Науковий вісник НЛТУ України
Scientific Bulletin of UNFU
ISSN 1994-7836 (print)
ISSN 2519-2477 (online)
https://doi.org/10.36930/40340412
Article received 09.05.2024
Article accepted 23.05.2024
UDC 004.[77+92]
Correspondence author: L. M. Zhuravchak liubov.m.zhuravchak@lpnu.ua

Л. М. Журавчак, В. Р. Сеньків, Н. Р. Сеньків
Національний університет "Львівська політехніка", м. Львів, Україна

РЕАЛІЗАЦІЯ БАГАТОКОРИСТУВАЦЬКОЇ ГРИ ЗАСОБАМИ РУШІЯ UNREAL ENGINE 5

Запропоновано новий підхід до реалізації багатокористувацьких ігор, який полягає у використанні засобів Unreal Engine для ефективного передавання даних мережею Інтернет, мінімізації затримки їх передавання та синхронізації станів гравців у спільному ігровому світі. Реалізація гри використовує алгоритми компенсації затримки пакетів, мережевих коливань та втрати пакетів і компенсацію на рівні гри у вигляді схем передбачення позиції. Для проектування кооперативної гри обрано архітектуру *listen server*, враховуючи дві її ключові переваги – зменшення витрат на підтримку гри внаслідок уникнення потреби оренди ігрових серверів, оскільки самі гравці почергово виступають серверами, та усунення затримки пакетів внаслідок локального розміщення сервера і клієнтів. Для синхронізації ігрових станів використано два механізми: реплікацію з метою постійної підтримки на клієнтах авторитетного стану гри, який відповідає тому, що є на сервері, та *RPC* (англ. *Remote Procedure Calls*) для одноразової передачі інформації. Проведено обчислювальний експеримент щодо локального передбачення переміщення персонажа для перевірки якості роботи алгоритмів передбачення. Реалізовано локальну симуляцію для зброї, яка відображає гравцю візуальні ефекти, та фізичну симуляцію для замаскування затримки у мережі. Інтегровано онлайн сервіси від Epic Games для під'єднання до ігрових світів інших гравців мережею Інтернет. Унаслідок тестування встановлено, що реалізована система передбачення руху забезпечила плавність переміщень персонажів і за достатньо великих затримок передавання пакетів даних у мережі (500 мс), і за реально непередбачуваних (1 с). Запропонований підхід має високу точність та ефективність у вирішенні завдань, які виникають у процесі розроблення багатокористувацьких ігор, внаслідок використання повторного відправлення пакетів для уникнення їхньої втрати та локальної симуляції й передбачення для компенсації затримки мережі. Майбутні дослідження полягатимуть в уточненні алгоритму передбачення для забезпечення підтримки плавного ігрового досвіду при використанні складних режимів руху, таких як плавання, скелелазіння, ковзання, рух канатом тощо.

Ключові слова: мінімізація затримки передавання пакетів; передбачення руху персонажа; локальна симуляція; компенсація затримки мережі; синхронізація ігрових станів гравців.

Вступ / Introduction

Індустрія відеоігор або індустрія інтерактивних розваг (англ. *Video Game Industry* або *Interactive Entertainment Industry*) – це високоінноваційний сектор, який швидко зростає останнім часом і став домінантним в індустрії розваг із значними економічними та соціальними наслідками [4]. Особливої популярності набрали багатокористувацькі ігри, які дають змогу кільком гравцям бачити дії один одного та співпрацювати в тому самому віртуальному світі в реальному часі.

Розроблення багатокористувацьких ігор має багато своїх особливостей, оскільки вимагає побудови програмної системи реального часу з передаванням і синхронізацією даних мережею Інтернет. До особливостей системи реального часу, серед яких можна навести мінімальний час її відгуку, додаються ще такі вимоги: мінімальна затримка передачі інформації через мережу, швидке доставлення інформації до інших гравців, синхронізація ігрового світу між усіма гравцями.

Реалізація багатокористувацьких ігор є тривалим і складним процесом внаслідок потреби поєднання багатьох програмних систем та вирішення нетривіальних завдань, пов'язаних з особливостями роботи ігрових механік у багатокористувацькому середовищі. Зважаючи на ці чинники, можна значно заощадити час на розроблення багатокористувацької гри, використавши для цього один з наявних ігрових рушіїв з потужною мережевою системою, зокрема, Unreal Engine, який останнім часом отримав значні оновлення та набув великої популярності.

Об'єкт дослідження – розроблення багатокористувацьких ігор засобами ігрового рушія Unreal Engine 5.

Предмет дослідження – методи та способи забезпечення плавного ігрового досвіду в багатокористувацьких іграх реального часу, що дасть змогу компенсувати затримки мережі та покращити якість процесу гри.

Мета роботи – розробити багатокористувацьку ко-

Інформація про авторів:
Журавчак Любов Михайлівна, д-р техн. наук, професор, кафедра програмного забезпечення.
Email: liubov.m.zhuravchak@lpnu.ua; https://orsid.org/0000-0002-1444-5882
Сеньків Віталій Романович, бакалавр, кафедра програмного забезпечення.
Email: vitalii.senkiv.pz.2020@lpnu.ua; https://orsid.org/0009-0001-4322-8182
Сеньків Назарій Романович, бакалавр, кафедра програмного забезпечення.
Email: nazarii.senkiv.pz.2020@lpnu.ua; https://orsid.org/0009-0004-4561-5006

Цитування за ДСТУ: Журавчак Л. М., Сеньків В. Р., Сеньків Н. Р. Реалізація багатокористувацької гри засобами рушія Unreal Engine 5. Науковий вісник НЛТУ України. 2024, т. 34, № 4. С. 93–101.
Citation APA: Zhuravchak, L. M., Senkiv, V. R., & Senkiv, N. R. (2024). Implementation of a multiplayer game using Unreal Engine 5. *Scientific Bulletin of UNFU*, 34(4), 93–101. https://doi.org/10.36930/40340412

оперативну пригодницьку гру як програмну систему реального часу для демонстрації можливих способів вирішення ключових завдань, пов'язаних з передаванням і синхронізацією даних мережею Інтернет.

Для досягнення зазначеної мети визначено такі основні *завдання дослідження*:

1) проаналізувати наявні літературні джерела, інтернет-ресурси та відеоконференції щодо визначення методів і способів реалізації особливостей роботи ігрових механік під час розроблення багатокористувацьких ігор;
2) дослідити основний функціонал рушія та інструменти, які він надає, для забезпечення ефективного передавання даних, мінімізації затримки такого передавання та під'єднання клієнтів до спільного ігрового світу мережею Інтернет;
3) спроектувати архітектуру для багатокористувацької кооперативної пригодницької гри, що забезпечить простий та ефективний спосіб обміну даними між гравцями;
4) розробити багатокористувацьку кооперативну пригодницьку гру з використанням технологій, які надає ігровий рушій Unreal Engine 5, що уможливить плавний досвід багатокористувацької гри внаслідок застосування алгоритмів передбачення та локальної симуляції.

Аналіз останніх досліджень та публікацій. У роботі [8] автори Steven Lee та Rocky Chang досліджують тему компенсації мережевої затримки в іграх, щоб покращити ігровий досвід у іграх-шутерах та мінімізувати проблеми, які виникають через тривалий час передачі пакетів. Вони аналізують проблему "пострілів із укриття", яка полягає у тому, що через велику затримку у мережі гравці можуть отримувати пошкодження навіть тоді, коли вони сховалися за укриттям. Для вирішення цієї проблеми автори розробили покращений спосіб компенсації затримки пакетів, який враховує стан мережі та розмір затримки, щоб забезпечити плавний досвід навіть для гравців із великою затримкою. Удосконалений підхід полягає у тому, щоб додати валідацію із сторони клієнта, оскільки позиція на сервері може бути застарілою та не відповідати дійсності. Клієнт надсилає серверу повідомлення, яке підтверджує або спростовує попадання, а також повідомляє свою поточну позицію.

У роботі [10] автори розглядають проблему розсинхронізації клієнтів та сервера внаслідок затримки у мережі та втрати пакетів і пропонують новий метод компенсації затримки мережі на рівні клієнта, який полягає у передбаченні руху інших гравців. Цей метод передбачення базується на поєднанні лінійної регресії та "підсилювальному навчанні". Основна ідея такого поєднання полягає у тому, щоб точно обчислювати наступну позицію гравця за допомогою передбачення, підсиленого машинним навчанням на основі попередніх синхронізованих станів гравців. Згідно з результатами дослідження такий підхід допоміг зменшити кількість помилок, спричинених мережевими затримками більше ніж у десять разів. Для цього автори виміряли середнє зниження влучності гравців, яке при затримці у 100 мс становило 0.0039 при використанні їхнього підходу та 0.55 без компенсації затримки.

Згідно з даними роботи [14] основною проблемою мережевих багатокористувацьких ігор є затримка мережевого передавання. Автори статті ілюструють вплив цієї проблеми на процес гри прикладом багатокористувацької гонки, де через затримку передавання інформації гравець може бачити свій автомобіль попереду автомобіля опонента, хоча насправді вони можуть бути на одному рівні. На таке відображення впливають застарілі дані про позицію автомобіля опонента. Отож, велика затримка може призводити до абсурдних ситуацій під час ігрового процесу. На жаль, вона спричинена фізичним рівнем передачі даних і не може бути повністю усунута. Автори цієї статті також описують підхід до боротьби зі затримкою, суть якого полягає у введенні затримки перед виконанням певної дії. Така затримка потрібна, щоб виділити час на пересилання мережею даних про позицію та час цієї позиції. Але, згідно з результатами аналізу, такий підхід не дуже добре працює зі значною затримкою (більше 200 мс), оскільки все-одно допускає достатньо великі відхилення від справжньої позиції.

Для уточнення боротьби зі затримкою та її компенсації у роботі [13] запропоновано ввести різноманітні схеми передбачення, які обчислюватимуть майбутні позиції гравців і які можна поділити на два типи: передбачення позиції та передбачення вводу. Перші пробують визначити наступну позицію, враховуючи попереднє переміщення. Схема передбачення вводу пробує вгадати наступний ввід з клавіатури чи джойстика на основі попереднього вводу, а наступна позиція обчислюється на основі передбаченого вводу. Для компенсації мережевої затримки позицію інших гравців обчислюють як інтерполяцію між двома позиціями: передбаченою та отриманою через мережу, що дає змогу зменшити розбіжність позицій між клієнтами.

У роботі [2] автори описують проблему впливу затримок мережі на рух персонажів у багатокористувацьких іграх та для її вирішення пропонують удосконалити алгоритм передбачення позиції гравців *Dead Reckoning*. Цей алгоритм є стандартним алгоритмом передбачення, базованим на законах Ньютона та припущенні про лінійну природу руху гравців. Автори вважають його не дуже вдалим для передбачення позиції в сучасних іграх, де гравець може змінити напрямок руху за власним бажанням у будь-який час, тому пропонують свою покращену версію вказаного алгоритму під назвою *Smart Reckoning*. Удосконалений алгоритм засновано на підході з застосуванням машинного навчання, що у разі нелінійного руху застосовує натреновану модель для передбачення позиції гравця.

У дослідженні [7] автор також намагається компенсувати мережеві затримки внаслідок передбачення позиції. Порівняно різні алгоритми передбачення (англ. *Input Prediction*, *Closest Average*, *Dead Reckoning*, *Closest Last Lap* та *DR/Input Prediction*) за різних значень мережевої затримки (150 мс, 300 мс, 450 мс, 600 мс та 750 мс) на основі двовимірної гри-гонки, при цьому виміряно похибку позиції автомобіля. Згідно з результатами досліджень алгоритм передбачення вводу *Input Prediction* показав себе найкраще.

Матеріали та методи дослідження. Під час аналізу методів вирішення поставлених завдань і способів їхньої практичної реалізації використано документацію ігрового рушія Unreal Engine 5 [12], компендіум розроблення багатокористувацьких ігор [11] та початковий код ігрового рушія. Для дослідження можливості під'єднання клієнтів до спільного ігрового світу застосовано документацію Epic Online Services [3]. Для перевірки якості роботи багатокористувацької гри в умовах різної затримки пакетів використано інструменти введення штучної затримки та втрати пакетів [18].

Результати дослідження та їх обговорення / Research results and their discussion

Огляд основних проблем багатокористувацьких ігор, спричинених мережею. У книзі "Networking and online games" [5] автори виокремлюють такі ключові проблеми у багатокористувацьких іграх: мережеві затримки, коливання мережі та втрату пакетів. "Незалежно від жанру гри, реалізм онлайн-гри залежить від того, наскільки добре основна мережа дає змогу учасникам гри спілкуватися вчасно та передбачувано", – стверджують вони. Мережеві затримки – це час, який витрачають на передавання пакета мережею Інтернет від одного учасника до іншого. Коливання мережі – це варіація мережевої затримки від одного пакета до іншого, тобто час затримки пакета не є сталим, а може змінюватися для кожного пакета, причому ці зміни можуть бути як плавними, так і різкими, залежно від стану мережі. Втрата пакетів – це ефект, коли пакет через певні умови мережі не досягає місця призначення та губиться десь по дорозі. Втрату пакетів визначають як відношення втрачених пакетів до надісланих та обчислюють за формулою

$$L = P_L / P_S,$$

де: L – коефіцієнт втрати пакетів; P_L – кількість втрачених пакетів; P_S – кількість надісланих пакетів.

Автори наголошують [5], що ці три проблеми мають значний негативний вплив на процес гри. Затримка мережі ламає відчуття того, що гра проходить у реальному часі, оскільки вона призводить до того, що гра не може одразу дати відповідь на дії користувача. Тому гравці не можуть вчасно реагувати на події, що може бути критичним для певних видів ігор. Велике коливання мережі може призвести до того, що рухи персонажів й інші дії можуть бути нестабільними, бо пакети приходять із різною затримкою. Втрата пакетів може призводити до пропусків певних ігрових станів, що є неприйнятним.

Серед основних причин виникнення цих проблем автори виділяють такі: обмежений час серіалізації, фізичне обмеження передавання даних кабелями, затримки через перевантаження мережі та зміна маршрутів передавання даних. Важливо зазначити, що внаслідок фізичного обмеження передавання даних кабелями виникають значні затримки, якщо два гравці знаходяться далеко один від одного. Так, наприклад, час, який потрібний електричному сигналу, щоб дійти від Лос-Анджелеса до Сіднея, становить близько 40 мс. Формула для обчислення затримки електричного сигналу є такою:

$$T = d / 300,$$

де: T – затримка електричного сигналу у мс; d – відстань до точки призначення у кілометрах, 300 – кількість км, яку пройде світло за 1 мс.

Зрозуміло, що довжина шляху прямо пропорційно пов'язана з імовірністю втрати пакета або його затримки на завантаженому вузлі. Коливання мережі спричинено двома чинниками: пакети можуть проходити різні маршрути, хоча зміна маршрутів відбувається нечасто, та пропускна здатність маршруту постійно змінюється залежно від навантаження маршрутизаторів й інших мережевих пристроїв.

Автори звертають увагу на те, що комп'ютерні ігри потребують більшого контролю над затримкою пакетів, мережевими коливаннями та втратою пакетів, аніж, наприклад, електронна пошта. У книзі розрізняють два способи вирішення цих трьох проблем. Перший – мережевий спосіб, другий – компенсація на рівні гри. Суть мережевого способу полягає в тому, щоб оптимізувати передавання пакетів на низькому рівні, наприклад, обробляти пакети за їхнім пріоритетом, і для систем реального часу цей пріоритет має бути вищим. Проте мережеві підходи мають свої недоліки, зокрема, маршрутизатор не може виявляти пакети з підробленими пріоритетами.

Загальна архітектура мережевої складової гри. Першим завданням, яке виникає під час розроблення багатокористувацької гри, є створення правильної та гнучкої архітектури, яка дасть змогу просто та ефективно обмінюватися даними. Вона повинна враховувати всі особливості взаємодії між користувачами у реальному часі.

Досліджено різні види архітектур, які використовують у багатокористувацьких іграх. Є два підходи до реалізації, перший – клієнт-серверна архітектура, другий – так звана *peer-to-peer* архітектура. Остання базується на тому, що всі гравці є рівноправними та напряму обмінюються даними між собою. На відміну від неї клієнт-серверна архітектура передбачає наявність сервера, який є авторитетним та має більше прав, ніж клієнти.

Загалом більшість багатокористувацьких ігор реального часу використовують клієнт-серверну архітектуру. Це зумовлено необхідністю авторитетного джерела, відносно якого синхронізують всі інші клієнти. Така архітектура дає змогу забезпечити однакове виконання процесів у всіх гравців і ускладнює можливості для шахрайювання. Суть клієнт-серверної архітектури полягає в поділі примірників програми на сервер та клієнтів. Усі клієнти обмінюються даними з сервером. За потреби обміну даними між клієнтами він теж відбувається через сервер, тобто клієнт надсилає дані на сервер, а той пересилає їх до іншого клієнта.

Вивчення можливостей, які надає ігровий рушій Unreal Engine [12], показало, що ігровий рушій використовує клієнт-серверну архітектуру, при цьому він дає змогу використання двох підвидів, а саме *dedicated server* та *listen server*. Суть архітектури *dedicated server* полягає в тому, що сервер розгортається окремим процесом, який не має графічного відображення та відповідає тільки за централізоване оброблення ігрової логіки та синхронізацію ігрових станів між клієнтами. Суть архітектури *listen server полягає* в поділі примірників програми на клієнтів і сервер. Однак особливістю цього підвиду порівняно із звичайним клієнт-сервером є те, що сервер також обробляє дані з гри, тобто він одночасно виконує і роль клієнта, і серверну логіку з оброблення та надсилання пакетів даних.

Під час реалізації та проектування кооперативної гри було використано архітектуру *listen server* (рис. 1).

Особливістю реалізації цієї архітектури в ігровому рушії Unreal Engine є те, що сервером може стати будь-хто з клієнтів, тобто він не є відокремленим, при цьому після завершення ігрової сесії і на початку наступної сервером може виступати вже інший клієнт, а колишній сервер буде тільки клієнтом. Такий підхід має дві ключові переваги для кооперативних ігор. Перша полягає в тому, що немає потреби в оренді ігрових серверів, оскільки самі гравці виступають серверами, що значно зменшує витрати на підтримку гри.

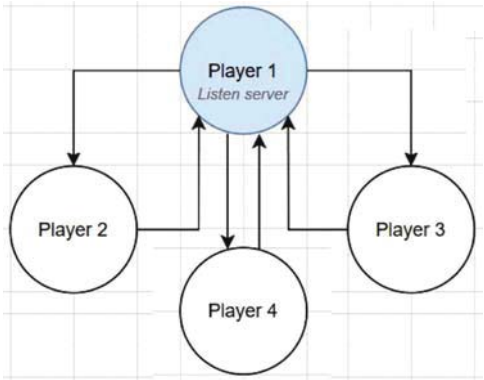

Рис. 1. Архітектура "listen server" / Listen server architecture

Другою є вирішення певною мірою проблеми затримки пакетів внаслідок локального розміщення сервера і клієнтів. Наприклад, якщо деяка група друзів захоче пограти у гру, то швидше за все вони будуть перебувати в одному регіоні, що зменшує час затримки порівняно із віддаленим сервером, найближчий із яких може перебувати в іншому географічному регіоні. Також архітектура *listen server* придатна для реалізації кооперативної гри, адже будь-хто з команди може виступати в ролі сервера.

Способи синхронізації ігрових станів з клієнтами та вирішення проблеми втрати пакетів. Внаслідок дослідження можливостей рушія, описаних у роботах [11, 12], встановлено, що для синхронізації ігрових станів використовують такі механізми як реплікація [16] та *RPC* (англ. *Remote Procedure Calls*) [15]. Реплікація – це спосіб синхронізації ігрового стану сервера та клієнтів внаслідок надсилання пакетів, які містять цей стан. В ігровому рушії Unreal Engine реплікація працює так, що інформацію відправляють тільки від сервера до клієнтів, щоб вони постійно підтримували авторитетний стан гри, який їм продиктував сервер. Реплікація відбувається постійно, тобто сервер кожен раз перевіряє, чи ігровий стан не змінився, і якщо так, то відправляє нові пакети. Основна суть реплікації полягає в тому, щоб постійно підтримувати на клієнтах авторитетний стан гри, який відповідає тому, що є на сервері.

У випадках, коли потрібно передавати дані із клієнтів на сервер або на інші клієнти, використовують *RPC*. На відміну від реплікації, яка відбувається постійно, *RPC* – це спосіб одноразової передачі інформації. В ігровому рушії Unreal Engine всі *RPC* поділяють на три категорії: *Server, Client* та *Multicast*.

Server RPC використовують для того, щоб передати інформацію із клієнта на сервер. Це корисно у випадках, якщо потрібно передати інформацію, яка може міститися тільки на клієнті, наприклад, інформацію про ввід користувача. Такі *RPC* використовують для того, щоб виконувати якісь дії, які мають бути перевірені сервером, наприклад, перевірка того, чи куля влучила у ціль, чи ні. Є правило: чим більше дій виконується із залученням серверної перевірки, тим складніше шахраювати у самій грі.

Client RPC – це спосіб передавання інформації із сервера на клієнта, його найчастіше використовують для того, щоб сповістити конкретного клієнта про певну подію, яка відбулася. Такий вид *RPC* можна використовувати як спосіб оптимізації синхронізації станів, що рідко змінюються.

Multicast RPC – це спосіб передавання даних від сервера до всіх клієнтів. Цей вид *RPC* подібний до реплікації, проте використовується у випадках, коли потрібно сповістити всім про те, що відбулася певна подія. Такі *RPC* використовують у косметичних цілях для сповіщення про події, що рідко відбуваються.

Проблему втрати пакетів, зазвичай, можна вирішити за допомогою пакетів-підтверджень, які сповіщають сервер про те, що відправлений пакет успішно дійшов. У випадку, якщо сервер не отримує відповідь про отримання пакета від клієнта у зазначеному проміжку часу, сервер ще раз намагається відправити пакет. В ігровому рушії Unreal Engine обидва ці підходи розроблено із урахуванням проблеми втрати пакетів. Реплікація вирішує проблему втрати пакетів, оскільки постійно намагається відтворити стан сервера на клієнтах, і якщо пакет не дійшов, то через певний проміжок часу спроба повторюється. При використанні механізму *RPC* є дві категорії: *Reliable* та *Unreliable*, які визначають, як обробляти втрату пакетів. У випадку *Reliable* при втраті пакета буде здійснено декілька спроб ще раз його відправити, і якщо вони всі будуть неуспішними, то зв'язок буде вважатися втраченим і цього клієнта буде від'єднано від сервера. Зазвичай, такі *RPC* використовують для того, щоб сповістити про важливі події, які не можна ігнорувати. У випадку *Unreliable* при втраті пакета жодного обробляння не буде. Такі *RPC,* в основному, використовують для неважливих подій, зазвичай косметичних ефектів, які не мають значного впливу на ігровий процес (рис. 2).

У розробленій грі використано обидва способи передавання даних. Так, наприклад, для правильної поведінки вогнепальної зброї у багатокористувацькому режимі використано категорію *Server RPC*, яка сповіщає сервер про постріл, щоб той зміг перевірити влучання та наніс пошкодження ворогу. Також використано категорію *Multicast RPC*, яка відображає візуальні ефекти пострілу для кожного клієнта. Відповідно *Server RPC* є *Reliable*, бо відображає важливу подію, а *Multicast RPC* є *Unreliable*, бо гравцям не дуже важливо, якщо через поганий стан мережі не з'явиться один візуальний ефект.

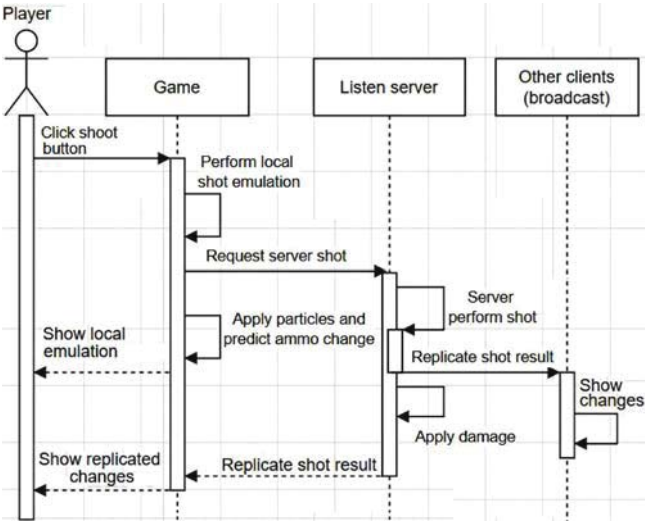

Рис. 2. Діаграма послідовності процесу пострілу / Shooting process sequence diagram

Згадані вище події використовують *RPC*, оскільки це події, які відбуваються відносно рідко, бо гравець не стріляє постійно, і є великі проміжки часу, коли він не буде використовувати зброю. Кількість здоров'я ігрового персонажа використовує реплікацію, бо важливо цю характеристику синхронізувати із станом гравця на сер-

вері. Принцип роботи процедури синхронізації відображено на рис. 2.

Локальна симуляція та передбачення. Як вже було вияснено з аналізу робіт [13, 14], для вирішення деяких проблем, пов'язаних із затримками пакетів у мережі, можна використовувати різноманітні алгоритми передбачення. Зокрема, передбачення часто застосовують для компенсації затримки позиції гравця.

Логіка алгоритмів передбачення базується на тому, що напрямок руху та швидкість змінюється не дуже часто, тому можна екстраполювати позицію об'єктів на основі швидкості та напрямку руху або на основі припущення про введення елементів керування. Як вже було згадано раніше, такі алгоритми можуть передбачати позицію безпосередньо або передбачати введення гравця, а вже на основі передбаченого введення визначати наступну позицію. Формула алгоритму передбачення позиції на основі швидкості є такою:

$$\overline{L_p} = P(\overline{L_0}, \overline{V}, t_0, t_1),$$

де: $\overline{L_p}$ – передбачена позиція гравця; P – функція передбачення позиції на основі швидкості; $\overline{L_0}$ – попередня позиція гравця; $\overline{V}$ – останній вектор швидкості гравця, отриманий з сервера; t_0, t_1 – час останньої синхронізації з сервером та час, для якого здійснюється передбачення, відповідно.

Найпростішими функціями передбачення є такі:

$$P = \overline{L_0} \text{ та } P = \overline{L_0} + \overline{V}(t_1 - t_0).$$

Перша з них робить припущення, що гравець перестає рухатись, але зазвичай це не відповідає дійсності, і таку функцію ніхто не використовує. Друга функція припускає, що гравець продовжує рухатись в тому самому напрямку зі сталою швидкістю. Цей варіант набагато кращий за попередній, але не враховує можливості пришвидшення. Для уточнення можливості передбачення пришвидшеного руху можна в функції передбачення врахувати ще й пришвидшення:

$$\overline{L_p} = P(\overline{L_0}, \overline{V}, t_0, t_1, \overline{a}),$$

де $\overline{a}$ – останній вектор пришвидшення гравця, отриманий із сервера.

Функцію передбачення для фізичних тіл, зокрема, передбачення гравітаційного впливу на гравця, часто вибирають у вигляді:

$$P = \overline{L_0} + \overline{V}(t_1 - t_0) + \frac{\overline{a}}{2}(t_1 - t_0)^2.$$

Такі алгоритми частково борються із втратою пакетів. Наприклад, гравець просто йде вперед, а один із пакетів з новою позицією втратився під час доставки. Без використання алгоритму передбачення позиції відбудеться зупинка персонажа, а з отриманням наступного пакета персонаж "перестрибне" зразу на два кроки замість одного. Такі "перестрибування" негативно впливають на ігровий процес, особливо в іграх, подібних до гонок, описаних у роботі [14], де позиція відіграє ключову роль. Алгоритми передбачення позиції дають змогу уникнути більшої кількості таких різких переміщень, оскільки навіть за відсутності нових даних із сервера позиція персонажа екстраполюватиметься на основі його попередньої швидкості та напрямку руху.

Алгоритми передбачення використовують у компоненті руху гравця [17] для забезпечення плавності в різних режимах руху з урахуванням затримки та втрати даних при передаванні. Для розуміння, як працює система передбачення та виправлення руху персонажа в ігровому рушії Unreal Engine, потрібно усвідомлювати, що один і той самий персонаж у багатокористувацькій грі залежно від середовища може мати одну з ролей: *Autonomous Proxy* (персонаж на комп'ютері гравця, який ним керує), *Authority* (копія персонажа гравця на сервері) та *Simulated Proxy* (копія персонажа гравця на іншому клієнті).

Алгоритм процесу синхронізації із передбаченням позиції через компонент руху гравця є таким.

Крок 1. Спочатку гравець, у якого роль *Autonomous Proxy*, виконує рух на основі фізичного введення гравця.

Крок 2. Відбувається збереження даних про рух у поточний момент часу, ці дані компресують для уникнення повторів і зменшення їхньої кількості, потім пересилають на сервер.

Крок 3. Відбувається надсилання на сервер збережених даних про рух та кінцеву позицію на клієнті.

Крок 4. Сервер на основі отриманих даних повторює такий самий рух, але з позиції гравця на сервері.

Крок 5. Якщо результат повторення руху збігається з кінцевою позицією *Autonomous Proxy* клієнта, до клієнта надсилається підтвердження правильності його руху.

Крок 6. Якщо обчислена на сервері позиція відрізняється від позиції *Autonomous Proxy*, до клієнта надсилається повідомлення з даними про виправлення позиції.

Крок 7. *Autonomous Proxy* відтворює ті самі рухи, починаючи зі своєї позиції на сервері, а не з локальної.

Крок 8. Після цього сервер реплікує дані про позицію гравця всім клієнтам, які мають роль *Simulated Proxy*.

Крок 9. *Simulated Proxy* застосовує репліковані дані про рух з урахуванням передбачень та механізму мережевого згладжування.

Система передбачень руху в ігровому рушії Unreal Engine виконується для гравців, які мають роль *Simulated Proxy*, тобто для подання персонажів інших гравців. Алгоритм загалом працює подібно до описаного вище принципу передбачення позиції на основі швидкості та напрямку.

За аналогією з передбаченням руху можна використати ще один подібний підхід, який дає змогу компенсувати затримку пакетів. Ним є локальна симуляція, коли частина ігрової логіки виконується локально паралельно із запитом до сервера. Такий підхід дає змогу зробити вигляд повної відсутності затримки і базується на тому, що з дуже високою ймовірністю результат дії, виконаної локально, буде збігатись з результатом сервера. Принцип роботи такий: клієнт локально виконує якусь дію і одночасно з цим відправляє на сервер запит на здійснення цієї дії. Поки відповідь із сервера ще не прийшла, гравець бачить результат локальної симуляції. Коли сервер виконав дію, він синхронізує стан клієнта зі своїм, і якщо локальна симуляція була вдалою, то ці стани будуть однаковими, що забезпечує відчуття відсутності затримки.

У розробленій кооперативній грі такий підхід використовується для стрільби з вогнепальної зброї. На рис. 2 можна побачити принцип роботи локальної симуляції, а на рис. 3 – синхронізацію стрільби у грі. При стрільбі частина логіки (відображення візуальних ефектів, фізичний вплив на об'єкти та зменшення кількості

набоїв) виконується локально без затримки. Паралельно з цим на сервері виконується аналогічний постріл, який синхронізується між усіма клієнтами.

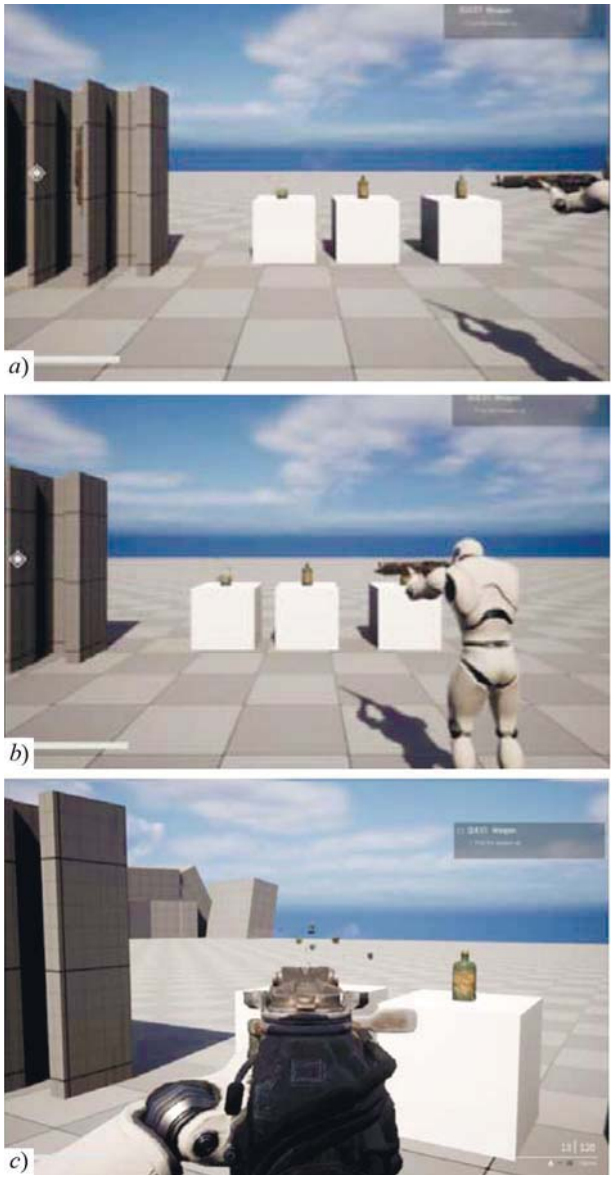

Рис. 3. Синхронізація стрільби між клієнтами / Shooting synchronization between clients

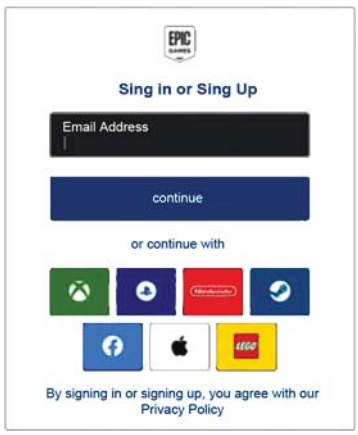

Рис. 4. Авторизація в EOS / Authorization using EOS

Серед методів вирішення завдання під'єднання до ігрового світу з використанням мережі Інтернет, як найпрактичніший та найфункціональніший, вибрано використання онлайн-сервісів, які надають різні компанії. Згідно з документацією ігрового рушія Unreal Engine [3], він надає підсистему OSS (англ. *Online SubSystem*), яка є набором інтерфейсів для імплементації різних багатокористувацьких функцій. Використано інтерфейси для авторизації користувача (рис. 4) та для роботи з ігровими сесіями, на основі останнього реалізовано функціональність пошуку (рис. 5) та приєднання до ігрової сесії (рис. 6).

Рис. 5. Пошук ігрових сесій та під'єднання до них через EOS / Searching for game sessions and connecting to them using EOS

Рис. 6. Два гравці в спільному ігровому світі (вигляд від першої особи першого гравця) / Two players in a common game world (first-person view of the first player)

Такі інтерфейси реалізують під конкретну платформу, яка надає свої сервіси. Після аналізу різних імплементацій цієї системи, зокрема Steam та EOS (англ. *Epic Online Services*), обрано останню, оскільки вона реалізує усі можливості системи та надає простий доступ до їхнього використання та відлагодження.

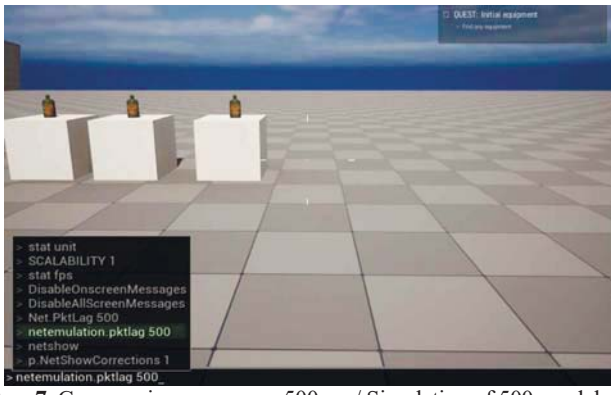

Рис. 7. Симуляція затримки в 500 мс / Simulation of 500 ms delay

Досліджено якість роботи системи передбачення руху персонажа при мережевій затримці в 500 мс. Для цього використано дві команди: *NetEmulation.PktLag 500* (рис. 7), яка симулює затримку пакетів 500 мс, та *p. NetShowCorrection 1* (рис. 8), яка вмикає візуалізацію синхронізації сервером позиції клієнта. Використання цих команд дає змогу побачити частоту розсинхронізації клієнта за різних дій.

Проведено два тести із штучною затримкою пакетів 500 мс: перевірку руху на рівній поверхні та при зіткненні з перешкодами. Кожен тест тривав близько 600 с і відбувався з використанням двох гравців. Внаслідок

виконання першого тесту не виявлено жодного виправлення, що свідчить про якісні алгоритми передбачення. При другому тесті виявлено тільки одне виправлення при зіткненні капсули гравця з ребром нерухомого об'єкта, але всі наступні спроби відтворити це виправлення були невдалими.

Рис. 8. Увімкнення візуалізації виправлень сервера для перевірки якості роботи передбачення / Enabling server correction visualization to asses prediction quality

Виконано аналогічні перевірки із затримкою в 1 с. Очікувалось, що відловити виправлення за такої великої затримки буде достатньо просто, проте впродовж 600 с перевірки виправлення відбулось тільки 3 рази, що значно менше, ніж очікувалось. Хоча за затримки в 1 с гру неможливо грати через значне відставання персонажів інших гравців, система передбачень достатньо добре себе показала навіть в таких критичних умовах.

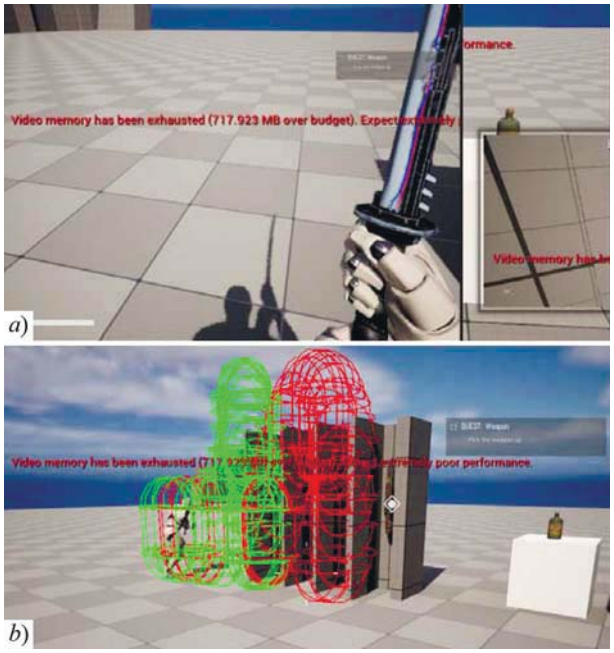

Рис. 9. Виправлення гравця сервером при зіткненні з перешкодою та штучною затримкою пакетів в 1 с / Player correction by the server upon collision with obstacle and artificial 1 second packet delay

Тест із зіткненнями теж показав себе краще, ніж очікувалось: тільки два виправлення за 600 с, одне з них можна побачити на рис. 9. Проте порівнюючи з попередніми тестами, затримка в 1 с інколи породжує достатньо великі виправлення: помічено виправлення приблизно на 5 м і з поворотом майже на 60 градусів.

Обговорення результатів дослідження. У роботі [21] для подолання проблем мережевої затримки пакетів автори пропонують використання алгоритмів планування із врахуванням трафіку. Такий підхід має мінімізувати час передавання пакетів внаслідок пошуку найменш завантаженого шляху, що сприяє ефективному передаванню. Це дослідження вирішує проблему в корені та оптимізує роботу мережі загалом, проте воно є дорожчим у реалізації порівняно з алгоритмами компенсації затримок, використаних у розробленому застосунку, оскільки потребує модифікації мережі та більшої обчислювальної потужності маршрутизаторів.

У своїй роботі [19] автори пропонують вирішувати проблеми затримки мережевих пакетів у багатокористувацьких іграх способом використання підходу до передбачення позиції гравця, заснованого на нейронній мережі в поєднанні з алгоритмом відстеження шляху. Згідно з результатами такий підхід забезпечує на 45 % меншу похибку порівняно з аналогами та вимагає менше даних для передавання мережею. Проте використання нейронних мереж потребує довгого процесу навчання та великої вибірки даних порівняно з детермінованими алгоритмами, а на складних моделях такий підхід працюватиме повільніше. Розроблений нами підхід, який ґрунтується на удосконалених алгоритмах передбачення та засобах Unreal Engine, надає точність, достатню для більшості ігор (хоч і нижчу, ніж запропонований у роботі [19]), але працює швидше, що дає змогу витратити зекономлений час для покращення програмного забезпечення інших систем.

У дослідженні [6] автор теж описує вплив затримок на досвід користувача. Як приклад, наводиться FPS (англ. *First Person Shooter*) гра, де від затримки залежить влучність стрільби та й загальне враження гравців також. Автор порівнює наявні методи компенсації та на основі моделі нейронної мережі глибокого навчання (англ. *Deep Learning*) намагається вирішити загальну проблему затримки без втручання в ігровий код. Згідно з дослідженням такий підхід передбачає і внутрішні ігрові стани, і дії зовнішніх щодо програми користувачів. Порівнюючи такий підхід із нашими результатами, то Він має перевагу як універсальний підхід до передбачення, який не вимагає змін в ігровому коді, проте потребує навчання й перетреновування нейронної мережі при додаванні в гру нових можливостей переміщення, а також розширення вибірки даних для тренування. Розроблений нами підхід завдяки засобам Unreal Engine є гнучкішим, оскільки система передбачення руху персонажа легко модифікується для врахування нових ігрових можливостей.

У своєму дослідженні [9] автори пропонують використати децентралізований підхід до ігрових світів, які обробляються сервером. Їхня ідея полягає у тому, щоб створити розподілену систему серверів, які можуть виступати у ролі одного цілого під час обробляння запитів від клієнтів. Автори дійшли висновку, що такий підхід надає кращу масштабованість та безпеку, бо у випадку, коли один із серверів зламається, гравці навіть не помітять цього. Внаслідок нашого дослідження отримано систему, яка також володіє особливістю стійкості до збоїв сервера, оскільки архітектура *listen server* передбачає, що якщо сервер перестав функціонувати, то один із гравців стає новим сервером, що також забезпечує безперебійну гру. Проте, на відміну від розподілених серверів, розроблена нами система не вимагає виділення віддалених серверів та не має накладних витрат на міжсерверну синхронізацію.

У своєму дослідженні [20] для вирішення проблеми пошуку ігрових сесій та під'єднання до них автори розробили свій плагін для рушія Unreal Engine, який надає функціонал роботи з ігровими сесіями. Такий підхід дав змогу авторам гнучко налаштовувати роботу із сесіями та надав більший контроль над ними. Проте запропонований нами підхід внаслідок використання EOS надає можливість працювати на різних платформах, починаючи від персональних комп'ютерів та завершуючи ігровими приставками й портативними консолями, а застосування власного плагіну, як у роботі [20], вимагає розроблення додаткового програмного забезпечення для таких випадків. Водночас використання EOS є дешевшим, бо не потребує оренди власних серверів (їх уже надає компанія Epic Games).

У дослідженні [1] автори розглядають підхід з використанням туманних технологій для вирішення низки проблем, пов'язаних із затримкою мережі в багатокористувацьких іграх. Запропоновано новий генетичний алгоритм, який допомагає оптимальніше вибирати туманні вузли для обчислення, щоб забезпечити найкращий досвід та мінімальні затримки. Таку систему можна поєднати з реалізованими нами алгоритмами передбачення із використанням засобів Unreal Engine, що дасть змогу покращити ігровий досвід навіть для віддалених гравців, чим плануємо зайнятися в подальших дослідженнях, Водночас, використавши складні режими руху (плавання, скелелазіння, ковзання, рух канатом тощо).

Отже, за результатами виконаної роботи можна сформулювати такі наукову новизну та практичну значущість результатів дослідження.

Наукова новизна отриманих результатів дослідження – набули подальшого розвитку відомі теоретичні підходи до забезпечення плавного ігрового досвіду у багатокористувацьких іграх, удосконалено практичне використання засобів сучасного ігрового рушія Unreal Engine 5 для спрощення та оптимізації процесу розроблення багатокористувацьких ігор.

Практична значущість результатів дослідження – результати дослідження реалізовано в ігровому програмному забезпеченні, яке демонструє можливі практичні способи використання систем ігрового рушія Unreal Engine 5 для ефективного вирішення завдань забезпечення плавного ігрового досвіду у багатокористувацьких іграх.

Висновки / Conclusions

Розроблено багатокористувацьку кооперативну пригодницьку гру як програмну систему реального часу, яка демонструє можливі способи вирішення ключових завдань, пов'язані з передаванням і синхронізацією даних мережею Інтернет. За результатами проведеного дослідження можна зробити такі основні висновки.

1. Визначено основні завдання, які виникають у процесі розроблення багатокористувацьких ігор, зокрема, щодо вибору архітектури та способу передавання даних. Проаналізовано проблеми мережевої затримки, коливання мережі та втрату пакетів. Встановлено основні причини, через які ці проблеми виникають. Запропоновано повторне відправлення пакетів для уникнення їхньої втрати. Використано локальну симуляцію та передбачення для компенсації затримки мережі та втрати пакетів.
2. Описано основні можливості рушія та інструменти, які він надає, для ефективного розроблення багатокористувацьких ігор. Досліджено основні архітектури багатокористувацьких ігрових додатків рушія Unreal Engine 5 та аргументовано доцільність вибору архітектури *listen server* для кооперативних ігор.
3. Створено кооперативну гру, у якій реалізовано згадані методи для боротьби з основними проблемами багатокористувацьких ігор. Досліджено та реалізовано можливість локального передбачення переміщення для уникнення різкої зміни позиції персонажа внаслідок серверних виправлень. Реалізовано локальну симуляцію для зброї, яка відображає гравцю візуальні ефекти, та фізичну симуляцію для замаскування затримки у мережі. Інтегровано онлайн-сервіси від Epic Games для реалізації можливості під'єднання до ігрових світів інших гравців мережею Інтернет.
4. Встановлено, що ігровий рушій Unreal Engine надає достатньо ефективну синхронізацію із сервером та можливість регулювати її частоту для певних змінних. За його допомогою вдається оптимізувати навантаження в мережі внаслідок можливості різної реплікації станів, наприклад, обмеження реплікації тільки до певного клієнта або початкової одноразової реплікації. Окрім цього, встановлено, що рушій Unreal Engine надає потужні алгоритми передбачення позиції персонажа, які забезпечують плавний ігровий досвід навіть за великих затримок.

References

1. Benamer, A. R., Boussetta, K., & Hadj-Alouane, N. B. (2021, December). A genetic algorithm for the placement of latency-sensitive multiplayer game servers in the fog. In 2021 *IEEE Global Communications Conference* (*GLOBECOM*) (pp. 1–6). IEEE. https://doi.org/10.1109/GLOBECOM46510.2021.9685952
2. Duarte, E. P., Pozo, A. T. R., & Beltrani, P. (2020). Smart Reckoning: Reducing the traffic of online multiplayer games using machine learning for movement prediction. *Entertainment Computing*, 33, 100336. https://doi.org/10.1016/j.entcom.2019.100336
3. Epic Dev | Home – Epic Online Services. URL: https://dev.epicgames.com/docs/epic-online-services.
4. Goh, E., Al-Tabbaa, O., & Khan, Z. (2023). Unravelling the complexity of the Video Game Industry: An integrative framework and future research directions. *Telematics and Informatics Reports*, 12, 100100. https://doi.org/10.1016/j.teler.2023.100100
5. Grenville Armitage, Mark Claypool, & Philip Branch. (2006, May 18). Networking And Online Games: Understanding And Engineering Multiplayer Internet Games. John Wiley & Sons Inc; 1st Edition, 218 p. URL: https://www.amazon.com/Networking-Online-Games-Understanding-Engineering/dp/0470018577
6. Halbhuber, D. (2022, November). To Lag or Not to Lag: Understanding and Compensating Latency in Video Games. *In Extended Abstracts of the 2022 Annual Symposium on Computer-Human Interaction in Play* (pp. 370–373). https://doi.org/10.1145/3505270.3558364
7. Larsson, E. (2016). Movement prediction algorithms for high latency games: A testing framework for 2 d racing games. URL: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A952076&dswid=1600
8. Lee, S. W., & Chang, R. K. (2018, June). Enhancing the experience of multiplayer shooter games via advanced lag compensation. *In Proceedings of the 9th ACM Multimedia Systems Conference* (pp. 284–293). https://doi.org/10.1145/3204949.3204971
9. Moll, P., Isak, S., Hellwagner, H., & Burke, J. (2021). A Quadtree-based synchronization protocol for inter-server game state synchronization. *Computer Networks*, 185, Article ID 107723. https://doi.org/10.1016/j.comnet.2020.107723
10. Motoo, T., Kawasaki, J., Fujihashi, T., Saruwatari, S., & Watanabe, T. (2021). Client-Side network delay compensation for online shooting games. *IEEE Access*, 9, 125678–125690. https://doi.org/10.1109/access.2021.3111180

11. Multiplayer network compendium. An Unreal Engine Blog by Cedric Neukirchen. URL: https://cedric-neukirchen.net/docs/category/multiplayer-network-compendium/
12. Networking overview for unreal engine | unreal engine 5.3 documentation. Epic Developer Community. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/networking-overview-for-unreal-engine? application_version=5.3
13. Pantel, L., & Wolf, L. C. (2002, April). On the suitability of dead reckoning schemes for games. *In Proceedings of the 1st workshop on Network and system support for games* (pp. 79–84). https://doi.org/10.1145/566500.566512
14. Pantel, L., & Wolf, L. C. (2002, May). On the impact of delay on real-time multiplayer games. *In Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video* (pp. 23–29). https://doi.org/10.1145/507670.507674
15. Remote procedure calls in unreal engine | unreal engine 5.3 documentation. (2024). Epic Developer Community. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/remote-procedure-calls-in-unreal-engine? application_version=5.3
16. Replicate actor properties in unreal engine | unreal engine 5.3 documentation. (2024). Epic Developer Community. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/replicate-actor-properties-in-unreal-engine? application_version=5.3
17. Understanding networked movement in the character movement component for unreal engine | unreal engine 5.4 documentation. Epic Developer Community. (2024). URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/understanding-networked-movement-in-the-character-movement-component-for-unreal-engine
18. Using network emulation. (2024). URL: https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Networking/NetworkDebugging/NetworkEmulation
19. Walker, T., Gilhuly, B., Sadeghi, A., Delbosc, M., & Smith, S. L. (2023). Predictive dead reckoning for online peer-to-peer games. *IEEE Transactions on Games*, 1–12. https://doi.org/10.1109/tg.2023.3237943
20. Wen, S., Li, D., Yuan, J., Huang, J., & Zhou, L. (2023, September). Design and Implementation of a Game Session Plugin based on Unreal Engine. *In Proceedings of the 8th International Conference on Cyber Security and Information Engineering* (pp. 142–149). https://doi.org/10.1145/3617184.3623451
21. Wijayasekara, S. K., Sasithong, P., Hsieh, H. Y., Saengudomlert, P., Chae, C. B., & Wuttisittikulkij, L. (2023). Optimization of packet transmission scheduling and node parent selection for 802.15.4 e time slotted channel hopping (TSCH). *ICT Express*. https://doi.org/10.1016/j.icte.2023.12.002
22. Yahyavi, A., & Kemme, B. (2013). Peer-to-peer architectures for massively multiplayer online games. *ACM Computing Surveys*, 46(1), 1–51. https://doi.org/10.1145/2522968.2522977

L. M. Zhuravchak, V. R. Senkiv, N. R. Senkiv

Lviv Polytechnic National University, Lviv, Ukraine

IMPLEMENTATION OF A MULTIPLAYER GAME USING UNREAL ENGINE 5

A novel approach to developing multiplayer games, leveraging the capabilities of Unreal Engine to optimize data transmission, reduce latency, and synchronize player experiences in shared game worlds is developed. The method incorporates specialized algorithms for compensating transmission delays, addressing network fluctuations and packet loss along with position prediction techniques to mitigate network latency effects. The primary goal of the research is to create a cooperative adventure game as a real-time software system, demonstrating practical solutions to key challenges in data transmission and synchronization over the Internet. By adopting a listen server architecture, we capitalize on its cost-effectiveness by eliminating the need for dedicated game servers (players take turns acting as servers) and reducing packet delay through localized hosting. Synchronization of game states is achieved through replication, ensuring consistent representation across clients, and Remote Procedure Calls (RPCs) for efficient data transmission. A computational experiment on local prediction of character movement was conducted to verify the quality of the prediction algorithms. Local simulation for weapons, providing players with visual effects, and physical simulation to mitigate network latency is implemented. Moreover, integration with Epic Games online services enables seamless connectivity with other players' game worlds. Extensive testing validates the effectiveness of predictive movement system, ensuring smooth character movements even under significant network delays (500 ms) and real-world unpredictability (1 s). Proposed approach demonstrates precision and efficiency in addressing challenges in multiplayer game development, facilitated by packet retransmission mechanisms and local simulation strategies. Future work will be focused on further refining prediction algorithms to accommodate various movement modes.

Keywords: packet transmission latency minimization; character movement prediction; local simulation; network latency compensation; player game state synchronization.