Форма №Н-6.01

Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «магістр»

на тему: «Ігровий онлайн застосунок на основі рушія UNITY»

Виконав:
магістрант 2 курсу
групи М-КН-21
спеціальності
122 «Комп'ютерні науки»
Борисов Максим Віталійович
керівник:
к.пед.н., доцент Петренко С.В.

Рівне – 2023

АНОТАЦІЯ

Борисов М. В. "Ігровий онлайн застосунок на основі рушія UNITY". – Кваліфікаційна робота на здобуття освітнього ступеня "магістр" за спеціальністю 122 Комп'ютерні науки – Рівненський державний гуманітарний університет – Рівне, 2023. - 66 с.

У кваліфікаційній роботі проаналізовано існуючі фреймворки та рішення для розробки сучасних ігрових додатків, а також здійснено їх порівняльний аналіз. Здійснено аналіз існуючого програмного забезпечення та методів для реалізації багатокористувацьких онлайн ігор та мережевої взаємодії в Unity. В цьому аналізі охоплені різні мережеві архітектури, протоколи та бібліотеки.

Було досліджено проблеми мережевої затримки, втрати пакетів та синхронізації. На основі отриманих результатів було запропоновано низку підходів для вирішення існуючих проблем. Окрім того розглянуто методи розробки ігрового процесу та проведено їх аналіз.

В ході практичної реалізації додатка було спроєктовано архітектуру програмної системи: описано структуру системи, побудовано діаграму класів та імплементовано корінь композиції. На кодовому рівні імплементовано серіалізацію даних та реплікацію об'єктів. Також було імплементовано алгоритми та методи для реалізації мережевих функцій, включаючи клієнт-серверний зв'язок, авторитетний сервер та синхронізовані дії гравців. Була розроблена модульна та масштабована архітектура для полегшення майбутнього розвитку та оновлень.

Подальший розвиток проблеми вбачаємо в покращенні імплементованих алгоритмів та додавання нових ігрових світів.

Ключові слова: онлайн ігри, шутер, ігровий рушій, Unity, мережа, протоколи передачі даних, серіалізація, клієнт, сервер.

ANNOTATION

Borysov M. V. "Online gaming application based on the Unity engine" – Qualification work for the Master's Degree in specialty 122 Computer Science – Rivne State University of the Humanities – Rivne, 2023. 66 p.

The qualification work analyzes existing frameworks and solutions for the development of modern game applications, as well as their comparative analysis. The analysis of existing software and methods for implementing multiplayer online games and networking in Unity is carried out. This analysis covers various network architectures, protocols, and libraries.

The problems of network latency, packet loss, and synchronization were investigated. Based on the results, several approaches were proposed to solve the existing problems. In addition, the methods of developing the game process were considered and analyzed.

During the application's practical implementation, the software system's architecture was designed: the system structure was described, a class diagram was built, and the composition root was implemented. At the code level, we implemented data serialization and object replication. We also implemented algorithms and methods for implementing network functions, including client-server communication, authoritative server, and synchronized player actions. A modular and scalable architecture was developed to facilitate future development and updates.

We see further development of the problem in improving the implemented algorithms and adding new game worlds.

Keywords: online games, shooter, game engine, Unity, network, data transfer protocols, serialization, client, server.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Rider – кросплатформне інтегроване середовище розробки програмного забезпечення для платформи . NET, що розробляється компанією JetBrains.

Unity – кросплатформне середовище розробки комп'ютерних ігор, розроблене американською компанією Unity Technologies.

Скриптинг – процес написання ігрового процесу.

LiteNetLib – легка та високопродуктивна мережева бібліотека UDP, призначена для розробки ігор, зокрема Unity.

Рендеринг – процес отримання зображення за моделлю з допомогою комп'ютерної програми.

Онлайн шутер – різновид комп'ютерних ігор, що поєднує основою якого є якого складає стрілянина по цілях, з гравцями підключеними через Інтернет.

Мультиплеєрна гра – є синонімом до слова онлайн гра.

Реплікація – повторення дій клієнта на інших клієнтах та сервері.

P2P чи Peer-to-peer – одна з мережевих топологій.

Пакет – одиниця даних які пересилаються мережею.

HTTPS – схема URI, яка зазвичай використовується для доступу до ресурсів Інтернет

Хедер – мета дані до пакету.

XML – запропонований консорціумом World Wide Web Consortium (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними. застосунками, зокрема, через Інтернет.

JSON – текстовий формат обміну даними, заснований на JavaScript.

Серіалізація – процес перетворення будь-якої структури даних у послідовність бітів.

Десеріалізація – процес перетворення структури бітів у раніше серіалізовану структуру даних.

RTT – тривалість, що вимірюється в мілісекундах, від моменту, коли клієнт надсилає запит, до моменту, коли він отримує відповідь від сервера.

TCP – протокол призначений для керування передаванням даних у комп'ютерних мережах, працює на транспортному рівні моделі OSI.

UDP – один із протоколів в стеку TCP/IP. Від протоколу TCP він відрізняється тим, що працює без встановлення з'єднання.

Ігровий рушій — програмний рушій, центральна програмна частина будь-якої відеогри, яка відповідає за всю її технічну сторону, дозволяє полегшити розробку гри шляхом уніфікації та систематизації її внутрішньої структури.

NDA – це угода, за якою сторони беруть на себе обов'язки у сфері використання і розкриття різного роду інформації (конфіденційної інформації, комерційної таємниці, іншої інформації з обмеженим доступом).

ВСТУП

Актуальність теми дослідження. Ігрова індустрія переживає значне зростання, споживачі витрачають мільярди доларів на відеоігри щороку. Зокрема, багатокористувацькі онлайн ігри стали домінуючою силою в індустрії розваг. Однак розробка таких ігор представляє унікальні виклики через їхню різножанровість. На відміну від традиційних локальних відеоігор, онлайн ігри вимагають ретельного розгляду архітектурних проблем, пов'язаних з мережевою комунікацією.

За кожною відеогрою стоїть комплексна система, за допомогою якої реалізовані всі аспекти гри. Такою системою є ігровий двигун, який складається з переліку двигунів, модулів, інструментів та підсистем, кожна з яких є незамінною та надає свій функціонал. Основні системи, які включає в себе ігровий двигун – це двигун рендерингу ("візуалізатор", 3D чи 2D), двигун фізики, звуку, а також певний функціонал, який забезпечують системи створення сценаріїв, набір візуальних інструментів, анімація, система скриптингу, штучний інтелект, засоби передачі даних через мережу, управління пам'яттю, локалізація, граф сцени. Використання двигуна для розробки ігор на сьогодні є фундаментальним принципом, оскільки це дає певний стек переваг, таких як: кросплатформність кінцевого продукту, можливість повторного використання його для розробки інших ігор, спрощений та швидший процес розробки в цілому, а також, з точки зору бізнесу, здешевлення розробки на всіх етапах.

Більшість двигунів на сьогодні є досить загальними, але все ж таки не надають можливості розробити будь-яку гру. Для розробки гри в тому чи іншому жанрі, можливостей двигуна може бути недостатньо. На допомогу приходить модульність, яка дає змогу реалізувати додатковий функціонал ігрового двигуна для потреби в розробці гри в тому чи іншому жанрі.

Актуальність теми магістерської роботи є нетривіальною, так-як всі розроблені популярні онлайн ігри мають закритий вихідній код через NDA. Методики розробки передаються через конференції по типу GDC. Тому комплексних робіт на цю тему майже немає. Після проведення досліджень було з'ясовано, що ця тема не є досить розвиненою для ігрового двигуна Unity. Один з розроблених інстументів для мережевої взаємодії в цьому рушії компанією Unity Technologies був покинутий та більше не підтримується. Кількість інструментів та аналогів, що існує не є достатнім і не дає можливості швидко і зручно реалізовувати онлайн світи.

Метою роботи є розробка онлайн шутера на основі ігрового рушія Unity.

Для досягнення поставленої мети окреслено такі **завдання дослідження**:

1. Схарактеризувати ігровий жанр шутер, засоби розробки онлайн шутер гри.
2. Розробити алгоритми та методи розробки онлайн ігор; визначити мережеву взаємодію в іграх.
3. Розробити архітектуру програмної системи.
4. Здійснити встановлення, тестування та обчислювальні експерименти.

Об'єктом дослідження є ігровий онлайн застосунок на основі рушія Unity.

Предметом дослідження є розробка ігрового онлайн застосунку на основі рушія Unity.

Для реалізації мети, розв'язання поставлених завдань у дослідженні використано різні **методи та підходи**, використовуючи ігровий рушій Unity та мережеву бібліотеку UDP LiteNetLib, які забезпечують потужну основу для розробки онлайн шутерів та реалізації ефективної і надійної мережевої комунікації. Це дослідження включає вивчення методів боротьби з втратою пакетів, зменшення затримок і підвищення узгодженості ігрового процесу в різних географічних точках.

Зміст розділів пояснювальної записки наступний:

Перший розділ описує основний набір функцій та можливостей онлайн шутер гри в особливості модулю мережевої взаємодії, наводить детальний опис прямих аналогів, а також описує обрані засоби реалізації та обґрунтовує їх вибір.

У другому розділі описуються усі реалізовані методи розробки онлайн шутер гри, їх теоретичне підґрунтя.

Третій розділ містить детальний опис програмної реалізації, з відповідними діаграмами, а також містить опис всіх складових систем, підсистем, класів та їх взаємодію.

Четвертий розділ містить опис системних вимог, демонструє процес запуску гри та можливі комбінації клавіш для гри. Також цей розділ містить таблиці з результатами тестів.

Практичне значення дослідження полягає в ґрунтовному аналізі методів синхронізації, алгоритмів прогнозування та компенсації й застосування кращих практик при імплементації ігрового застосунку. Також розроблений модуль для мережевої взаємодії може використовуватися і в іграх інших жанрів.

Апробація і впровадження результатів дослідження. Основні положення та результати дослідження обговорювалися і були схвалені на звітних наукових конференціях Рівненського державного гуманітарного університету та на XVI Всеукраїнській науково-практичній конференції «Інформаційні технології в професійній діяльності» (м. Рівне, 1 листопада 2023р.) було представлено тези «UNITY як ігровий рушій для гри жанру шутер»

Публікації. За матеріалами дослідження опубліковано тези на XVI Всеукраїнській науково-практичній конференції «Інформаційні технології в професійній діяльності» (м. Рівне, 1 листопада 2023р.) Результати дослідження висвітлено у публікації «UNITY як ігровий рушій для гри жанру шутер».

Структура та обсяг роботи. Магістерська робота складається зі вступу, чотирьох розділів, висновків до розділів, висновків до роботи, списку використаних джерел, додатків. Основний зміст викладено на 66 сторінках.

РОЗДІЛ 1
ЗАСОБИ РОЗРОБКИ ОНЛАЙН ШУТЕР ГРИ

Перед початком реалізації онлайн шутер гри є необхідним – чітко та точно описати задачу та окреслити межі розробки крім того оцінити та проаналізувати можливості аналогів та їх функціонал, а також обрати інструменти та методи розробки, що і описує даний розділ.

1.1 Характеристика ігрового жанру шутер

Онлайн шутери [1, с. 43] зробили революцію в ігровій індустрії, захопивши гравців своїми динамічними, конкурентними багатокористувацькими битвами, які об'єднують людей з усього світу. Ці ігри пропонують захопливий досвід, що характеризується боями з використанням вогнепальної зброї та інтенсивною взаємодією гравців у реальному часі.

Мультиплеєрні шутери охоплюють широкий спектр ігрових механізмів, але, як правило, вони наголошують на точному прицілюванні, стрільбі та русі. Гравці повинні відточувати свої навички влучності та прийняття тактичних рішень, щоб перехитрити супротивників.

Популярність онлайн шутерів продовжує стрімко зростати, мільйони гравців щодня беруть участь у багатокористувацьких баталіях. Привабливість шутерів зумовлена змагальним характером, можливостями соціальної взаємодії та постійними оновленнями, які розробники надають для того, щоб ігровий процес залишався свіжим і цікавим. З перших днів багатокористувацьких ігор такі ігри, як Doom і Quake, проклали шлях для розвитку жанру. Технологічний прогрес і високошвидкісне підключення до інтернету породили складні онлайн шутери, такі як Counter-Strike, Halo, Battlefield рисунок 1.1.

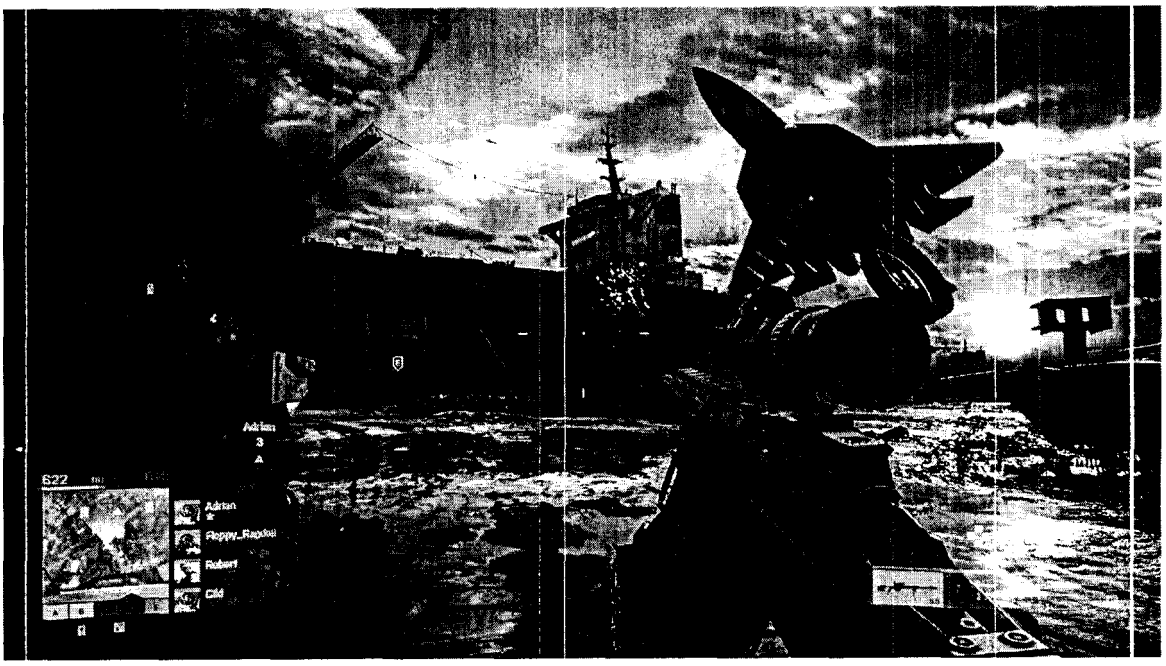

Рис. 1.1 Знімок екрану ігрового процесу гри Battlefield.

У межах жанру шутерів існують різні піджанри, що задовольняють різноманітні вподобання гравців. Яскравими прикладами є шутери від першої особи (FPS), шутери від третьої особи, тактичні шутери та ігри в жанрі королівської битви. Культовою грою залишається Counter-Strike, відома своїм стратегічним ігровим процесом і командними битвами. Франшиза Call of Duty також залишила значний слід, пропонуючи інтенсивні однокористувацькі кампанії поряд з потужними багатокористувацькими режимами. Останніми роками з'явилися ігри в жанрі королівської битви: Fortnite та Apex Legends зачаровують гравців своїм масштабним геймплеєм, що побудований на боротьбі до останнього бійця.

Онлайн шутери стали домінуючою силою в ігровій індустрії, забезпечуючи динамічний, конкурентний багатокористувацький досвід, який захоплює гравців по всьому світу. Постійні інновації та підтримка розробників стимулюють розвиток жанру, забезпечуючи постійну залученість і захопленість гравців. З розвитком технологій онлайн шутери, безсумнівно,

продовжуватимуть розширювати межі і дарувати гравцям захопливий досвід, яким вони зможуть насолоджуватися.

1.2 Задача розробки онлайн шутер гри

Метою цієї дипломної роботи є розробка онлайн шутера на основі ігрового рушія Unity [2, с. 22] та дослідження мережевих аспектів. Основна мета полягає у створенні захопливої та безперебійної багатокористувацької гри, одночасно вирішуючи проблеми мережевої затримки, втрати пакетів та синхронізації.

Для досягнення цієї мети були сформульовані наступні завдання:

- Проаналізувати існуюче програмне забезпечення та методи для реалізації багатокористувацьких онлайн ігор та мережевої взаємодії в Unity. Цей аналіз охоплюватиме різні мережеві архітектури, протоколи та бібліотеки.
- Визначити та дослідити основні підсистеми, необхідні для реалізації онлайн шутера. Сюди входять такі компоненти, як рух гравця, ігрова фізика, система підрахунку очок і підбір гравців.
- Вибрати відповідні інструменти, фреймворки та методи для реалізації ігрових і мережевих компонентів.
- Створити структуру програмної системи, включаючи організацію скриптів, активів та ресурсів в рамках проекту Unity. Це завдання передбачає розробку модульної та масштабованої архітектури для полегшення майбутнього розвитку та оновлень.
- Розробка алгоритмів і методів для реалізації мережевих функцій, включаючи клієнт-серверний зв'язок, авторитетний геймплей сервера та синхронізовані дії гравців. Це включає роботу з мережевими затримками,

управління взаємодією гравців та забезпечення справедливого ігрового процесу в різних мережевих умовах.

Потенційними користувачами онлайн шутера є також ігрові студії, що використовують ігровий рушій Unity. Однак основною цільовою аудиторією цього проекту є інді-розробники та інді-студії [3]. Така спрямованість пояснюється тим, що розроблена гра буде з відкритим вихідним кодом і вільно доступна для всіх бажаючих, які зможуть використовувати її та налаштовувати відповідно до своїх вимог.

Надаючи доступне і безкоштовне рішення, цей онлайн шутер має на меті дати можливість незалежним розробникам і студіям створювати захоплюючі багатокористувацькі ігри без фінансових обмежень, пов'язаних з пропрієтарними ігровими рушіями. Відкритість проекту заохочує співпрацю, інновації та обмін знаннями у спільноті інді-розробників ігор.

1.3 Існуючі рішення для розробки онлайн ігор

Коли справа доходить до створення онлайн шутерів в Unity, розробники мають на вибір кілька мережевих рішень. Давайте розглянемо та порівняємо фреймворки UNET, Mirror, Photon, SmartFoxServer, LiteNetLib.

UNET, Unity Multiplayer High-Level API [4], пропонує вбудовані мережеві функції в Unity. Однак, одним з його основних недоліків є його застарілий статус. UNET більше не підтримується та не оновлюється Unity Technologies, що означає, що розробники можуть зіткнутися з проблемами сумісності з новими версіями Unity. Крім того, відсутність офіційної підтримки UNET означає обмеженість ресурсів для усунення несправностей та вирішення потенційних проблем, пов'язаних з мережею.

MirrorNetworking [5], альтернатива UNET з відкритим вихідним кодом, має на меті усунути деякі з обмежень UNET. Однак Mirror також має свої недоліки. Як проект, керований спільнотою, він може мати меншу базу користувачів і менш вичерпну документацію порівняно з іншими рішеннями. Розробники можуть зіткнутися з труднощами у пошуку конкретних відповідей або отриманні своєчасної підтримки щодо складних мережевих питань. Крім того, хоча Mirror пропонує гнучкість, він може вимагати додаткових зусиль від розробників для впровадження розширених функцій та кастомних рішень. Також у даної бібліотеки є проблеми з оптимізацією та авторитарністю серверів.

Photon [6], наданий ExitGames, є популярним хмарним мережевим рішенням для Unity. Хоча Photon пропонує простоту використання та масштабованість, важливо звернути увагу на його цінову структуру. Розробники можуть зіткнутися з тим, що вартість хостингу може швидко зростати зі збільшенням кількості гравців. Крім того, оскільки Photon є хмарним рішенням, існує певна залежність від зовнішніх серверів, що може призвести до затримок або потенційних простоїв, якщо серверна інфраструктура зіткнеться з проблемами. Хоча дана бібліотека має і безплатне рішення, але воно підтримує тільки P2P архітектуру.

SmartFoxServer [7] – це потужне багатокористувацьке серверне рішення для ігор. Однак, одним з його головних недоліків є вартість ліцензії. SmartFoxServer вимагає комерційної ліцензії для масштабних проектів, що може бути значною інвестицією для інді-розробників або невеликих студій з обмеженим бюджетом. Крім того, крива навчання для SmartFoxServer може бути крутішою порівняно з іншими рішеннями, що робить його більш складним для розробників, незнайомих з адмініструванням серверів та мережевими концепціями. Також SmartFoxServer написаний на мові програмування Java, а це

значить що розробникам на Unity потрібно ще знати дану мову програмування, щоб писати серверну логіку.

LiteNetLib [8] – це надійна мережева бібліотека на основі UDP, яка забезпечує швидкий та ефективний зв'язок між клієнтами та серверами. Вона надає такі функції, як надійна та ненадійна доставка повідомлень, автоматична фрагментація пакетів, імітація втрати пакетів та затримок, що налаштовуються. Але головною проблемою даної бібліотеки є те, що вона є бібліотекою низького рівня, і реалізує тільки передачу пакетів по мережі.

Проаналізувавши існуючі рішення, ми прийшли до висновку реалізувати власний модуль мережевої взаємодію, на основі існуючої бібліотеки LiteNetLib. Детальне порівняння розробленого модуля з аналогами наведено в таблиці 1.1.

Таблиця 1.1

Порівняльна характеристика розробленого модуля з аналогами

Характеристика	Наш модуль	LiteNetLib	UNET	Photon	Photon (бесплатна версія)	Mirror
Безкоштовність	+	+	+	-	+	+
Повна авторитарність серверу	+	-	-	+	-	-
Оптимізація	+	+	-	+	+	-
Підтримка клієнт серверна взаємодії	+	-	+	+	-	+
Підтримка	+	+	-	+	+	+

Open source	+	+	-	-	-	+
Високорівнева інтергація з Unity	+	-	+	+	+	+

1.4 Засоби реалізації

1.4.1 Ігровий двигун Unity

Unity – популярний і широко використовуваний рушій для розробки ігор, відомий своєю універсальністю, простотою використання та потужними можливостями. Він надає розробникам повний набір інструментів та ресурсів для створення інтерактивних та візуально привабливих ігор на різних платформах. У цьому розділі ми обговоримо ключові особливості та функціональні можливості рушія Unity, з акцентом на його актуальність для розробки онлайн шутера.

Unity пропонує ряд можливостей, які роблять його ідеальним вибором для розробки онлайн шутерів. Ці можливості включають:

- Крос-платформна [9] підтримка: Unity підтримує безліч платформ, включаючи ПК, консолі, мобільні пристрої і навіть платформи доповненої/віртуальної реальності (AR/VR) [10, с. 84]. Це дає змогу розробникам орієнтуватися на широку аудиторію та максимізувати охоплення гри.
- Надійний графічний рушій: Графічний рушій Unity надає високоякісні можливості рендерингу, включаючи тіні в реальному часі, ефекти постобробки та підтримку новітніх технологій рендерингу. Ці функції

допомагають створювати візуально приголомшливе середовище та посилюють ефект занурення для гравців.

- Написання сценаріїв та кастомізація: Unity підтримує C# як основну мову сценаріїв, що дозволяє розробникам писати код для керування механіками ігрового процесу, реалізації мережевих функцій та управління ігровою логікою. Розширюваність рушія також дозволяє розробникам створювати кастомні інструменти та розширення редакторів для оптимізації робочого процесу.
- Фізичне моделювання: Вбудований фізичний рушій Unity забезпечує реалістичне виявлення зіткнень, динаміку жорстких тіл та суглобових систем. Ці можливості фізичного моделювання мають вирішальне значення для створення точних і правдоподібних взаємодій в ігровому світі, включаючи фізику снарядів і рух персонажів.
- Інтегроване середовище розробки (IDE): IDE Unity, що називається UnityEditor рисунок 1.2 , пропонує зручний інтерфейс, який спрощує розробку ігор. Він надає візуальний редактор для створення сцен, управління ресурсами та інструменти для налагодження, щоб полегшити процес розробки та тестування.
- Система анімації: Unity пропонує потужну систему анімації з підтримкою скелетної анімації, дерев змішування та автоматів. Це дозволяє розробникам створювати реалістичну анімацію персонажів та інтерактивні ігрові послідовності.
- Мобільна оптимізація: Unity пропонує функції оптимізації, спеціально розроблені для мобільних платформ, забезпечуючи ефективне використання пам'яті, оптимізацію продуктивності та сумісність з широким спектром мобільних пристроїв.

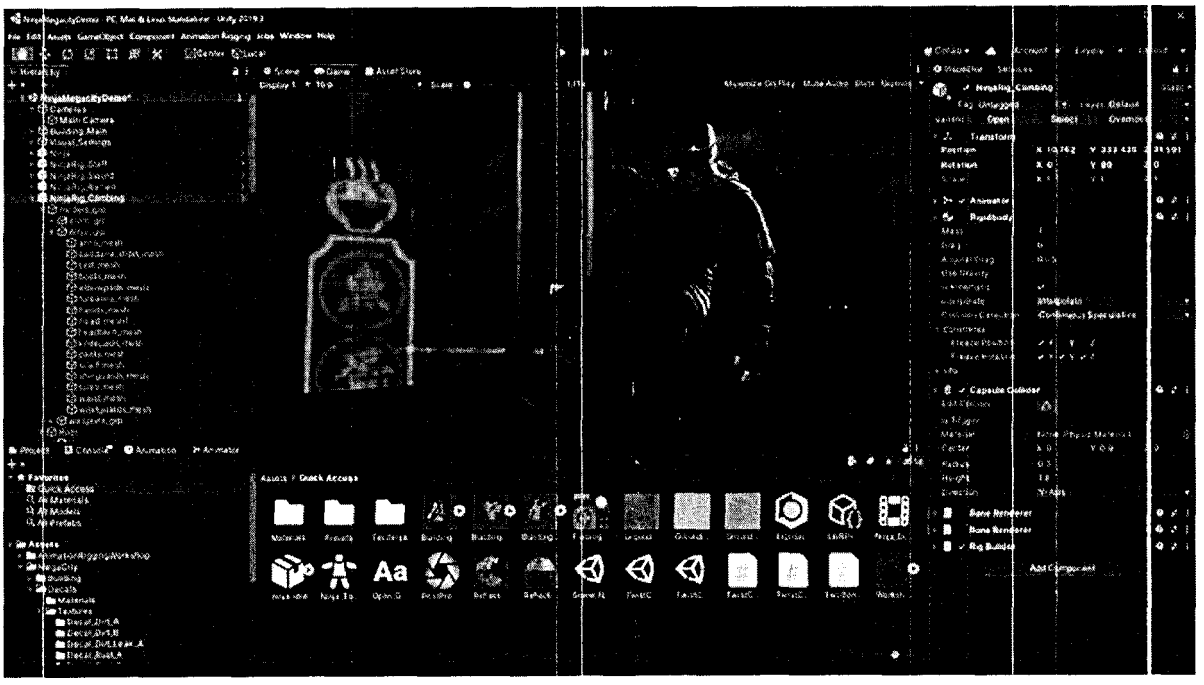

Рис. 1.2 Редактор ігрового рушія Unity.

1.4.2 Мова програмування, середовище розробки

C# [11, с. 34] – це основна мова програмування, що використовується в ігровому рушію Unity. Це універсальна і широко прийнята об'єктно-орієнтована мова програмування, яка забезпечує міцну основу для розробки ігор та інтерактивних додатків. У цьому розділі ми розглянемо значення C# в Unity та її внесок у розробку онлайн шутерів.

Інтеграція Unity з C# має декілька переваг для розробників ігор. Ось декілька ключових причин, чому C# є найкращою мовою для розробки ігор у Unity:

Звичний синтаксис: C# має синтаксис, схожий на інші популярні мови програмування, такі як Java [12, с. 21] та C++ [13, с. 15], що робить її відносно простою у вивченні та дозволяє ефективно співпрацювати з іншими розробниками: C# слідує об'єктно-орієнтованій парадигмі програмування, що

дозволяє розробникам створювати модульний і багаторазово використовуваний код. Цей підхід сприяє організації коду, інкапсуляції та підтримці, що робить його добре придатним для великомасштабних проектів з розробки ігор.

Unity надає багатий API , спеціально розроблений для C#, що пропонує прямий доступ до широкого спектру можливостей та функцій. Цей API дає розробникам можливість легко маніпулювати ігровими об'єктами, керувати сценами, працювати з фізичними симуляціями та впроваджувати мережеві функції.

Unity використовує архітектуру C#, керовану подіями, що дозволяє розробникам реагувати на певні ігрові події та запускати відповідні дії. Система подій та зворотних викликів Unity забезпечує гнучкий та інтуїтивно зрозумілий спосіб обробки вводу, анімації, зіткнень та інших подій, пов'язаних з грою.

Інтеграція з редакторами: C# легко інтегрується з редактором UnityEditor, забезпечуючи потужне середовище розробки. Редактор Unity пропонує інструменти для завершення коду, налагодження та візуальну систему сценаріїв (Unity'sPlaymaker) [14], які доповнюють досвід програмування на C#, роблячи розробку більш ефективною та впорядкованою.

Rider IDE [16] рисунок 1.3 -- це потужне інтегроване середовище розробки, розроблене компанією JetBrains, відомою своїми потужними можливостями для редагування, налагодження та рефакторингу коду. Rider пропонує безшовну інтеграцію з Unity, забезпечуючи ефективне та багатофункціональне середовище для розробки ігор у Unity.

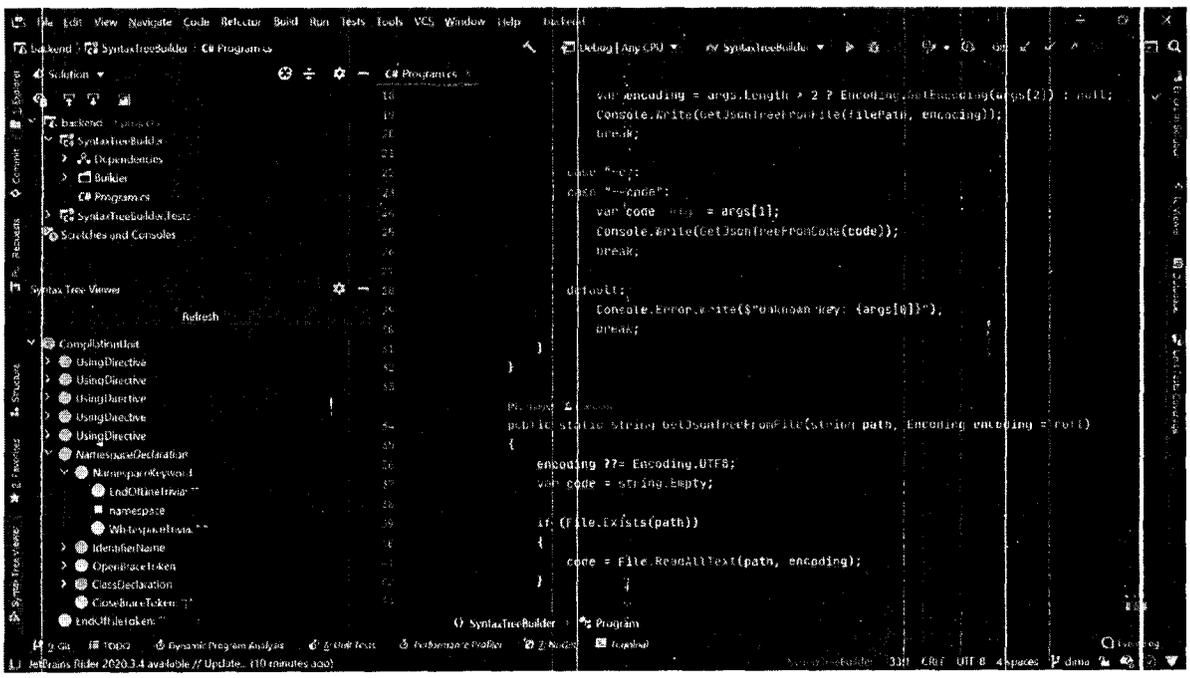

Рис. 1.3 Знімок IDE Rider.

Rider пропонує розширені можливості редагування коду, включаючи завершення коду, перевірку коду та інтелектуальну навігацію по коду. Ці функції допомагають розробникам швидше писати код, виявляти потенційні проблеми та легко орієнтуватися у великих проектах Unity, підвищуючи продуктивність та скорочуючи час розробки.

Надає надійні засоби налагодження для проектів Unity. Розробники можуть встановлювати точки зупинки, переглядати код, перевіряти змінні та аналізувати стек викликів для ефективного пошуку та виправлення помилок. Крім того, Rider пропонує інструменти профілювання для оптимізації продуктивності шляхом виявлення вузьких місць та оптимізації виконання коду.

Включає потужні інструменти рефакторингу [17 с. 67], які допомагають у реструктуризації та оптимізації коду. Розробники можуть безпечно перейменовувати змінні, витягувати методи, змінювати порядок коду та

виконувати інші автоматизовані рефакторинги коду, забезпечуючи зручність супроводу коду та покращуючи загальну якість кодової бази.

Легко інтегрується з популярними системами контролю версій, такими як Git, що дозволяє розробникам керувати змінами вихідного коду та ефективно співпрацювати з членами команди. Rider IDE надає візуальні інструменти для перевірки відмінностей, управління гілками та інші функції контролю версій безпосередньо в IDE.

1.4.3 Використані засоби Unity

Для розробки деяких систем онлайн гри були використані вбудовані інструменти Unity. На їх основі були реалізовані власні системи, які виступають як обгортки, над інструментами Unity, чи просто використовують даний інструмент. Перелік даних інструментів:

MeshRenderer – відповідає за рендеринг 3D-сіток у Unity. Він дозволяє відображати текстури, матеріали та шейдери на 3D-об'єктах, надаючи їм візуальне представлення у ігровому світі. Компонент MeshRenderer працює разом з компонентом MeshFilter для рендерингу складних 3D моделей з нанесеними текстурами та матеріалами.

MeshFilter – дозволяє прикріплювати дані тривимірної сіті до ігрових об'єктів. Він діє як контейнер для геометрії сіті, визначаючи форму і структуру 3D-об'єкта. MeshFilter у поєднанні з MeshRenderer дозволяє рендерити детальні 3D моделі у Unity.

Animator – є невід'ємною частиною створення анімації персонажів та керування їхньою поведінкою в Unity. Він дозволяє реалізовувати складні анімації, включаючи бездіяльність, ходьбу, біг та атаку. Використовуючи компонент Animator, розробники можуть створювати реалістичні рухи персонажів та плавні переходи між різними станами анімації.

ParticleSystem – дозволяє створювати та маніпулювати ефектами частинок в Unity. Він пропонує широкий спектр параметрів та налаштувань для створення різноманітних візуальних ефектів, таких як вогонь, дим, вибухи та магічні заклинання. Компонент ParticleSystem надає широкий контроль над поведінкою, розміром, кольором та рухом частинок, дозволяючи розробникам створювати візуально приголомшливі та динамічні ефекти.

Система освітлення Unity відіграє вирішальну роль у створенні реалістичного та візуально привабливого ігрового середовища. Вона включає в себе різні компоненти освітлення, такі як спрямовані прожектори, точкові прожектори, прожектори та зонні прожектори, які вносять свій внесок у загальне освітлення сцени. Налаштовуючи властивості освітлення та регулюючи тіні і відображення, розробники можуть створювати захоплюючі та атмосферні ігрові світи.

Колайдери є важливими компонентами для реалізації виявлення зіткнень та фізичної взаємодії в Unity. Unity пропонує ряд компонентів колайдерів, включаючи BoxCollider, SphereCollider, CapsuleCollider та MeshCollider, які визначають фізичні межі та форми ігрових об'єктів. Колайдери забезпечують точне виявлення зіткнень, реакцію на них та фізичне моделювання, що дозволяє реалістично взаємодіяти між об'єктами в ігровому світі.

GameObject та MonoBehaviour є фундаментальними компонентами в Unity. GameObjects слугують будівельними блоками ігрового світу, представляючи собою сутності, якими можна маніпулювати та взаємодіяти з ними. MonoBehaviour є базовим класом для скриптів у Unity, що дозволяє розробникам додавати до GameObjects власну поведінку та функціональність. Приєднавши скрипти MonoBehaviour до GameObjects, розробники можуть керувати ігровою логікою, реалізовувати обробку вводу та керувати взаємодією між ігровими об'єктами.

1.4.4 Бібліотека протоколу датаграм користувача LiteNetLib

LiteNetLib – це легка та високопродуктивна мережева бібліотека UDP, призначена для розробки ігор, зокрема Unity. Вона забезпечує надійний та ефективний комунікаційний рівень для багатокористувацьких ігор, дозволяючи швидко та безперебійно грати у мережевому режимі. У цьому розділі ми розглянемо можливості та переваги LiteNetLib та її інтеграцію з Unity.

LiteNetLib побудовано на транспортному рівні UDP (UserDatagramProtocol), який забезпечує низьку затримку та високу швидкість передачі даних. UDP добре підходить для багатокористувацьких ігор у реальному часі, де низька затримка має вирішальне значення, оскільки він забезпечує швидшу передачу даних у порівнянні з TCP (TransmissionControlProtocol). Дану бібліотеку розроблено для того, щоб бути легким та ефективним, що призводить до мінімального впливу на продуктивність гри. Він оптимізований для високошвидкісної обробки даних і низького використання пам'яті, що дозволяє іграм ефективно обробляти велику кількість одночасних з'єднань і мережевих об'єктів.

LiteNetLib розроблений для безперебійної роботи на різних платформах, включаючи Windows, macOS, Linux, iOS та Android. Ця крос-платформна сумісність дозволяє розробникам створювати багатокористувацькі ігри, в які можна грати на різних пристроях та операційних системах.

Висновки до першого розділу

У розділі схарактеризовано ігровий жанр шутер, проаналізовано основний набір функцій та можливостей онлайн шутер гри в особливості модулю мережевої взаємодії.

Визначені задачі розробки онлайн шутер гри, систематизовано існуючі рішення для розробки онлайн ігор, наведено детальний опис прямих аналогів.

У розділі також описано обрані засоби реалізації: ігровий двигун Unity, мова програмування та середовище розробки, використані засоби Unity, бібліотека протоколу датаграм користувача LiteNetLib та обґрунтовано їх вибір.

РОЗДІЛ 2
АЛГОРИТМИ ТА МЕТОДИ РОЗРОБКИ ОНЛАЙН ІГОР

Невід'ємною частиною онлайн ігор є мережева взаємодія між клієнтами, яка використовується для синхронізація та підтвердження дій гравця, з метою, щоб запобігти шахрайству. Також важливими частинами будь-якої гри є її користувацький інтерфейс та ігровий процес, що надає насолоду гравцю під час ігрової сесії. Далі опишемо мережеву взаємодії між клієнтами, розглянемо різні мережеві топології та архітектури побудови ігрового процесу.

2.1 Мережева взаємодія в іграх

Мережева взаємодія в іграх – це важливий аспект, який уможливлює багатокористувацький ігровий процес та онлайн взаємодію. Вона передбачає зв'язок і синхронізацію між кількома пристроями або гравцями через мережу. Мережа дозволяє гравцям з'єднуватися, взаємодіяти і змагатися в режимі реального часу, створюючи захопливий і динамічний ігровий досвід. Вона передбачає передачу даних, таких як введення гравців, оновлення стану гри та аудіовізуальної інформації для забезпечення послідовного та синхронізованого ігрового процесу. Ефективна мережева реалізація має важливе значення для мінімізації затримок, забезпечення чесного ігрового процесу та безперебійної і приємної багатокористувацької гри для гравців по всьому світу. Далі ми розглянемо методи та алгоритми для встановлення стабільної мережевої взаємодії.

2.1.1 Протокол пересилання даних

Мережева комунікація відіграє вирішальну роль у забезпеченні багатоко ристувацької функціональності в іграх. На відміну від веб-додатків, які в основному покладаються на HTTPS [18, с. 34] для зв'язку, мережева взаємодія в іграх вимагає іншого підходу через те, що інтерактивний ігровий процес працює в режимі реального часу і вимагає високої продуктивності рисунок 2.1.

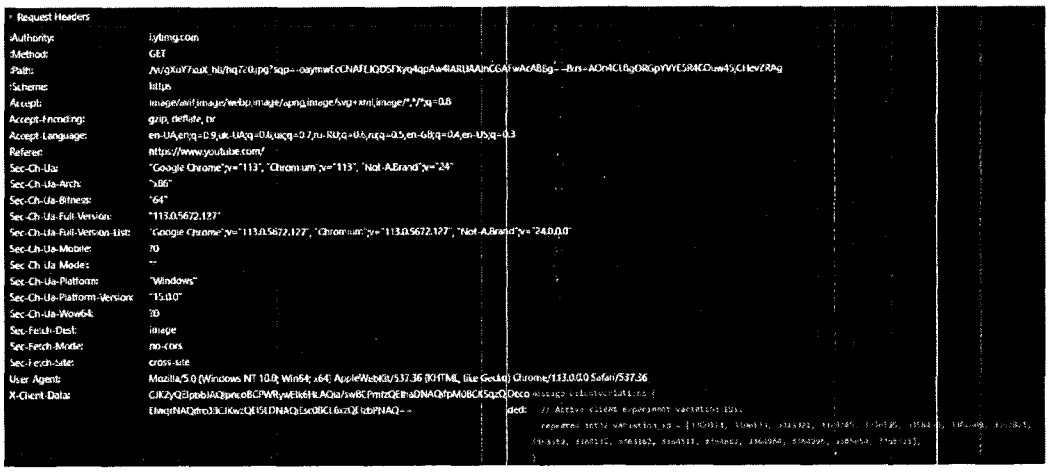

Рис. 2.1 Хедер HTTPS запиту до сайту youtube.com.

Як ми можемо побачити на рисунку 2.1. HTTPS протокол надсилає також інформацію яка не потрібна в ігрових додатках. Надсилання одного пакету в онлайн іграх відбувається як мінімум 30 раз на секунду, тому даний протокол не підходить для мережевої комунікації в ігрових додатках.

Ігри вимагають взаємодії між гравцями в режимі реального часу, де навіть незначні затримки можуть суттєво вплинути на ігровий процес. На відміну від веб-додатків, які можуть терпіти певні затримки в передачі даних, ігри повинні підтримувати низьку мережеву затримку, щоб забезпечити плавний ігровий процес. Це зумовлює необхідність застосування спеціально розроблених для ігор методів оптимізації для зменшення мережевих затримок, таких як мінімізація розміру пакетів, ефективне стиснення даних і визначення пріоритетів для критично важливих ігрових даних.

Тут на допомогу розробниками ігрових додатків приходить протокол користувацьких дейтаграм (UDP) [19]. На відміну від протоколу керування передачею (TCP), на основі якого побудований протокол HTTPS. UDP – це легкий протокол без підключення, який пропонує низькі накладні витрати і мінімальні затримки, що робить його добре придатним для додатків у реальному часі, таких як ігри. На відміну від TCP [20, с. 101], UDP не гарантує доставку повідомлень, впорядкування або надійність «з коробки» (див. Таблиця 2.1.). Однак цей компроміс дозволяє іграм надавати пріоритет швидкості та відгуку над безпомилковою передачею даних, оскільки певні мережеві збої можуть бути краще оброблені за допомогою специфічних для гри механізмів, таких як інтерполяція, прогнозування та узгодження на стороні клієнта.

Таблиця 2.1

Порівняльна характеристика TCP та UDP протоколів

Суб'єкт порівняння	**TCP**	**UDP**
Швидкість	Швидкість повільніша, ніж UDP.	Швидше, ніж TCP. Це транспортний протокол у режимі реального часу.
Надійність	Він надзвичайно надійний, оскільки завдяки процесу підтвердження забезпечує належну доставку пакету даних до вузла призначення.	Доставка пакетів даних не забезпечується, отже це ненадійний протокол.
Розмір заголовка	20 байт	8 байт

Для вирішення таких проблем, як включення порядкових номерів, підтверджень і методів узгодження даних. Крім того, природа UDP без з'єднання означає, що потрібні додаткові зусилля для управління взаємодією між клієнтом і сервером або іншим клієнтом. Така гнучкість дозволяє

розробникам адаптувати мережеве рішення до потреб своєї гри, забезпечуючи оптимальну продуктивність і швидкість реакції.

2.1.2 Топологія клієнт – сервер

У клієнт-серверній топології [21] один екземпляр гри називається сервером, а всі інші екземпляри гри називаються клієнтами. Кожен клієнт взаємодіє лише з сервером, а сервер відповідає за взаємодію з усіма клієнтами. Рисунок 2.2 ілюструє цю топологію.

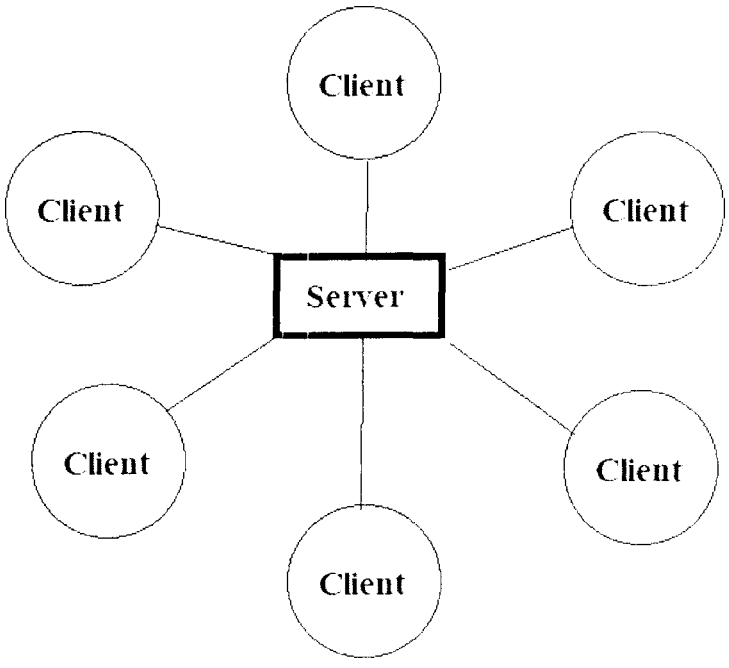

Рис. 2.2 Клієнт–серверна топологія.

У топології клієнт-сервер для n клієнтів існує загалом $O(2n)$ з'єднань. Однак вона є асиметричною, оскільки сервер матиме $O(n)$з'єднань (по одному з кожним клієнтом), в той час як кожен клієнт матиме лише одне з'єднання з сервером. З точки зору пропускної здатності, якщо є n клієнтів і кожен клієнт надсилає b байт в секунду даних, сервер повинен мати достатню пропускну

здатність, щоб обробити $b \times n$ вхідних байт в секунду. Аналогічно, якщо серверу потрібно відправляти c байт в секунду даних кожному клієнту, сервер повинен підтримувати $c \times n$ вихідних байт в секунду. Однак, кожному клієнту потрібно підтримувати лише c байт на секунду у низхідному потоці і b байт на секунду у висхідному потоці. Це означає, що зі збільшенням кількості клієнтів пропускна здатність, необхідна для сервера, буде лінійно зростати. Теоретично, вимоги до пропускної здатності для клієнта не змінюються залежно від кількості клієнтів. Однак на практиці підтримка більшої кількості клієнтів призводить до збільшення кількості об'єктів у світі, які потрібно реплікувати, що може призвести до незначного збільшення пропускної здатності для кожного клієнта.

Хоча це далеко не єдиний підхід до клієнт-сервер, більшість ігор, які реалізують клієнт-сервер, використовують авторитетний сервер. Це означає, що симуляція ігровим сервером гри вважається коректною. Якщо клієнт коли-небудь виявить незгоду з сервером, він повинен оновити свій ігровий стан на основі того, що сервер вважає ігровим станом.

Існує також підкласифікація типів серверів. Деякі сервери є виділеними, тобто що на них запускається лише ігровий стан і здійснюється зв'язок з усіма клієнтами. На виділеному сервері повністю відокремлений від будь-яких клієнтських процесів, що запускають гру. Це означає, що виділений сервер зазвичай є headless і не відображає жодної графіки. Цей тип серверів часто використовується у великобюджетних іграх, таких як Battlefield, що дозволяє розробнику запускати кілька процесів виділеного сервера на одній потужній машині.

Альтернативою виділеному серверу є сервер прослуховування. У цій конфігурації сервер також є активним учасником самої гри. Однією з переваг конфігурації сервера прослуховування є те, що вона може зменшити витрати на розгортання, оскільки немає необхідності орендувати сервери в дата-центрі натомість один з гравців може використовувати свій комп'ютер і як сервер, і як

клієнт. Однак недоліком сервера прослуховування є те, що машина, яка працює як сервер прослуховування, повинна бути достатньо потужною достатньо потужною і мати достатньо швидке з'єднання, щоб впоратися з таким підвищеним навантаженням. Сервер прослуховування іноді помилково називають P2P з'єднанням, але більш точний термін - одноранговий хостинг. Сервер все одно існує, просто так сталося, що його розміщує гравець у грі.

2.1.3 Топологія Peer-to-peer

В Peer-to-peer [22] топології кожен окремий учасник з'єднаний з кожним іншим учасником. Як видно з Рисунку 2.3, це означає, що між клієнтами передається велика кількість даних туди і назад між клієнтами. Іншими словами, кількість з'єднань є квадратичною функцією, для заданих n одноранговых комп'ютерів кожен з них повинен мати $O(n-1)$ з'єднань, що призводить до $O(n^2)$ з'єднань з'єднань по всій мережі. Це також означає, що вимоги до пропускної здатності для кожного одноранговых гравця зростають оскільки до гри підключається все більше і більше гравців. Однак, на відміну від клієнт-сервер, вимоги до пропускної здатності симетричні, тож кожен клієнт вимагатиме однакового обсягу доступної пропускної здатності як на виході, так і на вході.

Поняття авторитету є набагато більш розмитим у грі «peer-to-peer». Один з можливих підходів полягає в тому, що певні гравці мають владу над певними частинами гри, але на практиці таку систему може бути важко реалізувати. Більш поширений підхід в peer-to-peer іграх полягає в тому, що всі дії розподіляються між усіма учасниками гри, і кожен учасник імітує ці дії. Цю модель іноді називають моделлю спільного доступу до даних.

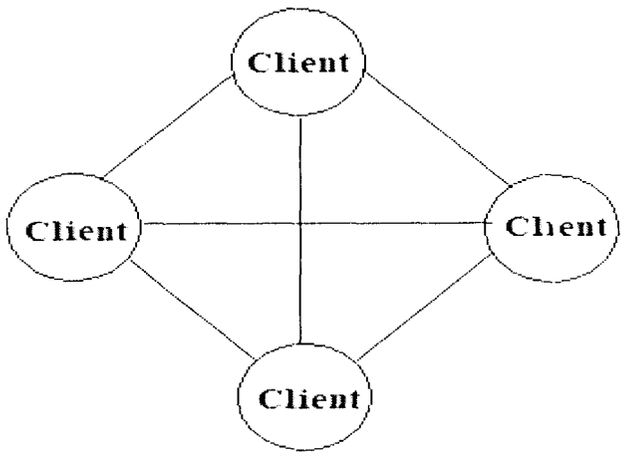

Рис. 2.3 Peer-to-peer топологія.

Крім того, важливо забезпечити узгодженість стану гри між усіма учасниками. Це означає, що реалізація гри має бути повністю детермінованою. Іншими словами, заданий набір вхідних даних повинен завжди призводити до однакових результатів. Кілька важливих аспектів цього включають використання контрольних сум для перевірки узгодженості стану гри між учасниками та синхронізація генерації випадкових випадкових чисел між усіма учасниками – обидві теми детально розглядаються далі у цій главі.

Ще однією проблемою, яка виникає в peer-to-peer мережі, є підключення нових гравців. Оскільки кожен учасник повинен знати адресу кожного іншого учасника, теоретично новий гравець може підключитися до будь-якого учасника. Однак сервіси пошуку партнерів, які показують список доступних ігор, зазвичай приймають лише одну адресу – в цьому випадку один з однорангових гравців може бути обраний так званим головним peer-to-peer гравцем.

2.1.4 Методи синхронізації клієнтів

Схарактеризуємо «Стан проти вводу». Це, мабуть, перше важливе рішення, яке потрібно прийняти при проектуванні того, як ваша гра буде

працювати в мережі. Щоб увімкнути мережеву багатокористувацьку гру, потрібно обмінюватися даними між усіма гравцями, щоб переконатися, що всі гравці мають однаковий ігровий стан. Виникає питання: що саме передавати? Існує три загальних підходи, і вони суттєво вплинуть на ігровий код.

1. Клієнт може надсилати серверу стан гри:

- Гравець А надсилає пакет в якому він інформує сервер що він переміщується на позицію (1;2);
- Сервер отримує запит і повідомляє всіх гравців, що гравець А зараз знаходиться у точці (1, 2)

2. Клієнт може надсилати данні воду користувача та отримувати стан гри з серверу:

- Гравець А надсилає пакет з даними вводу гравця «ВПЕРЕД» , та рухається відповідно, виконуючи логіку гри, виконуючи передбачення.
- Сервер отримує пакет, обробляє його та надсилає усім клієнтам.
- Клієнт отримує пакет з серверу, та перевіряє чи його передбачення було правильним

3. Клієнт надсилати дані вводу користувача всім клієнтам:

- Гравець А надсилає пакет з даними вводу гравця «ВПЕРЕД» всім гравцям.
- Гравець А отримує підтвердження від усіх гравців і переміщується на «ВПЕРЕД». Гравець А на екрані всіх інших гравців переміщується на «ВПЕРЕД».

Перший тип є найпростішим, але наймеш оптимізованим та ніяк не протидіє шахрайству. Тому в подальшому розглядати його ми не будемо.

Другий тип синхронізації є найпоширенішим в ігровій індустрії, а в жанрі шутер незамінним.

Третій тип використовується для ігрових додатків з топологіє peer-to-peer.

Таблиця 2.2

Порівняльна характеристика методів синхронізації клієнтів

Характеристика	**Другий тип**	**Третій тип**
Використання трафіку мережі	Високий	Низький
Перепідключення	Відносно легко, гравцеві просто потрібно отримати останній стан гри з серверу	Відносно складно. Вам потрібен вузол ретрансляції для зберігання всіх вводів клієнтів. Коли гравець перепідключиться, всі вводи будуть повторно надіслані та відтворені на стороні гравців
Розмір файлу відтворення	Великий	Маленький
Протидія шахрайству	Відносно легко	Майже неможливо
Детерміністична логіка	Не обов'язкова	Обов'язкова
Жанр ігор	Шутер	RTS

2.1.5 Затримка, пропускна здатність і втрата пакетів

Затримка, яку часто називають «пінг» – це часова затримка між тим, як гравець виконує дію, і тим, як він отримує відповідну відповідь. На неї насамперед впливає фізична відстань між гравцями та задіяна мережева інфраструктура. Вища затримка може призвести до затримки відповідей, що робить ігровий процес менш плавним. У динамічних іграх, таких як шутери, низька затримка має вирішальне значення для забезпечення взаємодії в реальному часі. На затримку, що вимірюється в мілісекундах, впливають різні

фактори, зокрема перевантаження мережі, неефективність маршрутизації та якість мережевого з'єднання. Візуалізація затримки представленна на рисунку 2.4.

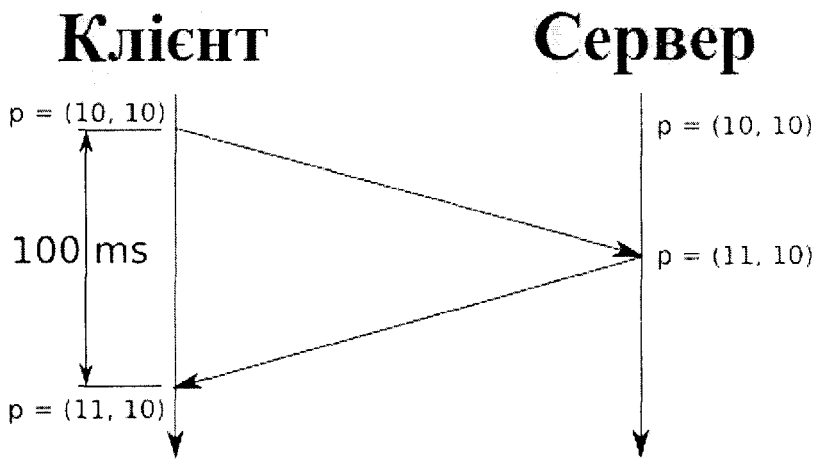

Рис. 2.4 Візуалізація затримки.

Щоб мінімізувати затримку, розробники ігор використовують кілька методів. Прогнозування на стороні клієнта дозволяє гравцям виконувати негайні дії локально, не чекаючи на підтвердження сервера, що робить ігровий процес плавнішим. Узгодження на сервері гарантує, що всі дії гравців будуть підтверджені та скориговані для підтримки синхронізації між гравцями. Крім того, такі методи, як компенсація затримки, допомагають пом'якшити вплив затримки, передбачаючи позиції інших гравців на основі їхніх минулих дій.

Пропускна здатність – це максимальна швидкість, з якою дані можуть передаватися через мережеве з'єднання. Зазвичай вона вимірюється в бітах на секунду (біт/с) і визначає пропускну здатність для передачі даних. В онлайн іграх пропускна здатність впливає на кількість інформації, яка може передаватися між гравцями і серверами, впливаючи на швидкість відгуку і візуальну якість ігрового процесу. Недостатня пропускна здатність може призвести до перевантаження даних, що спричиняє затримки і тремтіння в грі.

Для оптимізації використання пропускної здатності розробники ігор застосовують різні стратегії. Методи стиснення даних використовуються для зменшення розміру мережевих пакетів, що забезпечує більш ефективну передачу. Пріоритетність важливих ігрових даних над другорядними гарантує, що найважливішій інформації про ігровий процес буде надано пріоритет, зменшуючи вплив обмеженої пропускної здатності.

Втрата пакетів відбувається, коли пакети даних не досягають місця призначення, що призводить до пропусків або відсутності інформації в переданих даних. В онлайн іграх втрата пакетів може призвести до десинхронізації між гравцями, що призводить до непослідовності ігрового процесу. Поширеними причинами втрати пакетів є перевантаження мережі, несправність мережевого обладнання або ненадійне бездротове з'єднання.

Одним із поширених підходів є використання методів прямої корекції помилок, коли в пакети, що передаються, включається надлишкова інформація, яка дозволяє відновити втрачені дані. Механізми повторної передачі, такі як автоматичний запит на повтор , використовуються для запиту на повторну передачу втрачених пакетів. Крім того, методи інтерполяції та екстраполяції можуть бути використані для оцінки відсутніх даних на основі отриманої інформації, мінімізуючи візуальний вплив втрати пакетів.

Отже, затримка, пропускна здатність і втрата пакетів є критичними факторами, які впливають на продуктивність і загальний досвід багатокористувацьких онлайн ігор. Мінімізація затримок і втрат пакетів з одночасною оптимізацією використання пропускної здатності має першорядне значення для досягнення плавного ігрового процесу. Розробники ігор використовують різні методи про які ми поговоримо далі.

2.1.5.1 Алгоритм прогнозування на стороні клієнта

Прогнозування поведінки гравців [23] – це невід'ємна техніка, яка використовується в ігрових мережах для покращення реакції ігрового процесу та пом'якшення впливу мережевих затримок. Дозволяючи клієнтам моделювати та передбачати власні дії, цей підхід має на меті забезпечити більш плавний та захоплюючий ігровий досвід. У цьому розділі розглядається концепція прогнозування поведінки гравців, її значення в онлайн іграх і проблеми, пов'язані з її реалізацією.

В багатокористувацьких онлайн іграх затримка в мережі може спричинити затримку між дією гравця та її відповідним впливом на ігровий світ. Ця затримка може створювати помітне відставання, що негативно впливає на ігровий досвід. Клієнтське прогнозування вирішує цю проблему, дозволяючи клієнтам локально симулювати результат своїх дій до отримання підтвердження від сервера. Таким чином, гравці отримують миттєвий зворотній зв'язок, зменшуючи відчутну затримку і підвищуючи швидкість реакції гри.

Переваги клієнтського прогнозування особливо очевидні у швидкоплинних іграх, які вимагають швидкої реакції від гравців. Такі дії, як рух персонажа, стрільба або використання здібностей, можуть виконуватися локально, забезпечуючи гравцям миттєвий візуальний зворотній зв'язок. Така реакція в реальному часі покращує загальний ігровий досвід і сприяє відчуттю контролю та занурення в гру.

Однак реалізація клієнтського прогнозування створює проблеми, пов'язані з синхронізацією та підтримкою послідовного стану гри серед усіх гравців. Оскільки кожен клієнт незалежно прогнозує свої дії, можуть виникати розбіжності, коли прогнози відхиляються від авторитетного стану сервера. Ці розбіжності можуть призвести до непослідовного ігрового досвіду, коли гравці ніби телепортуються або демонструють незвичну поведінку.

Щоб вирішити ці проблеми, розробники використовують методи синхронізації, які узгоджують клієнтські прогнози з авторитетним станом сервера. Узгодження з сервером передбачає перевірку та коригування дій клієнта на основі точки зору сервера. Порівнюючи прогнозований стан клієнта з авторитетним станом сервера, можна виявити та усунути невідповідності. Цей процес гарантує, що всі гравці сприймають синхронізований стан гри, мінімізуючи візуальні артефакти, спричинені розбіжностями клієнтських прогнозів.

Окрім узгодження серверів, для згладжування візуальних невідповідностей використовуються методи інтерполяції та екстраполяції. Інтерполяція заповнює прогалини між отриманими оновленнями серверів, створюючи плавний і безперервний рух ігрових об'єктів. Екстраполяція оцінює майбутні позиції на основі поточних станів і швидкостей, забезпечуючи безперебійний візуальний досвід. Ці методи допомагають маскувати візуальні ефекти розбіжностей клієнтських прогнозів і сприяють більш згуртованому ігровому досвіду.

Важливо зазначити, що клієнтське передбачення може не підходити для всіх аспектів ігрового процесу. Дії зі значними наслідками для ігрового процесу або взаємодії з іншими гравцями часто вимагають валідації сервером, щоб підтримувати справедливість і запобігати шахрайству. Критичні дії, такі як нанесення шкоди або запуск подій, що змінюють хід гри, зазвичай покладаються на підтвердження сервера, щоб забезпечити узгодженість ігрового процесу для всіх клієнтів.

Отже, прогнозування поведінки гравців є цінною технікою в ігрових мережах, яка покращує швидкість реагування на ігрові дії, дозволяючи гравцям моделювати та прогнозувати свої власні дії. Вона вирішує проблеми, пов'язані із затримкою в мережі, пропонуючи гравцям більш безпосередній і захоплюючий ігровий досвід. Однак ретельна реалізація та синхронізація з сервером необхідні

для підтримки стабільного стану гри та запобігання несправедливим перевагам. Використовуючи такі методи, як узгодження, інтерполяція та екстраполяція серверів, розробники можуть досягти балансу між швидкістю відгуку та підтримкою справедливого і приємного багатокористувацького ігрового середовища.

2.1.5.2 Алгоритм компенсації затримки пострілу

Компенсація затримки вирішує фундаментальну проблему швидких мережевих ігор: надання клієнту досвіду без можливості довіряти клієнту.

Проблема полягає в тому, що жодна з машин у мережі не перебуває у точно однаковому стані гри. Те, що бачить один клієнт і на чому він базує свої дії, лише на 100% вірно для нього. Класичний приклад – визначення точного пострілу по віддаленому об'єкту; хоча клієнт бачить ціль прямо в прицілі, насправді вона вже перемістилася.

Якщо авторитетний сервер базує визначення влучень виключно на власному сприйнятті світу, ніхто ніколи ні в що не влучить навмисно.
Якщо ж клієнт має повноваження і може повідомити серверу, у що він влучив, система буде широко відкрита для простих експлойтів, що руйнують гру.

Компенсація затримки дозволяє серверу на мить побачити світ з точки зору кожного клієнта і вирішити, чи був він насправді в змозі зробити цей неможливий постріл. З іншого боку, це означає, що ціль може бути уражена, хоча самі гравці вважали, що їх безпечно відтягнули за стіну; однак, це буде набагато менш помітно.

На рисунку 2.5 зображено приклад компенсації затримки. Супротивник на клієнті знаходиться на позиції зображеної синіми та червоними лініями, які формують модель, але насправді гравець уже декілька десятків мілісекунд знаходиться на позиція, що відображає 3D модель, але клієнт цього не знає

через затримку, тому ми приходимо до компромісу, що клієнт має надіслати час в який було зроблено постріл та виконати всі анімації, навіть попадання, але не знімати життя в супротивника, це може зробити тільки сервер.

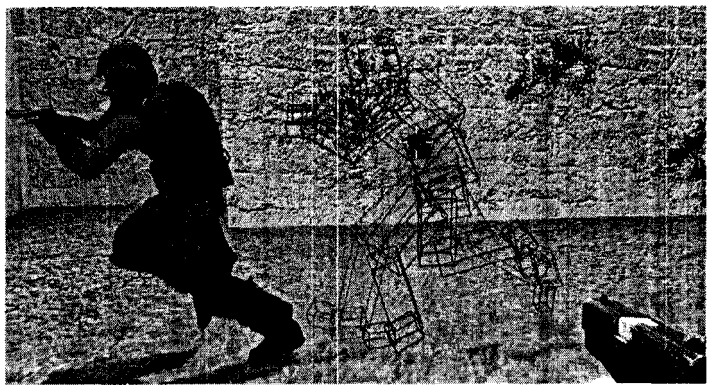

Рис.2.5 Приклад компенсації затримки пострілу в ігровому рушії Source.

2.2 Методи розробки ігрового процесу

У розробці ігор використовуються різні архітектурні патерни для структурування коду та логіки гри. Два найпоширеніші готові підходи – це «сутність-компонент-система» (ECS) [24] та «модель-представлення-контролер» (MVC) [25, с. 224]. Хоча MVC є популярним архітектурним патерном у розробці програмного забезпечення, він може бути не найкращим вибором для розробки ігор через притаманні йому обмеження. У цьому розділі ми розглянемо недоліки MVC у розробці ігор та висвітлимо переваги ECS як альтернативного підходу.

MVC, який розділяє додаток на три компоненти – модель(Model), представлення(View) та контролер(Controller), має на меті розділити проблеми та забезпечити модульність. Однак, при застосуванні до розробки ігор, MVC може спричинити кілька проблем.

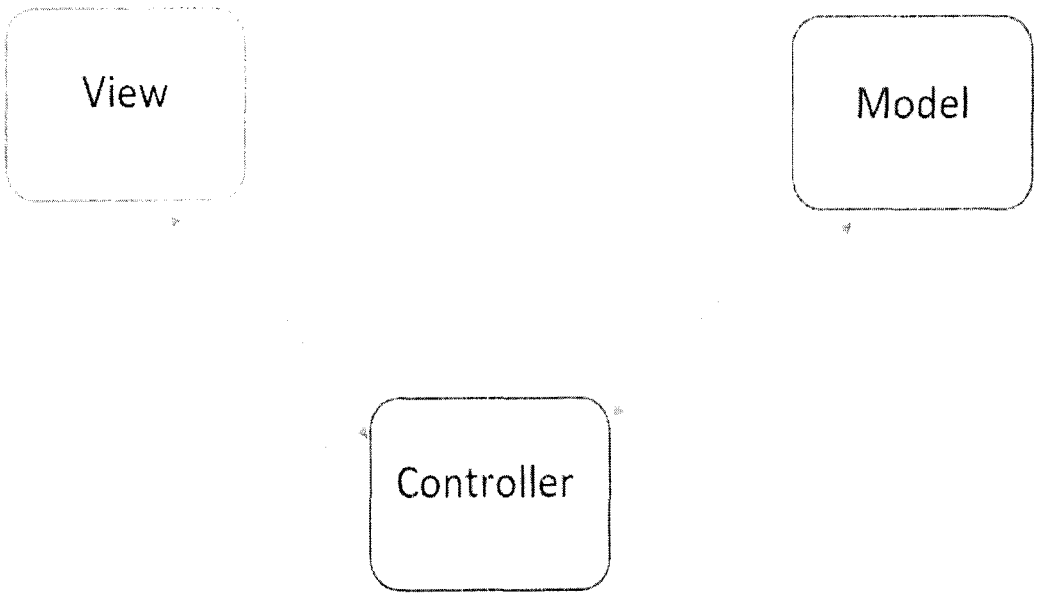

Рис. 2.6 Зображення відношення елементів патерну.

По-перше, ігрові об'єкти в MVC, як правило, тісно пов'язані з логікою презентації, що ускладнює розділення ігрової логіки та презентації. В іграх, де інтерактивність та швидкість реагування в реальному часі є критично важливими, такий зв'язок може призвести до проблем з продуктивністю та перешкоджати масштабуванню кодової бази.

По-друге, MVC не надає чіткої структури для обробки специфічної для гри поведінки та взаємодії. Ігри часто вимагають складних систем, які виходять за рамки традиційної парадигми Model-View-Controller. Реалізація специфічної для гри логіки в межах MVC може призвести до заплутаного коду і труднощів у підтримці та розширенні функціональності гри.

По-третє, MVC може призвести до збільшення кількості контролерів та роздутого шару представлення. Зі зростанням складності гри збільшується кількість контролерів і представлень, що ускладнює управління та навігацію по коду. Це може призвести до погіршення читабельності коду та його супроводу.

На противагу цьому, ECS пропонує кілька переваг для розробки ігор. Вона дотримується композиційного підходу, коли ігрові об'єкти складаються з невеликих компонентів, які можна використовувати повторно і які визначають їхню поведінку. Це дозволяє краще розділити завдання і сприяє модульності.

ECS надає кращі можливості для оптимізації продуктивності, дозволяючи системам працювати з підмножиною компонентів, які потребують обробки, що призводить до ефективного виконання. Вона також забезпечує кращу масштабованість, оскільки додавання нових функцій або модифікація існуючих може бути досягнута шляхом додавання або модифікації компонентів і систем без необхідності вносити значні зміни в існуючий код.

Крім того, ECS добре узгоджується з філософією проектування, орієнтованою на дані, де основна увага приділяється оптимізації доступу до даних і розміщення пам'яті для підвищення продуктивності. Це може призвести до кращого використання кешу, зменшення займаної пам'яті та підвищення продуктивності, що є життєво важливим для розробки ігор.

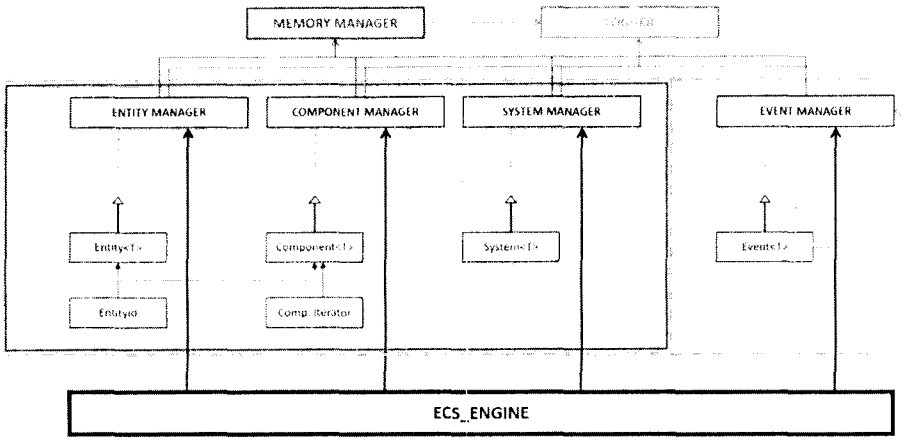

Рис. 2.7 Приклад реалізації патерну ECS.

Але ECS відходить від ООП парадигми та має проблеми у розростанні кодової бази. Коли в проекті може бути більше 1000 компонентів, при тому

деякі з них повторяють логіку один одного, через те, що ніхто не знає відповідальності всіх компонентів, та не всі компоненти є задокументовані. Також використання ECS саме в онлайн іграх, було заявлено лише розробниками гри Overwatch. Більшість ігор будують свою власну архітектуру, та використовують ECS для навантажених частин гри, а MVC для побудови користувацького інтерфейсу.

Висновки до другого розділу

У розділі схарактеризовані алгоритми та усі реалізовані методи розробки онлайн шутер гри, їх теоретичне підґрунтя.

Описано мережеву взаємодію в іграх, протокол пересилання даних, топологію клієнт–сервера, топологію Peer-to-peer, методи синхронізації клієнтів, Ззатримкау, пропускну здатність і втрату пакетів, алгоритм прогнозування на стороні клієнта, алгоритм компенсації затримки пострілу, методи розробки ігрового процесу.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ

Для досягнення реалізації всіх поставлених задач онлайн шутер-гри потрібно реалізувати набір підсистем, які будуть формувати її загальну структура та функціональність. Даний розділ детально описує структуру та архітектуру розробленої програмної системи та представляє реалізовані підсистеми з їх інтерфейсом використання.

3.1 Архітектура програмної системи

3.1.1 Структура системи

Розроблена онлайн гра складається з набору обов'язкових система та підсистем, кожна з яких виконує ряд функцій і забезпечує повне і правильне функціонування гри. Загальна структура наведена на рисунку

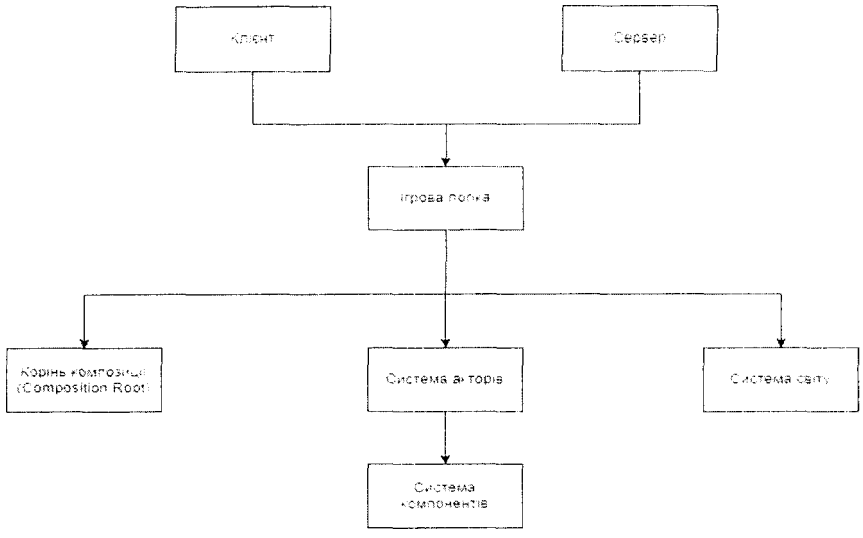

Рис. 3.1 Загальна структура онлайн шутер гри.

Головними модулями є: клієнт, сервер, ігрова логіка. Це три основні системи які в свою чергу складаються з підсистем, які виконують більш вузькоспрямовані задачі та формують загальну функціональність розробленої онлайн шутер гри.

3.1.2 Діаграма класів

Діаграма класів описує практичну реалізацію розробленої онлайн шутер гри за допомогою об'єктно-орієнтованої парадигми. Діаграма класів наведена на рисунку 3.2.

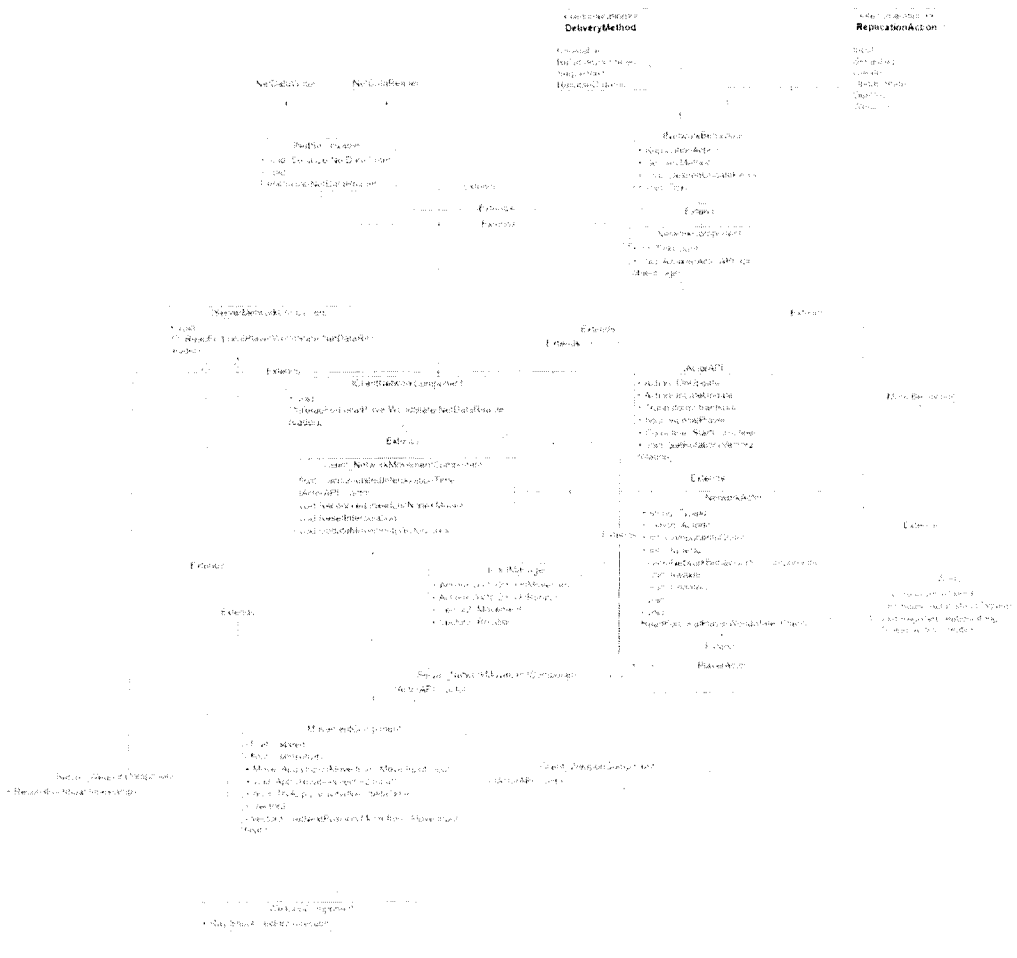

Рис. 3.2 Діаграма основних класів онлайн шутер гри.

Дана діаграма демонструє структуру основних елементів, набір класів, їх взаємодію та відношення між собою, описує ієрархічну структуру, демонструє інтерфейс, а також наводить та описує основний набір методів та атрибутів кожного з класів. Короткий опис класів та їх сфери відповідальності:

World – система, що відповідає за колізію та створення акторів на сцені.

GameClient – дає змогу комунікувати клієнту з сервером, за допомогою пакетів.

GameServer – дає змогу комунікувати серверу з клієнтами, за допомогою пакетів.

INetSerializable – інтерфейс бібліотеки LiteNetLib, що дозволяє використовувати систему серіалізації даної бібліотеки.

NetDataWrite – надає змогу запису інформації в бітову послідовність

NetDataReader – надає змогу читання інформації з бітової послідовності.

DeliveryMethod – описує тип доставки пакету.

ReplicationAction – є додатковими даними до пакету, який описує тип даних.

INetworkBehaviour – інтрефейс, який реалізує клас, що хоче ділитися даними з серевером, якщо є клієнтом, або з клієнтами якщо є сервером.

MonoBehaviour – базовий клас в Unity, для доступу до арі рушія.

NetworkActor – є базовим класом об'єкта, який має перебувати в ігровому світі. Зберігає в собі компоненти, та має можливість синхронізації данних.

IActorAPI – інтерфейс, що дозволяє компонентам отримувати доступ до подій реалізованих в рушії Unity.

PlayerActor – клас що зберігає в собі логіку гравця, та його компоненти.

NetworkComponent – базовий клас для компонентів, поділяються на серверні та клієнтні.

IServerNetworkComponent – інтерфейс, що має реалізувати серверний компонент.

IClientNetworkComponent - інтерфейс, що має реалізувати клієнтний компонент.

Client_NetworkMovementComponent – клієнтний компонент переміщення та повороту, реалізує в собі інтерполяцію між даними вводу гравця, алгоритм прогнозування на клієнті та відкат якщо прогнозування було неточним або гравець є шахраєм.

Server_NetworkMovementComponent – серверний компонент переміщення та повороту, реалізує в собі підтвердження дій гравця, та синхронізацію стану переміщення іншим клієнтам.

MovementComponent – логіка переміщення та повороту, виконується як на сервері так і на клієнті.

Client_WeaponComponent – клієнтний компонент зброї, містить реалізацію анімації, симуляцію попадань гравця, відправляє та застосовує дані з серверу.

Server_WeaponComponent – сервений компонент зброї, містить реалізацію компенсації затримки пострілу, відправляє та отримує інформацію клієнту.

WeaponComponent – містить в собі логіку обрахунку даних для променю пострілу.

3.2 Корінь композиції

Шаблон корінь композиції (compositionRoot) - це архітектурний принцип, який сприяє централізації конфігурації та композиції залежностей програми в одному місці. Корінь композиції виступає відправною точкою для вирішення залежностей і підключення різних компонентів програми.

Завдяки використанню патерну CompositionRoot [26] залежностями програми можна легко керувати та змінювати, що сприяє тестуванню, модуляризації та можливості заміни реалізацій, не впливаючи на решту кодової

бази. Корінь композиції зазвичай знаходиться у точці входу програми, наприклад, у методі Main.

Для ін'єкції залежностей ми реалізували патерн «локатор сервісів». Він забезпечує гнучкий та відокремлений підхід до вирішення залежностей у програмних додатках. Абстрагуючись від отримання залежностей, цей патерн сприяє модульності, супроводжуваності та можливості заміни реалізацій під час виконання.

Наша реалізація даного патерну знаходиться в класі GameServies, та має дані методи:

- RegisterSingletone – реєстрація простого сервісу C# який знаходитиметься в пам'яті весь життєвий цикл гри;
- RegisterMonobehSignletone – реєстрація сервісу що наслідує Unity клас Monobehaviour знаходитиметься в пам'яті весь життєвий цикл гри;
- RegisterSceneScoped – реєстація сервісу що наслідує Unity клас Monobehaviour та буде знищенний після зміни сцени в Unity;
- GetSceneScoped – отримання сервісу, що був зареєстрований через метод RegisterSceneScoped;
- GetSignletone – отримання сервірсу, що був зареєстрований через методи RegisterSignletone, RegisterMonobehSingletone;

3.3 Синхронізація клієнтів

3.3.1 Серіалізація об'єктів

Одним із суттєвих недоліків форматів JSON [27] та XML [28] є включення надлишкових символів. Ці символи слугують розміткою або розділювачами, додаючи структурованості даним. Однак вони також збільшують загальний розмір переданих даних, що призводить до нераціонального використання

мережевих ресурсів. Надлишкові символи збільшують розмір корисного навантаження, не надаючи жодної суттєвої користі, що негативно впливає на загальну продуктивність мережевого зв'язку.

JSON і XML спочатку були розроблені для серіалізації та обміну даними, а не спеціально для мережевої комунікації. Як наслідок, їм бракує певних оптимізацій, які мають вирішальне значення для ефективної передачі даних мережею. Наприклад, обидва формати не мають власної підтримки бінарного кодування даних, що призводить до необхідності додаткових етапів кодування/декодування.

```
<input>
<timestamp>15.43</timestamp>
<axis>15.15,42.12</axis>
<rotation>15.15,42.12</rotation>
</input>
```

Рис. 3.3 Клас Move. Input у форматі XML.

```
{
   "input": {
      "timestamp": 15.43,
      "axis": "15.15,42.12",
      "rotation": "15.15,42.12"
   }
}
```

Рис.3.4 Клас Move. Input у форматі JSON.

Серіалізовані дані в форматі XML мають 103 символи, у форматі JSON 101 і це ігровим додаткам не підходить, так як даний пакет може пересилатися 30 раз на секунду і це тільки один такий пакет, з часом таких пакетів буде сотні. Тому ми будемо пересилати данні у «сирому» форматі:

```
1
2  15.43|15.15,42.12|15.15,42.12
```

Рис. 3.5 Дані у «сирому» форматі, для можливості читання було додано символи «|» та «,» які не будуть серіалізовані.

Класи бібліотеки LiteNetLib реалізують упаковку даних у байти, нам залишається реалізувати інфтерфейсINetSerializable, записати дані у методі Serialize та десереалізувати дані у методі Deserialize, при цьому послідовність є важливою.

```
public void Serialize(NetDataWriter writer)
{
    writer.Put(Timestamp);
    writer.Put(Axis);
    writer.Put(Rotation);
}

Frequently called  2+ 15 usages  MykolaBulyha
public void Deserialize(NetDataReader reader)
{
    Timestamp = reader.GetFloat();
    Axis = reader.GetVector2();
    Rotation = reader.GetVector2();
}
```

Рис. 3.6 Серіалізація даних за допомогою бібліотеки LiteNetLib.

Також у оптимізації ми можемо піти ще далі, і використати ентропійне кодування, яке займається стисненням даних на основі того наскільки вони несподівані. Тобто не завжди гравець використовує клавіши «ВПЕРЕД» та «НАЗАД» і можемо використати бітову маску, щоб передати тільки ті дані що змінилися. Кодову реалізацію на даного методу, для серіалізації паке Move.Input продемонстровано на рисунку.

На відміну від використання автоматичних серіалізаторів, що серіалізують у формати JSON чи XML, розглянутий вище серіалізатор є найбільш оптимізованим, та використовується у ігрових додатках AAA класу [29]. Єдиним його мінусом є те що потрібно писати багато однотипного коду, але це ніщо, якщо ми говоримо про таку величезну оптимізацію.

```
public Entropy EntropyValue;

Frequently called | 1 usage | new *
public void Serialize(NetDataWriter writer)
{
    writer.Put((byte) EntropyValue);
    if (EntropyValue.HasFlag(Entropy.Timestamp))
    { writer.Put(Timestamp); }
    if (EntropyValue.HasFlag(Entropy.AxisX))
    { writer.Put(Axis.x); }
    if (EntropyValue.HasFlag(Entropy.AxisY))
    { writer.Put(Axis.y); }
    if (EntropyValue.HasFlag(Entropy.RotationX))
    { writer.Put(Rotation.x); }
    if (EntropyValue.HasFlag(Entropy.RotationY))
    { writer.Put(Rotation.y); }
}

Frequently called | 1 usage | new *
public void Deserialize(NetDataReader reader)
{
    var entropyValue = (Entropy)reader.GetByte();
    if (entropyValue.HasFlag(Entropy.Timestamp))
    { Timestamp = reader.GetFloat(); }
    if (entropyValue.HasFlag(Entropy.AxisX))
    { Axis.x = reader.GetFloat(); }
    if (entropyValue.HasFlag(Entropy.AxisY))
    { Axis.y = reader.GetFloat(); }
    if (entropyValue.HasFlag(Entropy.RotationX))
    { Rotation.x = reader.GetFloat(); }
    if (entropyValue.HasFlag(Entropy.RotationY))
    { Rotation.y = reader.GetFloat(); }
}
```

Рис. 3.7 Серіалізація пакету вводу за допомогою бібліотеки LiteNetLib та ентропійного кодування.

3.3.2 Реплікація об'єктів

Для реплікації об'єктів було створено інтерфейс INetworkBehaviour, у якому наявні поля та методи:

- DesiredUpdateTime – час, за яким надсилається пакет

- ReplicationAction – метадані.
- IsDirtyForSending – маркер, що потрібно надсилати пакет даного об'єкту.
- DeliveryMethod – тип доставки.
- Tick() – метод який виконується перед надсиланням пакету
- Serialize(NetDataReader) – метод що серіалізує пакет.

Кожен тік ігорового рушія перевіряємо у циклі for кожен об'єкт що релізує інтерфейс INetworkBehaviour чи настав час надсилання пакету, якщо час настав виконуємо метод Tick та ставимо маркер IsDirtyForSending в true, виконуємо цикл for до кінця. Коли цикл було виконано, виконуємо ще один цикл for у якому, якщо елемент має маркер IsDirtyForSending в true – знаходимо ActorId даного класу, TypeId та його ComponentIndex(для NetworkActor класів ComponentIndex завжди дорівнює -1, для компонентів це є індексом в списку компонентів в NetworkActor) записуємо дані в ReplicationHeaderрисуноку 3.8 серіалізуємо дані об'єкту, надсилаємо їх за допомогою методу Send в класі GameClient – якщо ми є клієнтом, та класі GameServer – якщо ми є сервером та ставимо маркер IsDirtyForSending в false.

Рис. 3.8 Реалізація класу ReplicationHeader.

Припустимо, що вище описаний алгоритм виконувався на клієнті. Коли сервер отримає даний пакет, за допомогою ReplicationAction він дізнається як їх трактувати, за допомгою ActorId та ComponentIndex хто є суб'єктом даного пакету. В більшості випадків клієнт надсилає ReplicationAction.Input що означає, що клієнт надсилає дані зчитані з клавіатури чи комп'ютерної миші.

3.3.3 Реалізація алгоритму прогнозування на стороні клієнта

Однією з головних проблем онлайн шутерів є плавність переміщення гравців. В цьому розділі описана реалізація прогнозування клієнтом, яке має вирішити цю проблему.

Наша реалізація полягає в тому, що зберігаємо пакет вводу кожних DesiredUpdateTime часу, у компоненті переміщення цей час становить 0.033 секунди, виконуємо даний пакет, отримуємо стан гравця через 0.033 секунди часу за допомогою розробленої формули для розрахунку переміщення.

Спочатку розглянемо як ми рахуємо зміщення, для цього спочатку треба дізнатися поточне обертання гравця, для даного обчислення нам потрібно дізнатися так звані «forwardvector» – вектор який репрезентує погляд вперед та «rightvector» - який репрезентує погляд в правий бік.

Інформацію про повороти в комп'ютерній графіці зберігають в кватерніонах, так відбувається через проблему блокування обертання (gimballock), яка притаманна кутам Ейлера. Проблема виникає через те, що кути Ейлера використовують фіксовані осі обертання, а коли дві осі стають паралельними, обертання навколо цих осей стають зв'язаними. Це ускладнює анімування складних рухів і досягнення плавних переходів. Формула конвертації з кутів Ейлера в кватерніони 3.1.

$$Q(\alpha, x, y, z) = cos\left(\frac{\alpha}{2}\right) + sin\left(\frac{\alpha}{2}\right) \times (x \times i + y \times j + z * k) \quad (3.1)$$

де Q – обертання в кватерніонах;

α – значення кутів ейлера;

x, y, z – скаляри вектора осі навколо якої відбувалося обертання;

I, j, k – базиси системи координат.

Далі використаємо формулу 3.2 для пошуку зміщення

$$V(x,y) = (Q_x \times F \times x + Q_z \times R \times z) \times (T|1 - T_2) \times s \quad (3.2)$$

де F – напрямок осі x;

R – напрямок осі y ;

Q_x – повороти у кватерніонах навколо осі x;

Q_z – повороти у кватерніонах навколо осі z;

x – дані вводу гравця для переміщення по осі x;

z – дані вводу гравця для переміщення по осі z;

T_1 – штамп часу попереднього пакету;

T_2 – штамп часу поточного пакету;

s – швидкість переміщення.

$$L(B,t,x,y) = V(x,y) \times t + B \quad (3.3)$$

Де L – нова позиція гравця;

V – зміщення відносно попередньої позиції;

t – поточне значення інтерполяції ;

B – попередня позиція.

Коли була за формулою 3.3 була порахована наступна позиція гравця, ми зберігаємо пакет з порахованими даними. Та виконуємо інтерполяцію з поточної позиції до порахованої в кожному кадрі за допомогою формули 3.4.

$$v = a + (b - a) \times \frac{s-T}{D} \quad (3.4)$$

де a – початкова позиція;

b – порахована позиція;

s – поточний час на клієнті;

T – час обновлення даних вводу;

D – час, з яким компонент відправляє пакети серверу, тоді й відбувається обов'язковий збір даних вводу;

v – позиція відображення гравця.

Але, якщо ми буде збирати данні вводу гравця, тільки в певні періоди часу, то це буди приводити до незрозумілих для гравця подій, а саме він натиснув кнопку, яка відповідає за переміщення, але його аватар не змінив свою позицію, бо кнопка була натиснута не в час збору даних воду. Ми вирішили цю проблему наступним чином, програма завжди виконує ввід гравця, але при цьому корегує минулі пакети.

Спочатку оглянемо як це працює на клієнті коли клієнт натискає кнопку для переміщення, викликається функція ResetInterpolation в якій ми зі збережених пакетів переміщення дістаємо перший не інтерпольований пакет викликається функція CorrectMoveRelativeToInterpolation в яку ми передаємо раніше отриманий не інтерпольований пакет, s та D. Далі обчислюється t за формулою 3.5.

$$t = 1 - \frac{s}{D} \tag{3.5}$$

та рахуємо позицію, до якої ми встигли провести інтерполяцію за формулою 3.3, додатково інвертувавши скаляри вводу, так як дані скаляри формують вектор напрямку.

В свій час на сервері ми розраховуємо t просто віднімаючи час минулого пакету від поточного та обчислюємо позицію за тією самою формулою, що й на клієнті.

Результат можемо побачити на рисунку 3.9.

Рис. 3.9 Скріншот переміщення гравця.

3.3.4 Реалізація алгоритму компенсації затримки пострілу

Одним з таких важливих рішень у іграх-шутерах є визначення влучень, тобто процес визначення того, чи влучив постріл у гравця. Він виконується щоразу, коли сервер отримує подію пострілу. Алгоритм для визначення влучень без компенсації затримки показано в псевдокоді 3.1. Тут ми припускаємо, що гра використовує зброю з можливістю визначення влучень, тобто що кулі рухаються з нескінченною швидкістю вздовж прямої лінії і, таким чином, будь-яке влучання є миттєвим. Для ясності, назви функцій, які виконуються на стороні сервера мають префікс «server», а функції на стороні клієнта – «client».

Спочатку розглянемо реалізацію виявлення влучень без компенсації затримки псевдокод 3.1, проблемою даного алгоритму є те, що кожен гравець грає в гру яка рендериться в минулому через затримку мережі. І якщо дозволимо гравцю вирішувати чи попав він чи ні, це дозволить використання читів [30], що ламає ігровий процес. Якщо ми дозволимо серверу вирішувати, чи попав гравець, то серверу потрібно знати, в якому стані знаходився світ під час пострілу гравця.

Псевдокод 3.1 – Виявлення влучень без компенсації затримки

```
1: Процедура ServerReceiveFire(shooter)
2:         Початок ← місцезнаходження камери с грільця;
3:         Кінець ← старт + напрямок прицілу стрільця × відстань трасування;
4:         Виконати трасування зіткнень від початку до кінця;
5:         Якщо виявлено влучання і влучений актор є гравцем, то
6:         Ціль ← актор в якого влучили;
7:         Якщо стрілець може пошкодити ціль, то
8:         Відправити результат стрільцю та мішені;
```

Через те, що ми реалізували передбачення переміщення клієнта, сервер зберігає історію всіх переміщень гравців. Ці дані ми використаємо для алгоритму псевдокод 3.2, реалізація якого знаходиться в компоненті Server_WeaponComponent.

Псевдокод 3.2 – Виявлення влучень з компенсацією затримки

```
1: процедура ServerReceiveFire(стрілець)
2:     для всіх гравців
3:     відкотимо стани гравця до часу коли гравець зробив постріл
за формулою 3.6
4:     початок ← місцезнаходження камери стрільця
5:     кінець ← початок + напрямок прицілу стрільця × відстань трасування
6:     виконати трасування зіткнення від початку до кінця
7:     якщо виявлено зіткнення і актор, який потрапив у нього, є гравцем, то
8:         ціль ← гравець, в якого влучили
9:         якщо стрілець може пошкодити ціль, то
10:            надіслати результат стрільцю та мішені
11:    для всіх гравців виконати
12:        відновити стани гравців
```

$$R = S + (E - S) \times t + G \quad (3.6)$$

де S – початкова позиція в пакеті;

E – кінцева позиція в пакеті;

G – RTT;

t – значення інтерполяції яке було в даного гравця в цей час.

E та S нам уже відомі залишилося знайти t, знайдемо його за формулою3.7

$$t = \frac{p - a}{b - a} \quad (3.7)$$

де p – штамп час пострілу;

a – час коли гравець був в позиції S;

b – час коли гравець був в позиції E.

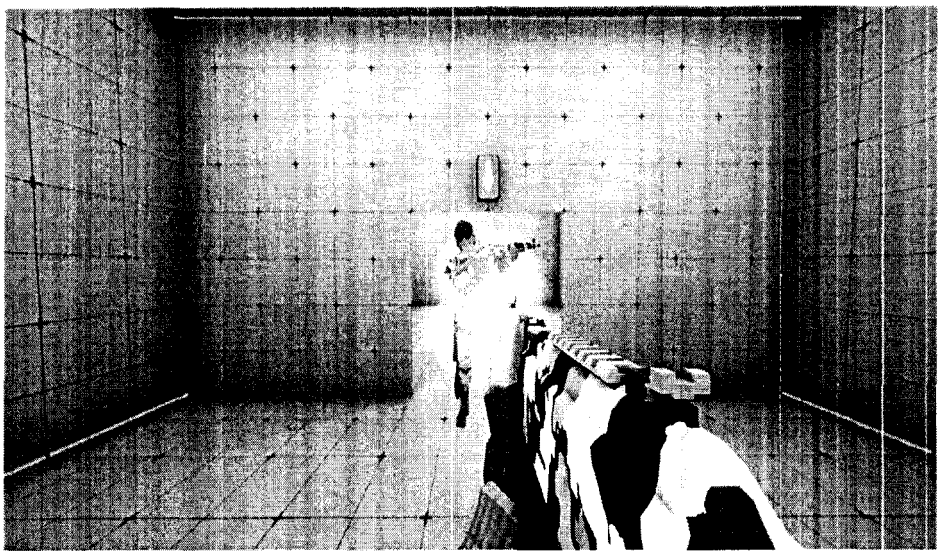

Рис. 3.10 Скріншот пострілу по іншому гравцю.

Висновки до третього розділу

У розділі міститься детальний опис програмної реалізації, описана архітектура програмної системи, структура системи з відповідними діаграмами, а також міститься опис всіх складових систем, підсистем, класів, корінь композиції, схарактеризовано синхронізація клієнтів, серіалізація об'єктів,

реплікація об'єктів, реалізація алгоритму прогнозування на стороні клієнта їх взаємодію, реалізація алгоритму компенсації затримки пострілу.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА ОБЧИСЛЮВАЛЬНІ ЕКСПЕРИМЕНТИ

Для плавного ігрового процесу розробленої гри опишемо технічні вимоги та процес встановлення розробленої програмної системи.

4.1 Встановлення програмного продукту

Для гри в розроблений ігровий додаток потрібно мати певний набір технічних характеристик обчислювальної машини .

Мінімальні технічні характеристики обладнання:

• Операційна система – Windows 10 64-розрядна;

• Процесор – Чотирьохядерний процесор Intel або AMD, 2,5 ГГц або швидше;

• Оперативна пам'ять – 8 ГБ;

• Постійна пам'ять – HDD 100 ГБ;

• Відеокарта – NVIDIA GeForce 470 GTX або AMD Radeon 6870 HD або краще. Рекомендовані технічні характеристики обладнання:

• Операційна система – Windows 10 64-розрядна;

• Оперативна пам'ять – 8 ГБ;

• Постійна пам'ять – SSD 100 ГБ;

• Відеокарта – NVIDIA RTX 2080 або AMD Radeon 5700+ XT або краще;

• Процесор – Дванадцяти ядерний процесор Intel або AMD, 3,4 ГГц або швидше.

Для запуску гри потрібно розархівувати файл ShooterGame.zip. Всі залежності програмного продукту містяться в архіві разом з виконуваним файлом, попередня інсталяція додаткових пакетів не потрібна. Спочатку потрібно запустити Server.exe, для запуску серверу, потім запустити Client.exe.

Коли Client.exe буде запущено з'явиться екран зображений на рисунку 4.1 де потрібно ввести адрес та порт серверу. Так як ми запустили сервер локально введемо такі данні у поле «Адрес» - localhost, у поле «Порт» - 5000.

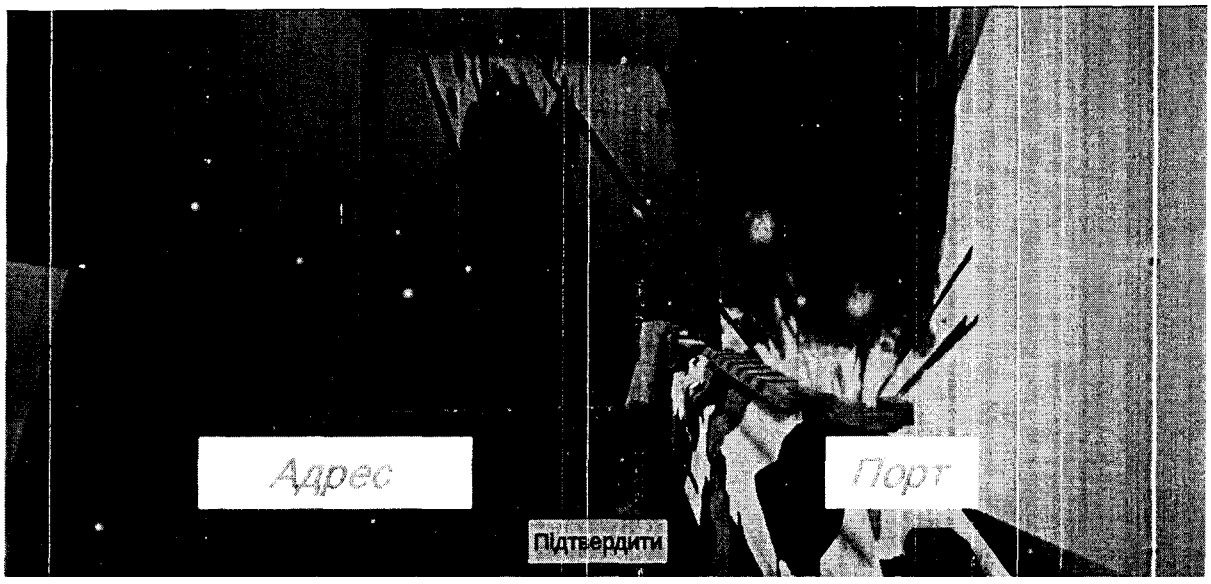

Рис. 4.1 Екран що зустрічає гравця при вході в гру.

Після того, як гравець підключиться до сесії, він з'явиться в кімнаті зображеній на рисунку 4.2.

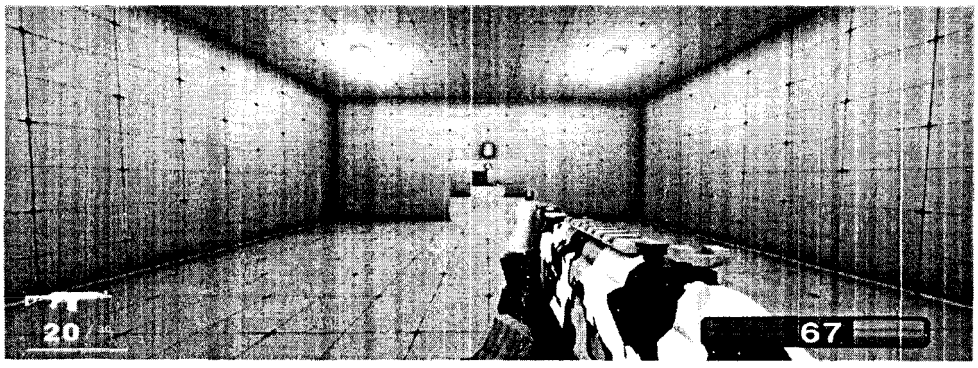

Рис. 4.2 Початкова кімната в якій з'являється гравець.

У лівому нижньому куті екрану знаходяться інформація про набої гравця зображені на рисунку 4.3.

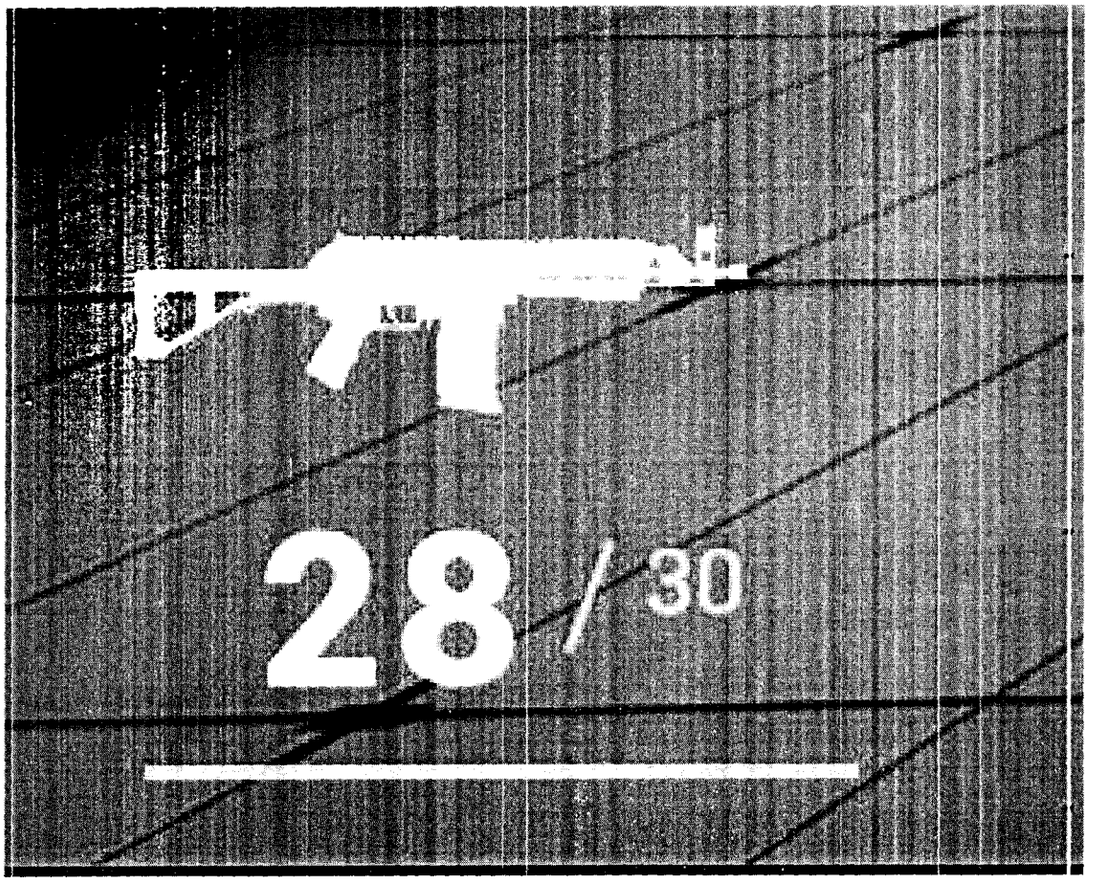

Рис. 4.3 Інформація про набої гравця.

У правому нижньому куті екрану знаходяться здоров'я гравця зображені на рисунку 4.3. Максимальним є значення 100, мінімальним 0. При значенні 0 гравець програє та зникає з карти для інших гравців, а у гравця, що програв з'являється екран зображений на рисунку 4.4. Якщо гравець хоче продовжити йому потрібно натиснути на зелену кнопку, якщо він хоче покинути гру – червону.

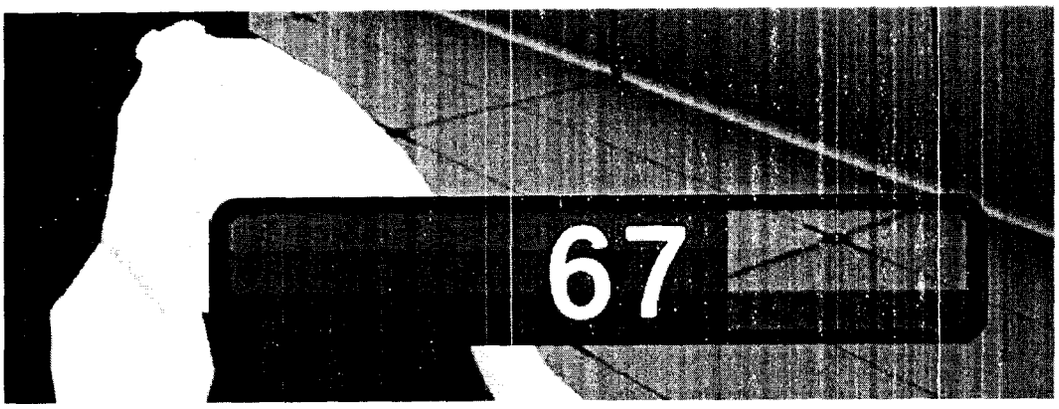

Рис. 4.4 Здоров'я гравця.

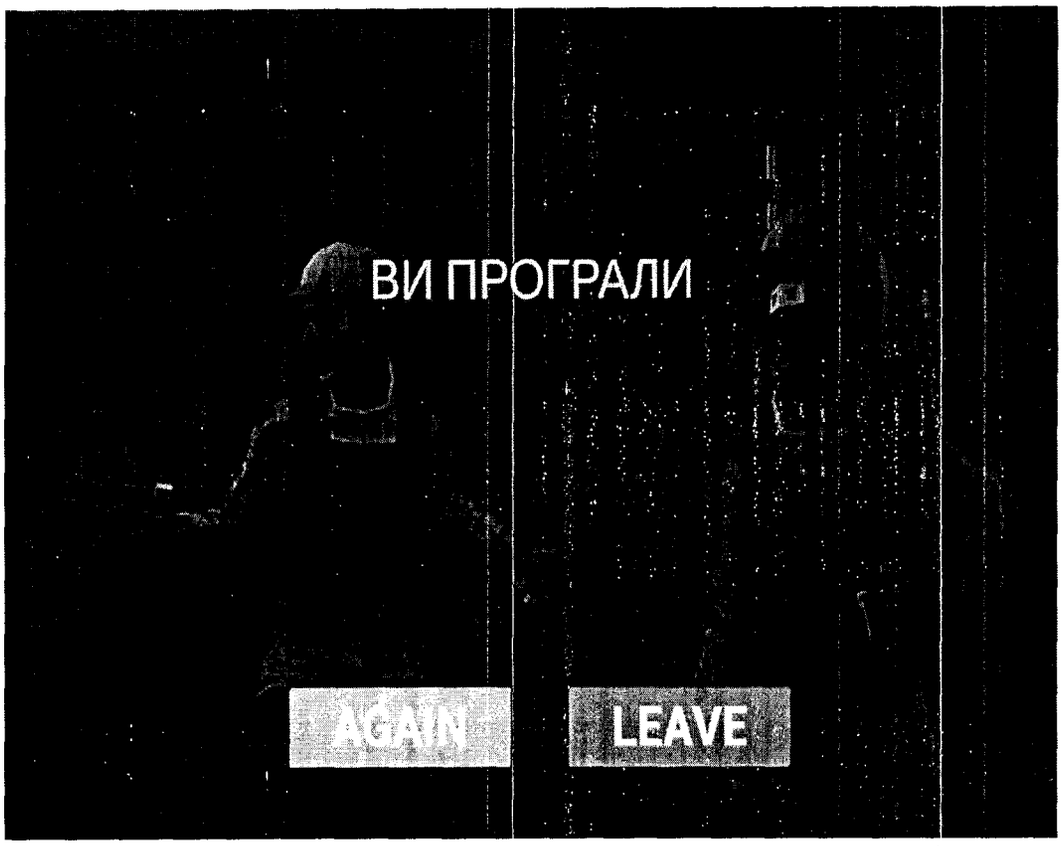

Рис. 4.5 Екран програшу.

Щоб не програти гравцю, потрібно збирати амуніцію розкидану на рівні, яка з'являється час від часу. На рисунку 4.6 зображено 2 типи амуніції:

- Куб зеленого кольору відновлює здоров'я;
- Куб фіолетового кольору відновлю набої.

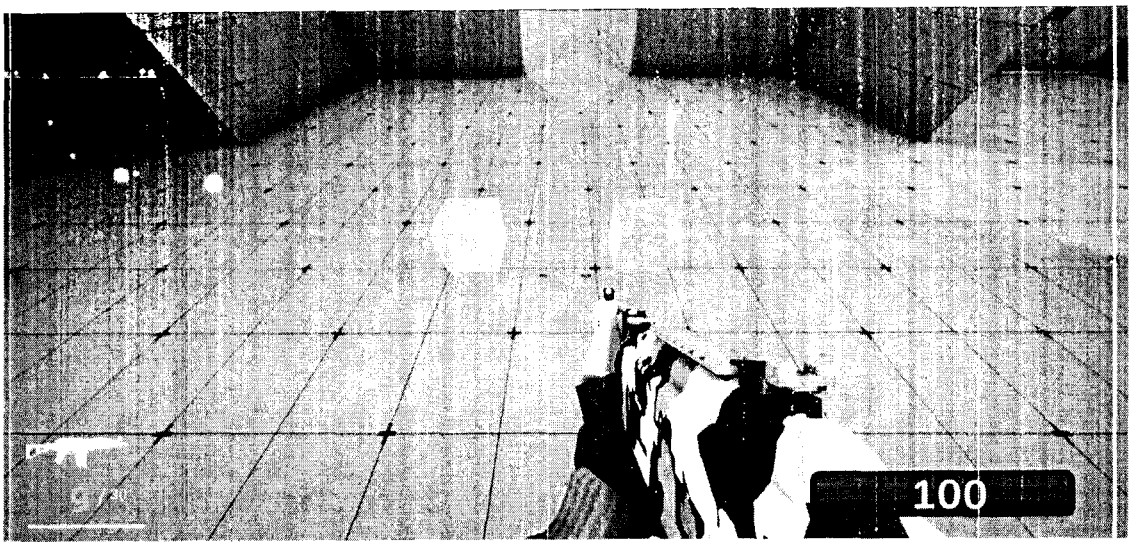

Рис. 4.6 – Типи амуніції, які може зібрати гравець.

Для переміщення використовувати кнопки на клавіатурі:

- Ц(W) – вперед;
- І(S) – назад;
- Ф(A) – вліво;
- В(D) – вправо;

Стрільба:

- Ліва кнопка миші(LMB) – власне постріл;
- К(R) – перезарядка;

4.2 Тестування розробленої гри

Під час тестування розробленої онлайн гри-шутера є кілька ключових характеристик, які можна порівняти, щоб переконатися, що гра відповідає бажаним стандартам продуктивності, ігрового процесу та користувацького досвіду. Ці характеристики включають в себе: пропускну здатність та кількість кадрів в секунду(FPS).

Спочатку пропускну здатність, яку ми буде вимірювати в мб/год та головним чинником в цьому тестуванні є кількість гравців в сесії.

Таблиця 4.1

Пропускна здатність гри відносно кількості гравців

Середня пропускна здатність	3 гравця	5 гравців	12 гравців
Вхідна здатність серверу	30.3 мб/год	52.8 мб/год	114.4 мб/год
Вихідна здатність серверу	110.5 мб/год	170 мб/год	254.4 мб/год
Вхідна здатність клієнту	70.7 мб/год	90.1 мб/год	119.8 мб/год
Вихідна зданість клієнту	15.4 мб/год	15.4 мб/год	15.4 мб/год

Для порівняння візьмемо один з найпопулярніших онлайн шутерів всіх часів CounterStrike: GlobalOffensive, де сумарне використання трафіку, $n + m$, де n вхідна пропускна здатність клієнту, а m вихідна, дорівнює 250 мб/год, у нашому ігровому додатку ця цифра становить 90.1 + 15.4 = 105.5 мб/година (беремо дані для 5 гравців через те, що в порівнюваній грі в сесії знаходиться саме така кількість клієнтів).

Тепер розглянемо кількість кадрів на секунду, для тестів використовувалась відеокарта GeForce GTX 1650, процесор IntelCore i7 8700k, Kingstone FURY DDR4 16GB.

Таблиця 4.2

Кількість кадрів на секунду відносно кількості гравців

Кількість гравців	Кі ькість кадрів в секунду
3	170
5	162
12	150

Дані показники є задовільними для плавної гри, тому що людське око бачить 24 кадрів в секунду, але чим більша кількість кадрів, тим плавнішою є ігрова картинка.

Висновки до четвертого розділу

У розділі схарактеризовано технічні вимоги та процес встановлення розробленої програмної системи для плавного ігрового процесу розробленої гри.

Схарактеризовано системні вимоги, продемонстровано процес запуску гри та можливі комбінації клавіш для гри.

Описано пропускну здатність гри відносно кількості гравців.

Проаналізовано тестування розробленої гри, представлені таблиці з результатами тестів.

ВИСНОВКИ

У кваліфікаційній роботі було поставлено завдання розробити онлайн шутер-гри на основі ігрового рушія Unity та дослідити мережеві аспекти цієї реалізації. Основна мета полягала у створенні онлайн шутер гри, одночасно вирішуючи проблеми мережевої затримки, втрати пакетів та синхронізації.

Для досягнення цих цілей було визначено виконане наступне:

- Проведено аналіз існуючого програмного забезпечення та методів для реалізації багатокористувацьких онлайн ігор та мережевої взаємодії в Unity. В цьому аналізі охоплені різні мережеві архітектури, протоколи та бібліотеки.
- Визначені та досліджені основні підсистеми, необхідні для реалізації онлайн шутер-гри, такі як рух гравця, стрільба.
- Обрані відповідні інструменти, фреймворки та методи для реалізації ігрових та мережевих компонентів.
- Створена структура програмної системи, включаючи організацію скриптів, активів та ресурсів в межах проекту Unity. Була розроблена модульна та масштабована архітектура для полегшення майбутнього розвитку та оновлень.
- Розроблені алгоритми та методи для реалізації мережевих функцій, включаючи клієнт-серверний зв'язок, авторитетний сервер та синхронізовані дії гравців.
- Проведено тестування розробленої гри.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Adams E., Rollings A. Fundamentals of game design (game design and development series). Prentice Hall, 2006. 600 с.
2. Mohmmed M. Developing games using unity: programming C# in unity engine. Independently Published, 2019. 149 с.
3. What are indie games? . gamemaker.io. URL: https://gamemaker.io/en/blog/what-are-indie-games (дата звернення: 31.05.2023).
4. Multiplayer and networking. URL: https://docs.unity3d.com/Manual/UNet.html (дата звернення: 01.06.2023).
5. Mirror. URL: https://mirror-networking.gitbook.io/docs/ (дата звернення: 20.05.2023).
6. Photon. URL: https://doc.photonengine.com/fusion/current/getting-started/fusion-intro (дата звернення: 17.05.2023).
7. Platform overview. URL: https://www.smartfoxserver.com/overview (дата звернення: 02.06.2023).
8. LiteNetLib. URL: https://revenantx.github.io/LiteNetLib/index.html (дата звернення: 18.05.2023).
9. Cross-platform mobile development. URL: https://www.techtarget.com/searchmobilecomputing/definition/cross-platform-mobile-development (дата звернення: 27.05.2023).
10. Virtual and augmented reality (VR/AR) / ред.: R. Doerner та ін. Cham : Springer International Publishing, 2022. 439 с.
11. Troelsen A., Japikse P. Pro C# 10 with .NET 6: foundational principles and practices in programming. 11-те вид. Apress, 2022. 1705 с.

12. Васильєв О. М. Програмування мовою Java. Тернопіль : Навчальна книга – Богдан, 2019. 696 с.

13. Meyers S. Effective modern C++. O'ReillyMedia, Ir corporated, 2014. 334 с.

14. Playmaker. URL: https://hutonggames.fogbugz.com/default.asp?W133 (дата звернення: 31.05.2023).

15. Heads-up display. URL: https://wow-how.com/articles/how-to-create-a-hud-that-matches-your-game-design-step-by-steb-guide (дата звернення: 03.06.2023).

16. Rider for unity. URL: https://www.jetbrains.com/lp/dotnet-unity/ (дата звернення: 04.06.2023).

17. Фрімен Е., Робсон Е. Head first. Патерни проєктування. Київ : Фабула, 2020. 672 с.

18. Ludin S. Learning HTTP/2: A Practical Guide for Beginners. O'Reilly Media, Inc, 2017. 136 с.

19. Mandl P. TCP und UDP Internals. Wiesbaden : Springer Fachmedien Wiesbaden, 2018. URL: https://doi.org/10.1007/978-3-658-20149-4 (дата звернення: 04.06.2023).

20. Dulaney E. Tcp/ip. Indianapolis, Ind : NewRiders, 1998. 359 с.

21. Посвістак В. С., Демківська Т. І. Клієнт-серверна архітектура та її використання при розробці програмного забезпечення. Інформаційні технології в науці, виробництві та підприємництві : зб. наукових праць молодих вчених, аспірантів, магістрів кафедри комп'ютерних наук та технологій / за заг. наук. ред.

22. P2P.URL: https://geode.apache.org/docs/guide/114/topologies_and_comm/topology_concepts/topology_types.html (дата звернення: 07.05.2023).

23. Gambetta G. Fast-Paced multiplayer (part II): client-side prediction and server reconciliation. URL: https://www.gabrielgambetta.com/client-side-prediction-server-reconciliation.html (дата звернення: 16.05.2023).

24. Entity component system FAQ. URL: https://github.com/SanderMertens/ecs-faq (дата звернення: 24.05.2023).

25. Asp.net mvc / T. Barcz та ін. O'Reilly Media, Incorporated, 2009. 492 с.

26. Composition root pattern: how to write modular software. URL: https://dev.to/nuculabs_dev/composition-root-pattern-how-to-write-modular-software-21p0 (дата звернення: 26.05.2023).

27. Json. URL: https://www.json.org/json-en.html (дата звернення: 1.05.2023).

28. XML. URL: http://w3schools.com/xml/xml_whatis.asp (дата звернення: 3.05.2023).

29. URL: https://www.trustedreviews.com/explainer/what-are-aaa-games-4259739 (дата звернення: 01.06.2023).

30. Cheating in video games: The A to Z. URL: https://blog.irdeto.com/video-gaming/cheating-in-games-everything-you-always-wanted-to-know-about-it/ (дата звернення: 27.05.2023).