ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ, ПРАВА ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Пояснювальна записка

до кваліфікаційної роботи на здобуття ступеня вищої освіти магістр

на тему: **«Моделювання інформаційного обігу у розподілених системах (на прикладі комп'ютерних ігор)»**

Виконав: здобувач вищої освіти
за освітньо-професійною програмою
Інформаційні управляючі системи та
технології спеціальності 126 Інформаційні
системи та технології ступеня вищої
освіти магістр
групи 126ІСТ_мд_21
Кваша А. В.
Керівник: Флегантов Л. О.
Рецензент: Петраш Р. В.

Полтава – 2023 року

ВСТУП

Актуальність роботи. Сучасний розвиток інформаційних технологій та розподілених систем породжує виклик для багатьох галузей і, в особливості, для індустрії комп'ютерних ігор. Моделювання інформаційного обігу у розподілених системах, зосереджене на контексті комп'ютерних ігор є темою високої актуальності та стратегічної важливості. Цей вибір обумовлений необхідністю вдосконалення технічних аспектів популярної на даний час галузі цифрових розваг. Дослідження цієї теми сприятиме вирішенню проблем, які виникають у зв'язку зі зростанням об'ємів даних, які створюють комп'ютерні ігри в умовах недостатньої швидкості їх передачі через географічне розподілення гравців та постійним прагненням до оптимізації ігрового процесу, що робить обрану тему дослідження надзвичайно актуальною.

Сучасний світ перебуває в стані технологічної трансформації, де комп'ютерні ігри виступають не лише як розважальний продукт, але й як справжня культурна та економічна сила, що впливає на розвиток людства та визначає його подальший розвиток та цифрову трансформацію. Вплив ігор на суспільство стає важливим предметом обговорень та численних досліджень. З одного боку, ігрова індустрія стає потужним економічним гравцем на ринку розваг, привертаючи мільйони користувачів та генеруючи вражаючі глобальні фінансові обороти. З іншого боку, розподілені системи, на яких базується сучасні мережі, зазнають тисків у зв'язку із збільшенням навантаження та з огляду на потреби у великому обсязі даних. Такий стан речей вимагає глибшого розуміння і оптимізації процесів інформаційного обігу у розподілених системах.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконана у відповідності до науково-дослідної ініціативної теми «Організаційно-методологічні аспекти впровадження інформаційно-комунікаційних систем і технологій в управлінні діяльністю сучасних організацій та підприємств за умов переходу до цифрової економіки» ДРН 0117U003099.

Мета роботи – аналіз моделі інформаційного обігу у розподілених системах на прикладі комп'ютерних ігор.

Завдання даної роботи є:

– аналіз теоретичних основ інформаційного обігу в комп'ютерних іграх;

– емпіричний аналіз інформаційного обігу на прикладі проєкту Unity Technologies;

– практична реалізація та програмування ігрового проєкту.

Об'єктом даного дослідження є інформаційний обіг у розподілених системах, зокрема в контексті комп'ютерних ігор.

Предметом дослідження є аналіз архітектур одноранговых мереж та клієнт-серверних систем, транспортних протоколів, а також практична реалізація ігрових механік та багатокористувацьких компонентів в рамках ігрових проектів.

Методи досліджень:

– аналіз літературних джерел: проведення докладного огляду вже наявних наукових публікацій стосовно інформаційного обігу у розподілених системах та комп'ютерних ігор;

– емпіричні спостереження: збір та аналіз конкретних даних з реальних ігрових середовищ з метою оцінки практичних аспектів інформаційного обігу;

– моделювання процесів: використання комп'ютерного моделювання для аналізу взаємодії розподілених систем та інформаційного обігу у гральних програмах;

– статистичний аналіз: використання статистичних математичних методів для обробки та аналізу отриманих даних.

Інформаційна база дослідження: технічні стандарти та вимоги до розподілених систем; теоретичні матеріали та наукові публікації про моделювання інформаційного обігу; технічна документація ігрових движків і платформ для мультиплеєрних ігор; аналітичні джерела про існуючі алгоритми обробки інформації в іграх, сучасні тенденції в ігрових системах; статистика та аналітика мережевих технологій; технічна документація інструментів для

моделювання мережевих взаємодій Unity та Unreal Engine, дані з офіційних сайтів ігрових платформ.

Наукова новизна полягає у розробці нових підходів до оптимізації інформаційного обігу у розподілених системах, зокрема, в контексті комп'ютерних ігор. Це включає в себе вивчення взаємодії розподілених систем і гравців, врахування соціокультурних аспектів гри та створення нових моделей оптимізації.

Практична значущість. Результати дослідження можуть бути використані для розробників ігор, та спрямовані на поліпшення якості гри та задоволення гравців. Аналіз соціокультурного впливу інформаційного обігу може також мати широкий застосунок у галузі дизайну ігор та медіа, що набуває все більшого поширення та актуальності.

Апробація результатів роботи шляхом оприлюднення доповідей на наукових міжнародних та студентських конференціях.

Публікаціїю. За результатами проведеного дослідження опубліковано тези доповідей: «Переваги програмного засобу Unreal Engine для створення відеогор» представлені на щорічній студентській науковій конференції (м. Полтава, 10 листопада 2022 року); «Інформаційний обіг у комп'ютерних іграх: технологія, геймдизайн і майбутнє» на І міжнародній науково-практичній конференції «Стратегічний менеджмент агропродовольчої сфери в умовах глобалізації економіки: безпека, іновації, лідерство» (м. Полтава, 28 вересня 2023 року); «Нетворкінг та його архітектури» на ХХ щорічному міждисциплінарному семінару (м. Полтава, 29 листопада 2023 року)

Структура та обсяг кваліфікаційної роботи. Пояснювальна записка кваліфікаційної роботи складається зі вступу, 3 розділів, висновків, списку використаних джерел (53 найменувань), 1 додатку. Кваліфікаційна робота містить 6 таблиць, 36 рисунків, викладена на 66 сторінках.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ТА МЕТОДОЛОГІЧНІ АСПЕКТИ МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОГО ОБІГУ В КОМП'ЮТЕРНИХ ІГРАХ

1.1 Основи інформаційного обігу у ігрових проєктах

Інформаційний обіг у ігрових проєктах визначається не лише обсягом інформації, яка передається, але й швидкістю обміну даними між гравцями та серверами. Це особливо важливо у великих багатокористувацьких іграх, де кожен момент може вплинути на геймплей та враження користувачів. Правильно спроектована ігрова мережа повинна забезпечувати стабільність підключення, низький пінг та високу пропускну здатність, щоб забезпечити плавність гри та максимально зменшити затримки.

Крім того, забезпечення безпеки інформаційного обігу у ігрових мережах є однією з ключових задач. У зв'язку зі збільшенням кількості кібератак та спроб обману в ігрових середовищах, розробники повинні вдосконалювати системи шифрування та аутентифікації, щоб захистити конфіденційні дані гравців.

Архітектура мережі відноситься до структурного та логічного розташування мережі. У ньому описано, як підключаються мережеві пристрої та правила, які регулюють передачу даних між ними. Планування архітектури мережі є життєво важливим, оскільки воно або покращує, або перешкоджає продуктивності всієї системи. Вибір неправильного середовища передачі або обладнання для певного очікуваного навантаження на сервер, наприклад, може спричинити уповільнення роботи мережі.

Індивідуальні особливості ігрових проєктів створюють різноманітні виклики, які визначають архітектуру ігрової мережі. Немає універсального рішення або «золотої архітектури», оскільки кожна гра може вимагати унікального підходу в залежності від специфіки геймплею, кількості гравців, графічного та звукового контенту, а також технічних обмежень.

У деяких випадках важливо мати ефективний механізм синхронізації даних між гравцями, щоб уникнути розходжень у геймплейі. У інших випадках пріоритет може бути наданий оптимізації пропускної здатності для масштабних багатокористувацьких баталій або подоланню географічних обмежень при використанні хмарних рішень.

Нові технології, такі як розподілені системи та блокчейн, також використовуються для покращення безпеки та управління ігровою економікою. Гнучкість у виборі технологій та їх інтеграція відображають постійний розвиток індустрії, яка намагається забезпечити якість і задоволення гравців у світі технологій, які швидко змінюються відповідно до вимог ринку.

У розробці багатокористувацьких ігор переважно існує дві можливі мережеві архітектури: однорангова (peer-to-peer – скорочено: p2p, що дослівно означає: рівний до рівного) та клієнт-сервер архітектура (рис. 1.1). У одноранговій архітектурі обмін даними здійснюється між будь-якою парою підключених гравців [1]. У той же час у клієнт-серверній архітектурі обмін даними відбувається окремо лише між гравцями та виділеним ігровим сервером [2].

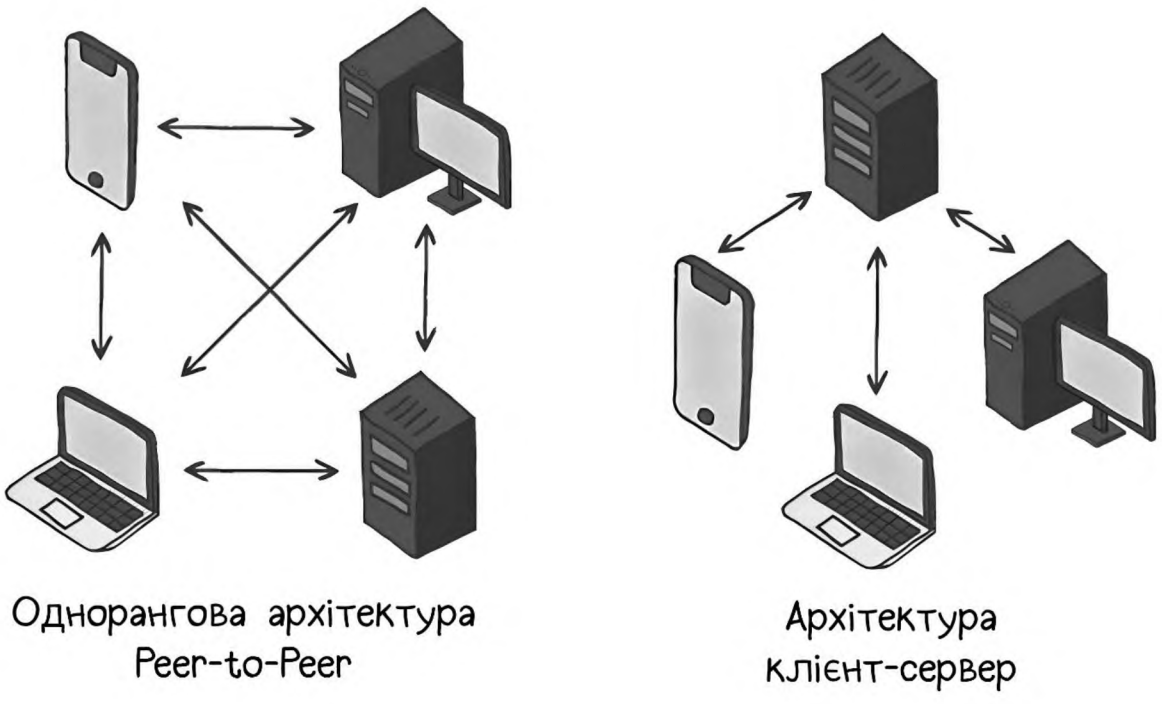

Рисунок 1.1 – Типи мережевих архітектур

Хоча зовні ці архітектури досить відрізняються, не заважаючи на це більшість багатокористувацьких ігор мають подібні структури, що складаються з обох типів ключових об'єктів: хостів та клієнтів. Хост – це активний суб'єкт, адміністратор, який створює сеанс гри, є його господарем та керує ним. Як правило, саме господарі відповідають за налаштування гри, вибір її режиму та запрошення інших гравців для приєднання у неї. Хост також має можливість видалення гравців з гри та змінювати налаштування її конфігурації. По суті, господар є головним адміністратором ігрової сесії [3]. З іншого боку, клієнти – це гравці та гості, які приєднуються до ігрового сеансу, створеного хостом. Вони, на відміну від хосту, не мають рівня контролю та адміністрації над грою, як господар, але все одно можуть грати та активно взаємодіяти з іншими гравцями під час динамічного процесу гри [3].

1.2 Архітектура однорангової мережі

Докладно розглянемо архітектуру однорангової мережі та її структуру. Отже, в одноранговій архітектурі обмін даними здійснюється між будь-якою парою підключених гравців. Одноранговий зв'язок – це структура мережі, яка найчастіше використовується саме в мережах спільного використання [1]. На початку використання цифрового зв'язку p2p використовувався такими веб-сайтами та програмним забезпеченням, як Napster, Limewire, MSN і WinMX, серед кількох інших. Сьогодні цей метод зберігається за різними клієнтами у Torrent та з іншими програмами для обміну файлами і базовим програмним забезпеченням, такими як : Fasttrack, Gnunet, Google Talk і Neonet. До того ж, саме ця мережева структура також використовується, щоб зробити світ ігор більш соціально інтерактивним та цікавим для гравців [2].

Ігри, які використовують цю структуру – відповідно називаються одноранговими іграми, інакше відомі як p2p-ігри. Це один з архітектурних методів, який саме і дозволяє грати гравцями онлайн [2]. Простіше кажучи,

однорангова мережа (p2p) відноситься до мережі комп'ютерів (та/або пристроїв), що спільно використовують робочі навантаження. Ці комп'ютери, або однорангові комп'ютери, надсилають повідомлення один одному в процесі гри безпосередньо. Повідомлення – це саме те, що контролює поточний стан гри. Це принципово відрізняється від мережної архітектури клієнт-сервер, де одна центральна машина як адміністратор контролює всю гру [1].

Через таку принципову архітектуру відпадає потреба в центральному сервері, що робить налаштування гри легшим і дешевшим. Відсутність головного серверу також запобігає проблемам, коли помилка сервера може торкнутися усіх без виключення клієнтів. Дана структура є оптимальним рішенням саме для проєкту з кількістю клієнтів до 10 учасників. Вона у цілому характеризується легшим шляхом обміну даними між гравцями. У протилежному випадку гравці також можуть зіткнутися зі специфічними проблемами із «господарем», про що було зазначено раніше. До того ж, якщо гравець, призначений господарем, залишає гру, або його зв'язок поганий, це може негативно вплинути на якість самого ігрового процесу інших гравців, а інколи навіть припинити гру взагалі. Інша проблема полягає в тому, що «хост» однорангового ігрового сервера має певну перевагу над іншими геймерами: вони мають майже нульовий пінг, що в конкурентній грі є недопустимим. Також існує велика проблема саме із недоброчесними гравцями, оскільки однорангові ігри вразливі до саботажу з боку гравців, які можуть навмисно вводити затримку у гру, або змінювати дані, коли немає центральної влади у формі нейтрального сервера. Цю проблему можна вирішити шляхом змушення одного з клієнтів обрати себе «господарем» гри. Ведучий ініціює гру, запрошує клієнтів приєднатися, а потім координує спілкування між ними [1].

Структура однорангової ігрової мережі зазвичай найкраще працює саме в іграх, де гравці знаходяться поруч один з одним. Це означає, що зв'язок у грі буде тим кращим, чим ближче гравці між собою знаходяться. Якщо ж гра у однорангові ігри відбувається на глобальному рівні, швидше за все з'єднання буде набагато повільнішим, ніж потрібно [3].

Деякі приклади однорангових ігор включають: Animal Crossing: New horizons; Super Smash Bros. Ultimate; GTA Online; Deamon Souls та ін.

У моделюванні однорангових ігор найчастіше зустрічаються два типи мережевих структур. До них відносяться: архітектура з гравцем у вигляді хосту та розподілена архітектура. Розглянемо їхню роботу докладніше (рис. 1.2). У випадку архітектури з гравцем у вигляді хосту один гравець діє саме як «сервер», відомий як хост. При цьому інші гравці можуть підключитися до цього хосту і хост передаватиме відповідні дані кожному гравцеві, який з'єднаний з ним. Інша форма p2p-ігор на прикладі розподіленої архітектури полягає в тому, що всі підключені елементи потрібні для підтримки єдиної мережі. При цьому в ній після встановлення з'єднання, мережа комп'ютерів надсилає повідомлення одне одному, щоб поширювати та балансувати обсяг роботи, необхідний для оптимальної підтримки певної гри [2].

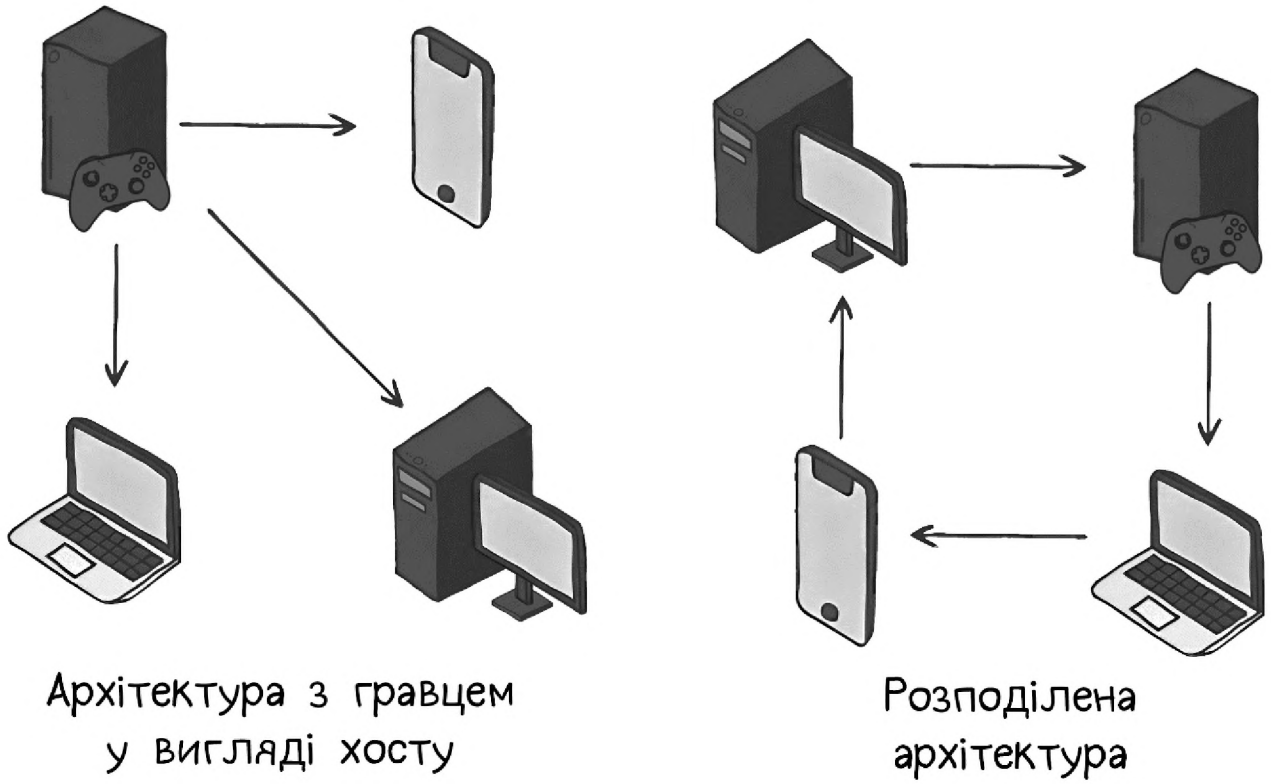

Рисунок 1.2 – Однорангові мережеві структури

Отже, розглянувши обидва варіанти однорангових мережевих структур, помітною стає ключова деталь роботи з'єднання p2p – це саме відсутність окремо виділеного елементу: сервера. В обох цих розглянутих випадках гра, або сеанс, покладаються на з'єднання у вигляді хосту, або гравців, замість фактичного сервера.

1.3 Архітектура клієнт-сервер

Окрім p2p однорангових мережевих структур існує також так званий «виділений сервер», де, як випливає із його назви, надається офшорний центральний сервер, до якого всі гравці можуть підключатися. При цьому виділений ігровий сервер, або спеціальний ігровий сервер, не є безпосереднім гравцем, а також не є частиною гри. Він, сервер, відноситься до елементу програми, завантаженої на окрему спеціальну машину, що безпосередньо запускає серверну частину гри для кількох гравців, іншими словами, програмне забезпечення. Також це не слід плутати з серверним обладнанням, що є фізичною базовою машиною, тобто апаратним забезпеченням. Відповідальність сервера полягає в моделюванні світу гри та надсиланні цих даних кожному підключеному гравцеві для безперешкодної взаємодії [3]. Таким чином архітектура клієнт-серверу ідеально підходить саме для високих навантажень, а виділені ігрові сервери призначені для одночасної роботи великої кількості гравців. Усе це досягається за допомогою процесу, який називається балансуванням навантаження. Балансування навантаження дозволяє розподіляти дані гри та вимоги до обробки між кількома серверами. У результаті такої роботи жоден сервер у кластері надмірно не перевантажується. Також цікавою особливістю є й те, що виділеному ігровому серверу не потрібен графічний процесор. Усе тому, що ігровому серверу не потрібно відтворювати графіку, оскільки зазвичай цим займаються пристрої користувачів. Натомість ігровому серверу потрібен потужний процесор, оперативна пам'ять і стабільне підключення до Інтернету,

щоб обробляти одночасне підключення кількох гравців, логіку самої гри та передавати дані між гравцями. Таким чином, інвестиції в графічний процесор для розміщення виділеного ігрового сервера не потрібні. Вони лише додадуть непотрібних витрат на хостинг. Усі ці нюанси призводять до того, що запуск виділеного сервера для ігор своїми власними силами може бути досить складним та комплексним. Оскільки для цього потрібне висококласне обладнання та надійне підключення до Інтернету. Крім того, постійне технічне обслуговування та оновлення даної системи можуть з часом виявитися дорогими. Це може бути справжньою незручністю для тих, хто хоче створити сервер для менших ігрових спільнот, оскільки вони можуть не мати ресурсів для фінансування виділеного сервера на триваліший термін. У той же час це не проблема за наявності достатньої кількості грошових активів, якщо існує можливість орендувати виділений сервер у такого постачальника серверів, як RedSwitches. При цьому професійні центри обробки даних забезпечують більшу пропускну здатність, кращу безпеку та цілодобовий моніторинг роботи [3].

Саме виділені ігрові сервери стоять за деякими дуже популярними багатокористувацькими іграми, такими як: Minecraft, 7 days to die, Counter Strike: Global Offensive, Rust.

Отже, хоча однорангова архітектура і використовується в багатьох іграх, саме архітектура клієнт-сервер є золотим стандартом, оскільки її легше реалізувати для більшої кількості клієнтів, вона вимагає меншої пропускної здатності та має більше шляхів запобіганню шахрайству. Причому при використанні такої архітектури майже завжди мають на увазі моделі саме із серверами-авторитетами, тобто моделі де сервер завжди правий. Наприклад, якщо клієнт думає, що він знаходиться в одних координатах, але сервер повідомляє йому, що вони інші, тоді клієнт повинен оновити свою позицію до позиції сервера, а не навпаки. Використання авторитетного сервера дозволяє розробникам легше виявляти та вже на ранньому етапі запобігти шахрайству у грі [3].

1.4 Транспортні протоколи

Вибір протоколу для передачі даних між вузлами ігри, будь то сервер чи клієнт – ще одна складова нетворкінгу. Протокол визначає те, як транспортувати дані між клієнтами та сервером і в якому саме форматі. Для цього доступні два основні інтернет-протоколи: TCP (Transmission Control Protocol) і UDP (User Datagram Protocol). Але при цьому також є можливість створити власний транспортний протокол на основі одного з попередніх, або скористатися бібліотекою, яка їх використовує [4].

Як і TCP, так і UDP протоколи базуються на IP. Інтернет-протокол (IP) – це протокол або набір правил для маршрутизації та адресації пакетів даних, щоб вони могли переміщатися по мережах і досягати правильного пункту призначення. Дані, що передаються в Інтернеті, поділяються на менші частини, які називаються пакетами. Інформація про IP додається до кожного пакету, і ця інформація допомагає маршрутизаторам надсилати пакети в потрібне місце. Кожному пристрою або домену, який підключається до Інтернету, призначається IP-адреса, і оскільки пакети спрямовуються на приєднану до них IP-адресу, дані надходять туди, куди вони потрібні.

Протоколи у мережі відіграють важливу роль у забезпеченні ефективного та стандартизованого обміну інформацією між різними пристроями та програмами. Вони визначають, як пристрої повинні взаємодіяти, щоб гарантувати правильність передачі даних, їх рівномірність та безпеку. Він включає в себе стандартизовані способи форматування даних, процедури обробки помилок, установлення та завершення з'єднань, а також інші аспекти, необхідні для ефективного обміну інформацією.

Розглянемо процес надсилання листа, щоб зрозуміти, навіщо потрібні протоколи. На конверті адреси написані в такому порядку: ім'я, вулиця, місто та поштовий індекс. Якщо в поштову скриньку опустити конверт із поштовим індексом, а потім адресою і так далі, пошта не доставить його. Існує узгоджений протокол написання адрес, щоб поштова система працювала. Таким же чином усі

пакети IP-даних мають надавати певну інформацію в певному порядку, а всі IP-адреси мають відповідати стандартизованому формату.

IP-адреса – це унікальний ідентифікатор, призначений пристрою або домену, який підключається до Інтернету. Кожна IP-адреса – це набір символів, наприклад «192.168.1.1». За допомогою DNS-перетворювачів, які перетворюють зрозумілі людині доменні імена в IP-адреси, користувачі можуть отримувати доступ до, наприклад, веб-сайтів, не запам'ятовуючи цю складну серію символів. Кожен IP-пакет міститиме як IP-адресу пристрою або домену, що надсилає пакет, так і IP-адресу одержувача, подібно до того, як адреса призначення та адреса для повернення включаються в пошту (рис. 1.3).

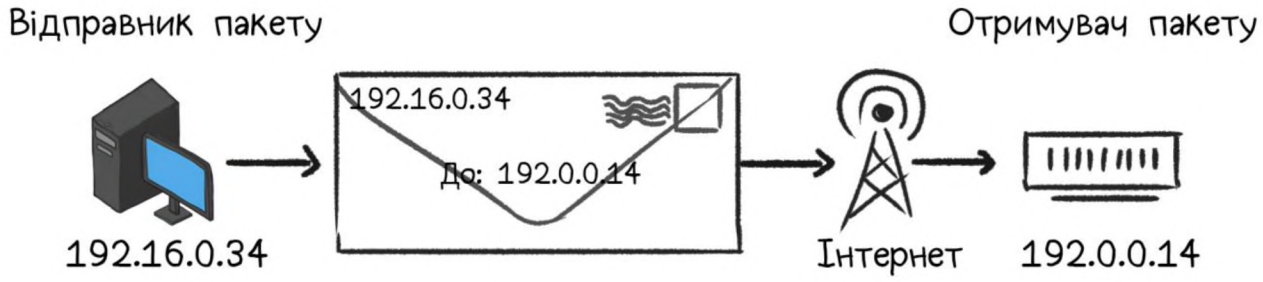

Рисунок 1.3 – Процес надсилання пакетів

IP-пакети створюються шляхом додавання IP-заголовка до кожного пакета даних перед його відправленням. IP-заголовок – це лише набір бітів (одиниць і нулів), у якому записується кілька фрагментів інформації про пакет, включаючи IP-адресу надсилання та отримання. IP-заголовки також повідомляють:

– довжину заголовка;

– довжину пакету;

– час життя (ttl) або кількість мережевих переходів, які може зробити пакет, перш ніж його буде відкинуто;

– який транспортний протокол використовується (tcp, udp тощо).

В цілому у заголовках IPv4 є 14 полів для інформації, хоча одне з них необов'язкове.

Інтернет складається з взаємопов'язаних великих мереж, кожна з яких відповідає за певні блоки IP-адрес, ці великі мережі відомі як автономні системи (AS). Різноманітні протоколи маршрутизації, включаючи BGP, допомагають маршрутизувати пакети між AS на основі їхніх IP-адрес призначення (рис. 1.4). Маршрутизатори мають таблиці маршрутизації, які вказують, через які AS мають проходити пакети, щоб досягти бажаного пункту призначення якомога швидше. Пакети переміщуються від AS до AS, поки не досягнуть тієї, яка бере на себе відповідальність за цільову IP-адресу. Потім ця AS внутрішньо направляє пакети до пункту призначення.

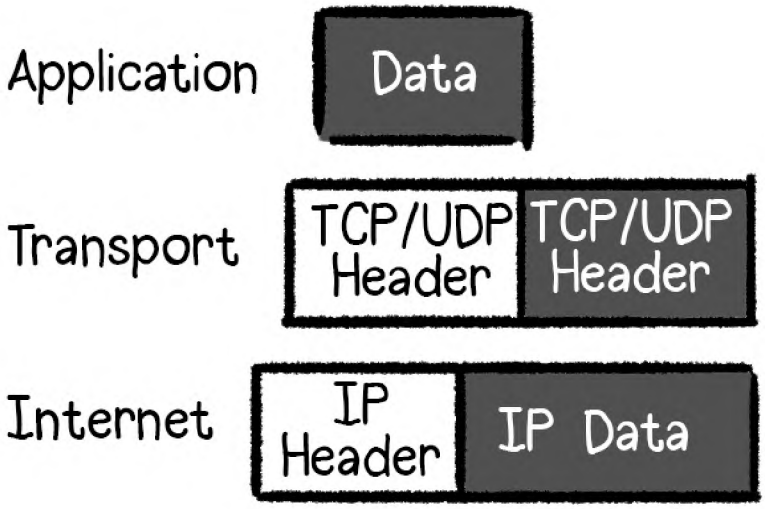

Рисунок 1.4 – заголовки пакетів на різних рівнях моделі OSI

Пакети можуть йти різними маршрутами до того самого місця, якщо це необхідно, так само як група людей, що їдуть до узгодженого пункту призначення, може вибрати різні дороги, щоб дістатися туди. Як тільки пакети прибувають до місця призначення, вони обробляються по-різному залежно від того, який транспортний протокол використовується в поєднанні з IP.

IP дозволяє передавати пакет від джерела до пункту призначення, але він не дає гарантії, що певний надісланий пакет врешті-решт прибуде до пункту призначення. Також немала вірогідність, що він надійде лише один раз, і що дані не будуть пошкоджені під час передачі, так і те, що послідовність пакетів

інформації буде надходити у правильному порядку. Крім того, цей пакет може містити лише обмежений розмір даних, який визначається MTU (Maximum Transmission Unit), тобто максимальним розміром блоку корисного навантаження пакету [3].

UDP є простішим протоколом порівняно з TCP, він працює як тонкий шар над IP. Він надає базовий рівень послуг, таких як надсилання та отримання пакетів даних, але він не забезпечує надійності, впорядкування та перевірку на помилки, що робить його менш актуальним для додатків, де важлива ця надійність. Навпаки, TCP надає повноцінний ряд функцій, котрі роблять його більш складним порівняно з UDP. Він гарантує, що дані будуть надійно доставлені в порядку їх відправлення, виявляє та виправляє помилки передачі і управляє потоком даних між двома хостами. Але всі ці функції мають негативну рису у вигляді додаткової затримки, особливо в мережах з великою пропускною здатністю. Таким чином, як ми бачимо, TCP дуже зручний і використовується в багатьох інших протоколах, таких як HTTP , FTP або SMTP.

Щоб зрозуміти, чому ці функції можуть спричиняти затримку, спочатку потрібно зрозуміти, як працює TCP. Розглянемо докладніше. Коли хост-джерело надсилає пакет на хост-адресат, він очікує отримання флаг підтвердження (ACK). Якщо він не отримує жодного підтвердження (оскільки його пакет було втрачено, або саме підтвердження було втрачено, чи з будь-якої іншої причини), через деякий час він знову надсилає пакет. Крім того, TCP гарантує, що пакети, які були надіслані, дійдуть та будуть опрацьовані в правильному порядку. Через це, поки втрачений пакет не було отримано, інші пакети не можуть бути оброблені навіть за умови, що вони вже були отримані хостом призначення. Усе це разом при поганому інтернет-з'єднанні буде гарантувати велику затримку. А як вже було зазначено вище, затримка є дуже важливим аспектом в багатокористувацьких відеоіграх, особливо в іграх, де гравець напряму керує діями персонажу, в таких як FPS проєкти [4]. Слід зазначити, що при використанні TCP протоколу використання алгоритму Nagle повинно бути вимкнено, оскільки він буферизує пакети перед їх надсиланням, отже також збільшує затримку (рис. 1.5).

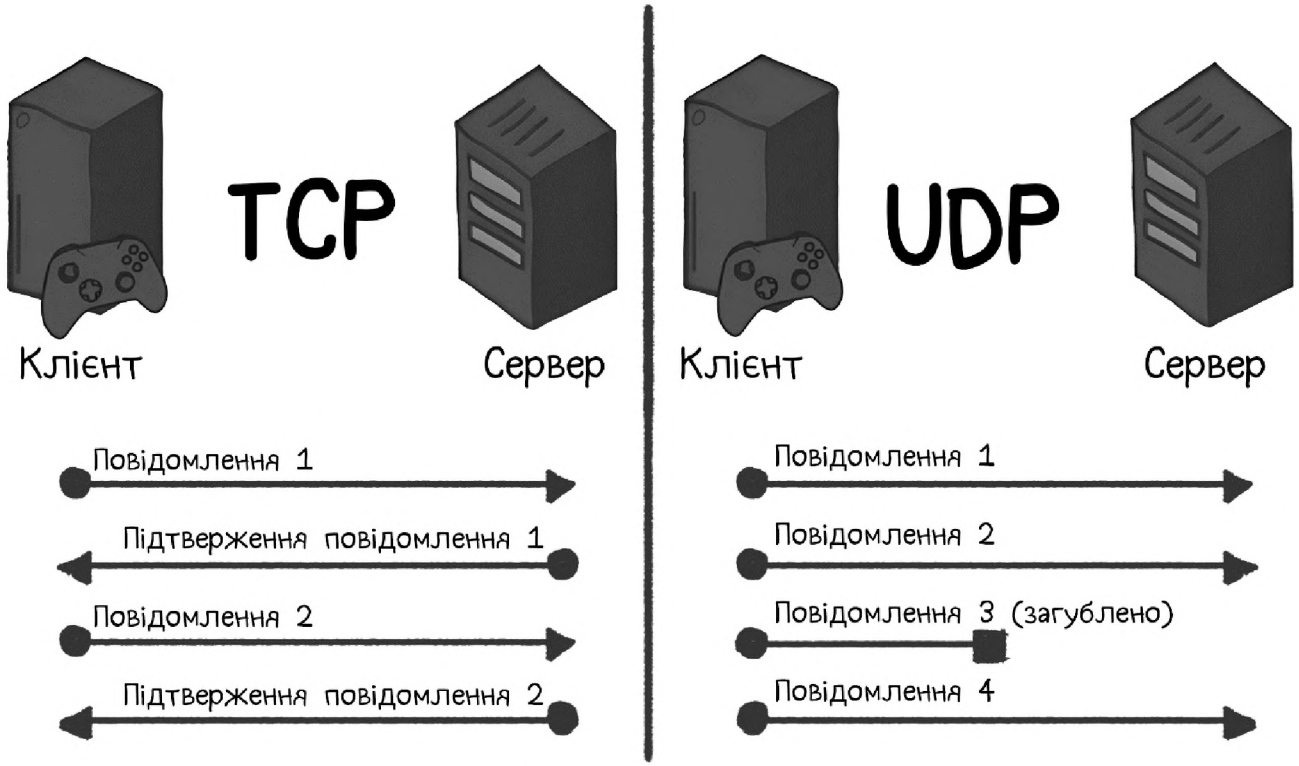

Рисунок 1.5 – Відмінність TCP та UDP

Розглянемо кілька сценаріїв, як змінений протокол, що використовує UDP, може бути ефективнішим, ніж TCP. Наприклад, він може позначати деякі пакети як надійні, а деякі інші як ненадійні. Таким чином, йому буде байдуже, прийде ненадійний пакет до місця призначення чи ні. Або він може керувати кількома потоками даних, щоб втрачений пакет в одному потоці не сповільнював інші потоки. Також може існувати потік для введення даних користувача і ще один для повідомлень чату. Таким чином, якщо повідомлення чату, що є нетерміновими даними, буде втрачено, це не сповільнить тригерне введення, яке є терміновим. Ось чому саме через цю особливість багато студій будують мережеву складову проєктів саме через протокол UDP [4]. При цьому навіть якщо TCP майже завжди є неоптимальним для мережі відеоігор, тим не менш він може бути непоганим рішенням та добре працювати у деяких жанрах. До того ж, може заощадити час на розробку. Наприклад, така затримка може зовсім не бути проблемою для покрокової гри або для гри, у яку можна грати лише в локальних мережах, де

затримка та швидкість втрати пакетів набагато менші, ніж в Інтернеті. Багато успішних ігор, таких як World of Warcraft, Minecraft або Terraria, використовують TCP. Однак більшість FPS використовують спеціальний протокол на основі UDP.

Якщо проєкту потрібне щось більш ефективне, ніж TCP, але немає можливості турбуватися про впровадження спеціального протоколу та тим самим занурюватися в багато проблем, пов'язаних з UDP, у відкритому доступі є багато готових рішень у вигляді мережевих бібліотек, а саме:

- yojimbo від Глена Фідлера;
- RakNet, який більше не підтримується, але форк SLikeNet;
- ENet – це бібліотека, створена для багатокористувацької гри FPS Cube;
- GameNetworkingSockets від Valve.

Серед цих бібліотек ENet є фаворитом, оскільки вона не лише проста у використанні, але й надійна. Крім того має чітку документацію та підручник для початку.

Висновки до розділу 1

У процесі розробки багатокористувацьких ігор вибір мережевої архітектури визначається рядом важливих факторів. Однорангова (p2p) архітектура стає оптимальною, коли немає ресурсів на використання виділеного серверу, коли система має працювати при відмові окремих її частин, і важлива можливість децентралізованого взаємодії між гравцями. Клієнт-серверна архітектура є більш затратною, але більш ефективна для завдань, що вимагають централізованого керування та ефективного використання ресурсів. Усе це може бути особливо корисним у великих та масштабних ігрових проєктах.

Незважаючи на широке використання однорангової архітектури у багатьох іграх, клієнт-серверна архітектура залишається стандартом через її легкість реалізації для великої кількості клієнтів, меншу потребу у пропускній здатності та більших можливостях запобігання шахрайству. Саме з використанням

авторитетного сервера розробники можуть ефективно виявляти та запобігати шахрайству ще на ранніх етапах розробки.

Щодо компонентів ігрової мережі, важливість правильного вибору транспортного протоколу (TCP або UDP) визначається конкретними вимогами певного проєкту. Саме транспортний протокол визначає, як дані пересуватимуться між клієнтами та сервером. TCP гарантує правильну доставку та обробку пакетів в правильному порядку, що особливо важливо у сценаріях, де послідовність даних є ключовою. З іншого боку, UDP надає невелику затримку, що дозволяє йому бути більш пріоритетним рішенням у великих ігрових проєктах, де швидкість важлива.

У сфері реалізації багатокористувацьких ігор існують готові рішення у вигляді мережевих бібліотек, заснованих на UDP. Їх використання полегшує розробку багатокористувацької складової ігрових проєктів, що заощаджує час на інші завдання. Ці бібліотеки надають готові інструменти та рішення для оптимізованої мережевої взаємодії, що сприяє швидкому та ефективному розгортанню ігор. При цьому важливо враховувати вимоги проєкту та обирати саме ті рішення, які найкраще відповідають його потребам.

Отже, вибір між TCP, UDP, або використанням бібліотеки, залежить від кількох факторів. По-перше, потреби самої гри: тут враховується те, чи потрібна їй дуже низька затримка? По-друге, потреби прикладного протоколу: чи потрібен цей надійний протокол?

Узагальнюючи усі викладене вище, розробка багатокористувацьких ігор вимагає уважного вибору мережевих архітектур та компонентів, з урахуванням конкретних вимог та особливостей проєкту, з метою створення ефективної та задовільної геймінг-мережі для потреб користувачів.

РОЗДІЛ 2
АНАЛІЗ ФУНКЦІОНУВАННЯ ІНФОРМАЦІЙНОГО ОБІГУ НА ПРИКЛАДІ ПРОЄКТУ UNITY TECHNOLOGIES

2.1 Передумови нетворкінгу

Ігрова компанія Unity Technologies пропонує проєкт-зразок, який розкриває питання нетворкінгу [38]. Зразок FPS було створено для внутрішньої перевірки функцій і пакетів, доступних під час Unity 2018.3 і доступний для завантаження, включаючи всі ресурси. Цей проєкт надає доступ до ігрового коду, який містить спеціальний код мережі з авторитетною серверною архітектурою, що підтримує до 16 гравців одночасно.

Основна ідея нетворгінгу ігрових проєктів полягає в тому, що у грі є клієнт (програма), який запускає традиційну гру Unity, що візуалізує зображення, відтворює звук і таке інше, а слідом йде сервер, який працює десь у хмарі і не має жодного стосунку до рендерингу. При цьому вони спілкуються за допомогою UDP-пакетів, які клієнт надсилає частіше, ніж сервер. Клієнт надсилає те, що називається командами користувача, які містять те, що він хоче зробити, тобто введення даних з миші та клавіатури. У свою чергу сервер надсилає оновлення про те, що відбувається в ігровому світі. Час, який витрачається на обмін всієї цієї інформації визначається пінгом, або затримкою. Затримка є однією із характеристик інтернету, а також вона є саме тим аспектом, що ускладнює розробку такого роду проєктів [39].

Простий UDP та TCP пакет включає в себе IP-адресу відправника та отримувача, а також дані користувача, які цим пакетом надсилаються (рис. 2.1). Поки цей пакет передається між клієнтом та сервером, є вірогідність того, що з його даними чим ним самим може щось трапитись. Наприклад, є вірогідність дублювання пакета, або він може раптово розділитися на дві частини, проходячи через маршрутизатор, який неправильно налаштований, або з якоїсь іншої причини у кінцевому підсумку прибути в двох примірниках на приймаючій

стороні. Також існує вірогідність того, що один з пакетів може переслатись трохи швидшим маршрутом, ніж інший, що у результаті зробить так, що вони прийдуть до отримувача не в тому порядку. Звичайно, ще існує проблема і повної втрати пакетів з якоїсь іншої причини, чи через комбінацію минулих [48].

Як було зазначено вище, використання TCP забезпечує надійність передачі без цих проблем, але передача за допомогою саме цього протоколу відбувається ціною затримки. Тож, єдиним рішенням буде просто продовжити роботу з UDP, та спробувати вирішити його специфічні проблеми.

Рисунок 2.1 – Вигляд UDP-пакетів

2.2 Вирішення низькорівневих проблем UDP

Для забезпечення коректного обміну даними, використовуючи протокол UDP, розробники застосовують різні алгоритми протидії його недолікам, а саме: алгоритми виявлення зміни порядку пакетів, дублювання та втрат. Також для подальшого налагодження та реалізації деяких інших функцій, розроблюється процедура визначення часу проходження туди та назад RTT [42].

Для вирішення проблеми зміни порядку та дублювання, розробники призначають пакетам порядкові номери. Через те, що порядковий номер

унікальний для кожного пакету, одержувач може з легкістю виявити пакети-дублікати або їх неправильний порядок і тим самим вирішити дві проблеми одночасно.

Рисунок 2.2 – Приклад пронумерованого пакету

Для вирішення проблеми загублених пакетів просто додається ще більше даних у вигляді додаткового порядкового номеру останнього побаченого пакету, з іншого боку. Розглянемо даний момент детальніше. Наприклад, існує пакет, який рухається від клієнта до сервера (рис. 2.3). На ньому написано, що він має номер 123, але останній вхідний пакет від серверу був пакет номер 54, так само і навпаки. Але цієї інформації буде замало, отже, у даному випадку додається детальніший набір даних у вигляді біт-маски про попередні 16 пакетів, яка називається ACK-маскою. Вона дає розуміння того, чи були отримані інші 16 пакетів. Наприклад, остання одиниця означає, що 54-й пакет схоже, що був отриманий, а от наступний – 0, означає, що пакет 53, той, який прийшов раніше, принаймні на момент відправки цього пакета, не було отримано (рис. 2.3). Тому в цьому випадку сервер, коли отримує цей пакет, не може дійсно зробити висновок, що 53 пакет було втрачено, але якщо він зберігає цю інформацію і цей нуль

залишається в інших пакетах, у кінцевому підсумку сервер таки зробить висновок, що він був втрачений [45].

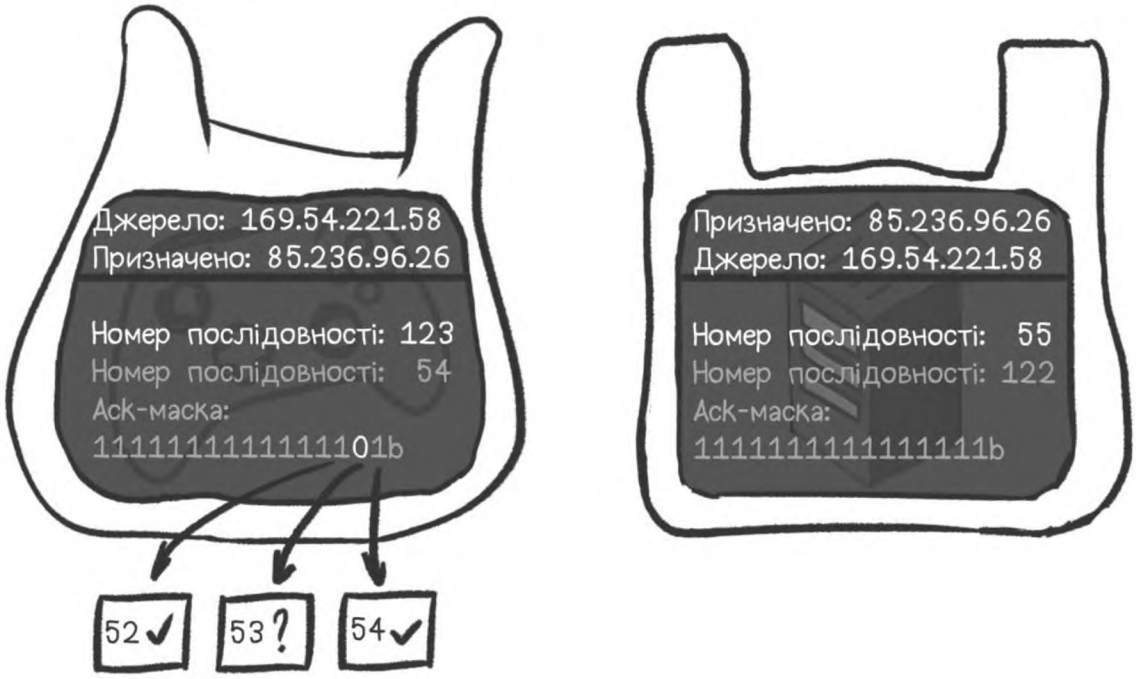

Рисунок 2.3– Приклад пакету з інформацією про останній отриманий пакет та ACK-маскою

Для налагодження та забезпечення роботи алгоритмів, які вирішують більш складні проблеми, необхідне визначення часу обміну пакетами, яке називається вимірювання RTT. Вимірювання RTT (Round-trip time), тобто часу проходження інформації туди і назад, відбувається на обох кінцях зв'язку (рис. 2.4). Клієнт, або сервер запускає секундомір у момент відправлення пакету. Після його надходження іншій стороні вона обробляє його та записує витрачений на це час у новий пакет разом із обробленою інформацією і надсилає його. Коли ж відправник отримав назад пакет, він зупиняє свій секундомір та від цього часу віднімає час, який інша сторона зазначила у пакеті. У зразку від Unity для заощадження ресурсів це вимірювання проводиться для кожного третього пакета, але визначення RTT можна налаштувати й по-іншому [48].

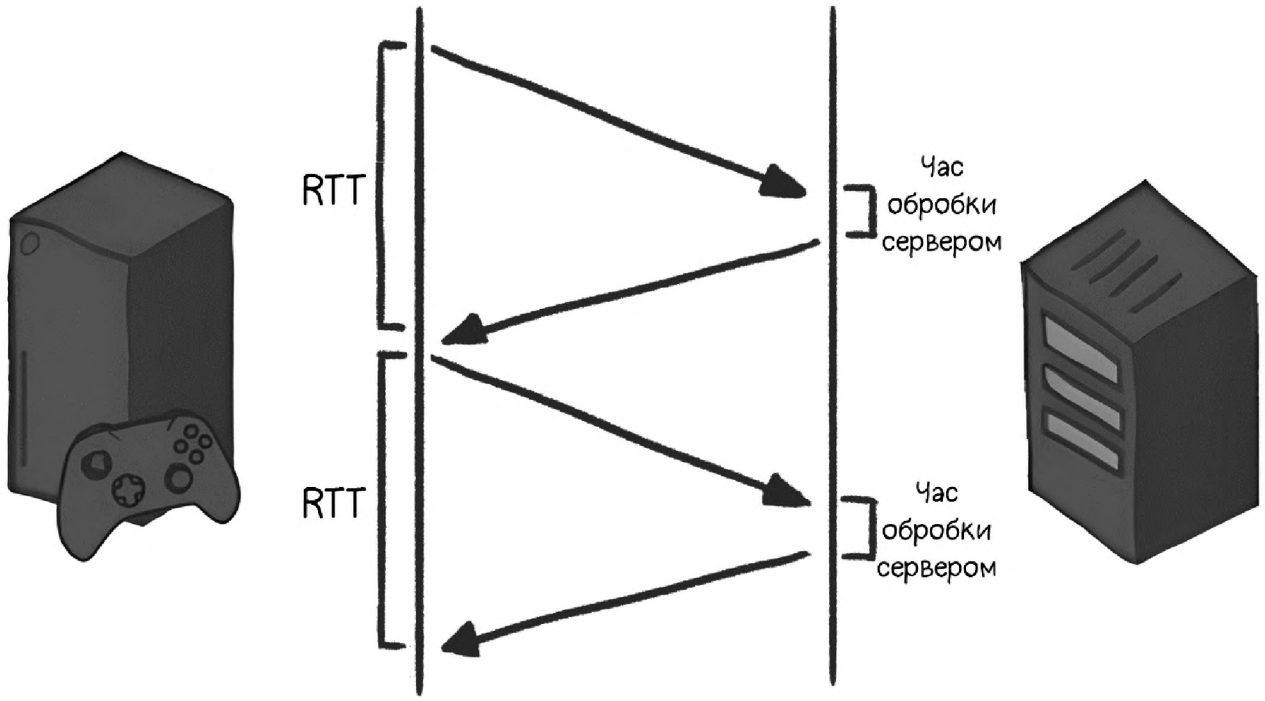

Рисунок 2.4 – Вимірювання RTT

2.3 Побудова циклу інформаційного обміну

Після вирішення низькорівневих проблем будується фактичний цикл. Параметри його роботи можуть відрізнятися від проєкту до проєкту та відповідно до апаратури клієнта. Отже, припустимо, що сервер працює з частотою у 20Гц, а клієнт працює зі стабільною швидкістю у 60 кадрів за секунду. Клієнт на кожному кадрі надсилатиме пакет команди користувача – пакети, які містить такі речі, як введення з клавіатури та миші і так далі. Отже, в будь-яку секунду 60 з цих пакетів будуть «у польоті» від клієнта до серверу, а сервер, з іншого боку, відправлятиме те, що називається знімком [36]. Це пакет, у якому сервер зазначає стан всього ігрового світу, який він надсилає лише 20 разів на секунду в конфігурації за замовчуванням. Якщо спробувати реалізувати цикл у такому вигляді, не додаючи жодного алгоритму оптимізації, то деякі речі можуть піти не так, як треба. Наприклад, може виявитись проблема із тим, що в один UDP пакет неможливо вмістити всю необхідну інформацію. Виявиться проблема, що не

зважаючи на потужність пристроїв гравців, на їхньому власному екрані вони самі та всі інші сутності рухатимуться ривками, 20 разів на секунду.

Для вирішення цих проблем необхідно подивитися на сервер та зрозуміти, як він працює. Сервер в якомусь сенсі схожий на повноцінну гру. Просто він не має клавіатури та екрану, але крім цього має цикл оновлень, який оновлює все, що рухається чи змінюється, щоб переконатись та заявити, що правильні клієнти отримують правильні бали, або таке інше. Єдине, що сервер може виводити, це різноманітну інформацію про стан ігрового світу та статус використання ним ресурсів обладнання у консольний додаток [26]. Таким чином замість клавіатури сервер приймає набори команд з мережі, які через малу частоту оновлень зберігаються в буфері, який називається чергою команд. А замість графічної інформації, яку виводить повноцінна гра, він надсилає пакети з вказівками своїм клієнтам через мережу.

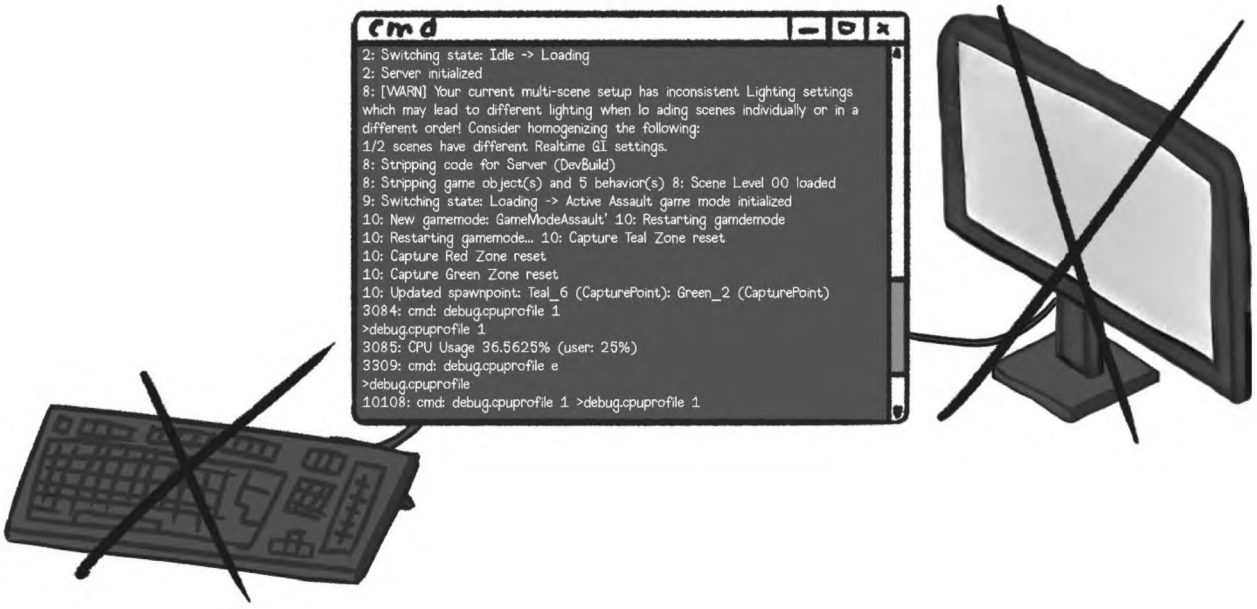

Рисунок 2.5 – Приклад виділеного серверу

Отже, сервер обробляє всі ці команди та виконує симуляцію лише один раз за визначений час. Одне його таке оновлення, або шаг, називається як у годинника, тіком [26]. Тіки, як і пакети, також нумеруються. Результатом його

роботи є кадр, який надсилається усім клієнтам. Через підвищену частоту прийому даних, у процесі роботи сервер встигає обробити одразу кілька пакетів користувачів і відправити їх у кадрі. Але існує вірогідність того, що інформація до нього зовсім не дійде, а тік виконувати потрібно. У такому випадку йому потрібно зробити деякі припущення. Наприклад, припущення в якому він буде вважати, що гравці просто нічого не роблять: не рухаються, не стріляють. Або, можливо, роблять те ж саме, що і минулий тік, або щось подібне. Отже, розглянемо ситуацію, де сервер все ж таки встиг ці пакети зловити і в цьому випадку пакети надходять не по порядку [36]. Наприклад, пакет під номером 92 приходить раніше за пакет 91. Через те, що UDP не можна довіряти, клієнт фактично пакує чотири різні команди користувача у кожен з цих пакетів. Першим йде той, що є потенційно найновішим, а потім чотири старіші. Вони надсилаються стиснутими та не займають багато місця. Завдяки цьому сервер обробляє ці команди у правильному поряду та завершує тік.

Сервер надсилає не весь світ, а відфільтрований стан із сутностями, які знаходяться навколо гравця. Це робиться із трьох причин. По-перше, весь стан може бути занадто великим, щоб передаватися на високій частоті. По-друге, клієнтів в основному цікавлять візуальні та аудіо дані, оскільки більшість ігрової логіки моделюється лише на ігровому сервері. Нарешті, у деяких іграх гравець не повинен знати деякі дані, такі як позиція суперника на іншому кінці карти, інакше він зможе відловити ці пакети та точно знати, куди йти, щоб знайти його.

Для розробки алгоритму вміщення обробленої інформації в один єдиний UDP пакет, для початку потрібно припустити, що не існує ніякого обмеження на розмір пакету та побудувати сам принцип. Сервер проходить через усі динамічні сутності у світі, або як їх називають «replicated» [21]. Таким тегом помічаються ті сутності, про які потрібно повідомити клієнту. Звичайно, це не стіни та підлоги, або таке інше, це саме ті сутності, які мають різні властивості та якось змінюють свій стан. Для передачі їхньої інформації необхідно перетворити її у формат, придатний для цього. Цей процес називається серіалізацією. Припустимо, що цих

сутностей, що мають деякі властивості, які потрібно передати, всього три на весь ігровий світ – це 2 ракети та гравець (рис. 2.8).

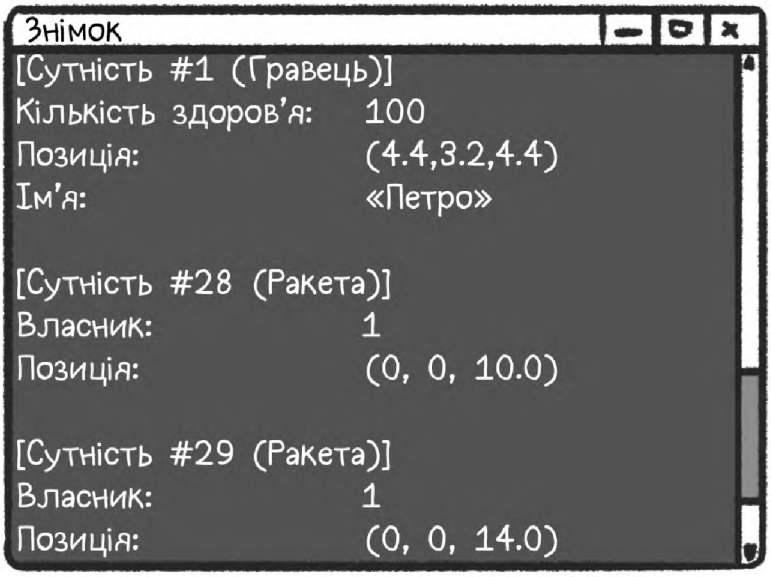

Рисунок 2.8 – Приклад обробленого сервером кадру

Пакування пакету в такому вигляді дуже зрозуміле для людини, але не достатньо ефективне з точки зору його використання (рис. 2.8). Обсяг даних, якими можуть обмінюватися клієнти та сервер, обмежений пропускною здатністю мережі. Стиснення даних може дозволити обмінюватися більшою кількістю даних у кожному знімку та мати швидшу частоту оновлення, або просто знизити вимоги до пропускної здатності. Отже, пакет потрібно трохи переписати і спростити (рис 2.9). Для клієнта достатньо написати список сутностей і по порядку відповідно їхню інформацію [8]. У такому вигляді пакету через те, що клієнт знає, що гравець містить саме такі параметри, знає їхню кількість і те, що вони знаходяться саме в такому порядку, він може просто подивитися на ці дані в одному потоці і не маючи жодного пояснення, використати їх за призначенням, не помилившись (рис. 2.9). А отже, може пропустити деякі та перейти, наприклад, до ракети і так далі. Це один із методів організації та вміщення великої кількості даних в один пакет.

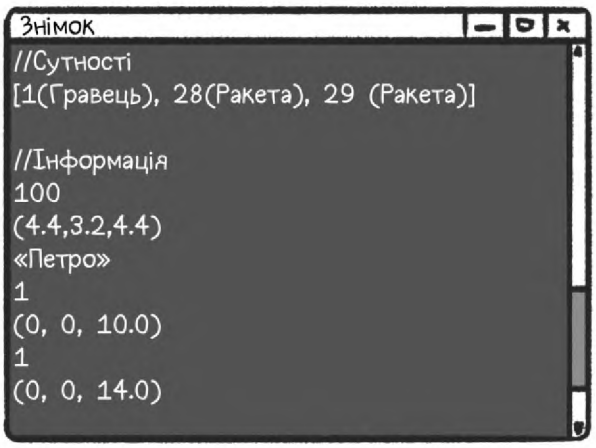

Рисунок 2.9 – Приклад стиснутого пакету

Для розгляду наступного методу переглянемо знімки 88 та 91, у яких не так багато речей було змінено (рис. 2.10). Тут подані усі ті самі сутності, які в основному не змінили свої параметри, але є деякі, що насправді рухаються, це ракети, що подорожують по осі Z [9]. Згадаємо ACK-маску, яка знаходилася в пакеті клієнта і те, що обидві сторони в цьому своєрідному циклі відстежують те, що прибуло на той чи інший бік. Завдяки цій ACK-масці сервер фактично знає, коли і яку інформацію клієнт уже бачив – це є ще одним місцем, де можна заощадити ресурси.

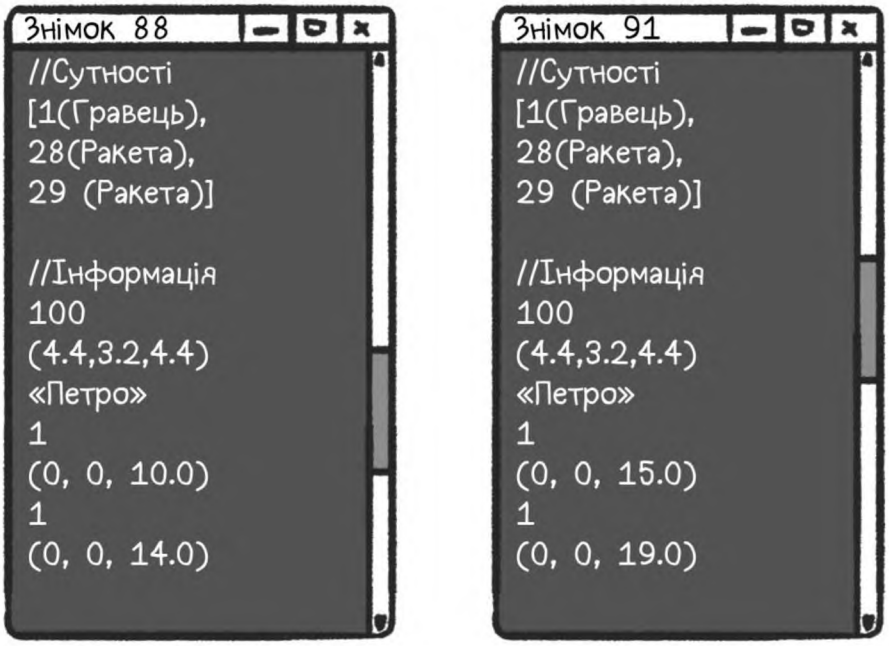

Рисунок 2.10 – Порівняння двох знімків

З огляду на те, що у клієнта вже є попередній знімок, на базі цього можна зробити ще один алгоритм стиснення. Суть цього стиснення полягає у тому, що інформація, яка все ще актуальна, не відсилається, а відсилається лише та, що змінилася, з огляду на старий знімок. Таким чином зменшується розмір пакету в залежності від активності дій самих гравців. При використанні цього методу пакет матиме зовсім іншу структуру, в якій спочатку йтиме базова лінія, яка говоритиме, що цей знімок повинен розглядатися як неповний і він пов'язаний зі знімком 88 та є його продовженням [12]. У такому пакеті буде присутній лише список нових сутностей, оновлення старих та в самому кінці сутності, які зникли. А замість минулої інформації, яка не змінювалася через те, що вся інформація йде у суворому порядку і без зазначення, що це за інформація, сервер записуватиме по одному рядку, який вказуватиме яку кількість рядків треба пропустити і що ці данні не потрібно оновлювати. У цьому випадку використовуються команди: skip 4 та skip 1 (рис. 2.11). У цьому разі клієнт все ще зберігатиме можливість реконструювати всю важливу інформацію із цих двох кадрів і зможе продовжити свою роботу.

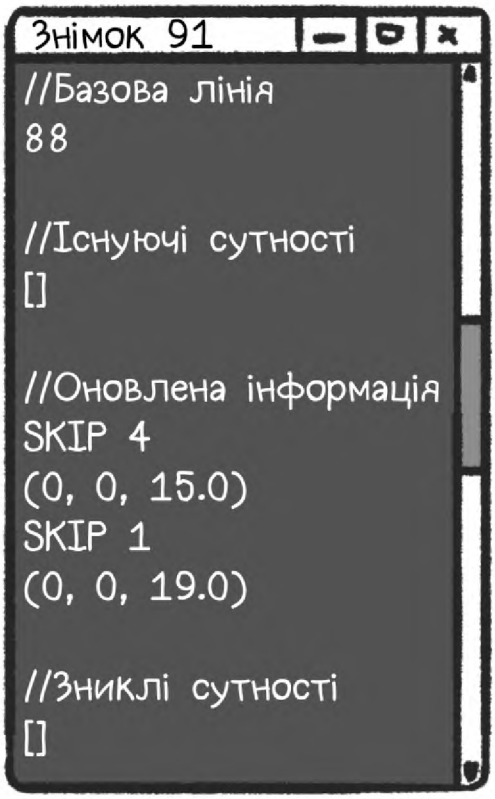

Рисунок 2.11 – Приклад пропуску релевантної інформації

Хоча такий метод і дає зменшення обсягів, але все ж найбільша частина залишається не зміненою – дані, які пересилаються, мають великі значення навіть при зміні лише на десяті. Отже, останнім методом стиснення є дельта-стиск. Стиснення в цьому методі відбувається завдяки надсиланню лише відмінностей між поточним станом гри та останнім станом, отриманим клієнтом. Припустимо, сервер створив знімок світу в тік 91, але в нього залишилися ще старі знімки. У даному випадку знімки 80, 85 і 88. Це не випадкові знімки, це знімки, про які сервер точно знає, що клієнт їх бачив [13]. Враховуючи, що у обох сторін є кадри 82, 85, 88 та те, що існують такі дані, які змінюються з постійною швидкістю, наприклад, це може бути ракета, яка рухається майже по прямій лінії з постійною швидкістю, рух якої здається цілком передбачуваним. Або це може бути рівень здоров'я гравця, показник якого не має жодної передумови до зміни і залишиться на одному рівні протягом тривалого часу. У цьому випадку сервер може спробувати запустити функцію передбачення, яка може створити передбачення того, як може виглядати кадр 91.

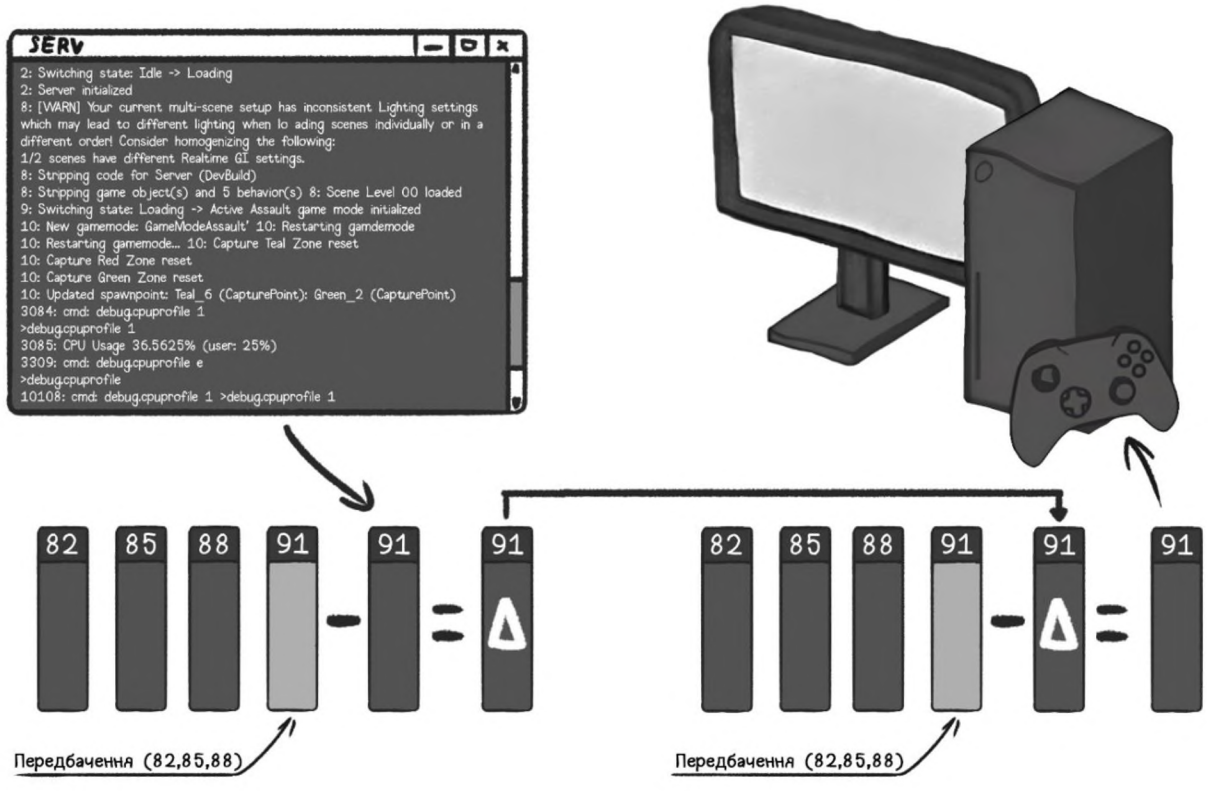

Рисунок 2.12 – Процес створення дельта-стану для клієнту

Сервер створює цей прогноз, але він може бути неправильним. Згодом він віднімає його від реального стану і отримує те, що називається дельта-станом, який, якщо він правильний, то повинен мати багато нулів. Оскільки він має багато нулів, пакет стає ще меншим і тепер сервер може відправити його клієнту, який, у свою чергу, має ті самі знімки, про які згадувалося раніше [14]. Тепер вже клієнт може запустити такий самий код передбачення. Через те, що це той самий код і ті самі дані, можна припустити, що він виробляє той самий оптимістичний прогноз 91. Тоді клієнт можете відняти свій прогноз від дельта-стану, який надіслав сервер, і використати його у грі – саме так виглядає процес створення дельта – стану для клієнту (рис. 2.12).

Отже, підсумовуючи, фінальний пакет буде виглядати саме як список базових лінії. Те, що називається схемами, списком нових і зниклих сутностей, списком оновлень, де крапки є стиснутою дельтою, що з'явилися завдяки використанню такого роду системи (рис. 2.13).

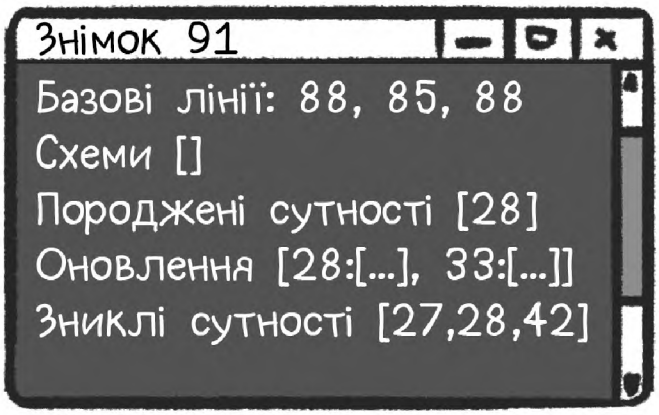

Рисунок 2.13 – Фінальний вигляд пакету

Також існують ще декілька методів компресування даних, серед них, зокрема, існує упаковка бітів. Цей метод полягає у використанні саме тієї кількості бітів, яка потрібна для представлення даної величини. Наприклад, якщо є перерахування, яке може приймати 16 різних значень, гра використовуватимете лише 4 біти замість цілого байта (8 бітів). Упаковка бітів особливо добре працює з квантуванням. Квантування – це техніка стиснення з втратами, яка полягає у

використанні лише підмножини можливих значень для кодування величини [26]. Найпростішим способом досягнення квантування є скорочення чисел з плаваючою комою.

Інші техніки полягають у використанні алгоритмів стиснення без втрат [41]:

– кодування Хаффмана з попередньо обчисленим кодом, яке є надзвичайно швидким і може дати хороші результати. Його використовували для стиснення пакетів у мережевому механізмі Quake3;

– zlib – це безкоштовна, юридично необтяжена бібліотека стиснення даних без втрат загального призначення для використання практично на будь-якому комп'ютерному обладнанні та операційній системі. Вона заснована на алгоритмі DEFLATE і використовується в багатьох програмах. Він підтримує різні стратегії, обробку помилок і параметри використання ресурсів;

– кодування довжини серії, можливо, є найпростішим алгоритмом стиснення, але воно дуже ефективне для певних типів даних. Це особливо підходить для стиснення ландшафтів із плиток або вокселів, де багато суміжних елементів схожі;

– існує також платна бібліотека Rad Game Tools під назвою Oodle Network Compression. На офіційній вебсторінці бібліотеки існує інформативний графік, де порівнюється ступінь стиснення кодування Хаффмана, zlib та їхнє рішення, яке виглядає дуже повчально.

Отже, коли інформація вже сформована, треба повернутися до процесу серіалізації та зробити так, щоб цю інформацію можна було передати. Першою ідеєю може бути використання читабельного формату, такого як JSON або XML. Але це було б зовсім неефективним і вимагало б велику пропускну здатність. Замість цього доцільно використовувати двійковий формат, який є набагато компактнішим. Таким чином пакети будуть містити лише купу байтів. При цьому потрібно пам'ятати, що існує одна проблема, до якої може призвести даний метод – це невідповідність порядку байтів, коли порядок байтів може відрізнятися від одного комп'ютера до іншого [48].

Також можна використовувати бібліотеки, які розроблені для допомоги серіалізації даних, наприклад:

– flatbuffers від google;

– cap'n proto від sandstorm;

– пластівці від шейна гранта та рендольфа вурхіса.

Альтернативою є самостійна розробка, що потребує багато знань та навичок, особливо якщо у коді є підхід, орієнтований на дані. Це також може дозволити виконати певну оптимізацію, якої не завжди можливо досягти за допомогою бібліотеки.

2.4 Інтерполяція на стороні клієнта

Кожен гравець очікує того, щоб на його екрані світ рухався максимально плавно: це можуть бути рухи інших гравців, ракети, рухомі платформи та будь-які інші рухомі речі, що динамічно змінюються у світі. Усі ці елементи повинні оновлюватися з достатньою швидкістю. А для цього потрібна велика потужність обладнання та пропускна здатність інтернету, надмірне використання яких не оптимальне. Отже, для оптимізації цього процесу існують спеціальні методи. Зокрема, є два способи досягти цього: за допомогою інтерполяції або екстраполяції. Інтерполяція використовує два останні стани та показує перехід від одного до іншого [46]. Її недолік полягає в тому, що вона викликає деяку затримку, оскільки клієнт завжди показує, що сталося в минулому. Екстраполяція полягає в передбаченні того, де зараз будуть сутності відповідно до останнього стану, отриманого клієнтом. Її недолік полягає в тому, що якщо суб'єкт повністю змінить свій напрямок, виникне велика похибка між прогнозом і реальним станом речей.

Отже, існує часова шкала з кадрами, які малює клієнтська сторона. Вона оновлюються зі швидкістю 60 кадрів на секунду. Клієнт намалював один кадр, вказівок щось змінити не було, отже, він малює наступний кадр, нічого не

змінюючи, а потім іще один. На цей кадр клієнт нарешті отримує пакет від серверу, в якому є вказівка пересунути персонаж, адже його позиція змінилася. Через те, що пакети з вказівками від серверу надходять лише 20 разів на секунду, клієнт малює на екрані персонажа, який 57 кадрів просто стоїть і лише 3 рази на секунду рухається з великими стрибками (рис. 2.14).

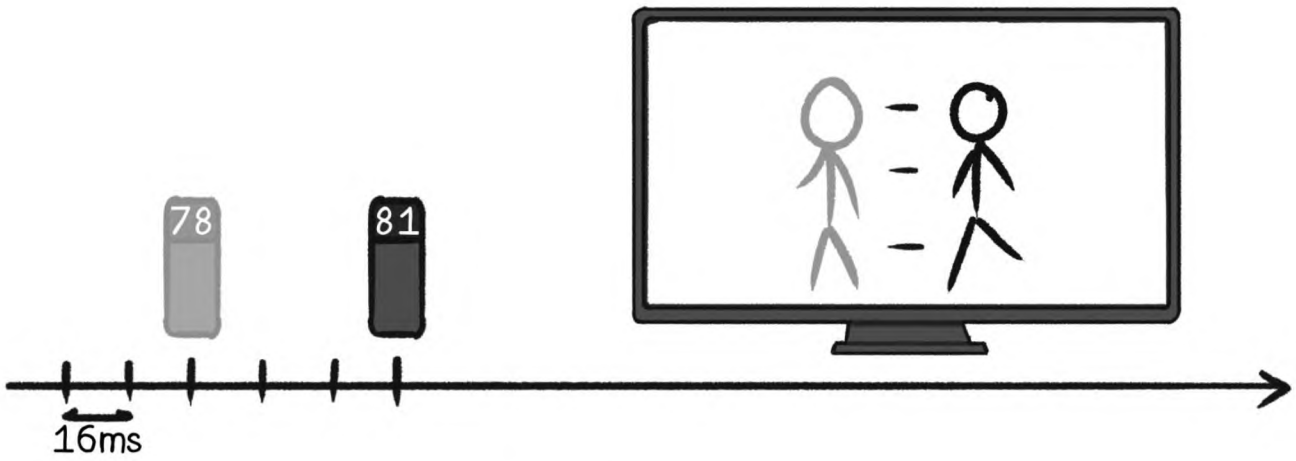

Рисунок 2.14 – Часова лінія клієнту

Відбувається це через те, що на момент оновлення пройшло 19 кадрів, за які персонаж не стояв, а рухався на сервері чи в іншого клієнта. Ідея полягає в тому, щоб згладити ці рухи шляхом застосування спеціального алгоритму за рахунок введення інтерполяції на стороні клієнта. Отже, для того, щоб зробити згладжування руху до отримання наступного 84-го пакету, потрібно взяти будь-який проміжок часу між оновленнями серверу, наприклад, проміжок часу між 78 і 81 пакетом, і зробити між ними інтерполяцію позиції персонажу, а потім за тим же алгоритмом інтерполювати інші кадри. Після цього персонаж буде рухатись в кожному кадрі, який малює клієнт [35]. Це є вирішенням проблеми руху персонажів, які пересуваються у гравця на екрані, але потрібно забезпечити плавний рух і персонажу самого гравця. Для цього відбувається прогнозування на стороні клієнта, яке насамперед просто дозволяє гравцю рухатися попереду сервера, наче сподіваючись, що сервер це дозволяє та не чекаючи його відповіді, як зображено на наступному малюнку (рис. 2.15). Однак, коли клієнт отримує

оновлення від сервера, він мусить перевірити правильність свого прогнозу. У протилежному випадку він повинен змінити свій стан відповідно до того, що він отримав від сервера, оскільки сервер є авторитетним. Ця техніка вперше була використана в Quake.

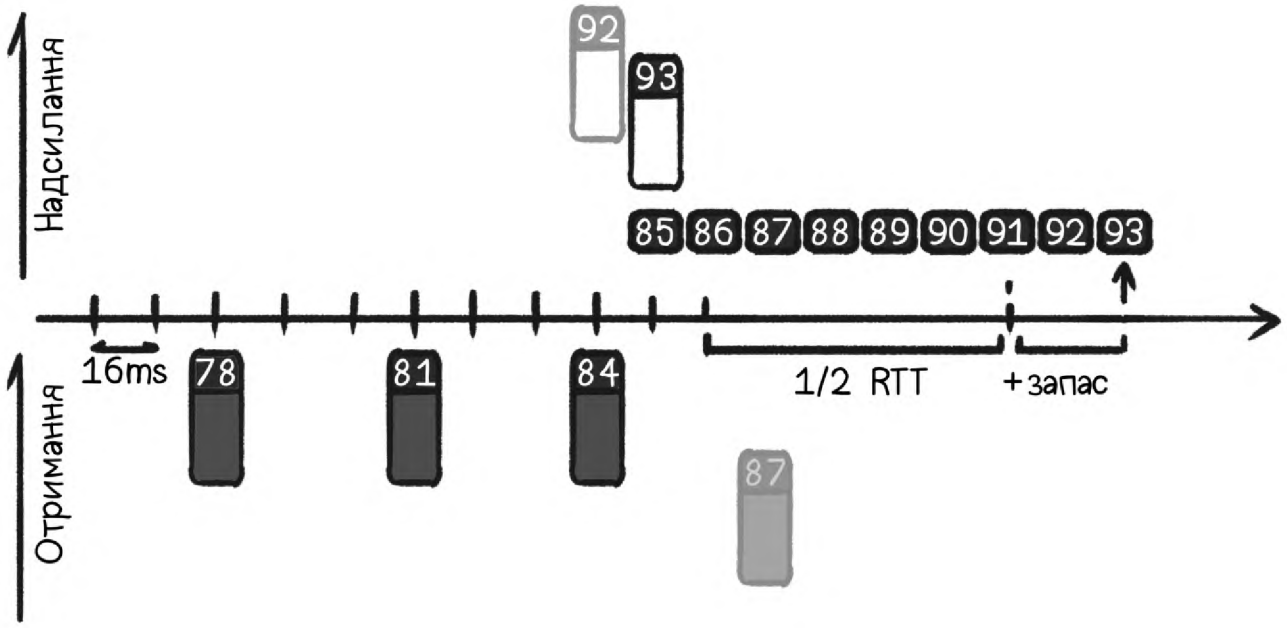

Рисунок 2.15 – Процес комунікації з сервером з точки зору клієнта

Із точки зору клієнта, до нього надходять оновлення від сервера приблизно кожні три кадри і виконується інтерполяція світу між ними. Через низьку частоту роботи серверу для забезпечення плавності руху персонажу гравця, дозволяється відтворювати тільки свої минулі команди руху і, отже, оновлювати світ таким чином, що запускається лише частину коду, саме та, яка використовується між сервером і клієнтом один раз на тік. Питання полягає в тому, на скільки довго клієнт може робити це? Час обміну інформації між клієнтом і сервером становить 100 мілісекунд, або це приблизно п'ять тіків серверу за умови, що він працює зі швидкістю 20 тіків. Це можна визначити, спробувавши вирахувати, котра година на сервері зараз, у конкретний момент. Клієнт бере кадр, який тільки-но отримав від сервера, наприклад 84, з нього дізнається, що йому, знадобилося 3 тіки, щоб прийти до нього. З цього клієнт може припустити, що сервер, ймовірно,

знаходиться в режимі пошуку 87-го пакету прямо зараз. Але якби клієнт відправив би щось на сервер водночас, воно б не надійшло негайно, а прийшло б через 50 мілісекунд, це приблизно 3 тіки [33]. Тому, якщо клієнт надішле щось прямо у даний момент, воно прибуде на сервер тільки тоді, коли той збиратиметься виконувати тік під номером 90. Таке запізнення пакету на сервер може призвести до проблеми з незареєстрованною дією, яку гравець вже виконав. Отже, для забезпечення вчасної обробки даних клієнт додає запобіжний захід у вигляді додаткових двох кадрів запасу і надсилає тік під номером 92. Завдяки цьому кадр прибуває на сервер раніше і трішки чекає у черзі. Тоді наступне питання полягає вже в тому, що клієнту потрібно робити вже у наступному кадрі? Тут є кілька варіантів, але зазвичай усі роблять один і той самий крок – це зробити повний відкат назад. Гра просто повертає назад увесь стан гравця до останнього отриманого пакету даних від сервера. У даному випадку це кадр 84. Але клієнт обраховує вже наступний кадр, тому він зараз знаходиться на кадрі 85. Це означає, що зараз йому потрібно зробити раніше описані обчислення наново і сказати, що сервер, ймовірно, знаходиться на 88-му кадрі [22]. Якщо ж він надішле щось зараз, воно надійде вже на 91 тік. Тому клієнт передбачає кадр аж до 93-го і відправляє цей пакунок. Якщо з якихось причин виходить так, що деякі кадри були втрачені, у результаті будуть утворюватися дірки в цьому потоці. Можливим рішенням цієї проблеми було б просте збільшення розміру інтерполяції, але такий варіант має деякі наслідки для ігрового досвіду у вигляді втрати пакетів. Тому на даному етапі ігровий досвід все ще чутливі до втрати пакетів.

2.5 Компенсація затримки на стороні серверу

Ще одна проблема полягає в тому, що гравцю хочеться, щоб все, що він робить у себе, одночасно однаково повторювалося як на сервері, так і в інших гравців. Гравець не хоче робити жодних поправок через проблеми зв’язку, чи

потужність серверу. Отже, останньою технікою, що є найдосконалішою та корисною лише для FPS, є компенсація затримки. За допомогою компенсації затримки сервер враховує затримку клієнта, коли він, наприклад, стріляє у грі в ціль [13]. Так, у ситуації, якщо гравець зробив постріл у певну ціль на своєму екрані, але насправді його ціль знаходиться в іншому місці через затримку, з точки зору гри було б несправедливо щодо гравця відмовити йому у зарахуванні попадання саме через технічну затримку. Таким чином для вирішення цієї проблеми сервер буде перемотувати назад у часі гру саме у той момент, коли гравець влучив, щоб імітувати те, що гравець бачив на екрані і перевірити зіткнення між його ціллю та мішенню.

Таким чином перша частина алгоритму вже реалізована на стороні клієнту. Якщо гравець стріляє зі зброї, що в тому ж кадрі попадає у ціль, грі знову доводиться компенсувати затримку, але вже на стороні сервера. Адже дивлячись на це його очима, виглядатиме все так, ніби є два клієнти, де один із них збиратиметься стріляти в іншого, на якого націлений. Отже, сервер отримав від гравця команду 92 з вказівками стріляти зі своєї зброї. Хоч він і думає, що попаде, це ніколи не станеться саме через затримку сервера. Треба згадати, що у цей час на стороні клієнта був виконаний прогноз, який йде попереду 84-го кадру, а той, у свою чергу, йде приблизно так само далеко вперед, як і повний час подорожі туди та назад [24]. Також у клієнта існував буфер, який повертається назад у часі з метою інтерполяції. Це означає, що те, що він бачив на той момент на екрані біля свого прицілу, було його діями на 92-му кадрі, на 180 мілісекунд попереду. Тому клієнт націлююся туди, де з його точки зору знаходиться сервер. Але сервер, на щастя, знає про ці затримки. Через те, що сервер зберігає круговий буфер історії всіх гравців за останні 128 кадрів ігрового світу, він знає час подорожі туди і назад та знає, який час інтерполяції у гравців, бо клієнт постійно оновлює сервер цією інформацією. Він просто може зробити те, що називається лаг-компенсацією: просто відтягне ворога назад за допомогою буфера історії до точки, де він був, коли клієнт на нього був націлений і зробив свій постріл

(рис. 2.15) Саме таким чином відбувається компенсація затримки на стороні серверу під час гри.

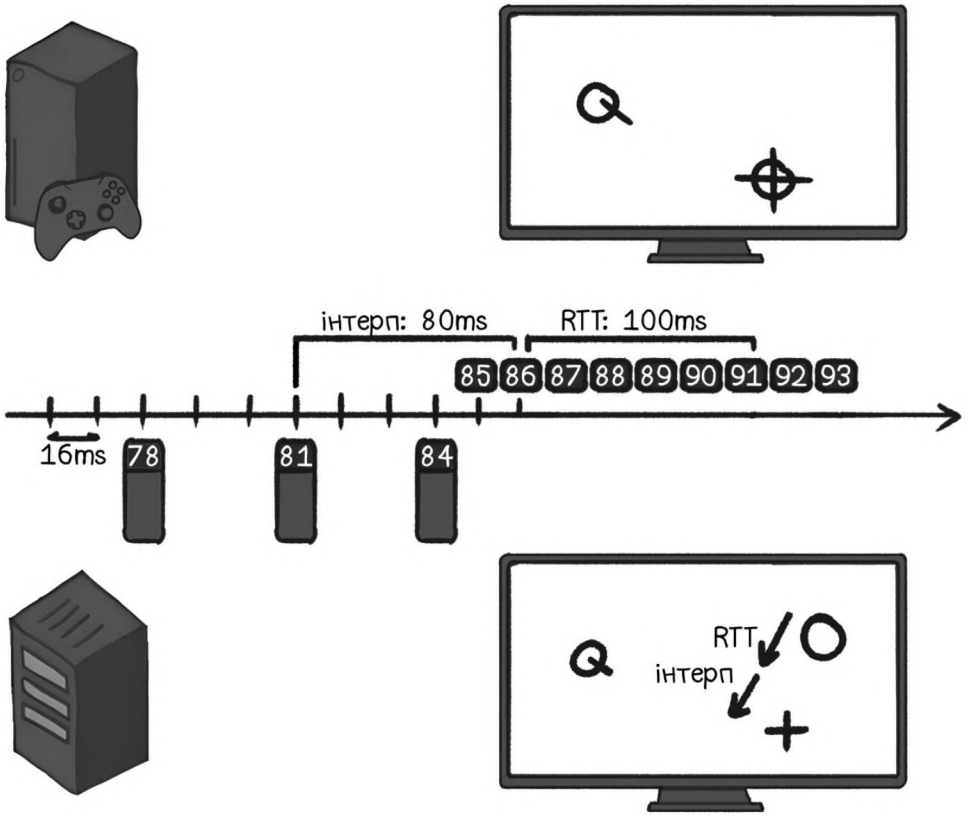

Рисунок 2.16 – Відмінність відображення між клієнтом та сервером та її компенсація

Висновки до розділу 2

У структурі мережевої взаємодії ігрових онлайн-проєктів на основі архітектури клієнт-сервер, ключовою складовою є саме UDP-пакет. За допомогою UDP-пакетів виконується обмін важливою інформацією, такою як: команди користувача від клієнта та оновлення про ігровий світ від сервера [6]. Основна проблема використання UDP полягає саме в його вразливості до порушень порядку пакетів, їх дублювання та втрат. Для подолання цих проблем упроваджуються різні алгоритми виявлення та коригування даних помилок.

Для вирішення проблеми дублювання та порушення порядку пакетів у кожен пакет вводиться власний порядковий номер та вказівки про останні отримані пакети. А для забезпечення імунітету до втрати одного з них, інформація про попередні пакети дублюється у нові пакети. При цьому обмін даними між клієнтом та сервером повинен бути оптимізований з урахуванням обмежень пропускної здатності, використовуючи можливості стискання даних [5].

Через низьку частоту оновлення сервера для забезпечення візуальної плавності гри на екрані гравця, використовується алгоритм її компенсації, який виконує інтерполяцію на стороні клієнта між пакетами серверу, що дозволяє плавно відображати рух об'єктів. Крім того, для синхронізації дій між гравцями використовується компенсація затримки на стороні серверу, що забезпечує однаковий ігровий досвід для всіх учасників ігрового процесу.

У цілому успішна розробка онлайн-проєкту вимагає уважної розробки його мережевого коду. Першорядне значення має прийняття фундаментальних архітектурних рішень із самого початку проєкту. Вибір між одноранговими або клієнт-серверними моделями, мережевими протоколами та планами масштабованості глибоко впливає на продуктивність гри та досвід гравців. Саме ігнорування цих варіантів на ранній стадії розробки може призвести до складних і дорогих переробок пізніше. Багатокористувацька гра – це не функція, яка прикріплюється до гри наприкінці, або щось, що можна зробити один раз на початку та забути пізніше [7]. Це безперервне зобов'язання про яке слід пам'ятати протягом усієї виробничої лінії. Регулярне тестування, налагодження та оновлення – необхідні для підтримки стабільної та успішної багатокористувацької гри. Нехтування цими важливими зусиллями може призвести до розчарування гравців і зниження інтересу спільноти до неї. Отже, реалізація функції багатокористувацької гри може бути складною, часто вимагаючи у 1,5 раза більше роботи та часу, ніж розробка гри для одного гравця. Це включає в себе мережевий код, синхронізацію, заходи проти шахрайства та інфраструктуру сервера.

РОЗДІЛ 3
ПРАКТИЧНІ АСПЕКТИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ПРОЄКТУ

3.1 Програмування механік ігрового об'єкту

У ході виконання магістерської кваліфікаційної роботи, на основі отриманої в ході попереднього аналізу інформації, було здійснено розробку автомобільної комп'ютерної гри. З огляду на це було розроблено список різних аспектів гри з урахуванням її специфіки та здійснена їх реалізація. Розглянемо це детальніше.

Для розробки спрощеної симуляції, у якості ігрового об'єкту було обрано автомобіль, і було проведено детальний аналіз його функціонування. У результаті цього встановлено, що автомобіль, як чотириколісний транспортний засіб, отримує обертальний момент від двигуна залежно від типу приводу. Зазвичай автомобілі оснащені двигуном внутрішнього згоряння, що через свою специфіку працюють лише в обмеженому діапазоні обертів за хвилину (рис. 3.1). Цей тип двигунів на відміну від електродвигуна не може почати обертатися з зупиненого стану. Це робить його неспроможним повністю зупинятися без припинення роботи.

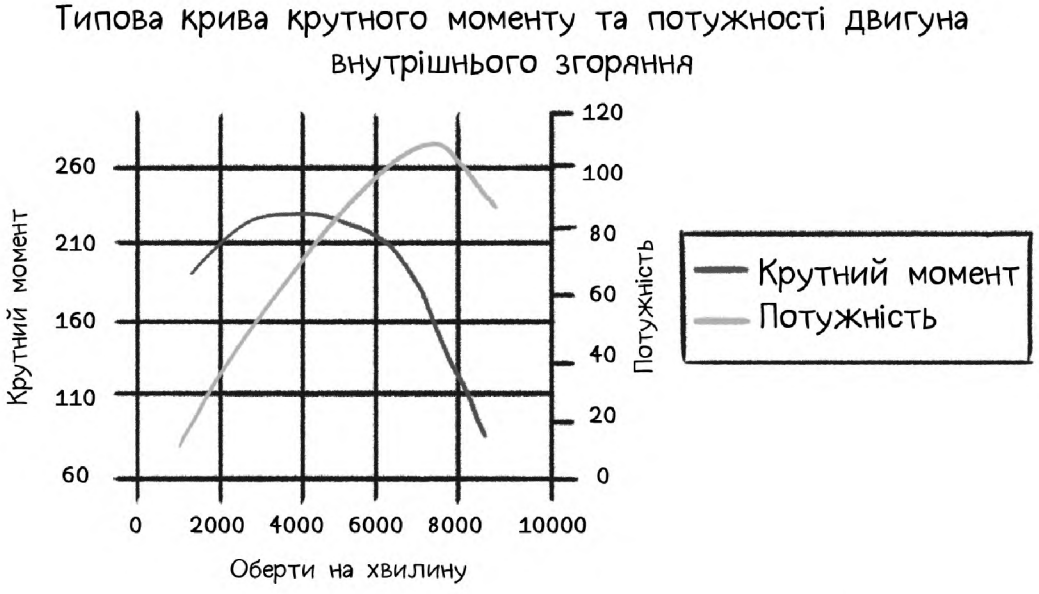

Рисунок 3.1 – Крива роботи двигуна автомобіля

У зв'язку з цим для забезпечення холостих обертів йому необхідний пристрій для від'єднання коліс, що називається зчепленням. Крім того, для ефективної роботи на різних швидкостях та задля їх досягнення використовується пристрій для зміни передавальних чисел, що змінює частоту обертання двигуна та силу, яку він передає на колеса. Пристрій, що використовується для збільшення ефективності, називається трансмісією. Вона має набір передач, що призначені для різних швидкостей.

На жаль, у Unity відсутні будь-які вбудовані рішення для симуляції основних елементів автомобілю, за винятком Wheel Collider, що призначене виключно для симуляції руху коліс. Однак у зв'язку з аркадним характером гри повноцінне моделювання всіх компонентів не доцільне.

Wheel Collider представляє собою спеціалізований вид колайдера, спроєктований для моделювання руху транспортних засобів, особливо тих, що мають колеса (рис. 3.2). У його спеціальний функціонал входять визначення зіткнень, фізика коліс і заснована на моделі ковзання фізика шин. Причому цей інструмент може використовуватися не лише для коліс, адже він був спеціально створений для ефективного моделювання руху транспортних засобів з колесами.

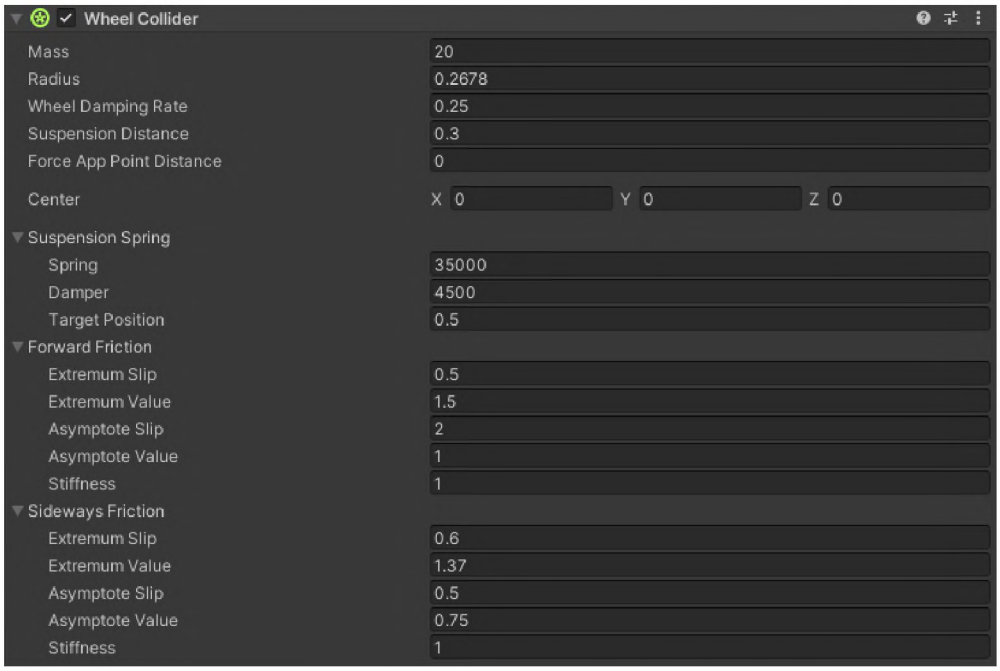

Рисунок 3.2 – Вікно налаштування Wheel Collider

Для передавання сили, що буде рухати тіло, до якого прикріплене колесо, у Wheel Collider існує змінна motorTorque. Вона визначає, скільки обертального моменту передається на колесо. Спершу вона перезаписуватиметься змінною в скрипті, що визначатиме максимальний обертальний момент двигуна без урахування його обертів за хвилину в певний конкретний момент часу. Для контролювання прискорення в функції призначення motorTorque, максимальний обертальний момент помножується на змінну, що визначає відкритість заслінки, діапазон значень якої від 0 до 1. У результаті автомобіль почне рухатись з постійним прискоренням, що неможливе у реальному світі, адже двигун внутрішнього згорання може прискорюватись лише до конкретного моменту та з певною швидкістю. Для симуляції саме цих характеристик використовується крива анімації, яка імітує криву обертального моменту та визначає силу, яку продукує двигун на основі його частоти обертів (рис. 3.3). Даний рисунок демонструє, що обертальний момент починається з нуля по вісі Y, поступово збільшуючись зі збільшенням значення частоти обертів двигуна по вісі X. А далі під кінець знову дорівнює нулю.

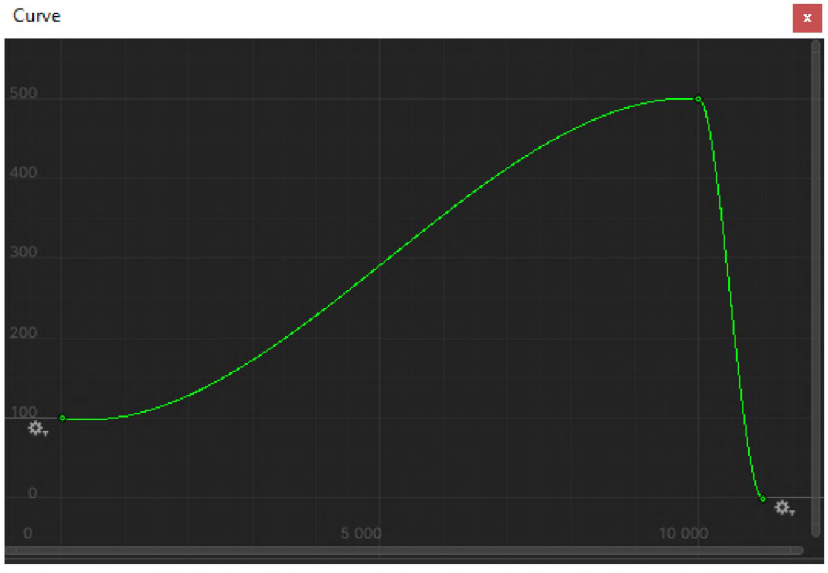

Рисунок 3.3 – Вікно налаштування Animation Curve

У реального автомобіля, коли зчеплення повністю вимкнене, швидкість обертання двигуна безпосередньо залежить від швидкості обертання коліс. На

основі цього факту можна обчислити кількість обертів двигуна, використовуючи значення WheelCollider.rpm. Це дає змогу визначити швидкість обертів двигуна та на основі цих показників використовувати криву анімації.

Маючи функцію визначення кількості обертів двигуна за хвилину під час руху, стає можливим обрахування сили, яку цей двигун виробляє, а отже цю силу можна передати й самому автомобілю. Але перед цим створюється можливість маневрування транспортним засобом. Відсутність керування напрямком руху призводить до того, що автомобіль може рухатися лише по прямій, що є дуже складним завданням через недосконаліть світу. Цей функціонал забезпечується модифікуванням значення змінної WheelCollider.steerAngle, яка перезаписується значенням максимального кута повороту колеса, заданого в налаштуваннях коду, помноженого на значення вводу гравця в межах від -1 до 1.

Після введення кривої обертального моменту двигуна, автомобіль прискорюватиметься до певного моменту з поступовим збільшенням сили, але при цьому рухатиметься дуже повільно через відсутність трансмісії. Саме через її відсутність передавальне число трансмісії, так би мовити, дорівнює одиниці, що означає, що частота обертів двигуна напряму дорівнюватиме частоті оберту привідного колеса. А отже, з такими числами двигун спочатку матиме мінімальну потужність, а при досягнені максимальних обертів автомобіль теоретично їхатиме зі швидкістю 748.9 км/год, якщо це дозволять закони фізики та якщо він встигне до цієї швидкості розігнатися. Тож, для оптимізації роботи двигуна, для забезпечення швидшого прискорення та обмеження швидкості транспорту до тієї, до якої він може розігнатися, необхідна реалізація трансмісії. Для цього використовується змінна типу лист зі значеннями передатних чисел та простий перемикач, що бере дані із листу та записує їх у змінну поточного співвідношення. Ця змінна використовується в обчисленні motorTorque шляхом множення її на минулі калькуляції. Також аналогічним чином вона використовується для визначення частоти оберту двигуна відповідно до швидкості обертання колеса. У результаті цього автомобіль пришвидшується з

реалістичною динамікою та має максимальну швидкість і трансмісію, що і відповідає дійсному руху автомобіля.

У реальному житті двигун може прискорюватися сам по собі, набираючи оберти навіть коли автомобіль не рухається, або коли зчеплення ввімкнуте. Отже і потенційному гравцю обов'язково захочеться натиснути на педаль акселератора в очікуванні саме цього результату. Через те, що обороти двигуна обраховуються відповідно до швидкості автомобіля, в симуляції такого ефекту не відбудеться, оберти двигуна все одно будуть дорівнювати нулю. Спочатку, для вирішення цієї проблеми вводиться умова, що при зупинці автомобіля, коли оберти падатимуть нижче рівня робочих, перестане обраховуватися частина коду «else», яка відповідає за обрахування частоти обертів двигуна від колес. Таким чином реалізується функціонал автоматичного зчеплення, значення якого у цьому випадку дорівнюватиме 0, а отже, можна припустити, що значення трансмісії теж відповідно дорівнюватиме 0. Також завдяки оператору «або», ця функція може примусово деактивуватись, якщо сам гравець натискатиме на відповідну клавішу зчеплення.

Отже, ця функція перестає обраховувати значення двигуна на основі значень від колес, що в свою чергу дає можливість перезаписувати їх іншими значеннями. Потім, коли передатне число трансмісії вже дорівнює 0, виконується основне тіло умови, де функція Mathf.SmoothDamp, що має три змінні, а саме: поточні оберти, бажані оберти та швидкість їх набору чи втрачання, починає змінювати оберти двигуна. Якщо ж автомобіль не матиме ніякого вводу від користувача, ця функція намагатиметься повернути значення обертів двигуна до необхідних для імітації холостого ходу, які є за замовчуванням. В іншому випадку гравець задаватиме цільову швидкість оберту двигуна, перезаписуючи дані холостого ходу перед цією функцією, до яких вона плавно прийде.

Іншим важливим аспектом є звук. Його значення у сприйнятті інформації становить майже 50%. Реальний автомобіль з двигуном внутрішнього згоряння не працює беззвучно – його двигун виробляє характерний для нього звук. Для відтворення цього ефекту використовується простий однорідний запис роботи

реального. Він відтворюється, змінює гучність та частоту за допомогою скрипту в залежності від раніше обрахованих обертів двигуна. Такий метод простий у виконанні, але звучить він штучно, а отже не досконалий. Цей недолік виправляється додаванням набору записів реального двигуна, тільки на різних обертах та при різних обставинах, що змінюються відповідно до обертів. Для зменшення помітності переходу між ними, вони також модифікуються зміною гучності і частоти скриптом, який виконує цю симуляцію. Таким чином, використовуючи усі ці напрацювання було створено й інші автомобілі з різними налаштуваннями.

3.2 Розробка UI та UX

Повноцінна гра повинна мати інтуїтивно- зрозумілий інтерфейс, завдяки якому користувач матиме можливість перемикатися між різними екранами, режимами та активувати чи змінювати певні параметри під себе. Першим таким елементом завжди є головне меню, через яке гравець потрапляє до інших розділів, кількість та контекст яких різниться від проєкту до проєкту. Імплементація ігрового інтерфейсу за своєю суттю є достатньо прямолінійним процесом і мало чим відрізняється від верстки сайту. Як і в тілі html так і на полотні Unity створюються контейнери, що містять у собі функціональні або декоративні елементи. Найпростіше головне меню з усіма необхідними кнопками виглядає як набір різних контейнерів із заданими розмірами, параметрами вирівнювання, відносним положенням на екрані та наповненням (рис. 3.4). У вікні налаштування Wheel Collider серед цих контейнерів головними елементами є кнопки. При натисканні на них, за схожим принципом з java scrip в html, активується частина коду, за яку ця кнопка відповідає. Він приховує та виводить на екран потрібні полотна. Наприклад, кнопка під назвою «Ride» приховує головне меню та відкриває меню вибору треку, а кнопка «Garage», також приховує головне меню і відкриває вже меню, у якому буде відбуватися вибір автомобіля та зміна

положення камери. Наступні кнопки за аналогічним принципом відкриватимуть вже інші полотна та виконуватимуть різні функції, відповідно до контексту гри.

Рисунок 3.4 – Вікно налаштування Wheel Collider

Розглянемо меню вибору автомобіля (рис. 3.5). Воно має три кнопки, серед яких одна в лівому верхньому куті, що як і попередньо розглянуті, просто перемикає полотна і виконує мінорі функції, в даному випадку повертає користувача у попереднє меню. А стан вже двох інших кнопок відслідковується скриптом, який відповідає за зміну автомобіля. У цьому скрипті кнопки «PREV» та «NEXT» відповідають за збільшення чи зменшення індексу змінної на один, яка визначає номер поточного автомобіля. Функції цього скрипта викликаються перед першим кадром для початкового налаштування сцени та при зміні індексу обраного автомобіля, тобто при натисканні на минулі клавіши. У результаті роботи скрипта активуються частина коду, яка прибирає попередній автомобіль зі світу та на основі нового значення додає автомобіль з відповідним індексом у масиві. Це відбувається таким чином, що при активації скрипту функцією Destroy(SpawnedCar) спочатку руйнується автомобіль, який вже існує, індекс змінюється. Слідом до сцени додається новий автомобіль з відповідним індексом «CarToSpawn» за допомогою функції SpawnedCar = Instantiate(Car[CarToSpawn].Prefab, position, rotation), в якій попередньо зазначається положення та обертання параметрами «position» та «rotation»

відповідно. У кінці сутність та номер нової машини записується в пам'ять для повторення минулих дій у майбутньому. Таким чином користувач має змогу візуально бачити зміну автомобіля, а гра на основі записаного індексу може додавати обраний автомобіль вже в іншій сцені.

Рисунок 3.5 – Меню вибору автомобіля.

Маючи готову модель автомобілю та меню його вибору, наступним кроком є створення меню, завдяки якому можна його випробувати на одному із представленому в грі треку. Це просте меню з вже знайомою кнопкою назад та вікном вибору треку, де на основі ще одного масиву, використовуючи цикл foreach автоматично, за шаблоном, у вікно додаються кнопки у вигляді мініатюр існуючих локації (рис. 3.6).

Рисунок 3.6 – Меню вибору треку

При натисканні на мініатюру треку, виконується вивантаження поточної сцени з пам’яті та завантаження сцени з обраним треком. Зміна сцен виконується завдяки вбудованій фунції SceneManager.LoadScene(i), де і – це номер сцени в налаштуваннях поточної версії розробки (рис. 3.7). Завдяки тому, що в циклі йде автоматичний рахунок його активацій, це значення можна присвоювати в нумерації кнопок, а отже і для обирання сцени, тобто для використання його замість індексу «і».

Рисунок 3.7 – Меню налаштування робочої версії розробки

Після завантаження нової сцени автоматично запускаються її скрипти, серед яких схожий за принципом обирання автомобілю скрипт, який просто додає його на певну локацію. Також додається інтерфейс гравця з міні-картою треку та телеметрією автомобілю. До телеметрії відносяться два текстові поля з поточною передачею, швидкістю автомобіля та тахометер, який автоматично розраховує кількість поділок на основі характеристик автомобіля та виводить їх на екран з потрібним відступом одна від одної, після чого в тілі update оновлює на ньому стрілку. У свою чергу міні-карта – це картинка, на якій у режимі реального часу оновлюється зображеннями з ще одніє камера, але з ортографічною проекцією, що прив’язана до основної. Вона знаходиться над машиною зверху на відстані і направлена строго вниз та в налаштуваннях сортування в неї вимкнені всі сутності для рендерингу окрім дорожнього полотна, а із чотирьох каналів RGBA вона бачить лише один. В результаті, це дає змогу бачити камері лише форму дороги без зайвих об’єктів та освітлення.

Таким чином гравець отримує можливість завантажуватися на різні локації разом із обраною машиною та одразу отримує весь необхідний інтерфейс для неї (рис. 3.8).

Рисунок 3.8 – Обраний трек з машиною

Для вільного пересування у грі гравець повинен мати змогу переходити і в минулі сцени. Для того, щоб для повернення назад гру не доводилося повністю закривати та запускати заново, щоб зробити щось інше, створюється спеціальне нове вікно з функціями повернення назад у гараж та перезапуску поточної локації (рис. 3.9). Також зазвичай це меню виконує іще одну функцію – функцію призупинення гри. Хоч і функція паузи не завжди дозволена розробниками, це меню все одно називається меню паузи. У цьому випадку при його виклику зупиняється ігровий процес та відкривається доступ до раніше згаданих функцій перезапуску карти, повернення до гаражу та відновлення ігрового процесу. Тут використовується вже існуючий скрипт, який переносить гравця на іншу сцену, тільки вже з іншими номерами сцен, необхідними для того, щоб перезавантажити карту і повернутися до гаражу. Інтерфейс гравця ховається параметром SetActive(false), елементу «HUD», а меню паузи навпаки – показується, використовуючи те ж саме налаштування параметру SetActive(true), тільки вже для елементу «PauseMenu». У коді виглядає це як елемент.SetActive (значення). У зворотному випадку «значення» змінюється на «true» та «false». За можливість зупинки гри відповідає функція Time.timeScale = 0f, де нуль – це множник часу. За замовчуванням він дорівнює одиниці, а при викликанні меню паузи його множник автоматично змінюється на нуль. Через те, що цей параметр є основним налаштуванням гри і не змінюється автоматично, перед виконанням функції коду, яка припускає зміну локації, чи при виході з меню паузи, потрібно повернути це значення на 1 чи на інше значення, при якому гра повинна працювати в звичайному режимі.

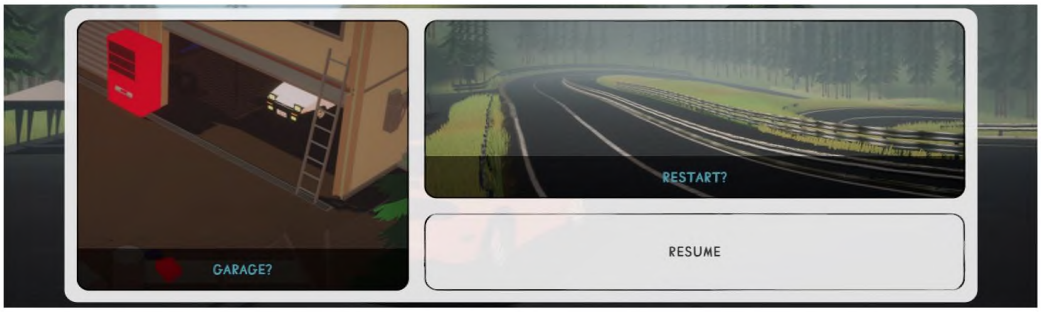

Рисунок 3.9 – Меню паузи

3.3 Принципи програмування ігрових режимів

Кожна гра має свою власну мету. Мета даної гри досить примітивна і розроблена для візуалізації обраної концепції. При завантаженні на трек від гравця потрібно лише просто збивати об'єкти своїм автомобілем, щоб заробляти бали. Об'єктами гри виступають прості дорожні конуси, зображені на малюнку (рис. 3.10), при збиванні яких нараховується певна кількість балів, що відображаються в інтерфейсі гравця в лівому верхньому куті. Ця механіка починає свою реалізацію з самого конуса, з яким пов'язаний скрипт, що відповідає лише за увімкнення регдолу (симуляції фізичної взаємодії) для нього лише тоді, коли стикається з автомобілем. Це зроблено для заощадження ресурсів комп'ютеру.

Рисунок 3.10 – Дорожні конуси

Механізм нарахування балів вбудований в ігровий об'єкт (автомобіль), де окремий скрипт стежить за тим, чи зіткнувся кузов машини саме із конусом.

Після виконання цієї умови просто виконується частина коду, яка відповідає за нарахування балів, це простий оператор інкременту. А вже для оновлення даних на екрані використовується скрипт інтерфесу, що опитує скрипти автомобілю і бере необхідні значення (рис. 3.11).

Рисунок 3.11 – Вигляд лічильника в інтерфейсі гравця

3.4 Реалізація багатокористувацької складової гри

Для створення багатокористувацької частини гри, щоб не витрачати час на те, що вже було розроблено, логічним рішенням є використання готового рішення для змагальних ігор. А отримані раніше знання про те, як працює обмін даними в такого роду проєктах, використовувати вже для налагодження та оптимізації цього рішення. Цим рішенням є додаток Netcode for Entities, розроблений на основі ECS (Entity Component System), створеного для продуктивності та масштабованості. Netcode for Entities – це частина Data Oriented Technology Stack (DOTS) Unity, що надає повноважний сервер із структурою прогнозування клієнта, яку можна використовувати для створення ігор для кількох гравців.

Спочатку налаштовується спосіб обміну даними між клієнтом і сервером. Обмін в Netcode for Entities досягається створенням іншого світу для сервера та кожного клієнта (через пакет Entities). Щоб обмінюватися даними між сервером і

клієнтом, до основної сцени додається підсцена. Це робиться у відповідному меню, яке відкривається при натисканні правої кнопки миші у вікні ієрархії. Для кращого розуміння краще назвати цю нову сцену, як «SharedData». Після чого окремі всесвіти можна переглянути у вікні ієрархій сутностей, що зображено на малюнку (рис. 3.12).

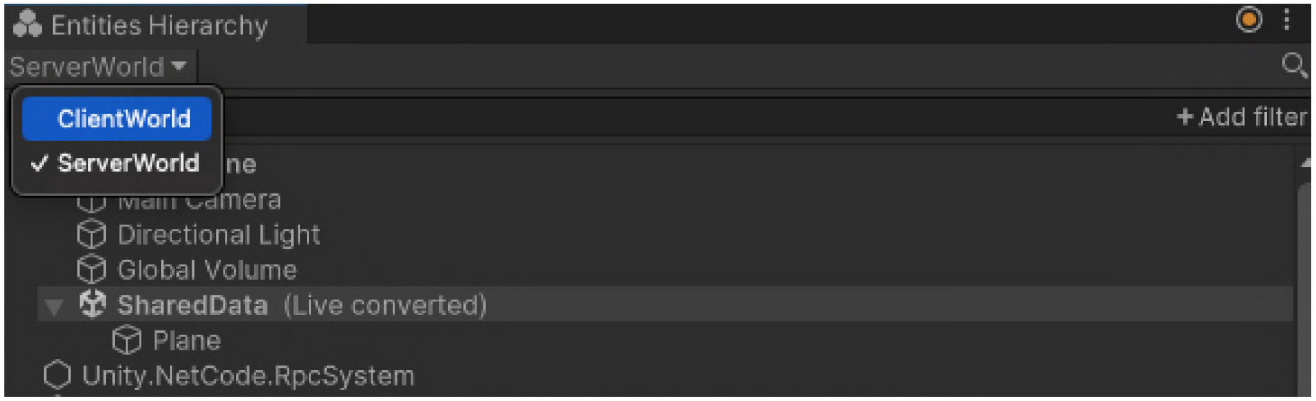

Рисунок 3.12 – Ієрархія сутностей

Щоб увімкнути зв'язок між клієнтом і сервером, потрібно встановити з'єднання. У Netcode for Entities найпростішим способом досягнення цього є використання функції автоматичного підключення. Функцію автоматичного підключення можна використовувати, успадкувавши від ClientServerBootstrap, а потім встановивши AutoConnectPort для обраного порту.

Створюємо файл із назвою Game.cs у папці Assets і пишемо до нього код, в якому буде створюватись спеціальна завантажувальна програма, що вмикатиме автоматичне підключення.

Після підключення можна почати спілкування. Важливою концепцією в Netcode for Entities є концепція InGame. Коли з'єднання позначено текстом «InGame», симуляція повідомляє про готовність розпочати синхронізацію.

Спілкування з Netcode for Entities виконується за допомогою RPCs. Отже, щоб продовжити, створюємо RPC, який діє як повідомлення «Перейти в гру» (наприклад, повідомимо серверу, що клієнт готовий почати отримувати знімки).

Створюємо фай під назвою GoInGame.cs у папці Assets і пишемо до нього код, який буде комунікувати з сервером.

Для синхронізації рухомих ігрових об’єктів між клієнтом та сервером, необхідно створити визначення мережевого об’єкта, який називається привидом.

Для цього створюємо куб у сцені та створюємо його Prefab, перетягнувши у папку Asset. Потім цей куб можна замінити на модель машини, просто додавши необхідні компоненти в середину Prefab, які будуть автоматично змінювати модель для візуалізації відповідних автомобілів. Завдяки системі Prefab ці дії автоматично застосовуються і до інших його інстанцій (рис. 3.13).

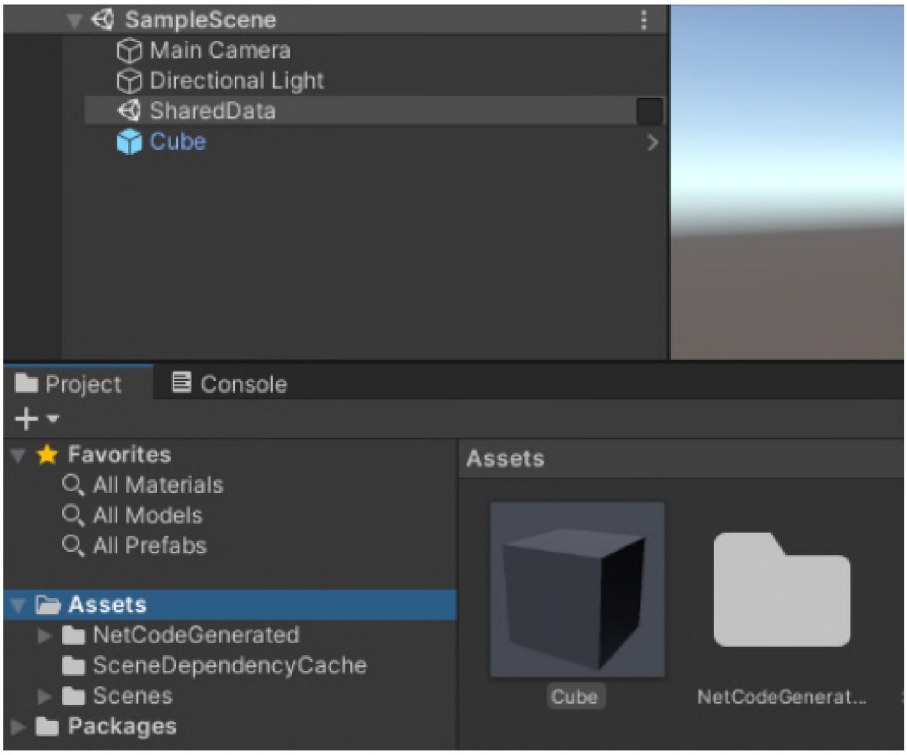

Рисунок 3.13 – Вигляд Prefab у вікні проєкту

Щоб ідентифікувати та синхронізувати Cube Prefab всередині Netcode для Entities, потрібно створити IComponent. Для цього створюємо новий скрипт під назвою CubeAuthoring.cs. Створивши цей компонент, додаємо його до Cube Prefab. Слідом у вікні інспектору додаємо до нього ж компонент Ghost Authoring (рис. 3.14). Після цього Unity автоматично серіалізує компоненти Translation і Rotation. Перш ніж можна буде пересувати куб, потрібно змінити деякі налаштування в щойно доданому компоненті Ghost Authoring Component:

– потрібно поставити прапорець «Has Owner». Це автоматично додає та перевіряє нову властивість під назвою Support Auto Command Target;

– змінюємо режим привида за замовчуванням на передбачений власником. Після чого потрібно встановити член NetworkId компонента Ghost Owner. Це гарантуватиме, що контролер передбачатиме свій власний рух.

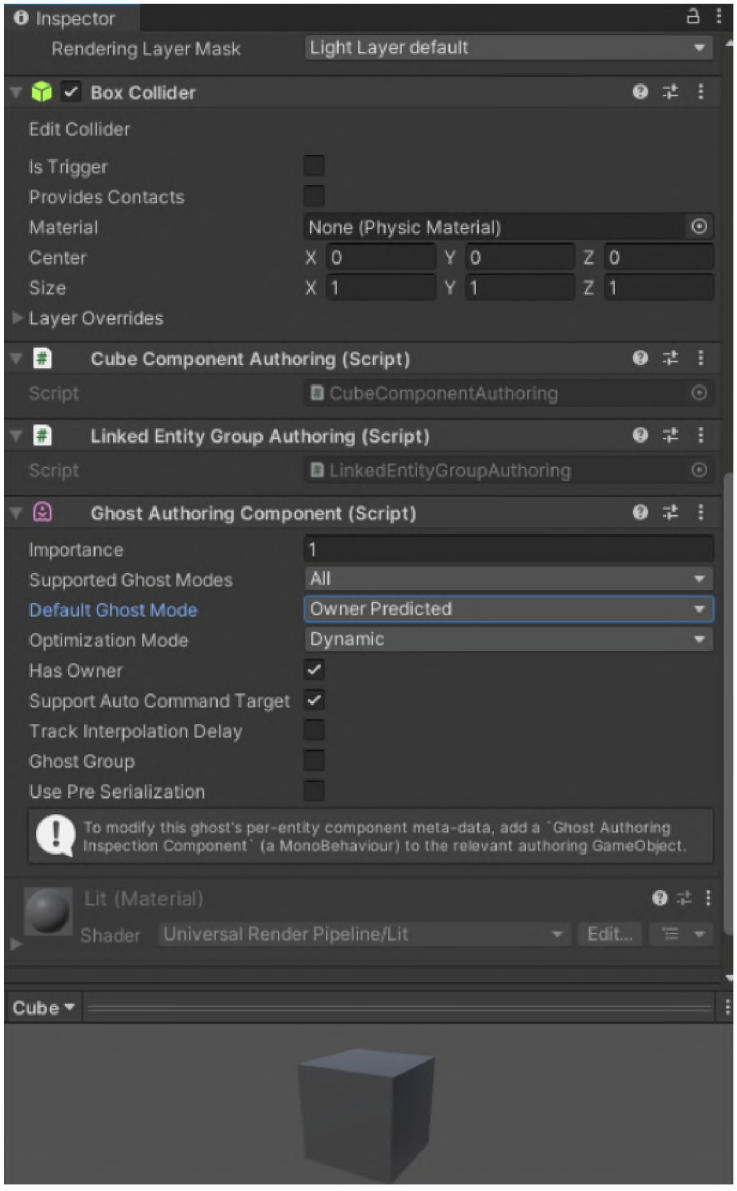

Рисунок 3.14 – Вікно інспектору

Далі потрібно повідомити Netcode для Entities, яких привидів використовувати, зробити це потрібно посилаючись на префаби з підсцени. Для цього створюємо новий компонент для спавнера під назвою CubeSpawnerAuthoring.cs. Після цього додаємо його на ігрове поле. Далі можна

спробувати додати префаб-привід на сцену, тобто побачити, як працює багатокористувацька складова. Для цього необхідно оновити файл GoInGame.cs, який тепер повинен надсилати RPC запит «GoInGame» серверу, коли клієнт будете готовий розпочати синхронізацію. Якщо зараз натиснути запуск симуляції, то можна побачити скопійований куб у грі у відповідній ієрархії сутностей (рис. 3.15).

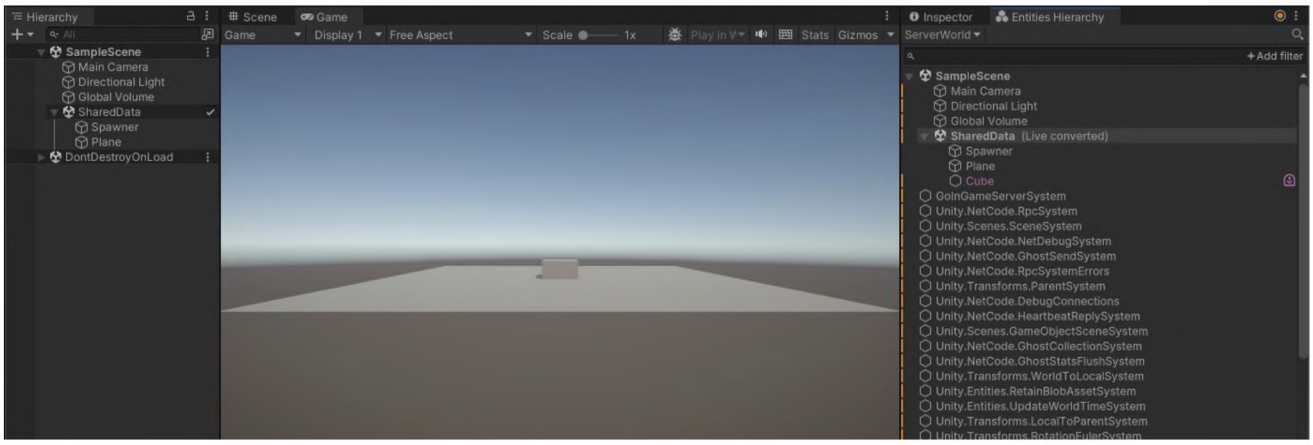

Рисунок 3.15 – Тестування реплікації

3.5 Економічне обґрунтування проєкту

Виробництво гри – це складний і багатоетапний процес, для його успішної реалізації необхідно враховувати різні аспекти. Для організації цих аспектів розробляється різного роду документація, яка є необхідною для розробки та виробництва гри (Таблиця 3.1). Грамотна виконана документація допомагає команді розуміти вимоги до гри, розподіляти завдання, визначати технічні параметри та забезпечує загальне розуміння всієї концепції гри. Вона служить важливим засобом спілкування між членами команди розробників та іншими зацікавленими сторонами, такими як тестувальники чи менеджерами. Її регулярне оновлення та виправлення дозволяють тримати всю команду на одній хвилі та забезпечувати якісний результат у вигляді готової гри.

Таблиця 3.1 – Основні види документації

Види документації	Опис документації
Концепція гри	Опис ідеї гри, її теми та цільової аудиторії. Опис основного геймплею та механіки гри. Сценарій гри та основні події.
Дизайн гри	Геймдизайн-документ, який включає в себе опис всіх аспектів гри, від головних персонажів до рівнів та завдань. Макети інтерфейсу користувача (UI/UX). Художній дизайн (арт-бук, концепції персонажів, місць тощо).
Технічна документація	Технічна специфікація гри з описом архітектури, технологій та інструментів, які будуть використовуватися. Опис алгоритмів та логіки гри. Документація з кодування, яка містить коментарі та пояснення до програмного коду.
Проєктний план	Розклад розробки та виробництва. Розподіл завдань між командою розробників. Бюджет та ресурси для реалізації проєкту.
Тестування та якість	План тестування для різних етапів розробки. Відзвітування про виявлені помилки та їхні виправлення. Документація забезпечення якості (QA).
Маркетинг та PR	Маркетингова стратегія для просування гри. Прес-релізи та інші матеріали для преси. Соціальні мережі та інтернет-сайт гри.
Юридична документація	Ліцензійні угоди та правова документація. Угоди з авторськими правами. Захист інтелектуальної власності.

Виконання етапів зазначених у документації вимагає використання різноманітних програмних засобів та матеріалів (Таблиця 3.2). Правильний інструментарій може значно полегшити та прискорити роботу розробників, а також покращити якість готового продукту. Набір інструментів може змінюватися в залежності від розміру та специфіки самої студії, а також від конкретних вимог та обраного підходу до розробки. Наприклад, стандартним ПЗ для розробки 3D моделей та анімацій в індустрії є Maya, але вона поширюється за схемою підписки, котра є достатньо дорогою для новачків індустрії. Отже вони шукають доступний та потужний засіб для роботи з тривимірною графікою, цим засобом є Blender. Його функціональність та відкритий код роблять його привабливим вибором для широкого кола користувачів.

Таблиця 3.2 – Офіційні ринкові ціни від розробників

Найменування матеріалів	Вартість, $
Програмний пакет Blender	безкоштовно
Програмний пакет Photoshop	604
Інтегроване середовище розробки (IDE) Unity	безкоштовно
Довідкові та навчальні матеріали	200
Сервісні програми	225
Разом	1029

При розрахунку щомісячної заробітної плати, враховується фіксований щомісячний оклад (ставка), кількість робочих днів відповідно до розкладу роботи на поточний місяць та фактична кількість робочих днів, яку працівник відпрацював у зазначений період. Розрахунок виконується за наступною формулою:

$$З = \frac{Т\,м}{Ч\,ф \times Ч\,м} \qquad (3.1)$$

де Тм – місячний посадовий оклад (ставка) працівника, у доларах США;

Чм – час роботи з графіку за цей місяць, днів;

Чф – час, фактично відпрацьований працівником цього місяця, робочих днів.

У таблиці 3.3 наведено детальний перелік фахівців, які є критично важливими для успішної реалізації проєкту протягом двох років. У таблиці подано розгорнуті деталі, включаючи необхідну кількість спеціалістів кожної професії та стандартні місячні зарплати, які служать орієнтиром для визначення бюджету проекту. Важливо враховувати, що вказані заробітні плати можуть змінюватися в залежності від рівня досвіду та кваліфікації найманих фахівців.

Крім того, в контексті компенсаційної політики можуть бути передбачені можливі преміальні виплати, що враховують досягнення та внесок працівників у реалізацію проекту. Ці премії можуть бути встановлені на основі визначених критеріїв, таких як вчасне виконання завдань, якість результатів та інші показники ефективності. Розгляд заробітної плати та можливих премій визначає

об’єктивні фінансові витрати проекту та допомагає забезпечити адекватні умови для залучення та утримання висококваліфікованих фахівців.

Таблиця 3.3 – Відомості про середні місячні зарплати фахівців.

Найменування професії	Кількість осіб	Плата за місяць у доларах США	Плата за місяць у гривнях
Художник з копцепт-арту	1	1461	53738
2D Художник	1	946	34796
Дизайнер	1	1100	40460
3D Моделер	1	1000	36782
Аніматор	1	1200	44138
Програміст	1	1874	68929
Дизайнер рівнів	1	1000	36782
Гейм-дизайнер	1	1300	47816
Разом на місяць		10881	400227

На основі даних із минулої таблиці Соціальні збори для всього персоналу є обов’язковим елементом фінансового плану компанії і включають в себе різноманітні соціальні та страхові внески, які спрямовані на соціальний захист працівників. Соціальні збори на весь персонал розрховуються за формулою:

$$С = Зп \times Н \div 100\% \quad (3.2)$$

де С – сума соціальних відрахувань;

Зп – заробітна плата;

Н – норма соціальних відрахувань.

Відповідно: 10881* 22%/100% = 2393$

На основі задіяної кількості людей у проєкті та їх потреб, визначається необхідна кількість обладнання, яке зазначене у таблиці 3.4. Через масштабність, стилістику та жанр комп’ютерної гри, для виробництва використовують переважно комп’ютери без екзотичного, вузькоорієнтованого обладнання.

Застосування комп’ютерів як основного засобу роботи визначається їхньою універсальністю та спроможністю вирішувати різноманітні завдання, що

виникають під час розробки гри. Це підтримує ефективність та гнучкість у виробництві, враховуючи динамічний та змінний характер проєкту.

Таблиця 3.4 – Вартість комп'ютерного обладнання

Найменування	Ціна 1шт. у доларах США	Ціна 1шт. в грн.	Кількість, шт.
Комп'ютер	1000	36782	8
Комп'ютер периферія	200	7356	8
Графічний планшет	45	1655	5
Інша техніка	1000	36782	1
Разом	2245	82576	-

Комп'ютерну техніку можна віднести лише до основних засобів. Розрахунок амортизації лінійним методом провадиться за формулою:

$$A = C \times H \div 100\% \quad (3.3)$$

де А – Річна сума амортизаційних відрахувань, грн.;

Н – Норма амортизаційних відрахувань, %;

С – Середньорічна вартість основних засобів.

Відповідно: 2245 * 20% / 100% = 449$.

Капітальні витрати при розробці гри включають в себе витрати на придбання тривалого майна або активів, які зазначені у таблиці 3.5. Ці матеріали розраховуються на використовуються протягом тривалого періоду, зазвичай більше одного року. Ці витрати є важливими для студій розробки ігор, оскільки вони визначають основні активи, необхідні для створення та утримання гри. Капітальні витрати становлять важливу частину загального фінансового плану проєкту та враховуються при визначенні бюджету розробки гри. Ці витрати визначають технічну інфраструктуру та активи, які будуть використовуватися протягом тривалого часу та можуть впливати на конкурентоспроможність продукту.

Таблиця 3.5 – Капітальні витрати

Найменування економічних елементів витрат	Сума, $	Сума, грн.
Матеріали, програмне забезпечення	2329	85665
Комп'ютерне обладнання	1245	45793
Разом	3329	122448

Загальний кошторис витрат, представлений у таблиці 3.6, є стратегічним інструментом управління фінансами, який дозволяє планувати, розподіляти та контролювати ресурси під час усього процесу розробки гри. Він включає в себе оцінку всіх можливих витрат, які э необхідними для успішного завершення проєкту.

Кошторис витрат є основою для контролю над фінансами та ефективним управлінням розробкою гри, допомагаючи уникнути непередбачених ситуацій та забезпечити успішне завершення проєкту в рамках визначеного бюджету.

Таблиця 3.6 – Кошторис витрат

Найменування економічних елементів витрат	Сума, $	Сума, грн.
Витрати на оплату праці	10881	400227
Відрахування на соціальні потреби	2393	88019
Амортизація основних фондів	449	16515
Інші витрати (комунальні послуги)	500	18391
Разом:	14023	393717

У розрахунках прийнято середній за червень 2023р. курс долара по відношенню до гривень: 1 $ = 36,78 грн.

Повна вартість проєкту становить: 122448 грн. + 393717 грн. = 516165 грн.

Висновки до розділу 3

У ході даної роботи було поступово успішно досягнуто, поставлені на початку ключові проміжні цілі, які визначили певні етапи розробки гри. Серед них: створення користувальницького інтерфейсу, завдяки якому гравець може переміщуватися у розробленій грі, розробка основних механік автомобілю, реалізація демонстраційної мети гри та розроблений мережевий код до неї.

Завдяки розробленому мережевому коду, гра отримала багатокористувацький режим, котрий потенційно розширяє її аудиторію та підвищує реіграбельність. Це робить проєкт більш соціальним, що в перспективі сприятиме його популярності та тривалому успіху.

У результаті реалізації цих моментів вдалося розробити повноцінний демонстративний проєкт. Напрацювання якого, можуть служити важливим фундаментом для подальшого розвитку та удосконалення гри.

В економічній частині проведено аналіз фінансових аспектів розробки комп'ютерної гри. З урахуванням витрат на розробку, соціальні потреби, а також на амортизацію основних фондів. Ретельний розгляд витрат на персонал, інфраструктуру дозволяє зробити висновок про реалістичність та стійкість економічного підґрунтя проєкту.

ВИСНОВКИ

Під час розробки багатокористувацьких ігор досліджено та розглянуто дві основні мережеві архітектури: однорангову (p2p) та клієнт-серверну. Однорангова архітектура найбільш підходить для сценаріїв використання в невеликих проєктах, де важлива децентралізація, відмовостійкість та економія як матеріальних, так і обчислювальних ресурсів. Тоді як клієнт-серверна архітектура є стандартом для великих проєктів та забезпечує централізоване управління, вона легко-масштабована, має більший потенціал для запобігання шахрайству. Саме вона володіє характеристиками, важливими для змагального проєкту.

Ключовими компонентами мережевих ігор є транспортний протокол (TCP або UDP) та розроблений під них програмний код для оптимізації їх використання та усунення проблем, пов'язаних з ними. Вибір між TCP і UDP залежить від конкретних вимог певного проєкту, де TCP гарантує послідовну доставку пакетів, а UDP забезпечує меншу затримку. Використання готових мережевих бібліотек на основі UDP сприяє швидкій та ефективній розробці багатокористувацької складової гри.

Забезпечення інтегрованої безпеки та надійності системи обміну даними з використанням протоколу UDP вимагає розробку спеціальних алгоритмів, спрямованих на виявлення та виправлення можливих помилок. Зокрема, це включає в себе врахування ситуацій порушення порядку отримання пакетів, виявлення та ігнорування дубльованих пакетів інформації, а також зменшення вразливості до втрати пакетів. Розробка ефективних алгоритмів дозволяє забезпечити надійність та цілісність передачі даних, зробивши обмін інформацією стійким до різноманітних помилок, які можуть виникнути під час передачі через мережу, що в кінцевому результаті гарантує належний користувацький досвід.

Щодо обміну даними між клієнтами та сервером важливо враховувати обмеження пропускної здатності. Через це існує необхідність у використанні різних алгоритмів компресії та відсіканні актуальної інформації для оптимізації обміну даними. Введення унікальних порядкових номерів та вказання останніх

отриманих пакетів допомагає у вирішенні проблеми втрати, зміни порядку та дублювання пакетів.

З метою підтримки візуальної плавності гри створюються різні алгоритми, які працюють на стороні клієнта. Серед них є алгоритм інтерполяції, що згладжує рух інших об'єктів, створюючи проміжні дані на основі раніше отриманих від серверу пакетів. Також для забезпечення плавного руху персонажу, яким керує гравець, використовується спеціальний алгоритм, що дозволяє виконувати певні рухи без дозволу серверу. А для синхронізації дій гравців відбувається компенсація затримки вже на стороні сервера, яка на основі даних із минулого керує поточними та прогнозує майбутнє. Тим самим зменшує вимогливість до якості інтернету користувача та урівнює гравців із різною якістю з'єднання.

Успішна реалізація багатокористувацької ігрової мережі вимагає уважного підходу до вибору архітектури та компонентів, оптимізації обміну даними, а також упровадження заходів для забезпечення стійкості та ефективності мережевої взаємодії.

В економічній частині висвітлено аспекти фінансування розробки комп'ютерної гри, включаючи витрати на створення, врахування соціальних потреб і амортизацію основних активів. Досконалий розгляд витрат на персонал та інфраструктуру, який дозволяє зробити висновок про реалістичність та стійкість фінансового фундаменту проєкту.

Достовірність отриманих у ході роботи результатів обумовлюється шляхом застосування загальноприйнятих методів наукових досліджень. А також завдяки використанню основних принципів нормативної теорії ухвалення рішень та відповідного математичного апарату.