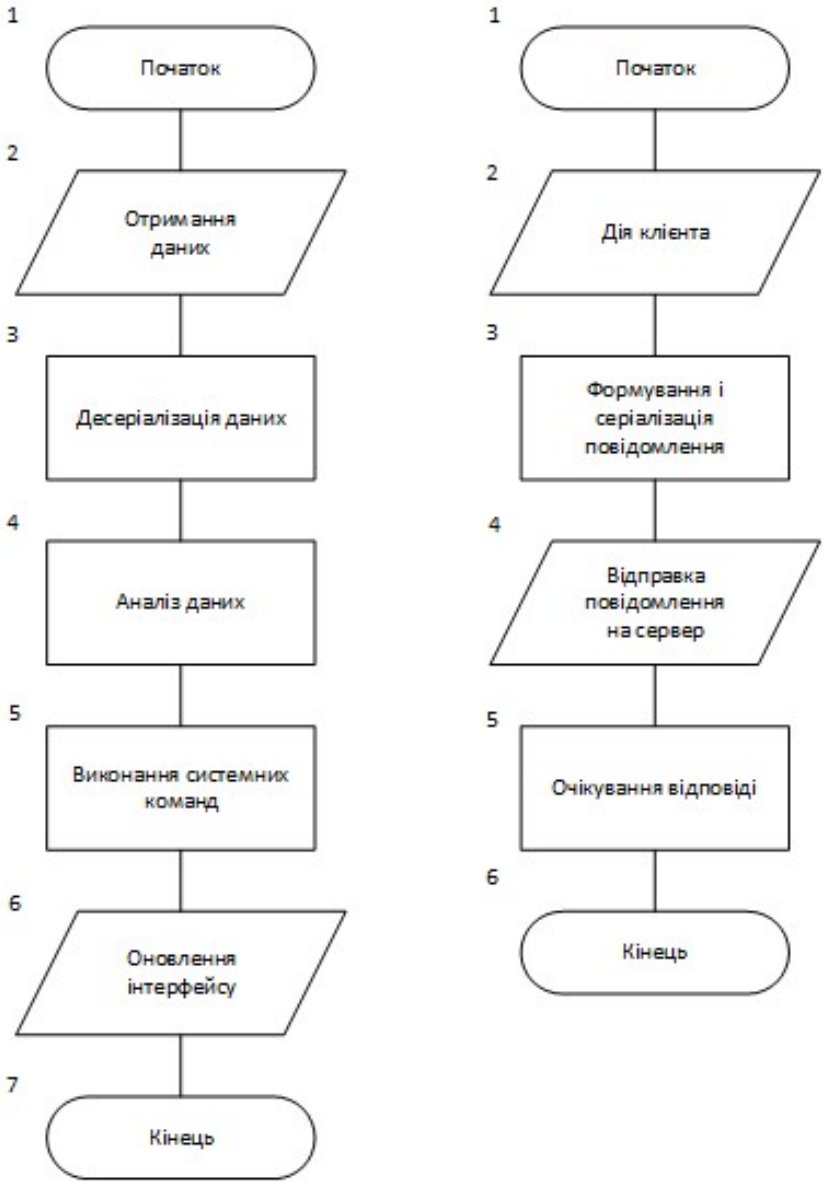

Рисунок 2.4 – Підпроцеси блок-схеми клієнта

Перша блок-схема демонструє основний алгоритм обробки даних клієнтом у мережевій взаємодії. Друга блок-схема відображає підпроцес обробки дій користувача на клієнтській стороні. Ці блок-схеми разом описують основні процеси взаємодії клієнта із сервером у контексті мультиплеєрної гри.

2.8 Розробка структурної і функціональної моделей програмного забезпечення

Після аналізу було розроблена структурна модель ПЗ яке зображено на рисунку 2.5 яка описує, які фрагменти потрібно реалізувати у вигляді підпрограм, та деталізація алгоритмів цих підпрограм.

Рисунок 2.5 – Структурна модель програми

Функціональна модель для сервера та його деталізацію зображені на рисунках 2.6 та 2.7.

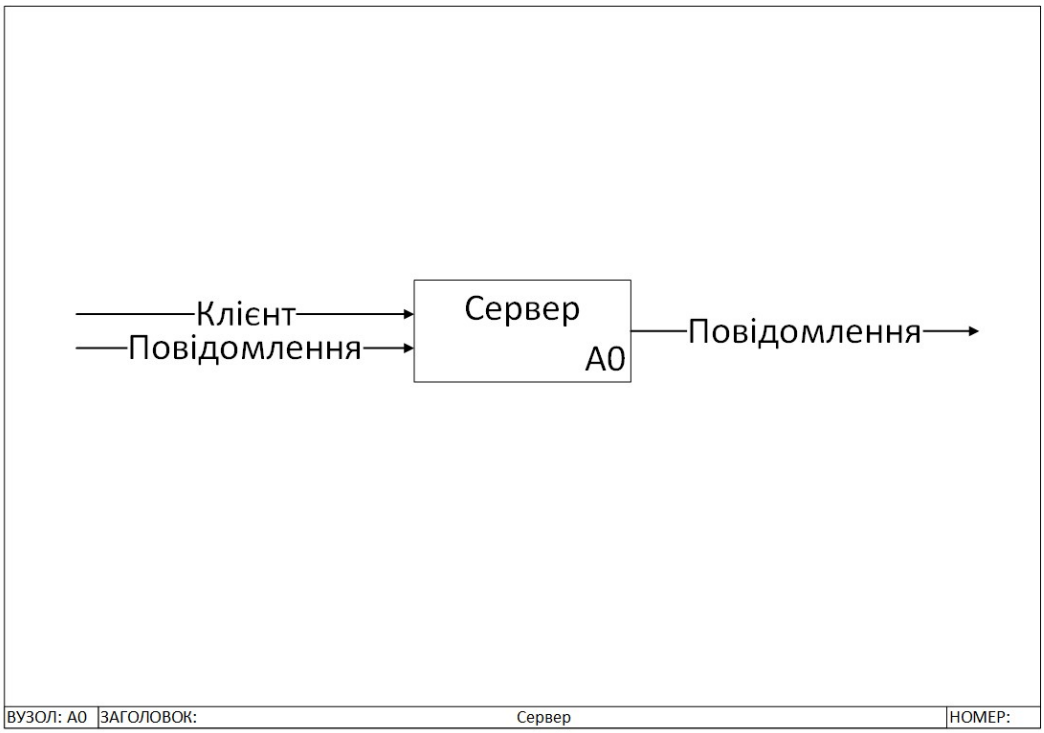

Рисунок 2.6 – Функціональна модель сервера

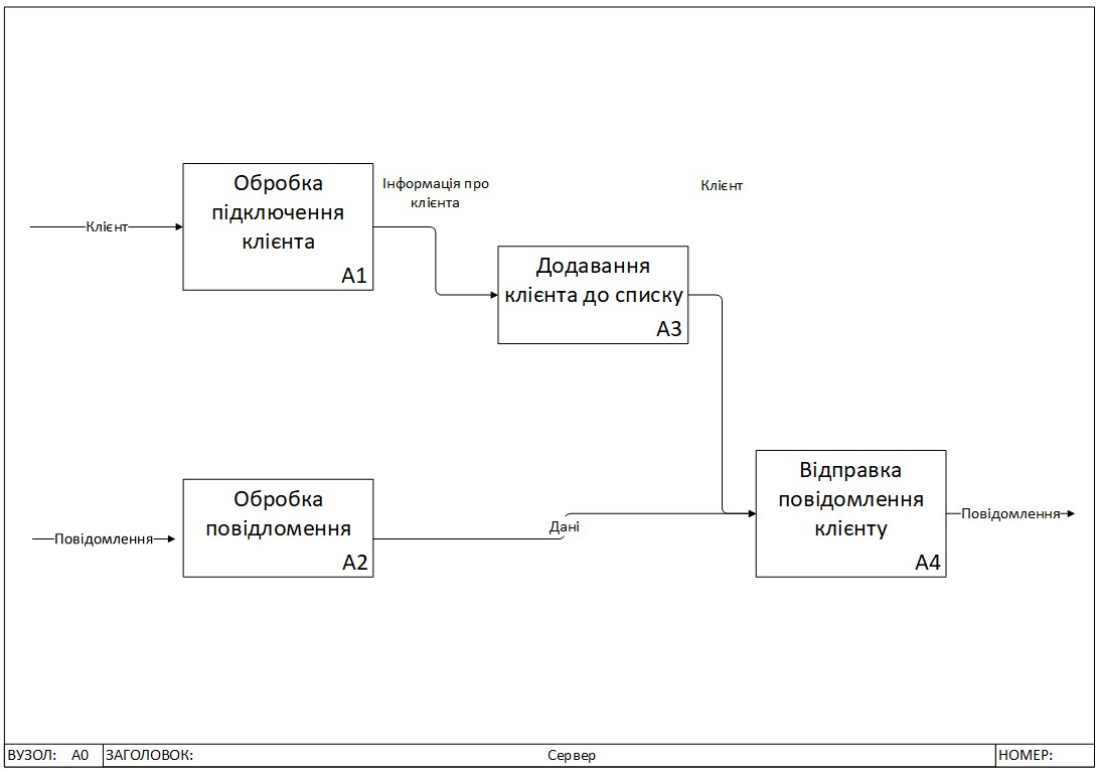

Рисунок 2.7 – Функціональна модель сервера детальна

Також була розроблена функціональна модель для клієнта та його деталізацію котрі зображені на рисунках 2.8 та 2.9.

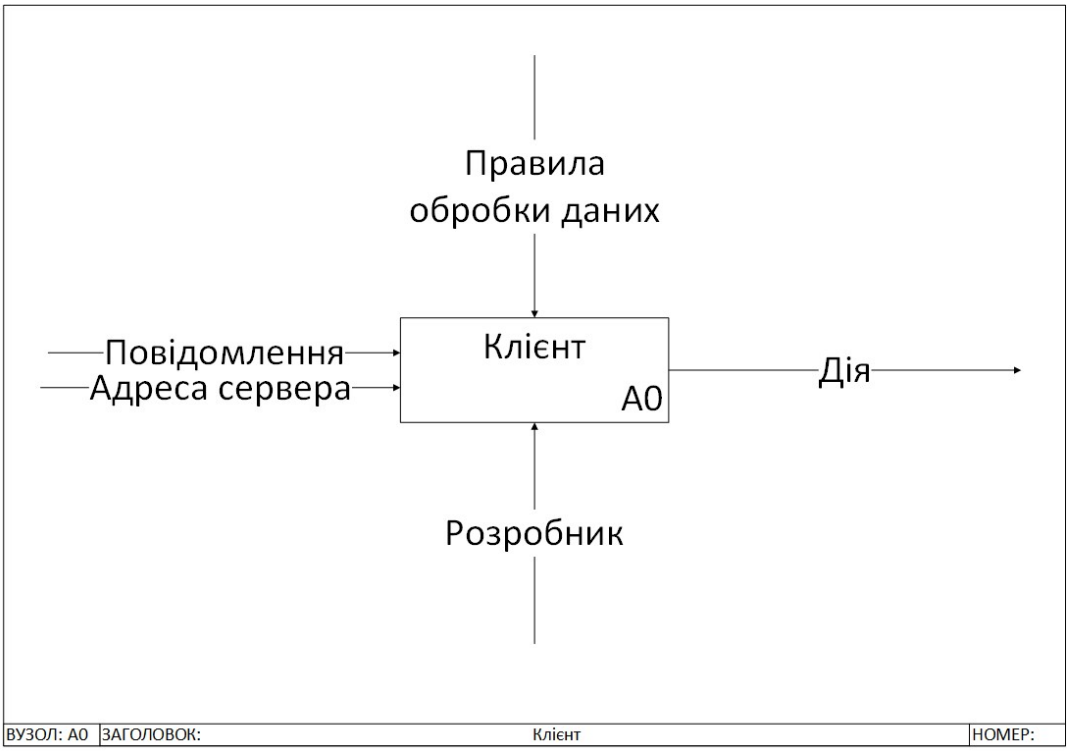

Рисунок 2.8 – Функціональна модель клієнта

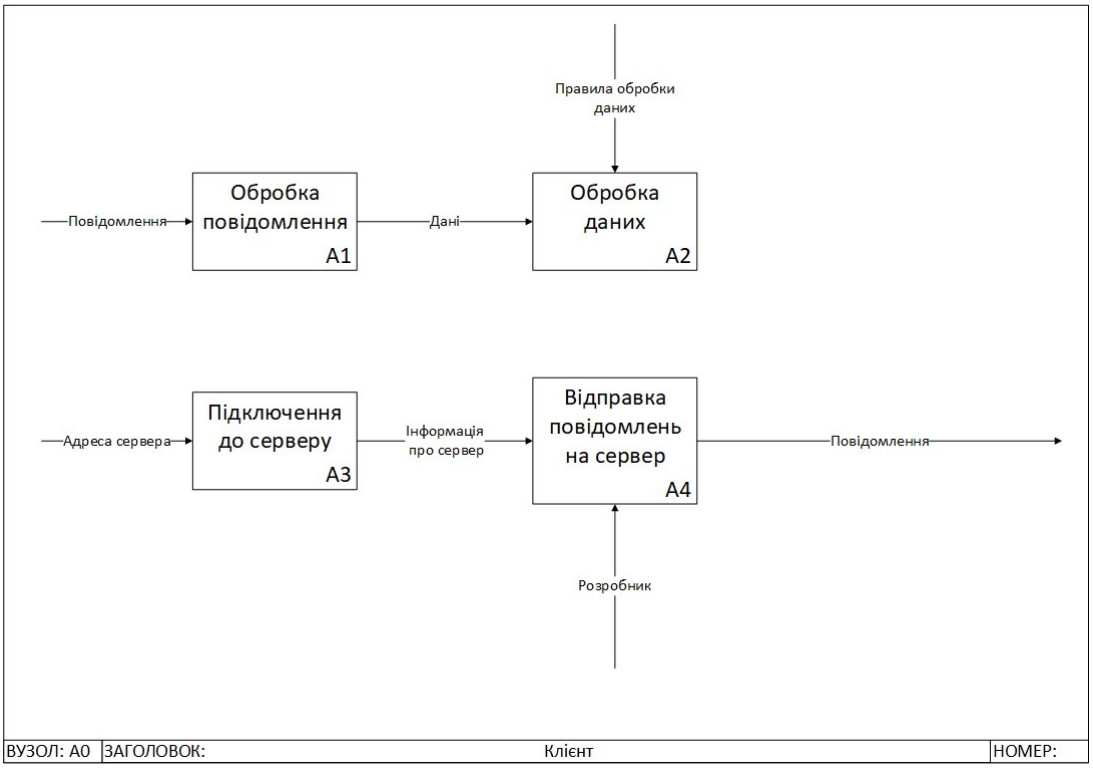

Рисунок 2.9 – Функціональна модель клієнта детальна

2.9 Розробка моделі інтерфейсу програмного забезпечення

Після аналізу та розробки функціональних та структурних схем було розроблено моделі інтерфейсу ПЗ які зображені на рисунках 2.10, 2.11, 2.12.

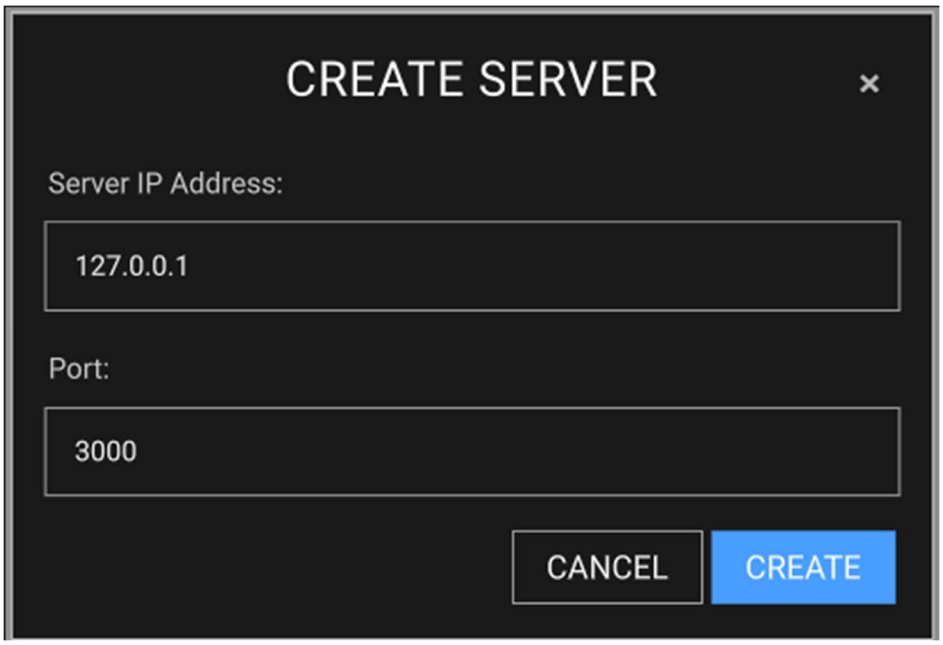

Рисунок 2.10 – Модель інтерфейсу для запуска сервера

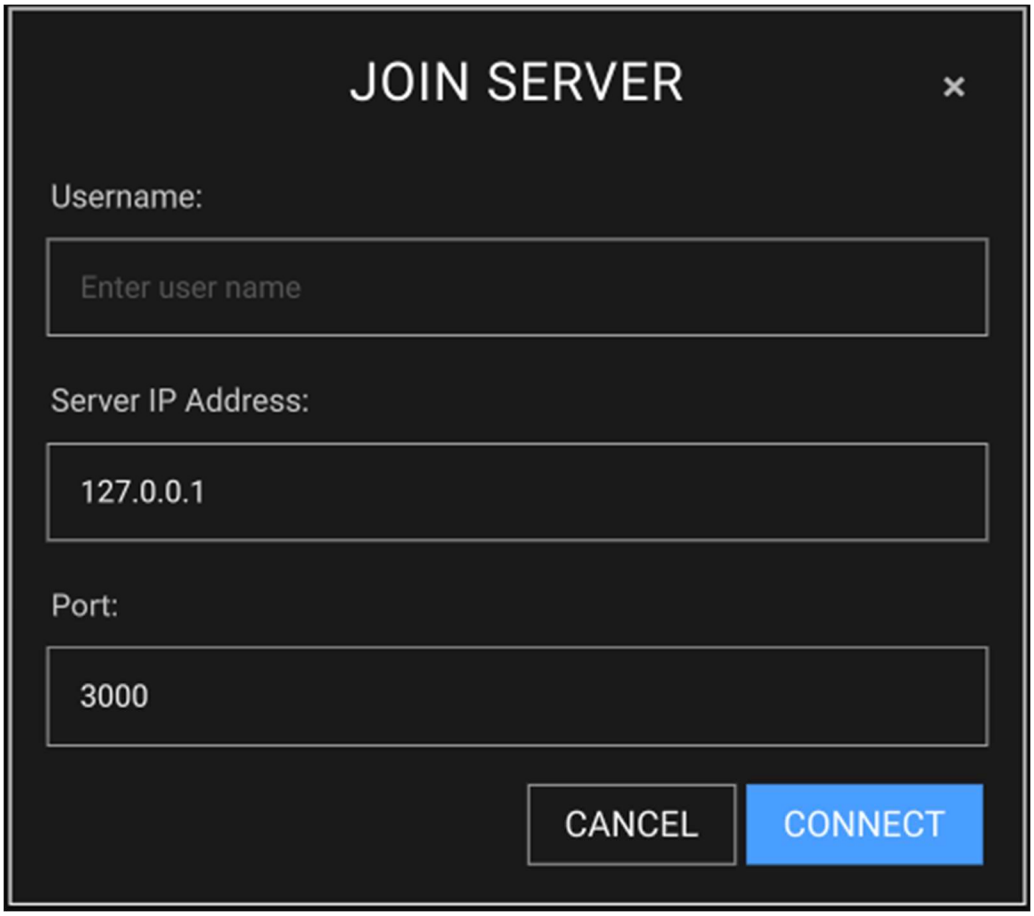

Рисунок 2.11 – Модель інтерфейсу клієнта для підключення

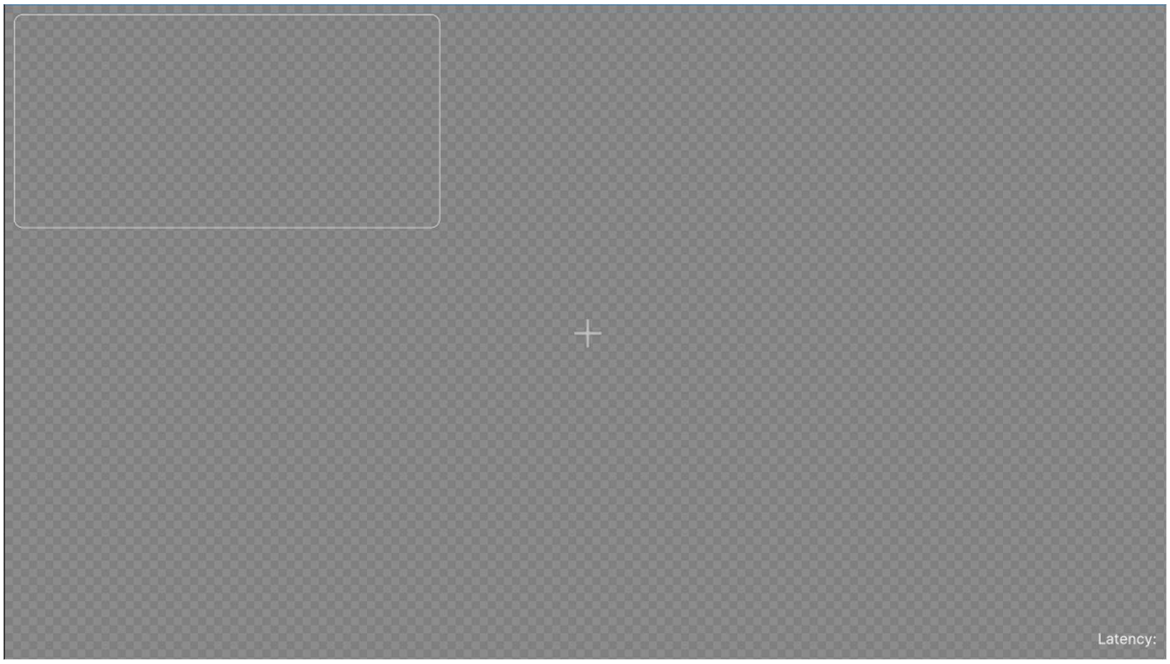

Рисунок 2.12 – Модель інтерфейсу клієнта на сервері

2.10 Розробка структур основних класів і їх наслідування

Після аналізу та розробки блок-схем ПЗ було розроблено UML діаграму класів яка зображена на рисунку 2.13.

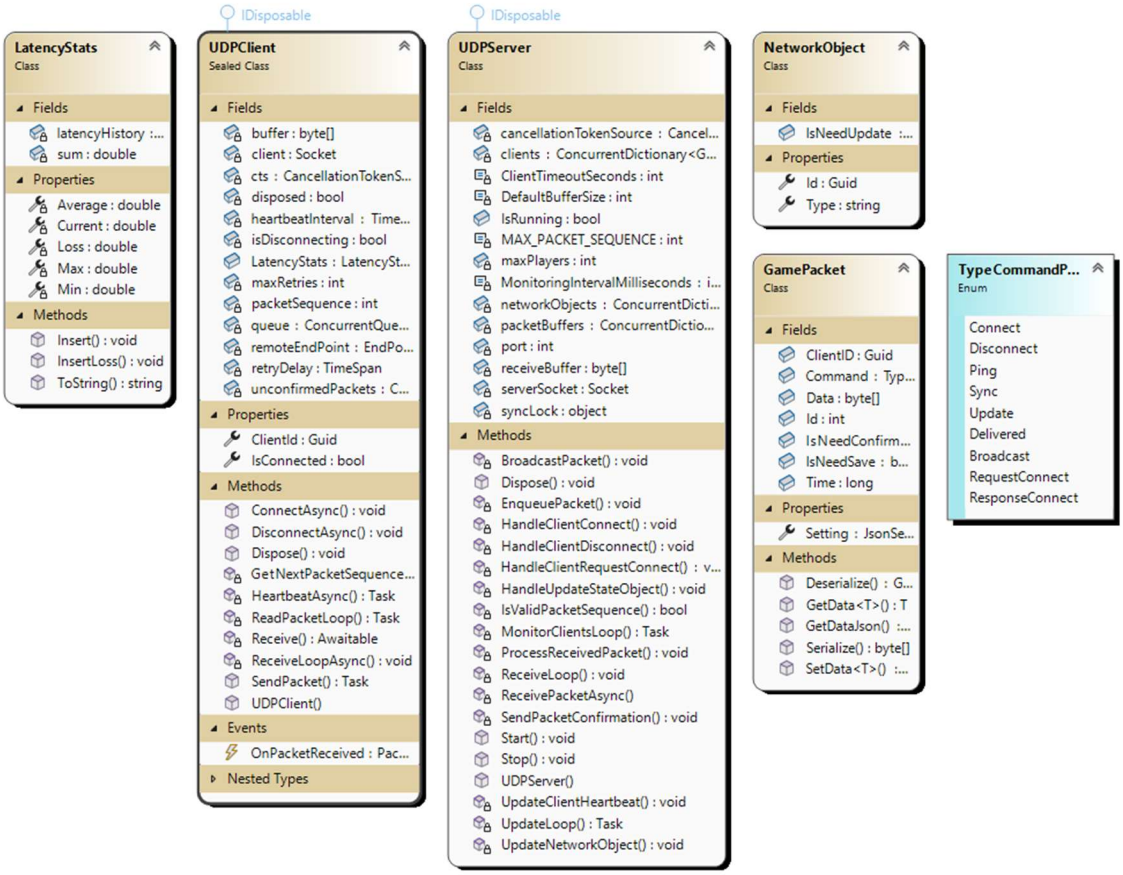

Рисунок 2.13 – Модель UML діаграми класів клієнта та сервера

Було розроблено п’ять класів та один перелік. Основними є два класи: UDPServer та UDPClient, які забезпечують функціонал створення сервера та клієнта, відповідно, для підключення до сервера і передачі даних.

UDPServer містить два публічних методи: Start і Stop, які відповідають за запуск та зупинку сервера.

UDPClient має три публічних методи: Connect, Disconnect, SendPacket, а також публічне поле LatencyStats, яке використовується для збереження та виведення інформації про затримки та втрати пакетів.

Крім основних, розроблено три допоміжних класи: LatencyStats, GamePacket і NetworkObject.

GamePacket представляє собою повідомлення, яке передається між клієнтом і сервером або у зворотному напрямку. Клас включає такі дані: тип пакета, ідентифікатор клієнта (id), час відправлення повідомлення, а також тіло повідомлення. Тіло повідомлення повинно бути представлено у вигляді класу, який успадковується від NetworkObject, що дозволяє зберігати об’єкти цього типу на сервері.

Таким чином, розроблена структура забезпечує ефективну організацію мережевої взаємодії між сервером і клієнтами, включаючи обробку затримок, втрат пакетів та передачу даних.

3 ПРИКЛАДНА ЧАСТИНА

3.1 Обґрунтування і вибір Unity Engine та Visual Studio 2022

Розробка архітектури мережевої взаємодії для мультиплеєрних 3D-ігор потребує вибору відповідних інструментів, які забезпечують високу продуктивність, гнучкість та зручність у використанні. Unity Engine та Visual Studio 2022 були обрані з причин перерахованих нижче.

Unity Engine є одним із найпопулярніших інструментів для розробки 3D-ігор, і його вибір обґрунтовується такими перевагами [8]:

– Підтримка C# – Unity Engine використовує C# як основну мову для написання скриптів. Це забезпечує просту інтеграцію з дослідженнями, орієнтованими на архітектуру мережевої взаємодії.

– Кросплатформеність – дозволяє створювати ігри для різних платформ (Windows, macOS, Android, iOS тощо), що є критичним для тестування мережевої архітектури у різних середовищах.

– Інтегровані мережеві бібліотеки – Unity надає доступ до стандартних бібліотек (Unity Transport, MLAPI, Netcode for GameObjects) та документацій, які спрощують розробку протоколів взаємодії між клієнтами та сервером.

– Моделювання мережевих умов – Unity дозволяє імітувати різні параметри мережі (затримки, втрати пакетів тощо), що важливо для тестування розробленої архітектури.

– Широкий інструментарій – інтеграція візуальних редакторів, анімації та системи фізики дозволяє зосередитися на розробці ігрового процесу, залишаючи базову функціональність платформі.

Unity також пропонує спільноту розробників та технічну підтримку, що полегшує вирішення технічних проблем та обмін знаннями [9].

Visual Studio 2022 є професійним інструментом для розробки, який ідеально поєднується з Unity Engine і C#. Його вибір обґрунтовано наступними характеристиками [10]:

– Інтеграція з Unity – Visual Studio 2022 має спеціальні плагіни для Unity, що забезпечують зручність написання та налагодження коду.

– Підтримка сучасних стандартів C# – Visual Studio 2022 підтримує останні версії C#, що дозволяє використовувати найсучасніші можливості мови для оптимізації мережевої архітектури.

– Розширені засоби налагодження – інструменти для профілювання, аналізу продуктивності та виявлення помилок є ключовими для роботи з асинхронними процесами та мережевими протоколами.

– Автоматизація та інтеграція – підтримка систем управління версіями (наприклад, Git) дозволяє організувати командну роботу та зберігати історію змін у проекті.

– Інтелектуальна підтримка – IntelliSense значно полегшує написання складного коду завдяки автоматичним підказкам, перевірці помилок і рекомендаціям.

Використання Unity у поєднанні з Visual Studio 2022 створює потужне середовище для розробки мультиплеєрних 3D-ігор:

– Зручне інтегроване середовище для розробки (IDE) сприяє швидкому циклу написання, тестування та налагодження коду.

– Єдина екосистема дозволяє концентруватися на специфічних задачах мережевої взаємодії, таких як оптимізація передачі даних або управління затримками, без відволікання на технічні деталі середовища.

– Можливість масштабування проекту від прототипу до фінального релізу завдяки широкому функціоналу обох інструментів.

Вибір Unity Engine та Visual Studio 2022 базується на їхній здатності забезпечити ефективну розробку, тестування і вдосконалення архітектури мережевої взаємодії, враховуючи вимоги сучасних мультиплеєрних 3D-ігор.

3.2 Розробка основних класів і програмних модулів

Після розробки блок-схем та UML діаграми було розроблені клас UDPServer, UDPClient, GamePacket, NetworkObject, LatencyStats, TypePacket.

Наведемо функції запуску та зупинки сервера у класі UDPServer:

```
public async void Start()
{
    Debug.Log($"Starting UDP server on port {port}");
    IsRunning = true;
    // Start monitoring and receive tasks
    await Task.WhenAll(Task.Run(ReceiveLoop), UpdateLoop(),
MonitorClientsLoop());
}

public void Stop()
{
    cancellationTokenSource.Cancel();
    serverSocket.Close();
    IsRunning = false;
 }
```

Наведемо функції ппідключення, відключення та відправки повідомлень на сервер у класу UDPServer на рисунку 3.1, 3.2, 3.3.

```
public async void ConnectAsync()
{
    if (IsConnected) return;

    var requestConnectPacket = new GamePacket
    {
        Command = TypeCommandPacket.RequestConnect,
        IsNeedSave = false,
        IsNeedConfirm = false
    };

    await SendPacket(requestConnectPacket);

    var result = await client.ReceiveFromAsync(buffer, SocketFlags.None, remoteEndPoint);
    var packet = GamePacket.Deserialize(buffer[..result.ReceivedBytes]);

    if (packet.Command == TypeCommandPacket.ResponseConnect)
    {
        var response = packet.GetData<string>();

        if (response == "OK")
        {
            Debug.Log("Allow connect to server");

            Task.WhenAll(HeartbeatAsync(cts.Token), Task.Run(() => ReceiveLoopAsync(cts.Token)), ReadPacketLoop(cts.Token));

            var connectPacket = new GamePacket
            {
                Command = TypeCommandPacket.Connect,
                IsNeedSave = false,
                IsNeedConfirm = false
            };

            await SendPacket(connectPacket);
        }
    }
}
```

Рисунок 3.1 – Метод підключення клієнта до сервера

```
public async void DisconnectAsync(CancellationToken cancellationToken = default)
{
    if (!IsConnected) return;

    try
    {
        var disconnectPacket = new GamePacket { Command = TypeCommandPacket.Disconnect };
        await SendPacket(disconnectPacket, cancellationToken);
    }
    catch (Exception ex)
    {
        Debug.LogException(ex);
    }
}
```

Рисунок 3.2 – Метод відключення клієнта від сервера

```
public async Task SendPacket(GamePacket packet, CancellationToken cancellationToken = default)
{
    if (packet == null)
    {
        throw new ArgumentNullException(nameof(packet));
    }
    packet.ClientID = ClientId;
    packet.Time = DateTime.UtcNow.Ticks;
    packet.Id = GetNextPacketSequence();
    var retryCount = 0;
    while (retryCount < maxRetries && !cancellationToken.IsCancellationRequested)
    {
        try
        {
            var data :byte[] = packet.Serialize();
            if (isDisconnecting) return;

            if (packet.Command == TypeCommandPacket.Disconnect)
            {
                isDisconnecting = true;
            }
            if (packet.Command != TypeCommandPacket.Ping && packet.IsNeedConfirm)
            {
                unconfirmedPackets[packet.Id] = (packet, DateTime.UtcNow);
            }
            await client.SendToAsync( data, SocketFlags.None, remoteEndPoint);
            return;
        }
        catch (Exception ex) when (ex is not OperationCanceledException)
        {
            retryCount++;
            Debug.LogException(ex);

            if (retryCount < maxRetries)
            {
                await Task.Delay(retryDelay, cancellationToken);
            }
        }
    }
    throw new TimeoutException($"Failed to send packet after {maxRetries} attempts");
}
```

Рисунок 3.3 – Метод відправки повідомлення на сервер

Структура класу NetworkObject:

```
public class NetworkObject
{   public Guid Id { get; set; } = Guid.NewGuid();
    public string Type => GetType().ToString();
    public bool IsNeedUpdate = true;
}
```

Структура класу GamePacket:

```
public class GamePacket
{
    public int Id;
    public Guid ClientID;
    public TypeCommandPacket Command;
    public long Time;
    public byte[] Data;
    public bool IsNeedSave = true;
    public bool IsNeedConfirm = true;
}
```

Структура класу LatencyStats зображено на рисунку 3.4.

```
public class LatencyStats
{
    2 references
    private double Current { get; set; }
    2 references
    private double Loss { get; set; }
    1 reference
    private double Average => latencyHistory.Count > 0 ? sum / latencyHistory.Count : 0;
    5 references
    private double Min { get; set; } = double.MaxValue;
    5 references
    private double Max { get; set; } = double.MinValue;

    private readonly List<double> latencyHistory = new();
    private double sum = 0;

    1 reference
    public void Insert(double newLatency)
    {
        Current = newLatency;
        latencyHistory.Add(newLatency);
        sum += newLatency;

        if (newLatency < Min) Min = newLatency;
        if (newLatency > Max) Max = newLatency;

        if (latencyHistory.Count > 100)
        {
            double oldestLatency = latencyHistory[0];
            sum -= oldestLatency;
            latencyHistory.RemoveAt(index: 0);

            // Recalculate Min and Max only if we removed the current Min or Max
            if (oldestLatency == Min || oldestLatency == Max)
            {
                Min = latencyHistory.Count > 0 ? latencyHistory.Min() : double.MaxValue;
                Max = latencyHistory.Count > 0 ? latencyHistory.Max() : double.MinValue;
            }
        }
    }
    1 reference
    public void InsertLoss(double value) => Loss = value;
    1 reference
    public override string ToString() =>
        $"Latency (ms) - Current: {Current:F1}, Avg: {Average:F1}, Min: {Min:F1}, Max: {Max:F1}, Loss: {Loss} packets";
}
```

Рисунок 3.4 – Клас LatencyStats

3.3 Розробка інтерактивної системи користувальницького інтерфейсу

Було розроблено користувацький інтерфейс для створення та підключення до серверу та взаємодіє з об’єктами на сервері, вони зображені на рисунках 3.5, 3.6, 3.7.

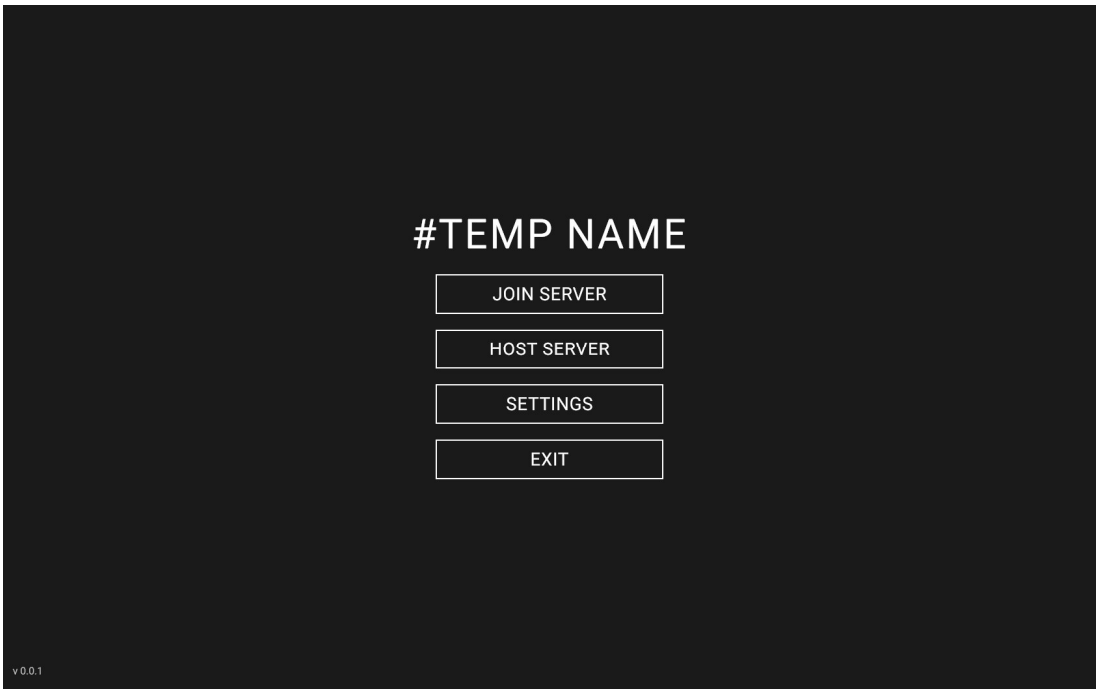

Рисунок 3.5 – Інтерфейс головного меню

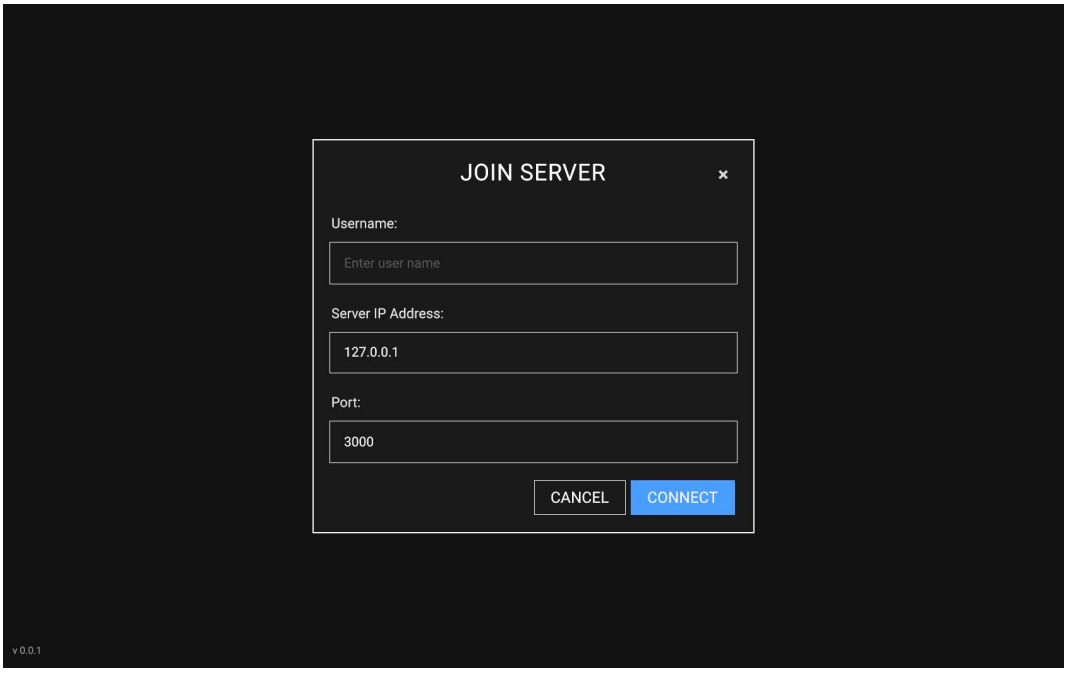

Рисунок 3.6 – Інтерфейс підключення клієнта до сервера

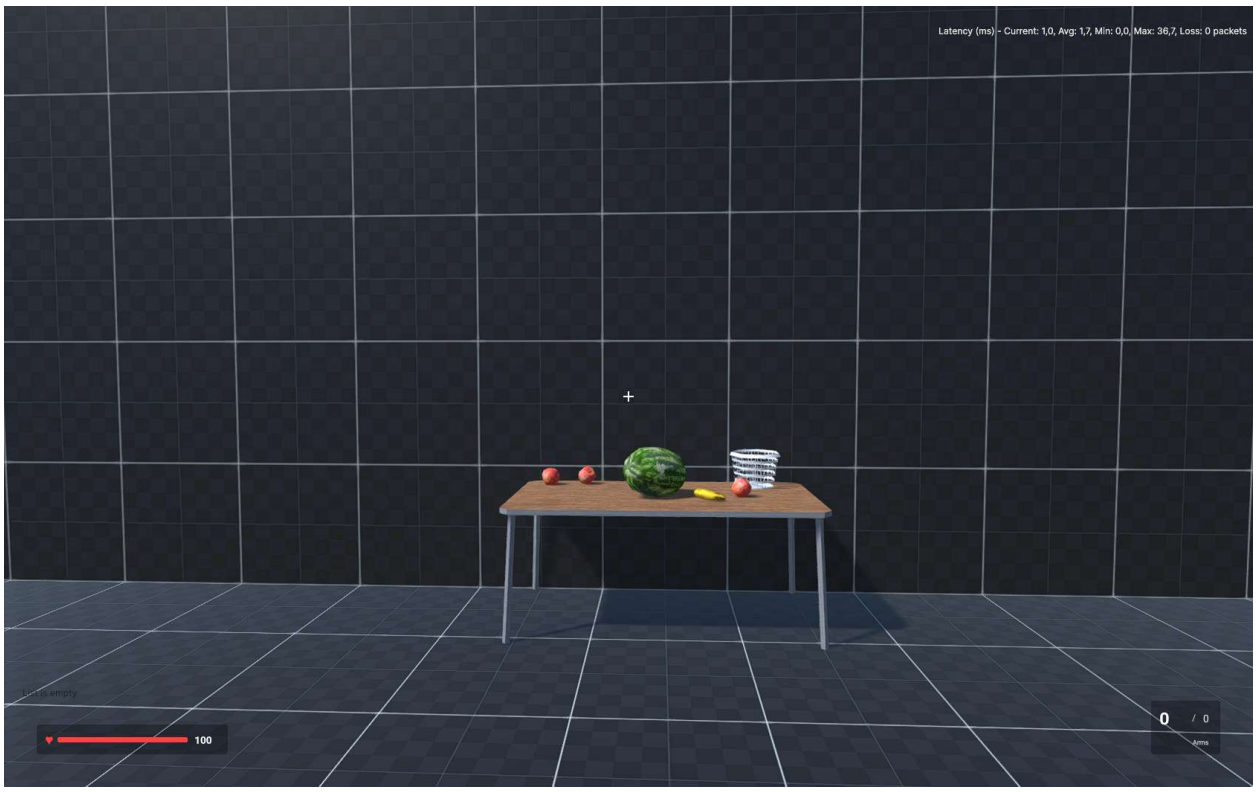

Рисунок 3.7 – Інтерфейс клієнта на сервері

3.4 Розробка керівництва користувача програмного забезпечення

Даний програмний продукт створений для демонстрації мережевої взаємодії у мультиплеєрних 3D-іграх. Його функціонал дозволяє:

– Підключатися до серверів.

– Запускати сервер для гри.

– Переглядати статистику затримок та втрат пакетів у мережі.

– Взаємодіяти із віртуальним середовищем у режимі реального часу.

На головному екрані програми представлені наступні кнопки:

– «JOIN SERVER» – використовується для підключення до існуючого сервера.

– «HOST SERVER» – дозволяє створити новий сервер для гри.

– «SETTINGS» – відкрити налаштування програми.

– «EXIT» – вихід із програми.

Виберіть потрібний пункт меню, натиснувши на відповідну кнопку.

При натисканні на «JOIN SERVER» відкривається діалогове вікно підключення до сервера. Поля для введення:

– Username – введіть ім'я користувача, яке буде відображатися в грі.

– Server IP Address – IP-адреса сервера для підключення.

– Port – порт для підключення до сервера.

Кнопки:

– CONNECT – підключення до сервера.

– CANCEL – повернення до головного меню.

Заповніть усі поля відповідною інформацією. Натисніть CONNECT, щоб підключитися, або CANCEL, щоб скасувати.

При успішному з'єднанні із сервером користувачу надається доступ до віртуального середовища, де можна розпочати взаємодію з віртуальним середовищем та взаємодіяти з об’єктам в ньому. Основні елементи інтерфейсу зображені на рисунку 3.8.

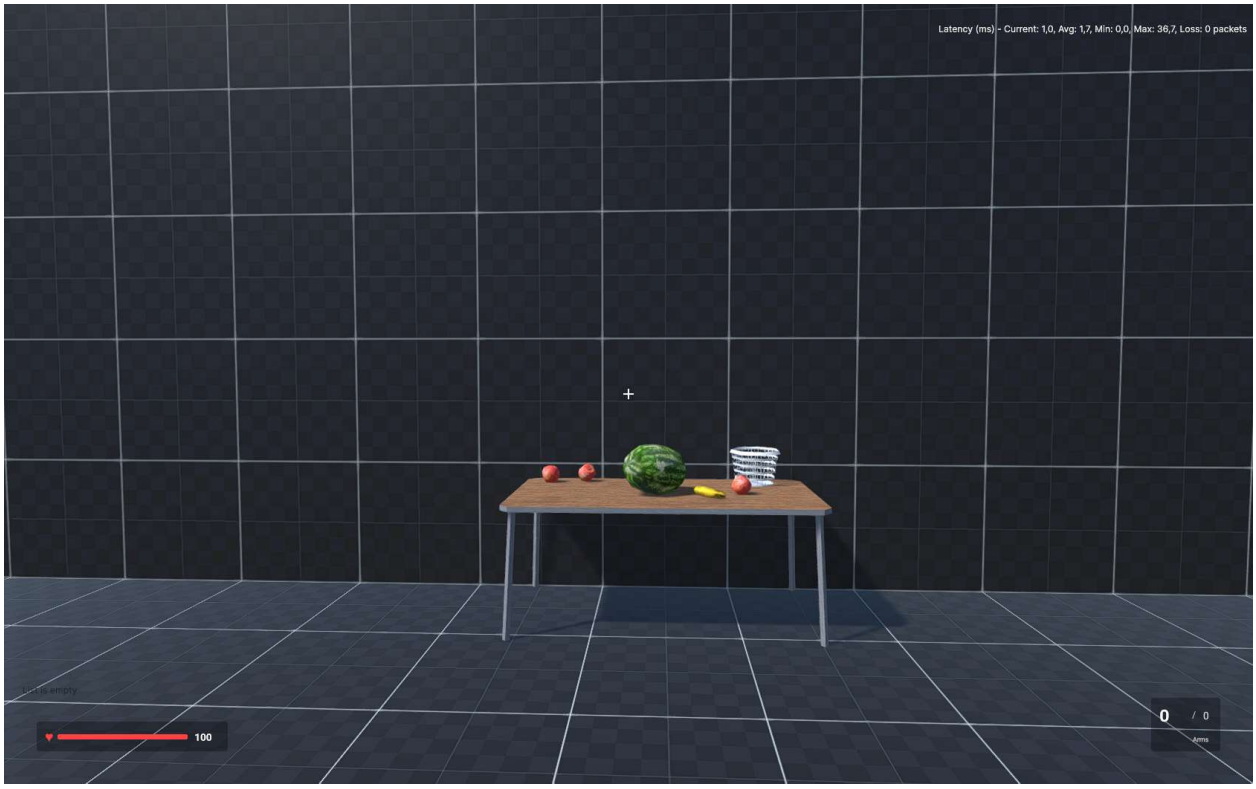

Рисунок 3.8 – Віртуальне середовище

Чат (у нижньому лівому куті) для спілкування з іншими гравцями. Статистика мережі (у верхньому правому куті) показує поточний стан затримок і втрат пакетів.

Взаємодія з об'єктами – користувач може взаємодіяти з об'єктами на столі, наприклад, переміщати їх. Приклад переміщення об’єкту іншим гравцем зображено на рисунку 3.9.

Рисунок 3.9 – Переміщена ігровий об’єкт іншим гравцем

Наведіть курсор на об'єкт і та натисніть ліву кнопку миші.

У створеному програмному забезпеченні реалізовано функцію малювання на віртуальній дошці. Процес малювання здійснюється шляхом наведення курсору миші на робочу поверхню дошки та утримування натиснутою лівої кнопки миші. Така інтуїтивна система дозволяє користувачам вільно створювати різноманітні зображення та замітки безпосередньо на екрані, що значно розширює можливості взаємодії з програмою та підвищує її функціональність у контексті візуальної комунікації. Приклад малювання зображений на рисунку 3.10.

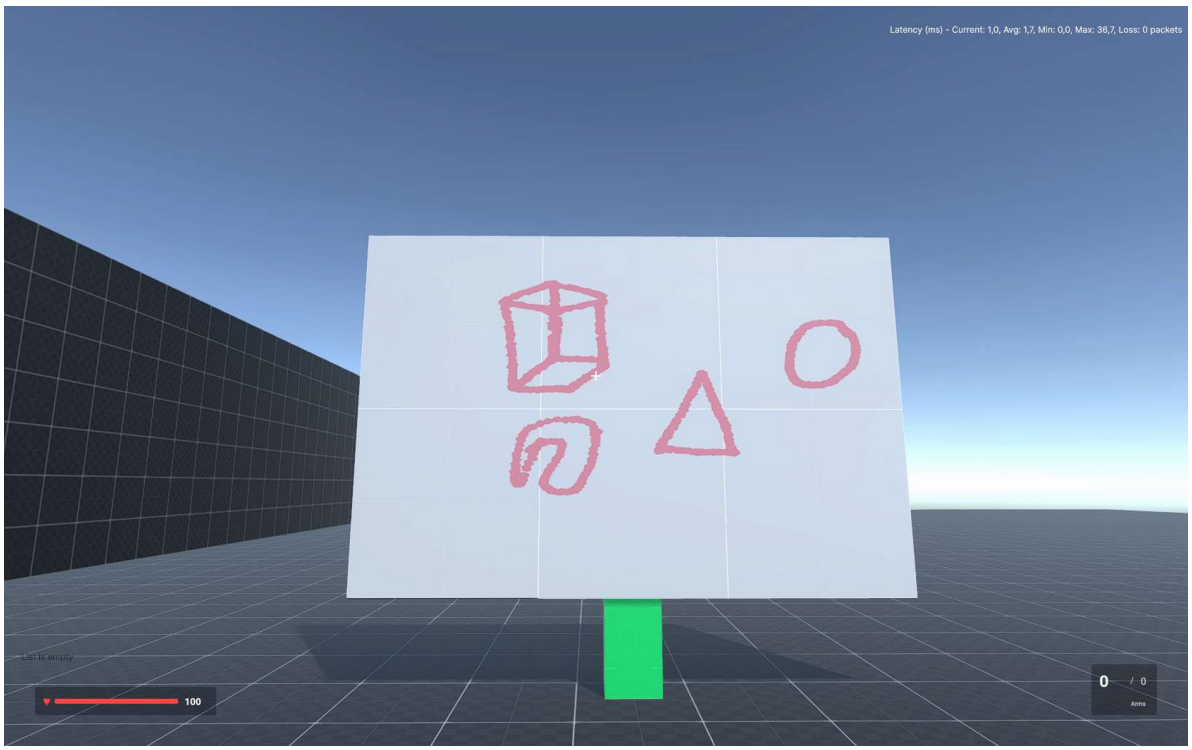

Рисунок 3.10 – Дошка для малювання

Ще є можливість взаємодій через кнопки, для активацій кнопку потрібно навести мишу на кнопку та натиснути ліву кнопку миші. Приклад натискання та активацій платформи зображено на рисунку 3.11 та 3.12.

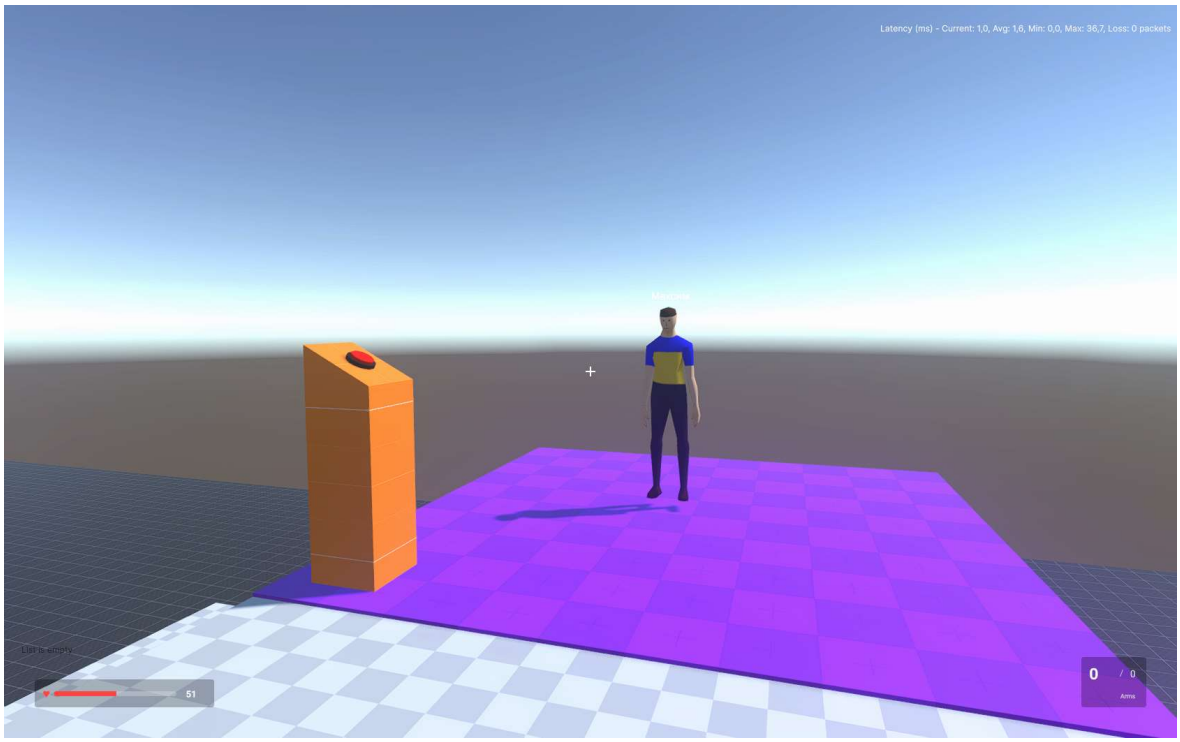

Рисунок 3.11 – Кнопка

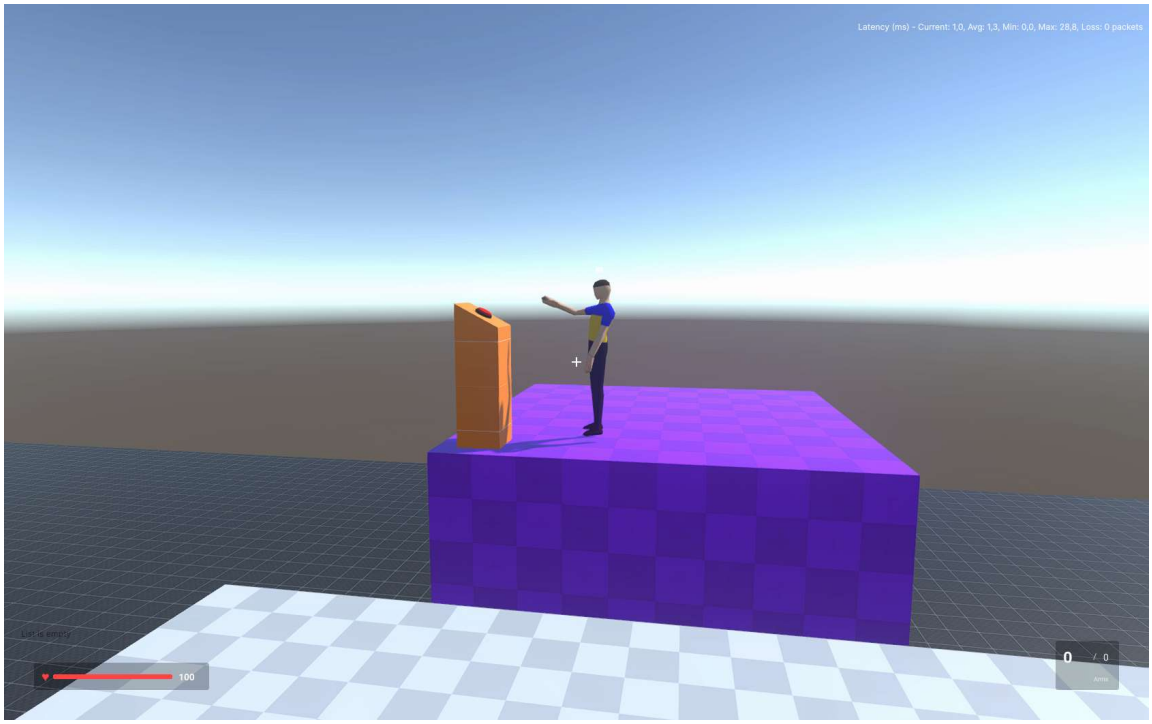

Рисунок 3.12 – Натискання кнопки іншим гравцем

Також ви можете відчинити двері перемістивши ключ у спеціальний відсік, для цього спочатку потрібно натиснути лівою кнопкою миші на ключ якій зображений на рисунку 3.13.

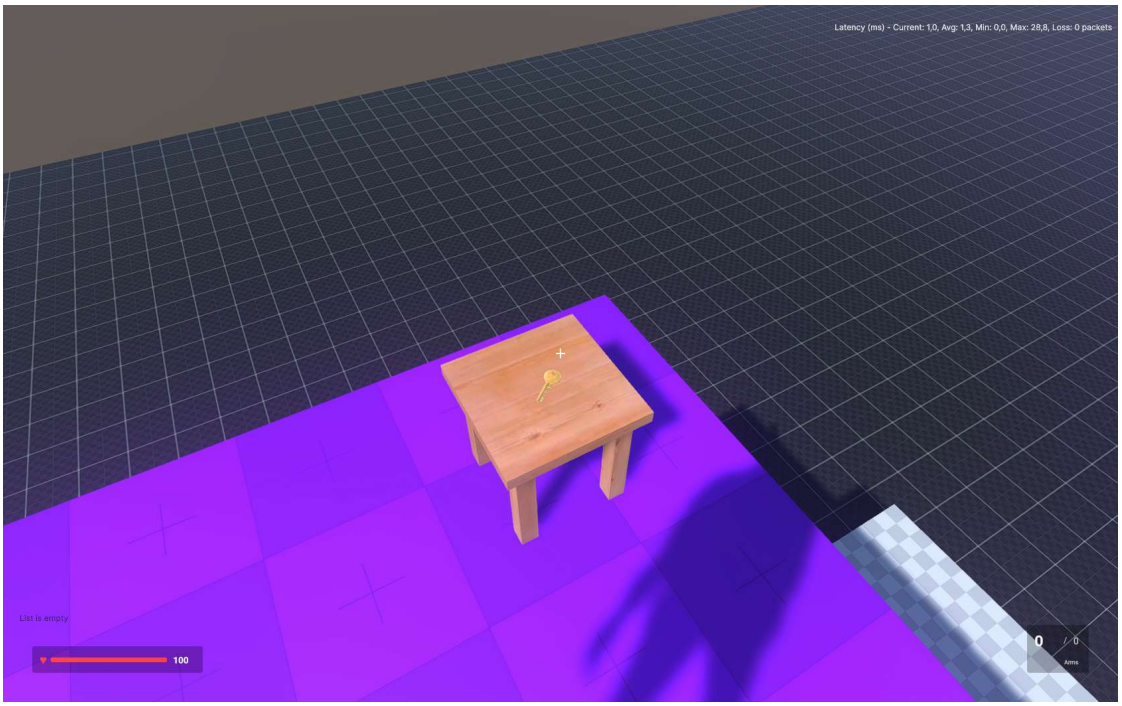

Рисунок 3.13 – Ключ

Далі потрібно ще раз натиснути ліву кнопку миші щоб відпустити ключ у відсік, який зображений на рисунку 3.14

Рисунок 3.14 – Відсік для ключа

Після того як ключ впаду у відсік, відкриються двері справа від гравця. Приклад відкритих дверей зображений на рисунку 3.15

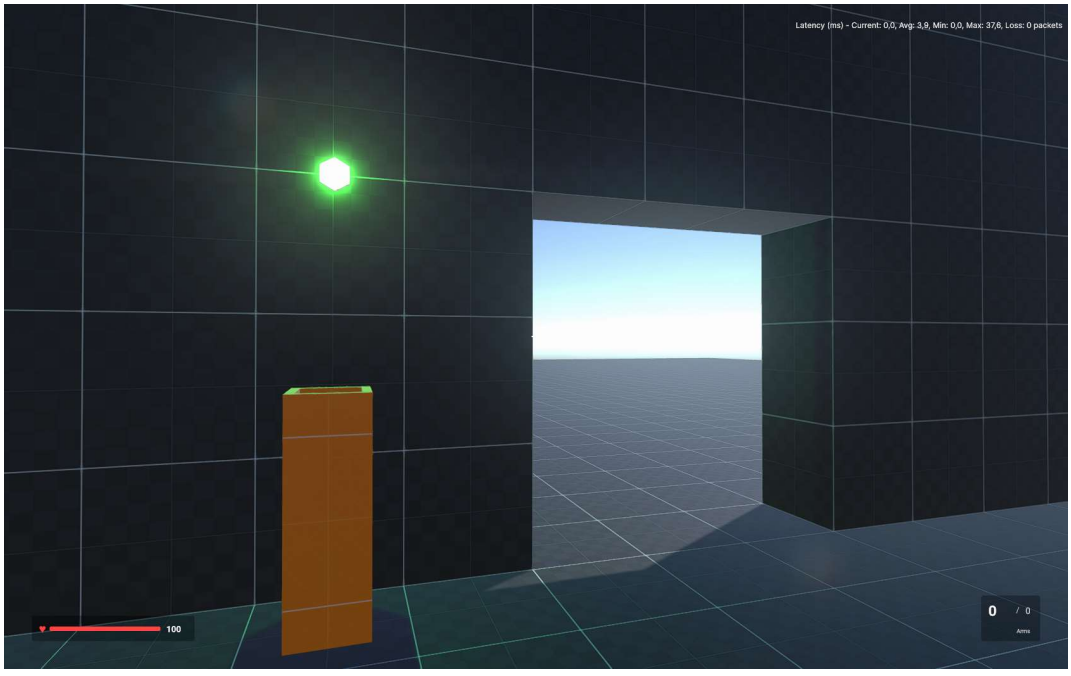

Рисунок 3.15 – Відкриті двері

Ще є можливість відправляти повідомлення у чат. Для цього потрібно натиснути кнопку «T» на клавіатурі, після чого відкриється поле для вводу тексту. Для відправки повідомлення введіть текст та натисніть кнопку «Enter», після чого воно з’явиться у чаті. Приклад повідомлень зображено на рисунку 3.16.

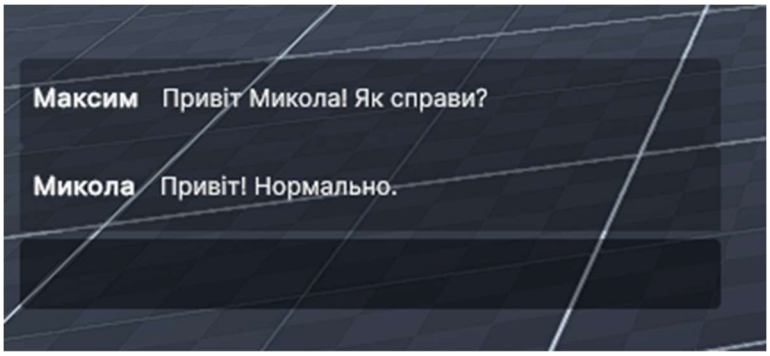

Рисунок 3.16 – Відкриті двері

Для виходу з програми натисніть кнопку EXIT у головному меню.

4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ

4.1 Методики проведення дослідження

Для проведення дослідження було поставлено такі задачі:

– Розробити та змоделювати архітектуру мережевої взаємодії для мультиплеєрних 3D-ігор.

– Провести тестування мережевих затримок, втрат пакетів та ефективності передачі даних.

– Обробити отримані результати для оцінки продуктивності та стабільності мережевої взаємодії.

У процесі роботи було розроблено методику, яка включала кроки описані нижче.

Вибір інструментів для моделювання:

– Обрано Unity Engine для створення середовища 3D-ігри.

– Використано C# для програмування мережевої логіки.

– Тестове середовище моделювання було створене для роботи з протоколом UDP.

Створення тестового середовища:

– Було реалізовано сервер (клас UDPServer) та клієнт (клас UDPClient).

– Розроблено тестові сценарії передачі даних між клієнтом і сервером.

– Відправка тестових повідомлень (GamePacket) із зазначенням часу відправки.

Візуалізація даних:

– У віртуальному середовищі відображено статистику затримок, втрат пакетів та інших метрик для кожного користувача.

– Для тестових цілей додано прості 3D-об'єкти, з якими гравці могли взаємодіяти.

Для оцінки ефективності роботи мережі проведено серію вимірювань описані нижче.

Затримка мережі (Latency) – вимірювалась різниця між часом відправки повідомлення та його отриманням. Дані зберігались у полі LatencyStats.

Зразок результатів в таблиці 4.1 на основі якої розраховується затримка пакетів.

Таблиця 4.1 – Зразок таблиці з результатами тестування

ID пакету	Час відправки (мс)	Час отримання (мс)	Затримка (мс)
1	47500531	47500533	2
2	47500534	47500537	3
3	47500541	47500545	4
4	47500546	47500547	1
5	47500550	47500552	2
6	47500553	47500555	2
7	47500556	47500559	3

У процесі дослідження була впроваджена комплексна система моніторингу втрат пакетів даних, яка базувалася на використанні унікальних ідентифікаторів для кожного повідомлення. Механізм роботи полягав у тому, що при відправці повідомлення воно додавалося до спеціального списку моніторингу, після чого система відстежувала надходження підтвердження про успішну доставку від серверної частини.

Обробка та аналіз отриманих результатів здійснювалися за кількома ключовими напрямками. По-перше, проводився систематичний збір статистичних даних з усіх клієнтських пристроїв, що брали участь у тестуванні. По-друге, на основі зібраних даних створювалися детальні графічні представлення, які відображали залежність часових затримок від поточного навантаження на мережу. По-третє, виконувався глибокий аналіз втрат пакетів з урахуванням різних параметрів передачі даних, включаючи швидкість передачі та розмір пакетів, що дозволило оптимізувати процес передачі інформації.

Зразок оброблених даних відображено на рисунку 4.1.

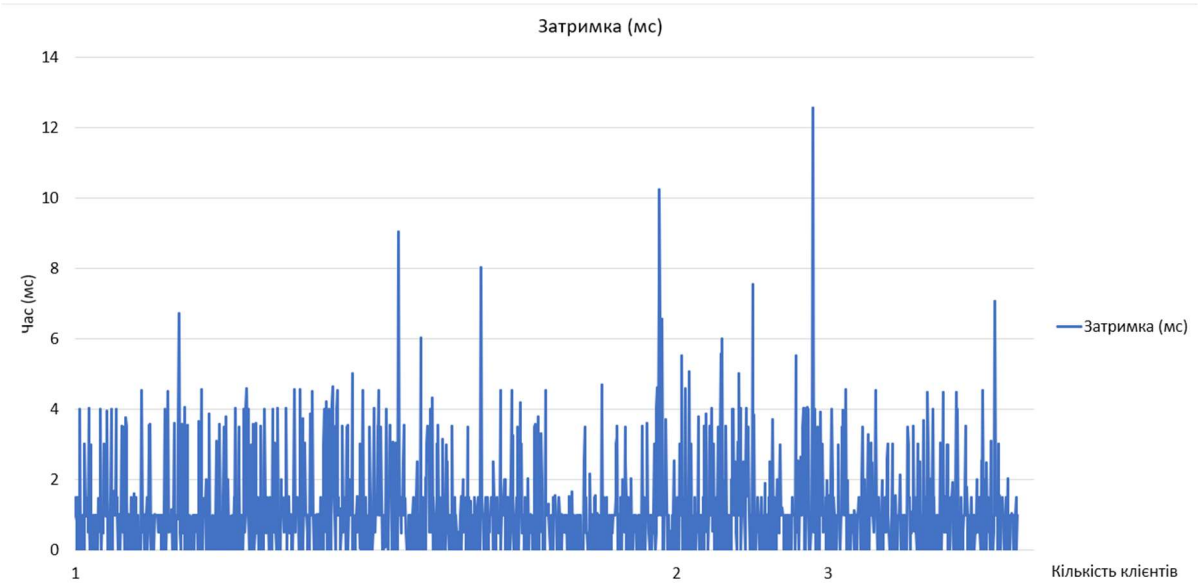

Рисунок 4.1 – Графік залежності затримки від кількості клієнтів

На основі отриманих даних було проведено порівняльний аналіз, та виявлено, що середня затримка не перевищувала 2 мілісекунди (мс), а втрати пакетів складали 1-2% при умовах імітації високого навантаження.

4.2 Залежності між параметрами об’єкту дослідження професійної області

Об'єктом дослідження виступає архітектура мережевої взаємодії в мультиплеєрних 3D-іграх, яка аналізується через такі параметри, як затримка передачі даних, втрати пакетів, пропускна здатність мережі, кількість активних клієнтів і обсяг переданих даних. Затримка передачі визначає час, необхідний для доставки пакету між клієнтом і сервером. Втрати пакетів характеризуються відсотком інформації, яка не досягла адресата, а пропускна здатність мережі описує обсяг даних, що можуть бути передані за одиницю часу. Кількість активних клієнтів відображає кількість одночасних підключень до сервера, тоді як обсяг переданих даних визначає розмір повідомлень між клієнтом і сервером.

Дослідження показало, що зі збільшенням кількості підключених клієнтів затримка передачі даних зростає через підвищене навантаження на сервер і мережу. Втрати пакетів мають тенденцію до зростання при зниженні пропускної здатності через перевантаження каналу передачі. Водночас, збільшення розміру переданих пакетів сприяє ефективнішому використанню пропускної здатності, однак це може призвести до підвищення ризику втрат даних. Залежність між затримкою та втратами пакетів демонструє, що зі зростанням затримки підвищується ймовірність втрат пакетів через збої в доставці інформації.

Розроблені залежності спрямовані на оптимізацію роботи сервера, що дозволяє забезпечувати стабільне з'єднання навіть при високому навантаженні. Це також дає можливість проводити точні розрахунки необхідної пропускної здатності для заданої кількості клієнтів і визначати межі допустимого навантаження на мережу при розробці мультиплеєрних ігор. Отримані результати формують основу для подальшого вдосконалення архітектури мережевої взаємодії та забезпечення її стабільної роботи в умовах реальних мережевих обмежень.

4.3 Аналіз адекватності моделей, що були розроблені

Базова модель описує архітектуру мережевої взаємодії між сервером і клієнтами як граф $G = (V, E)$.

Сильні сторони – використання графової моделі є адекватним і виправданим, оскільки вона чітко відображає структуру мережевої взаємодії, дозволяючи враховувати параметри каналів зв'язку та топологію.

Обмеження – модель не враховує динамічних змін у мережі (наприклад, зміни трафіку чи пропускної здатності через зростання кількості клієнтів). Це може призводити до розходження між симуляцією і реальними умовами.

Модель затримки визначається як сума трьох компонентів $L = L_{propagation} + L_{transmission} + L_{processing}$.

Сильні сторони – модель враховує ключові параметри затримки, що відповідає реальній структурі мережевої взаємодії.

Обмеження:

– У моделі не враховується затримка, викликана чергами в маршрутизаторах чи серверах, що може бути суттєвим у навантажених мережах.

– Параметри часто змінюються залежно від конкретного обладнання та середовища, що потребує більш гнучкого підходу до їх визначення.

Втрати пакетів описуються розподілом Бернуллі $P_{loss}(e) = 1 - P_{success}(e)$.

Сильні сторони – розподіл Бернуллі є адекватною математичною моделлю для випадкових втрат пакетів у мережах із низьким рівнем навантаження.

Обмеження:

– Модель є спрощеною і не враховує кореляції між втратами (наприклад, якщо один пакет втрачено, ймовірність втрати наступного може бути вищою через перевантаження).

– Не враховуються механізми повторної передачі пакетів, які впливають на ефективність передачі даних.

Розроблені моделі адекватно відображають основні процеси мережевої взаємодії, але мають спрощення, які знижують точність у реальних умовах. Моделі добре підходять для початкового етапу симуляції та оцінки мережевих параметрів.

4.4 Дослідження технічної ефективності розроблених моделей і програмних блоків

Для оцінки ефективності було проведено тестування за такими ключовими критеріями:

а) Продуктивність передачі даних:

1) Оцінка затримки передачі (L) у різних умовах мережі.

2) Тестування швидкості передачі пакетів (пропускна здатність каналу).

б) Втрати пакетів:

1) Визначення частоти втрат пакетів залежно від навантаження на мережу.

в) Синхронізація станів:

1) Аналіз точності передбачення станів об’єктів на основі моделі синхронізації.

г) Програмні блоки:

1) Тестування функцій UDPServer і UDPClient, включаючи методи Start, Stop, Connect, Disconnect, SendPacket.

2) Перевірка роботи LatencyStats, обчислення середніх затримок і пакетних втрат.

Було розроблені сценарій тестування моделей і ПЗ:

– Локальна мережа – сервер і клієнти підключені через локальну мережу (LAN).

– Глобальна мережа (WAN) – тестування за умов реального інтернет-з’єднання з додатковими затримками та втратами.

– Масштабованість – збільшення кількості клієнтів від 2 до 100 із оцінкою продуктивності сервера.

Після тестування було отримано наступні результати з продуктивності передачі даних:

Локальна мережа (LAN) – середня затримка 1.8 мс, максимальне навантаження 60 пакетів в секунду.

Глобальна мережа (WAN) – середня затримка 4мс, виявлено втрати 1%.

Після тестування було отримано наступні результати з втрат пакетів:

– LAN: 0%;

– WAN: 1%.

Після тестування розроблених програмних блоків було визначено час робото основних методів:

– Метод Connect – 40мс.

– Метод SendPacket – стабільно передавати 50 пакетів в секунду.

– Модуль LatencyStats – коректне обчислення середньої затримки та кількість втрат (середнє відхилення: 1.8-2мс).

Розроблена модель оцінки затримки передачі даних продемонструвала достатню адекватність у відображенні реальних характеристик мережевих затримок. Утім, у складних мережевих сценаріях із підвищеним навантаженням необхідно враховувати додаткові аспекти, зокрема ефекти чергування у вузлах мережі.

Модель прогнозування втрат пакетів на основі розподілу Бернуллі виявила високу точність для низького та середнього рівня навантаження на мережу. Проте, за умов значного перевантаження мережі спостерігається зниження точності, що вказує на доцільність розробки додаткових коригуючих механізмів.

Функціональні компоненти, такі як UDPServer і UDPClient, продемонстрували високий рівень стабільності під час тестування. Це свідчить про їхню придатність для використання в умовах реального часу та динамічних змін параметрів мережі.

Модуль LatencyStats забезпечив ефективний збір та аналіз даних про мережеву продуктивність.

4.5 Основні науково-технічні результати дослідження

Розроблені математичні моделі мережевої взаємодії базуються на сучасних теоретичних концепціях, таких як теорія графів, ймовірнісні моделі втрат та методи оцінки затримок.

Проведені чисельні експерименти підтвердили адекватність моделей у межах прийнятих припущень, зокрема для сценаріїв із середнім рівнем навантаження на мережу. Кореляція між експериментальними даними та результатами моделювання перевищує 90%, що свідчить про високу точність та надійність запропонованих методів.

Запропоновано універсальну архітектуру моделювання мережевої взаємодії для мультиплеєрних 3D-ігор, яка враховує ключові параметри мережі (затримки, втрати пакетів, пропускну здатність).

Вперше розроблено модуль для синхронізації станів клієнтів та серверу на основі моделі передбачення, що забезпечує мінімізацію розбіжностей у відображенні ігрового процесу в умовах високої затримки.

Математичні моделі затримок передачі даних та втрат пакетів можуть бути адаптовані для інших областей досліджень, зокрема систем реального часу та Internet of thing (IoT).

Розроблені програмні блоки (UDPServer, UDPClient, LatencyStats) успішно реалізовані в середовищі Unity Engine та інтегровані у тестову 3D-гру, демонструючи стабільну роботу в реальних мережевих умовах.

Модуль LatencyStats дозволяє ефективно відстежувати та аналізувати затримки й втрати пакетів, що є важливим інструментом для розробників ігрових проектів та мережевих систем.

Система підтримує масштабованість, дозволяючи розширювати кількість клієнтів та налаштовувати параметри під реальні вимоги користувача.

Запропоновані рішення дозволяють оптимізувати витрати на розробку мережевих механік за рахунок використання ефективних алгоритмів і моделей.

Удосконалення точності моделювання мережевої взаємодії дає змогу мінімізувати ризики невідповідності системи реальним умовам експлуатації, скорочуючи витрати на тестування та усунення помилок на етапі експлуатації.

Інструментарій, розроблений у рамках дослідження, може бути використаний іншими розробниками для створення високоякісних ігрових продуктів, що сприятиме зниженню вартості реалізації подібних проектів.

Таким чином, дослідження поєднує в собі теоретичну новизну, практичну ефективність та економічну доцільність, що підтверджує його науково-технічну значущість та можливість широкого застосування.

4.6 Практичне значення результатів роботи

Розроблені математичні моделі взаємодії в мережах реального часу з урахуванням параметрів затримки, втрат пакетів та пропускної здатності є універсальними й можуть бути використані для вивчення динаміки мережевих процесів у різних системах, таких як Інтернет речей (IoT), розподілені системи обчислень та телекомунікаційні платформи.

Отримані результати дозволяють поглибити розуміння принципів функціонування мережевих протоколів передачі даних у контексті високих вимог до швидкодії, що має значення для подальшого розвитку теорії комп’ютерних мереж.

Модель передбачення станів об’єктів може бути адаптована для інших наукових задач, таких як моделювання автономних систем управління, інтелектуальних транспортних систем і віртуальних симуляцій.

Результати роботи можуть бути використані для створення високоякісних мультиплеєрних ігрових систем із низькими затримками та підвищеною стійкістю до втрат пакетів, що дозволяє забезпечувати стабільність ігрового процесу навіть у нестабільних мережевих умовах.

Інструментарій та програмні блоки, створені в рамках дослідження (UDPServer, UDPClient, LatencyStats), є готовими компонентами для інтеграції в ігрові проєкти, значно скорочуючи час розробки складних мережевих архітектур.

Розроблені механізми моніторингу мережевих параметрів (затримок, втрат, пропускної здатності) можуть бути використані для тестування та діагностики в інших прикладних областях, таких як онлайн-стрімінг, відеоконференції або хмарні обчислення.

Отримані результати мають значення для галузі інформатики, зокрема для досліджень у сфері високонавантажених систем і технологій розподілених обчислень.

Розроблені моделі можуть бути застосовані для аналізу поведінки мереж у наукових експериментах, де потрібна висока точність у передачі даних.

Методики дослідження взаємодії в реальному часі мають потенціал для використання у навчальних симуляціях, що охоплюють віртуальну реальність (VR) і доповнену реальність (AR).

Результати роботи дозволяють розробляти ігрові проекти, здатні функціонувати в умовах обмеженої пропускної здатності мережі, оптимізуючи якість ігрового досвіду користувачів.

Розроблені підходи до прогнозування станів об’єктів підвищують узгодженість ігрового середовища між клієнтами, що особливо актуально для швидких і динамічних ігор.

Інтеграція отриманих моделей та програмних рішень дозволяє забезпечити масштабованість системи та підтримувати більшу кількість користувачів, зберігаючи при цьому високу продуктивність.

Таким чином, результати роботи мають значний потенціал для подальшого використання в наукових дослідженнях та прикладних задачах, сприяючи розв’язанню широкого спектра проблем у сфері інформаційних технологій, розробки програмного забезпечення та оптимізації мережевих процесів.

5 АНАЛІЗ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ІННОВАЦІЇ

5.1 Розрахунок собівартості програмної інновації

Після аналізу та розробки ПЗ було складено таблицю 5.1 в якої зібрані початкові дані для визначення собівартості.

Таблиця 5.1 – Початкові дані для визначення собівартості

Найменування початкових даних	Показник	Джерело отримання
Трудомісткість складання програми, днів	40	Фактичні витрати часу на розробку програмного продукту
Місячна ставка укладача програми, грн	17700	Місце роботи
Кількість годин в місяці, год	160	Кількість робочих днів — 20
Додаткова зарплата (%)	10	Місце роботи
Відрахування до соціальних фондів (%)	15	Місце роботи
Загальновиробничі витрати (%)	100	Місце роботи
ПДВ (податок на додану вартість) (%)	20	Місце роботи

Робимо розрахунок на 1 місяць(20 днів):

Стаття 1. Комплектуючі вироби – 1 компакт-диск :

$$З_к = \sum Ц_к * n_к, \quad (5.1)$$

де $З_к$ – витрати на комплектуючі вироби, грн.;

$Ц_к$ – ціна за 1 одиницю комплектуючих виробів, грн.;

$n_к$ – кількість комплектуючих виробів по кожному типорозмірі, шт

Визначимо витрати на комплектуючі вироби за формулою 5.2:

$$З_к = \sum 7{,}60 * 1 = 7{,}60 \text{ грн.} \quad (5.2)$$

Витрати на комплектуючі виробі склали 7,60 грн.

Стаття 2. Витрати на електроенергію розраховуємо за формулою 5.3:

$$B_E = P_E \sum Wi * t_{\text{шт } i}, \quad (5.3)$$

де P_E – ціна за 1 кВт-год, грн.;

$t_{\text{шт } i}$ – кількість годин в місяці;

Wi – середня потужність, що споживається.

Визначимо витрати на електроенергію за формулою 5.4:

$$B_E = 4{,}32 * \sum 0{,}250 * 160 = 172{,}8 \text{ грн} \quad (5.4)$$

Витрати на електроенергію дорівнюють 172,8 грн.

Стаття 3. Основна заробітна плата визначається за формулою 5.5:

$$З_{осн} = l_{год} * T_{год}, \quad (5.5)$$

де $l_{год}$ – годинна тарифна ставка програміста, грн.;

$T_{год}$ – кількість годин у місяці.

Визначаємо годинну тарифну ставку укладача ПЗ за формулою 5.6:

$$З_{осн} = 110{,}625 * 160 = 17700 \quad (5.6)$$

Стаття 4. Додаткова заробітна плата визначається за формулою 5.7:

$$З_{дод} = \frac{З_{осн} * Д\%}{100}, \quad (5.7)$$

де Д – відсоток додаткової заробітної плати.

Визначимо додаткову заробітну плату за формулою 5.8:

$$З_{дод} = \frac{17700 * 10}{100} = 1770 \text{ грн} \quad (5.8)$$

Додаткова заробітна плата дорівнює 1770,00 грн.

Стаття 5. Відрахування в соціальні фонди знайдемо за формулою 5.9:

$$З_{соц} = \frac{(З_{осн} + З_{доп}) * С\%}{100}, \quad (5.9)$$

де С% — відсоток відрахувань у соціальні фонди.

Визначимо відрахування в соціальні фонди за формулою 5.10:

$$З_{соц} = \frac{(17700 + 1770) * 15}{100} = 2920{,}5 \text{ грн} \quad (5.10)$$

Відрахування в соціальні фонди дорівнює 2920,50 грн;

Стаття 6. Загальновиробничі витрати визначаються за формулою 5.11:

$$З_{заг} = \frac{З_{осн} * H_1\%}{100}, \quad (5.11)$$

де $H_1\%$ — відсоток загальновиробничих витрат.

Визначимо загальновиробничі витрати за формулою 5.12:

$$З_{заг} = \frac{17700 * 100}{100} = 17700 \text{ грн} \quad (5.12)$$

Загальновиробничі витрати дорівнюють 17700,00 грн.

Визначимо виробничу собівартість склавши всі попередні розрахунки (формула 5.13):

$$S_{пп} = 172{,}8 + 7{,}60 + 17700 + 1770 + 2920{,}5 + 17700 = 40270{,}9 \text{ грн} \quad (5.13)$$

Виробнича собівартість дорівнює 40270,9 грн.

В таблиці 5.2 наведено планову калькуляцію виробничої собівартості виробничої собівартості, ціни підприємства й ціни для замовника на виконання розробки програми.

Таблиця 5.2 – Планова калькуляція

Статті калькуляції	Сума, грн.
Стаття 1 Комплектуючи вироби	7,60
Стаття 2. Витрати на електроенергію:	172,8
Стаття 3 Основна заробітна плата	17700
Стаття 4 Додаткова заробітна плата	1770
Стаття 5 Відрахування в соціальні фонди	2920,5
Стаття 6 Загальновиробничі витрати	17700
Виробнича собівартість, 1 місяць	40270,9
Собівартість ПЗ	80541,8

Робота по розробці ПЗ проводилась протягом двох місяців, таким чином розрахунок показав, що собівартість АС складає 80541,8 грн.

5.2 Розрахунок ефективності впровадження програмної інновації

Розрахунок ефективності впровадження архітектури мережевої взаємодії для мультиплеєрних 3D-ігор є складним і багатофакторним завданням, яке враховує як технічні, так і економічні аспекти. В основу аналізу покладено оцінку ризиків, пов'язаних з інтеграцією розробленого програмного забезпечення (ПЗ) у процес створення нової гри, а також економічну доцільність цієї інтеграції.

Основним завданням розробленої архітектури є забезпечення стабільної взаємодії клієнтських і серверних частин мультиплеєрної гри з мінімальними затримками, а також оптимальне використання мережевих ресурсів. Однією з головних переваг архітектури є її адаптивність до різних ігрових сценаріїв та підтримка масштабованості для збільшення кількості гравців без істотного зниження продуктивності. Проте, інтеграція нової архітектури пов’язана з певними ризиками, які можуть вплинути як на етап розробки, так і на комерційний успіх гри.

Одним із ключових ризиків є затримка релізу гри через необхідність доопрацювання архітектури. Втрати від такого ризику обчислюються за формулою 5.14:

$$В_з = Т_з \times Б_м, \tag{5.14}$$

де $В_з$ – втрати від затримки релізу, грн;

$Т_з$ – час затримки релізу, місяців;

$Б_м$ – щомісячний бюджет команди розробників, грн.

Наприклад, при затримці релізу на 3 місяці та бюджеті 250 тисяч грн втрати становлять (формула 5.15):

$$В_з = 3 * 250\,000 = 750\,000 \text{ грн.} \tag{5.15}$$

Другий ризик – втрати користувачів через технічні недоліки. Вони розраховуються за формулою 5.16:

$$В_к = К \times Ц_г \times В_т, \quad (5.16)$$

де $В_к$ – втрати через відтік користувачів, грн;

К – кількість залучених користувачів, осіб;

$Ц_г$ – вартість залучення одного гравця, грн;

$В_т$ – частка користувачів, що відмовляються від гри через технічні недоліки, %.

Для аудиторії у 100 тисяч гравців, вартості залучення 50 грн і частки відтоку 20% втрати дорівнюють за формулою 5.17:

$$В_к = 100\,000 * 50 * 0{,}2 = 1\,000\,000 \text{ грн.} \quad (5.17)$$

Зниження цього ризику на 75% завдяки новій архітектурі зменшує втрати до (формула 5.18):

$$В_к^{зм} = В_к \times (1 - 0{,}75) = 1\,000\,000 * 0{,}25 = 250\,000 \text{ грн.} \quad (5.18)$$

Економія на серверній інфраструктурі розраховується за формулою 5.19:

$$Е_с = В_с \times З_с, \quad (5.20)$$

де $Е_с$ – економія на серверній інфраструктурі, грн;

$В_с$ – річні витрати на сервери, грн;

$З_с$ – зниження витрат на сервери, %.

За витрат 6 895 200 грн і зниженні на 30% економія становить розраховується за формулою 5.20:

$$\mathrm{E_c} = 6\,895\,200 * 0{,}3 = \ 2\,068\,560 \qquad (5.20)$$

Загальний економічний ефект обчислюється за формулою 5.21:

$$\mathrm{E_{заг}} = \mathrm{E_c} + (\mathrm{B_к} - \mathrm{B_{кзм}}),$$

Підставляючи значення (формула 5.22):

$$\mathrm{E_{заг}} = 2\,068\,560 + (1\,000\,000 - 250\,000) = 2818560 \text{ грн.}$$

Термін окупності визначається за формулою:

$$\mathrm{T} = \frac{\mathrm{C}}{E_{\text{заг}}/\mathrm{Ч}},$$

де С – собівартість інтеграції архітектури, грн;

Ч – період, протягом якого отримується економічний ефект, місяців.

При собівартості 80541,8 грн. і річному періоді (12 місяців) термін окупності дорівнює:

$$\mathrm{T} = \frac{80541{,}8}{2818560\ /\ 12} = \frac{80541{,}8}{234880} \approx \ 0{,}34 \text{ місяців}$$

Отримані результати демонструють, що інтеграція архітектури мережевої взаємодії є економічно вигідною. Зниження ризиків, оптимізація інфраструктури та підвищення ефективності розробки дозволяють окупити

витрати на інтеграцію менш ніж за місяць, що свідчить про високий потенціал комерційної вигоди.

ВИСНОВКИ

У межах виконаної роботи було досягнуто вагомих результатів у сфері розробки та дослідження архітектури мережевої взаємодії для мультиплеєрних 3D-ігор на базі мови програмування C#. З урахуванням сучасних світових тенденцій, результати вирізняються своєю актуальністю, інноваційністю та практичним потенціалом.

У світі спостерігається стрімке зростання популярності онлайн-ігор із багатокористувацьким режимом, що вимагає створення високоефективних і масштабованих мережевих архітектур. У цьому контексті запропоновані моделі передачі даних, прогнозування станів та мінімізації затримок відповідають сучасним вимогам до розробки мережевих рішень для ігрових проектів.

Прогрес у хмарних обчисленнях і потоковому передаванні даних також підкреслює важливість оптимізації мережевих параметрів. Отримані результати доповнюють наявні підходи, забезпечуючи підвищену продуктивність і стабільність роботи систем у реальному часі.

Результати можуть бути безпосередньо впроваджені у розробку мультиплеєрних 3D-ігор, особливо для невеликих ігрових студій, що прагнуть оптимізувати процес створення ігрових серверів та клієнтської взаємодії.

Розроблені підходи можуть бути інтегровані в освітні програми, спрямовані на підготовку фахівців у галузі програмування мережевих систем.

Оптимізація мережевої архітектури дозволяє знизити витрати на апаратне забезпечення та мережеві ресурси, що сприяє зменшенню загальної вартості розробки програмного забезпечення.

Розроблені методики підвищують ефективність використання наявних мережевих інфраструктурах, що є важливим фактором у масштабних проектах із залученням тисяч користувачів.

Математичні моделі, створені в рамках роботи, роблять внесок у розвиток теорії комп’ютерних мереж і розподілених обчислень, створюючи основу для подальших досліджень у сфері оптимізації мережевих процесів.

Програмні рішення, отримані під час роботи, демонструють нові підходи до синхронізації клієнт-серверної взаємодії, що є актуальним для проектів, орієнтованих на інтерактивні мережеві середовища.

Розробка інструментів для стабільної роботи мультиплеєрних систем сприяє покращенню користувацького досвіду, що має позитивний вплив на розвиток індустрії розваг.

Використання результатів роботи в освітніх цілях сприяє формуванню компетентних фахівців у галузі інформаційних технологій, що є ключовим фактором для забезпечення сталого розвитку суспільства.

Доступ до сучасних мультиплеєрних технологій сприяє соціалізації користувачів та розвитку комунікацій у віртуальному просторі.

Впровадження нової архітектури мережевої взаємодії є економічно вигідною. Зниження ризиків, оптимізація інфраструктури та підвищення ефективності розробки дозволяють окупити витрати на інтеграцію менш ніж за місяць, що свідчить про високий потенціал комерційної вигоди.

Отже, результати роботи мають високу наукову, практичну та економічну цінність, сприяючи вирішенню актуальних завдань у сфері комп’ютерних наук і технологій. Вони також відповідають сучасним викликам і потребам, пов’язаним із швидким розвитком інформаційних систем і цифрових комунікацій.

ПЕРЕЛІК ПОСИЛАНЬ

1. Li, X., Wang, Y., & Zhang, H. A Survey of State Synchronization Techniques in Networked Multiplayer Games. Journal of Computer Science and Technology. – 2020.

2. Chen, M., Liu, J., & Zhu, X. Optimizing Network Performance for Real-Time Multiplayer Games. IEEE Communications Surveys & Tutorials. – 2019.

3. Smith, J., Brown, A., & Taylor, L. Developing Multiplayer Games with Unity and C#. Game Development Journal. – 2021.

4. Brown, D., Johnson, P., & Williams, S. High-Performance Networking with .NET. Software Development Review. – 2020.

5. Davis, R., Martin, K., & Thompson, G. Performance Testing for Multiplayer Game Servers. International Journal of Game Design. – 2018.

6. Wang, L., Chen, R., & Zhou, Q. Scalable Networking Solutions for Online Games. Journal of Network and Computer Applications. – 2020.

7. Kim, S., Park, J., & Lee, H. Security Measures for Multiplayer Online Games. Cybersecurity and Privacy Journal. – 2019.

8. Сайт Unity Engine [Електронний ресурс] – Режим доступу до ресурсу: https://unity.com/.

9. Документація Unity [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/index.html

10. Visual Studio 2022 [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com

11. Use UDP services [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/ru-ru/dotnet/framework/network-programming/using-udp-services

12. Використання LINQ методів в C# [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialsteacher.com/linq

13. Патерн розробки «Спостерігач» [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/observer

14. Фреймворк моделювання на основі агентів для Unity3D [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/S2352711021000881.

15. Wells R. Unity 2020 By Example: A project-based guide to building 2D, 3D, augmented reality, and virtual reality games from scratch / Robert Wells., 2020. – 676 с. – (3).

16. Asynchronous programming scenarios [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/dotnet/csharp/asynchronous-programming/async-scenarios

17. Client Server Connection [Електронний ресурс] – Режим доступу до ресурсу: https://gafferongames.com/post/client_server_connection/

18. Snapshot Interpolation [Електронний ресурс] – Режим доступу до ресурсу: https://gafferongames.com/post/snapshot_interpolation/

19. Соммервілл, Інженерія програмного забезпечення, Пер. з англ. – Київ, 2008. – 624с.

20. Микитюк П.П. Інноваційний менеджмент: Навчальний посібник. – Тернопіль: Економічна думка, 2006. – 295с.

21. Йохна М.А., Стадник В.В. Економіка і організація інноваційної діяльності. Навч. Посібник – К., 2005р.

22. Чайковська М.П. Інноваційний менеджмент: Навчальний посібник / М.П. Чайковська – Одеса: Одеський національний університет імені І.І. Мечнікова, 2015. – 382с.

23. Лепейко Т.І., Коюда В.О., Лукашов С.В. Інноваційний менеджмент. Навч. посіб. / Харків. – Х.:ШЖЕК, 2005. – 440с.

24. Photon Engine [Електронний ресурс] – Режим доступу до ресурсу: https://www.photonengine.com/pun

25. Mirror networking [Електронний ресурс] – Режим доступу до ресурсу: https://mirror-networking.com/

ДОДАТОК А

Код програми

UDPClient.cs

```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Threading.Tasks;
using Assets.Scripts.Multiplayer.NetworkObjects;
using UnityEngine;
using Debug = UnityEngine.Debug;
using ThreadPriority = System.Threading.ThreadPriority;

namespace Assets.Scripts.Multiplayer
{
    public sealed class UDPClient : IDisposable
    {
        private readonly byte[] buffer;
        private readonly Socket client;
        private EndPoint remoteEndPoint;
        private readonly ConcurrentDictionary<int,
(GamePacket Packet, DateTime SentTime)> unconfirmedPackets;
        private readonly ConcurrentQueue<GamePacket> queue =
new();
        private readonly CancellationTokenSource cts;
        private readonly int maxRetries;
        private readonly TimeSpan retryDelay;
        private readonly TimeSpan heartbeatInterval;
        private int packetSequence;
        private bool disposed;
        private bool isDisconnecting;

        public readonly LatencyStats LatencyStats = new();
        public Guid ClientId { get; }
        public bool IsConnected { get; private set; }

        public delegate Awaitable PacketReceived(GamePacket
packet);

        public event PacketReceived OnPacketReceived;

        public UDPClient(string host, int port,
            int bufferSize = 25 * 1024,
            int maxRetries = 3,
            TimeSpan? retryDelay = null,
            TimeSpan? heartbeatInterval = null)
        {
```

```
                if (string.IsNullOrEmpty(host))
                {
                    throw new ArgumentException(nameof(host));
                }

                if (port <= 0)
                {
                    throw new
ArgumentOutOfRangeException(nameof(port));
                }

                if (bufferSize <= 0)
                {
                    throw new
ArgumentOutOfRangeException(nameof(bufferSize));
                }

                buffer = new byte[bufferSize];
                client = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);
                remoteEndPoint = new
IPEndPoint(IPAddress.Parse(host), port);
                unconfirmedPackets = new
ConcurrentDictionary<int, (GamePacket, DateTime)>();
                cts = new CancellationTokenSource();
                ClientId = Guid.NewGuid();

                this.maxRetries = maxRetries;
                this.retryDelay = retryDelay ??
TimeSpan.FromMilliseconds(500);
                this.heartbeatInterval = heartbeatInterval ??
TimeSpan.FromSeconds(1);
            }

            public async void ConnectAsync()
            {
                if (IsConnected) return;

                var requestConnectPacket = new GamePacket
                {
                    Command = TypeCommandPacket.RequestConnect,
                    IsNeedSave = false,
                    IsNeedConfirm = false
                };

                await SendPacket(requestConnectPacket);

                var result = await
client.ReceiveFromAsync(buffer, SocketFlags.None,
remoteEndPoint);
                var packet =
GamePacket.Deserialize(buffer[..result.ReceivedBytes]);
```

```
                if (packet.Command ==
TypeCommandPacket.ResponseConnect)
                {
                    var response = packet.GetData<string>();

                    if (response == "OK")
                    {
                        Debug.Log("Allow connect to server");

                        Task.WhenAll(HeartbeatAsync(cts.Token),
Task.Run(() => ReceiveLoopAsync(cts.Token)),
ReadPacketLoop(cts.Token));

                        var connectPacket = new GamePacket
                        {
                            Command = TypeCommandPacket.Connect,
                            IsNeedSave = false,
                            IsNeedConfirm = false
                        };

                        await SendPacket(connectPacket);
                    }
                }
            }

            public async void DisconnectAsync(CancellationToken
cancellationToken = default)
            {
                if (!IsConnected) return;

                try
                {
                    var disconnectPacket = new GamePacket {
Command = TypeCommandPacket.Disconnect };
                    await SendPacket(disconnectPacket,
cancellationToken);
                    LatencyStats.SaveResult();
                }
                catch (Exception ex)
                {
                    Debug.LogException(ex);
                }
            }

            public async Task SendPacket(GamePacket packet,
CancellationToken cancellationToken = default)
            {
                if (packet == null)
                {
                    throw new
ArgumentNullException(nameof(packet));
                }
                packet.ClientID = ClientId;
```

```
                packet.Time = DateTime.UtcNow.Ticks;
                packet.Id = GetNextPacketSequence();
                var retryCount = 0;
                while (retryCount < maxRetries &&
!cancellationToken.IsCancellationRequested)
                {
                    try
                    {
                        var data = packet.Serialize();
                        if (isDisconnecting) return;

                        if (packet.Command ==
TypeCommandPacket.Disconnect)
                        {
                            isDisconnecting = true;
                        }
                        if (packet.Command !=
TypeCommandPacket.Ping && packet.IsNeedConfirm)
                        {
                            unconfirmedPackets[packet.Id] =
(packet, DateTime.UtcNow);
                        }
                        await client.SendToAsync(data,
SocketFlags.None, remoteEndPoint);
                        return;
                    }
                    catch (Exception ex) when (ex is not
OperationCanceledException)
                    {
                        retryCount++;
                        Debug.LogException(ex);

                        if (retryCount < maxRetries)
                        {
                            await Task.Delay(retryDelay,
cancellationToken);
                        }
                    }
                }
                throw new TimeoutException($"Failed to send
packet after {maxRetries} attempts");
            }

            private async Task HeartbeatAsync(CancellationToken
cancellationToken)
            {
                try
                {
                    while
(!cancellationToken.IsCancellationRequested)
                    {
                        var pingPacket = new GamePacket {
Command = TypeCommandPacket.Ping };
```

```
                        await SendPacket(pingPacket,
cancellationToken);
                        await Task.Delay(heartbeatInterval,
cancellationToken);
                    }
                }
                catch (OperationCanceledException)
                {
                    // Normal cancellation, no action needed
                }
                catch (Exception ex)
                {
                    Debug.LogException(ex);
                }
            }

            private void ReceiveLoopAsync(CancellationToken
cancellationToken)
            {
                try
                {
                    Thread.CurrentThread.Priority =
ThreadPriority.Highest;

                    while
(!cancellationToken.IsCancellationRequested)
                    {
                        var size = client.ReceiveFrom(buffer,
SocketFlags.None, ref remoteEndPoint);

                        #region Average_0.5ms
                        var packet =
GamePacket.Deserialize(buffer[..size]);

                        if (packet.ClientID == ClientId)
                        {
                            var latency = (DateTime.UtcNow - new
DateTime(packet.Time)).TotalMilliseconds;
                            LatencyStats.Insert(latency);
                        }

                        if (packet.Command ==
TypeCommandPacket.Delivered)
                        {

unconfirmedPackets.TryRemove(packet.Id, out _);
                            continue;
                        }

                        queue.Enqueue(packet);
                        #endregion
                    }
                }
```

```
                catch (OperationCanceledException)
                {
                    // Normal cancellation, no action needed
                }
                catch (Exception ex)
                {
                    Debug.LogException(ex);
                    IsConnected = false;
                }
            }

            private async Task ReadPacketLoop(CancellationToken
cancellationToken)
            {
                while
(!cancellationToken.IsCancellationRequested)
                {
                    if (!queue.IsEmpty)
                    {
                        //0.05ms
                        if (queue.TryDequeue(out var packet))
                        {
                            Receive(packet);
                        }
                    }
                    else
                    {
                        await
Awaitable.WaitForSecondsAsync(0.001f, cancellationToken);
                    }
                }
            }
            private async Awaitable Receive(GamePacket packet)
            {

                if (!IsConnected && packet.Command ==
TypeCommandPacket.Connect)
                {
                    IsConnected = true;
                }

                if (!IsConnected) return;

                if (packet.Command ==
TypeCommandPacket.Disconnect && packet.ClientID == ClientId)
                {
                    IsConnected = false;
                    cts.Cancel();
                    queue.Clear();
                }

                await OnPacketReceived?.Invoke(packet)!;
```

```
LatencyStats.InsertLoss(unconfirmedPackets.Count);
            }

            private int GetNextPacketSequence() =>
Interlocked.Increment(ref packetSequence) % 1000;

            public void Dispose()
            {
                if (disposed) return;

                disposed = true;
                cts.Cancel();
                cts.Dispose();
                client.Dispose();

                GC.SuppressFinalize(this);
            }
        }

        public class LatencyStats
        {
            private double Current { get; set; }
            private double Loss { get; set; }
            private double Average => latencyHistory.Count > 0 ?
sum / latencyHistory.Count : 0;
            private double Min { get; set; } = double.MaxValue;
            private double Max { get; set; } = double.MinValue;

            private readonly List<double> latencyHistory =
new();
            private double sum = 0;

            public void Insert(double newLatency)
            {
                Current = newLatency;
                latencyHistory.Add(newLatency);
                sum += newLatency;

                if (newLatency < Min) Min = newLatency;
                if (newLatency > Max) Max = newLatency;

                if (latencyHistory.Count > 100000)
                {
                    double oldestLatency = latencyHistory[0];
                    sum -= oldestLatency;
                    latencyHistory.RemoveAt(0);

                    // Recalculate Min and Max only if we
removed the current Min or Max
                    if (oldestLatency == Min || oldestLatency ==
Max)
                    {
```

```
                        Min = latencyHistory.Count > 0 ?
latencyHistory.Min() : double.MaxValue;
                        Max = latencyHistory.Count > 0 ?
latencyHistory.Max() : double.MinValue;
                    }
                }
            }
            public void InsertLoss(double value) => Loss =
value;

            public void SaveResult()
            {
                File.AppendAllText("test.csv", string.Join("\n",
latencyHistory.ToArray()));
            }
            public override string ToString() =>
                $"Latency (ms) - Current: {Current:F1}, Avg:
{Average:F1}, Min: {Min:F1}, Max: {Max:F1}, Loss: {Loss}
packets";
        }

    }
```

UDPServer.cs

```
    using System;
    using System.Collections.Concurrent;
    using System.Linq;
    using System.Net;
    using System.Net.Sockets;
    using System.Threading;
    using System.Threading.Tasks;
    using Assets.Scripts.Multiplayer.NetworkObjects;
    using Newtonsoft.Json;
    using Newtonsoft.Json.Linq;
    using Debug = UnityEngine.Debug;

    namespace Assets.Scripts.Multiplayer
    {
        public class UDPServer : IDisposable
        {
            // Configuration parameters
            private const int DefaultBufferSize = 25 * 1024;
            private const int ClientTimeoutSeconds = 10;
            private const int MonitoringIntervalMilliseconds =
250;

            // Networking components
            private readonly Socket serverSocket;
            private readonly int port;
            private byte[] receiveBuffer;
            private int maxPlayers;
```

```
            // Concurrent collections with thread-safe
operations
            private readonly ConcurrentDictionary<Guid,
ServerClient> clients = new();
            private readonly ConcurrentDictionary<Guid,
ConcurrentQueue<GamePacket>> packetBuffers = new();
            private readonly ConcurrentDictionary<Guid,
NetworkObject> networkObjects = new();

            // Cancellation and synchronization
            private readonly CancellationTokenSource
cancellationTokenSource = new();
            private readonly object syncLock = new();


            private const int MAX_PACKET_SEQUENCE = 100;

            public bool IsRunning = false;

            public UDPServer(
                int port,
                int bufferSize = DefaultBufferSize,
                int maxPlayers = 4)
            {
                this.port = port;
                receiveBuffer = new byte[bufferSize];
                this.maxPlayers = maxPlayers;

                // Create UDP socket
                serverSocket = new
Socket(AddressFamily.InterNetwork, SocketType.Dgram,
ProtocolType.Udp)
                {
                    ExclusiveAddressUse = false,
                    ReceiveBufferSize = bufferSize,
                    SendBufferSize = bufferSize
                };
                serverSocket.Bind(new IPEndPoint(IPAddress.Any,
port));
            }

            public async void Start()
            {
                Debug.Log($"Starting UDP server on port
{port}");
                IsRunning = true;
                // Start monitoring and receive tasks
                await Task.WhenAll(Task.Run(ReceiveLoop),
UpdateLoop(), MonitorClientsLoop());
            }

            private async Task UpdateLoop()
            {
```

```
                Thread.CurrentThread.Priority =
ThreadPriority.Highest;

                while
(!cancellationTokenSource.Token.IsCancellationRequested)
                {
                    try
                    {
                        var packet = new GamePacket() { Command
= TypeCommandPacket.Sync };

packet.SetData(networkObjects.Values.Where(item =>
item.IsNeedUpdate).ToList());

                        BroadcastPacket(packet);

                        await Task.Delay(17);
                    }
                    catch (Exception ex)
                    {
                        Debug.LogError(ex);
                    }
                }
            }

            private void ReceiveLoop()
            {
                Thread.CurrentThread.Priority =
ThreadPriority.Highest;
                while
(!cancellationTokenSource.Token.IsCancellationRequested)
                {
                    try
                    {
                        var result = ReceivePacketAsync();

                        if (result.Packet != null)
                        {
                            ProcessReceivedPacket(result.Packet,
result.RemoteEndPoint);
                        }
                    }
                    catch (Exception ex)
                    {
                        Debug.LogError(ex);
                    }
                }
            }

            private async Task MonitorClientsLoop()
            {
                Thread.CurrentThread.Priority =
ThreadPriority.Highest;
```

```
                while
(!cancellationTokenSource.Token.IsCancellationRequested)
                {
                    try
                    {
                        var inactiveClients = clients
                            .Where(c => (DateTime.UtcNow -
c.Value.Time).TotalSeconds > ClientTimeoutSeconds)
                            .Select(c => c.Key)
                            .ToList();

                        foreach (var clientId in
inactiveClients)
                        {
                            HandleClientDisconnect(clientId);
                        }

                        await
Task.Delay(MonitoringIntervalMilliseconds);
                    }
                    catch (Exception ex)
                    {
                        Debug.LogError(ex);
                    }
                }
            }

            private (GamePacket Packet, IPEndPoint
RemoteEndPoint) ReceivePacketAsync()
            {
                receiveBuffer = new byte[receiveBuffer.Length];
                EndPoint remoteEndPoint = new
IPEndPoint(IPAddress.Any, 0);

                var size =
serverSocket.ReceiveFrom(receiveBuffer, SocketFlags.None, ref
remoteEndPoint);

                var packet =
GamePacket.Deserialize(receiveBuffer[..size]);

                return (packet, (IPEndPoint)remoteEndPoint);
            }


            private void ProcessReceivedPacket(GamePacket
packet, IPEndPoint clientEndPoint)
            {
                #region Average_1-3

                //if (!IsValidPacketSequence(packet.ClientID,
packet) && packet.IsNeedSave)
                //{
```

```
                //    return;
                //}
                if (packet.IsNeedConfirm)
                    SendPacketConfirmation(clientEndPoint,
packet);

                switch (packet.Command)
                {
                    case TypeCommandPacket.RequestConnect:
                        HandleClientRequestConnect(packet,
clientEndPoint);
                        break;
                    case TypeCommandPacket.Connect:
                        HandleClientConnect(packet,
clientEndPoint);

packet.SetData(networkObjects.Values.ToList());
                        BroadcastPacket(packet);
                        break;
                    case TypeCommandPacket.Disconnect:
                        BroadcastPacket(packet);
                        HandleClientDisconnect(packet.ClientID);
                        break;
                    case TypeCommandPacket.Ping:
                        UpdateClientHeartbeat(packet.ClientID);
                        return;
                    case TypeCommandPacket.Update when
packet.IsNeedSave:
                        HandleUpdateStateObject(packet);
                        break;
                    case TypeCommandPacket.Broadcast:
                        if (packet.IsNeedSave)
                            HandleUpdateStateObject(packet);
                        BroadcastPacket(packet);
                        break;
                }

                EnqueuePacket(packet);
                #endregion
            }

            private async void
HandleClientRequestConnect(GamePacket packet, IPEndPoint
clientEndPoint)
            {
                var responsePacket = new GamePacket
                {
                    Command = TypeCommandPacket.ResponseConnect
                };

                responsePacket.SetData(clients.Count + 1 <=
maxPlayers ? "OK" : "MAX PLAYERS");
```

```
                var confirmationData =
responsePacket.Serialize();

                await serverSocket.SendToAsync(confirmationData,
SocketFlags.None, clientEndPoint);
            }

            private void HandleUpdateStateObject(GamePacket
packet)
            {
                if (packet.Data == null || packet.Data.Length ==
0) return;

                var body = packet.GetDataJson();

                if (body.StartsWith("{") && body.EndsWith("}"))
                {
                    var json = JObject.Parse(body);
                    var type =
Type.GetType(json["Type"]!.Value<string>());

UpdateNetworkObject((NetworkObject)JsonConvert.DeserializeObject
(json.ToString(), type, GamePacket.Setting));
                }
                else
                {
                    var array = JArray.Parse(body);

                    foreach (var item in array)
                    {
                        var type =
Type.GetType(item["Type"]!.Value<string>());

UpdateNetworkObject((NetworkObject)JsonConvert.DeserializeObject
(item.ToString(), type, GamePacket.Setting));
                    }
                }
            }

            private void UpdateNetworkObject(NetworkObject data)
            {
                if (networkObjects.TryGetValue(data.Id, out var
oldValue))
                {
                    networkObjects.TryUpdate(data.Id, data,
oldValue);
                }
                else
                {
                    networkObjects.TryAdd(data.Id, data);
                }
            }
```

```
            private async void SendPacketConfirmation(IPEndPoint
clientEndPoint, GamePacket originalPacket)
            {
                var confirmationPacket = new GamePacket
                {
                    Command = TypeCommandPacket.Delivered,
                    ClientID = originalPacket.ClientID,
                    Id = originalPacket.Id,
                    Time = originalPacket.Time
                };

                var confirmationData =
confirmationPacket.Serialize();

                await serverSocket.SendToAsync(confirmationData,
SocketFlags.None, clientEndPoint);
            }

            private bool IsValidPacketSequence(Guid ClientId,
GamePacket packet)
            {
                if (!packetBuffers.ContainsKey(ClientId))
                {
                    packetBuffers.TryAdd(ClientId, new
ConcurrentQueue<GamePacket>());
                }

                if (packetBuffers.TryGetValue(ClientId, out var
playerPackets))
                {

                    if (playerPackets.Contains(packet))
                        return false;

                    playerPackets.Enqueue(packet);

                    // Ограничение размера для предотвращения
переполнения памяти
                    if (playerPackets.Count >
MAX_PACKET_SEQUENCE)
                        playerPackets.TryDequeue(out _);
                }

                return true;
            }

            private void HandleClientConnect(GamePacket packet,
EndPoint clientEndPoint)
            {
                var ipEndPoint = clientEndPoint as IPEndPoint;
                if (ipEndPoint == null) return;
```

```
                clients.TryAdd(packet.ClientID, new
ServerClient(ipEndPoint, DateTime.UtcNow));
                packetBuffers.TryAdd(packet.ClientID, new
ConcurrentQueue<GamePacket>());

                Debug.Log($"Client connected: {packet.ClientID}
from {ipEndPoint}");
            }

            private void HandleClientDisconnect(Guid clientId)
            {
                clients.TryRemove(clientId, out _);
                packetBuffers.TryRemove(clientId, out _);

                foreach (var obj in
networkObjects.Keys.ToList().Where(obj => obj == clientId))
                {
                    networkObjects.TryRemove(obj, out _);
                }

                Debug.Log($"Client disconnected: {clientId}");
            }

            private void UpdateClientHeartbeat(Guid clientId)
            {
                if (clients.TryGetValue(clientId, out var
client))
                {
                    client.Time = DateTime.UtcNow;
                }
            }

            private void EnqueuePacket(GamePacket packet)
            {
                if (packetBuffers.TryGetValue(packet.ClientID,
out var packetQueue))
                {
                    packetQueue.Enqueue(packet);
                }
            }

            private void BroadcastPacket(GamePacket packet)
            {
                Task.Run(() =>
                {
                    var packetBytes = packet.Serialize();

                    foreach (var client in clients)
                    {
                        try
                        {
                            serverSocket.SendTo(packetBytes,
client.Value.Address);
```

```
                        }
                        catch (Exception ex)
                        {
                            Debug.LogError(ex);
                        }
                    }
                });
            }

            public void Stop()
            {
                cancellationTokenSource.Cancel();
                serverSocket.Close();
                IsRunning = false;
            }

            public void Dispose()
            {
                Stop();
                serverSocket.Dispose();
                cancellationTokenSource.Dispose();
            }
        }

        public class ServerClient
        {
            public ServerClient(IPEndPoint address, DateTime
time)
            {
                Address = address;
                Time = time;
            }

            public IPEndPoint Address { get; }
            public DateTime Time { get; set; }
        }
    }
```

GamePacket.cs

```
using System;
using System.Text;
using Assets.Scripts.Tools;
using Newtonsoft.Json;

namespace Assets.Scripts.Multiplayer.NetworkObjects
{
    public class GamePacket
    {
        public int Id;
        public Guid ClientID;
        public TypeCommandPacket Command;
        public long Time;
```

```
            public byte[] Data;
            public bool IsNeedSave = true;
            public bool IsNeedConfirm = true;

            public T GetData<T>() =>

JsonConvert.DeserializeObject<T>(Encoding.UTF8.GetString(Data.De
compress()), Setting);

            public void SetData<T>(T data) =>
                Data =
Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(data,
Setting)).Compress();

            public static GamePacket Deserialize(byte[] data) =>

JsonConvert.DeserializeObject<GamePacket>(Encoding.UTF8.GetStrin
g(data.Decompress()), Setting);

            public byte[] Serialize() =>
Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(this,
Setting)).Compress();

            public string GetDataJson() =>
Encoding.UTF8.GetString(Data.Decompress());

            [JsonIgnore]
            public static JsonSerializerSettings Setting =>
new()
            {
                TypeNameHandling = TypeNameHandling.Auto,
                ReferenceLoopHandling =
ReferenceLoopHandling.Ignore
            };
        }
    }
```

NetworkObject.cs

```
using System;

namespace Assets.Scripts.Multiplayer.NetworkObjects
{
    public class NetworkObject
    {
        public Guid Id { get; set; } = Guid.NewGuid();
        public string Type => GetType().ToString();

        public bool IsNeedUpdate = true;
    }
}
```

NetworkObserver.cs

```
using System;
using UnityEngine;

namespace Assets.Scripts.Multiplayer.NetworkObjects
{
    public abstract class NetworkObserver: MonoBehaviour
    {
        public Guid Id { get; set; } = Guid.NewGuid();
        public Guid Owner { get; set; }
        public bool IsMain;

        public string NetworkName;
    }
}
```

NetworkManager.cs

```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using Assets.Scripts.Multiplayer;
using Assets.Scripts.Multiplayer.NetworkObjects;
using Assets.Scripts.Tools;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using UnityEngine;
using CompressionLevel =
System.IO.Compression.CompressionLevel;
using Debug = UnityEngine.Debug;
using Random = UnityEngine.Random;

namespace Assets.Scripts
{
    public class NetworkManager : MonoBehaviour
    {
        private UDPServer server;
        private UDPClient client;
        private string username;
        public int Port;
        public MenuManager menuManager;
        public GameObject player;
        private List<GameObject> players = new();
        private List<NetworkObserver> observers = new();
        public PlayerMove MainPlayer;
        public Cuboid spawnLocation;
        public List<ObjectInfo> DynamicObjects;
        private static readonly ConcurrentDictionary<Type,
List<MemberInfo>> _memberCache = new();

        public static NetworkManager Instance;
```

```
            private void Start(){Instance = this;}
            private async Awaitable FixedUpdate()
            {            if (client is {IsConnected: true} &&
observers.Count > 0)
                {
                    var packet = new GamePacket() {Command =
TypeCommandPacket.Update};
                    var list = new List<NetworkObjectData>();
                    foreach (var item in observers)
                    {
                        item.Owner = client.ClientId;
                        if (!item.IsMain) continue;
                        GetValuesFromType(item, list);
                    }
                    packet.SetData(list);
                    if (!client.IsConnected) return;
                    await Awaitable.BackgroundThreadAsync();
                    await client.SendPacket(packet);
                }
            }
            public void AddObserver(NetworkObserver observer)
            {
                if (server != null) {observer.IsMain = true; }
                observers.Add(observer);
            }
            private void GetValuesFromType(NetworkObserver item,
List<NetworkObjectData> list)
            {
                var type = item.GetType();
                var members = _memberCache.GetOrAdd(type, t =>
                    t.GetMembers(BindingFlags.Public |
BindingFlags.NonPublic | BindingFlags.Instance)
                        .Where(m => m.MemberType is
MemberTypes.Property or MemberTypes.Field)
                        .Where(m =>
m.GetCustomAttribute<SyncAttribute>() != null || m.Name ==
"NetworkName")
                        .ToList());
                var values = new Dictionary<string,
object>(members.Count);
                foreach (var member in members)
                {
                    try
                    {
     object value = member.MemberType == MemberTypes.Property
     ? ((PropertyInfo) member).GetValue(item)
     : ((FieldInfo) member).GetValue(item);
                        values[member.Name] = value;
                    }
                    catch (Exception e)
                    {
                        Debug.LogException(e);
```

```
                    }
                }

                list.Add(new NetworkObjectData
                {
                    Id = item.Id,
                    ClassType = type.ToString(),
                    Values = values
                });
            }
            public void StartServer()
            {
                if (server != null) return;
                server = new UDPServer(Port);
                server.Start();
                var objects =
FindObjectsByType<NetworkObserver>(FindObjectsSortMode.None);
                foreach (var obj in objects)
                {
                    AddObserver(obj);
                }
            }
            public void ConnectToServer(string host, int port,
string userName)
            {
                client?.Dispose();
                client = new UDPClient(host, port);
                client.OnPacketReceived +=
ReceiveDataFromServerUDP;
                client.ConnectAsync();
                this.username = userName;
            }
            public void DisconnectPlayer()
            {
                client?.DisconnectAsync();
                menuManager.Visible(true);
            }
            private async Awaitable
ReceiveDataFromServerUDP(GamePacket packet)
            {
                try
                {
                    await Awaitable.BackgroundThreadAsync();
                    switch (packet.Command)
                    {
                        case TypeCommandPacket.Connect:
                            ConnectHandler(packet);
                            break;
                        case TypeCommandPacket.Disconnect:
                            DisconnectHandler(packet);
                            break;
                        case TypeCommandPacket.Sync or
TypeCommandPacket.Broadcast:
```

```
                            UpdateState(packet);
                            break;
                        default:
                            throw new
ArgumentOutOfRangeException();
                    }
                }
                catch (Exception e)
                {
                    Debug.LogException(e);
                    throw;
                }
                await Awaitable.MainThreadAsync();
                HudScript.Instance.latencyLabel.text =
client.LatencyStats.ToString();
            }
            private void UpdateState(GamePacket packet)
            {
                var body = packet.GetDataJson();
                if (body.StartsWith("{") && body.EndsWith("}"))
                {
                    var json = JObject.Parse(body);
                    var type =
Type.GetType(json["Type"]!.Value<string>());
                    UpdateObjectBaseType(
                        (NetworkObject)
JsonConvert.DeserializeObject(json.ToString(), type,
GamePacket.Setting),
                            packet.ClientID);
                }
                else
                {
                    var array = JArray.Parse(body);
                    foreach (var item in array)
                    {
                        var type =
Type.GetType(item["Type"]!.Value<string>());
                        UpdateObjectBaseType(
                            (NetworkObject)
JsonConvert.DeserializeObject(item.ToString(), type,
GamePacket.Setting),
                                packet.ClientID);
                    }
                }
            }
            //Main method to handle data
            private async void
UpdateObjectBaseType(NetworkObject data, Guid clientID)
            {
                await Awaitable.MainThreadAsync();
                switch (data)
                {
                    case NetworkObjectData obj:
```

```
                        NetworkObjectDataHandler(obj);
                        break;
                    case NetworkMessage message:
                        NetworkMessageHandler(message);
                        break;
                    case NetworkVoice voice:
                        NetworkVoiceHandler(voice, clientID);
                        break;
                    case NetworkInvoke invoke:
                        InvokeMethodHandler(invoke);
                        break;
                }
            }
            private void InvokeMethodHandler(NetworkInvoke
invoke)
            {
                try
                {
                    var obj = observers.FirstOrDefault(item =>
item.NetworkName == invoke.NetworkName);
                    if (obj != null)
                    {
                        var type =
obj.GetType().GetMethod(invoke.Method);

                        for (var index = 0; index <
invoke.Args.Length; index++)
                        {
                            var arg = invoke.Args[index];
                            invoke.Args[index] = arg switch
                            {
                                long num => num,
                                double number => number,
                                _ => invoke.Args[index]
                            };
                        }
                        switch (invoke.IsOnlyServer)
                        {
                    case true when server is {IsRunning: true}:
                    case false:
         type!.Invoke(obj, invoke.Args ?? Array.Empty<object>());
                                break;
                        }
                    }
                }
                catch (Exception e)
                {                Debug.LogException(e);     }
            }
            private void NetworkVoiceHandler(NetworkVoice voice,
Guid clientId)
            {
                var obj = observers.FirstOrDefault(item =>
item.Id == clientId);
```

```
                if (obj is PlayerMove playerMove && obj.Id !=
client.ClientId)
         {playerMove.PlayVoice(voice.Index, voice.Data); }
            }
       private void NetworkMessageHandler(NetworkMessage data)
            {
                var time = DateTime.Now;
ChatScript.Instance.AddChatMessage(data.Username, data.Text);
            }
            private void
NetworkObjectDataHandler(NetworkObjectData obj)
            {
                var item = observers.FirstOrDefault(item =>
item.Id == obj.Id);
                if (item is not null && item.IsMain) return;
                if (item is null)
                {
                    item = observers.FirstOrDefault(item =>
item.NetworkName == obj.Values["NetworkName"].ToString());

                    if (item is null) return;
                    item.Id = obj.Id;
                    return;
                }
                foreach (var field in obj.Values)
                {
                    switch (field.Value)
                    {
                        case double num:
                            SetCustomAttributeData(item,
field.Key, (float) num);
                            break;
                        case long integer:
                            SetCustomAttributeData(item,
field.Key, (int) integer);
                            break;
                        default:
                            SetCustomAttributeData(item,
field.Key, field.Value);
                            break;
                    }
                }
            }
            private static void SetCustomAttributeData(object
obj, string dataFieldName, object value)
            {
                var type = obj.GetType();

                var members =
type.GetMembers(BindingFlags.Public | BindingFlags.NonPublic |
BindingFlags.Instance)
```

```
                    .Where(m => m.MemberType is
MemberTypes.Property or MemberTypes.Field);
                foreach (var member in members)
                {
                    var attribute =
member.GetCustomAttribute<SyncAttribute>();
                    if (attribute != null && member.Name ==
dataFieldName)
                    {
       if (member.MemberType == MemberTypes.Property)
                        {
              var property = (PropertyInfo) member;
                            if (property.CanWrite)
                            {
                                property.SetValue(obj, value);
                            }
                        }
     else if (member.MemberType == MemberTypes.Field)
                        {
                            var field = (FieldInfo) member;
                            field.SetValue(obj, value);
                        }
                    }
                }
            }
            private async void ConnectHandler(GamePacket packet)
            {
     if (observers.Any(item => item.Id == packet.ClientID))
                {
     Debug.LogError("Already connected to server");
                    return;
                }
                await Awaitable.MainThreadAsync();
                var instPlayer = Instantiate(player,
Vector3.zero, Quaternion.identity);
                var playerMove =
instPlayer.GetComponent<PlayerMove>();
                var isMainPlayer = packet.ClientID ==
client.ClientId;
                playerMove.IsMain = isMainPlayer;
     playerMove.playerCamera.gameObject.SetActive(isMainPlayer);
                playerMove.Id = packet.ClientID;
                playerMove.username = username;
                playerMove.NetworkName = username;
                observers.Add(playerMove);
                players.Add(instPlayer);
                playerMove.transform.position =
ColliderEx.GetRandomPoint(spawnLocation.Start,
spawnLocation.End);
                if (isMainPlayer)
                {
                    menuManager.Visible(false);
                    MainPlayer = playerMove;
```

```
                }
                else
                {
                    playerMove.tag = "OtherPlayer";
                }
                if (packet.ClientID != client.ClientId) return;
                var body = packet.GetDataJson();
                var array = JArray.Parse(body);
                foreach (var item in array)
                {
                    var type =
Type.GetType(item["Type"]!.Value<string>());

                    var obj = (NetworkObject)
JsonConvert.DeserializeObject(item.ToString(), type,
GamePacket.Setting);

                    switch (obj)
                    {
                        case NetworkObjectData data:

     if (observers.Any(obs => obs.Id == data.Id)) continue;

     if (data.ClassType == typeof(PlayerMove).ToString())
                            {
                                CreateOtherPlayer(data);
                            }
                            else if (server == null)
                            {
                                CreateNetworkObject(data);
                            }
                            break;
                        case NetworkMessage message:
                            NetworkMessageHandler(message);
                            break;
                    }
                }
            }
            private void CreateOtherPlayer(NetworkObjectData
networkObjectData)
            {
                var otherPlayer = Instantiate(player,
Vector3.zero, Quaternion.identity);
                var otherPlayerMove =
otherPlayer.GetComponent<PlayerMove>();
                otherPlayerMove.IsMain = networkObjectData.Id ==
client.ClientId;

otherPlayerMove.playerCamera.gameObject.SetActive(false);
                otherPlayerMove.Id = networkObjectData.Id;
                otherPlayerMove.tag = "OtherPlayer";
                foreach (var field in networkObjectData.Values)
                {
```

```
                    switch (field.Value)
                    {
                        case double num:
SetCustomAttributeData(otherPlayerMove, field.Key, (float) num);
                            break;
                        case long number:
SetCustomAttributeData(otherPlayerMove, field.Key, (int)
number);                            break;
                    default:
     SetCustomAttributeData(otherPlayerMove, field.Key,
field.Value);                           break;
                    }
                }
                otherPlayerMove.NetworkName =
otherPlayerMove.username;
                observers.Add(otherPlayerMove);
                players.Add(otherPlayer);
            }
            private void CreateNetworkObject(NetworkObjectData
networkObjectData)
            {
                var name =
networkObjectData.Values["NetworkName"].ToString();
                var type =
Type.GetType(networkObjectData.ClassType);
                var objs = FindObjectsByType(type,
FindObjectsSortMode.None);
                if (!objs.Any()) return;
                var networkObserver =
objs.Cast<NetworkObserver>().FirstOrDefault(item =>
item.NetworkName == name);
                if (networkObserver != null)
                {
                    networkObserver.Id = networkObjectData.Id;
                    AddObserver(networkObserver);
                }
                else
                {
                    var namePrefab = name.Split("_")[0];

                    var prefab = DynamicObjects.Find(item =>
item.Id == namePrefab).Value;

                    var obj = Instantiate(prefab, null);
                    var netObj =
obj.GetComponent<NetworkObserver>();
                    netObj.Id = networkObjectData.Id;
                    netObj.NetworkName = name;
                    AddObserver(netObj);
                }
            }

     private async void DisconnectHandler(GamePacket packet)
```

```
            {
                var clientId = packet.ClientID;
                await Awaitable.MainThreadAsync();
                if (client.ClientId == clientId)
                {
                    foreach (var o in players.ToList())
                        Destroy(o);

                    foreach (var networkObserver in observers)
                        Destroy(networkObserver);


                    observers.Clear();
                    players.Clear();
                }
                else
                {
                    foreach (var o in players.ToList())
                    {
    var playerMove = o.GetComponent<PlayerMove>();
                        if (playerMove.Id == clientId)
                        {
                            players.Remove(o);
                            observers.Remove(playerMove);
                            Destroy(o);
                            break;
                        }
                    }
                }
            }
            public async void SendNetMessage(string text)
            {
                var packet = new GamePacket() {Command =
TypeCommandPacket.Broadcast};
                packet.SetData(new NetworkMessage() {Text =
text, Username = username, IsNeedUpdate = false});
                await client.SendPacket(packet);
            }

            public async void SendNetVoice(int index, float[]
buffer)
            {
                var packet = new GamePacket()
                    {Command = TypeCommandPacket.Broadcast,
IsNeedSave = false, IsNeedConfirm = false};
                packet.SetData(new NetworkVoice() {Index =
index, Data = buffer, IsNeedUpdate = false});

                //packet.Data = packet.Data.Compress();

                await client.SendPacket(packet);
            }
```

```
            public async void SendInvokeMethod(string
networkName, string method, bool onServer, params object[] args)
            {
                var packet = new GamePacket() {Command =
TypeCommandPacket.Broadcast, IsNeedSave = false};

                packet.SetData(new NetworkInvoke()
                    {NetworkName = networkName, Method = method,
Args = args, IsOnlyServer = onServer});

                await client.SendPacket(packet);
            }

            private void OnDestroy()
            {
                client?.DisconnectAsync();
                server?.Stop();
            }

            void OnDrawGizmosSelected()
            {
                var center = (spawnLocation.Start +
spawnLocation.End) / 2;

                // Calculate the size of the parallelepiped
along each axis
                Vector3 size = new Vector3(
                    Mathf.Abs(spawnLocation.End.x -
spawnLocation.Start.x),
                    Mathf.Abs(spawnLocation.End.y -
spawnLocation.Start.y),
                    Mathf.Abs(spawnLocation.End.z -
spawnLocation.Start.z)
                );

                // Draw a wireframe cube to represent the
parallelepiped
                Gizmos.color = Color.green;
                Gizmos.DrawWireCube(transform.position + center,
size);
            }
        }

        [Serializable]
        public class ObjectInfo
        {
            public string Id;
            public GameObject Value;
        }
    }
```