МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

Кафедра комп'ютерних наук та прикладної математики

"До захисту допущена"
Зав. кафедри комп'ютерних наук та
прикладної математики
д.т н., проф. Ю.В. Турбал
«____» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

Розробка багатокористувацького режиму для стратегії в реальному часі з використанням Godot Engine

Виконав: Ковтунець Владислав Володимирович _____
(прізвище, ім'я, по батькові) (підпис)

група ІПЗ-41

Керівник: к.т.н., доцент, доцент Климюк Ю. Є _____
(науковий ступінь, вчене звання, посада, прізвище, ініціали) (підпис)

Рівне – 2025

Національний університет водного господарства та природокористування

(повне найменування вищого навчального закладу)

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

Кафедра **комп'ютерних наук та прикладної математики**
Освітньо-кваліфікаційний рівень **бакалавр**
Галузь знань **12 Інформаційні технології**
(шифр і назва)
Спеціальність **121 Інженерія програмного забезпечення**
(шифр і назва)
Спеціалізація –

«ЗАТВЕРДЖУЮ»
Завідувач кафедри
д.т н., проф. Ю.В. Турбал
«____» _____ 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ковтунцю Владиславу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи *"Розробка багатокористувацького режиму для стратегії в реальному часі з використанням Godot Engine"*

керівник роботи *Климюк Юрій Євгенійович, к.т.н., доцент, д кафедри комп'ютерних наук та прикладної математики.*

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання, посада)

затверджені наказом по університету від *"22" квітня 2025 року С №-525.*

2. Термін подання роботи студентом *30 травня 2025 року.*

3. Вихідні дані до роботи: *архітектура багатокористувацької гри у реальному часі, особливості мережевого обміну в Godot Engine, підключення клієнтів до хоста, обробка команд гравців і синхронізація стану гри.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *провести реалізацію багатокористувацького режиму у стратегіях, розробити прототип багатокористувацької гри з використанням Godot Engine, реалізувати базовий функціонал з підключенням клієнтів, обробкою дій та синхронізацією, провести тестування та оцінити стабільність з'єднання.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Мультимедійна презентація

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1	*доцент Климюк Ю.Є.*	*21.10.24*	*08.12.24*
Розділ 2	*доцент Климюк Ю.Є.*	*21.10.24*	*18.03.25*
Розділ 3	*доцент Климюк Ю.Є.*	*21.10.24*	*14.05.25*

7. Дата видачі завдання *21 жовтня 2024 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	*Вибір та узгодження теми кваліфікаційної роботи*	*02.10.24 – 20.10.24*	*виконав*
2	*Аналіз літературних джерел і визначення підходів до реалізації мережевої гри*	*23.10.24 – 08.12.24*	*виконав*
3	*Розробка архітектури клієнт-серверної взаємодії*	*11.12.24 – 26.01.25*	*виконав*
4	*Реалізація базового функціоналу сервера та клієнтів*	*29.01.25 – 23.02.25*	*виконав*
5	*Реалізація логіки гри та синхронізації стану*	*26.02.25 – 18.03.25*	*виконав*
6	*Тестування на кількох клієнтах, відлагодження*	*18.03.25 – 22.03.25*	*виконав*
8	*Створення графічних матеріалів та інтерфейсів*	*25.03.25 – 26.04.25*	*виконав*
9	*Оформлення пояснювальної записки*	*01.04.25 – 31.05.25*	*виконав*

Здобувач _____ **Ковтунець В.В.**
(підпис) (прізвище та ініціали)

Керівник _____ **Климюк Ю.Є.**
(підпис) (прізвище та ініціали)

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота: 51 с., 7 рисунків, 17 джерел.

Мета роботи: проєктування та реалізація багатокористувацького режиму для гри «Legends of Draxxor» у жанрі стратегії в реальному часі з використанням ігрового рушія Godot Engine. У межах цієї мети передбачено створення архітектури, що забезпечує можливість взаємодії кількох гравців у спільному ігровому середовищі з урахуванням вимог до стабільності з'єднання, синхронізації стану гри та продуктивності системи.

Об'єкт дослідження – процес розробки багатокористувацької гри у жанрі стратегії в реальному часі, з акцентом на використання інструментів і можливостей ігрового рушія Godot Engine.

Предмет дослідження – архітектура та технології реалізації мережевої взаємодії у багатокористувацькій стратегічній грі в реальному часі на базі Godot Engine.

Методи вивчення – аналіз технічної документації та практичних рішень з реалізації багатокористувацької взаємодії в ігрових рушіях, дослідження можливостей мережевого плагіна GdSteam, експериментальне впровадження клієнт-серверної логіки за допомогою функціоналу Steam API у Godot Engine, а також тестування, налагодження й оцінка стабільності та продуктивності з'єднання між гравцями.

Ключові слова: GODOT ENGINE, БАГАТОКОРИСТУВАЦЬКИЙ РЕЖИМ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, GDSTEAM, ІГРОВА СИНХРОНІЗАЦІЯ, РОЗРОБКА ІГОР, РЕАЛЬНИЙ ЧАС.

ВСТУП

У сучасних умовах стрімкого розвитку ігрової індустрії особливого поширення набувають багатокористувацькі ігри, що дозволяють гравцям взаємодіяти в режимі реального часу. Такий формат суттєво підвищує інтерес до ігрового продукту, забезпечуючи соціальну складову, конкуренцію та розширення геймплейних можливостей. Жанр стратегій у реальному часі (RTS) зберігає свою актуальність, оскільки поєднує тактичне мислення, стратегічне планування і швидку реакцію, що особливо цікаво у багатокористувацькому — рушій з відкритим вихідним кодом, що активно розвивається та підтримується спільнотою розробників. Попри численні переваги, реалізація повноцінного багатокористувацького режиму в середовищі Godot залишається складним завданням через обмежені вбудовані інструменти.

Актуальність теми зумовлена недостатньою кількістю готових рішень для створення RTS-ігор з мультиплеєром (багатокористувацьким режимом) у Godot. Це ускладнює процес розробки для незалежних розробників і потребує створення надійного прототипу, що міг би слугувати шаблоном для подальших проєктів. Окрему увагу в цьому контексті заслуговує інтеграція зі Steam API, яка надає широкі можливості для організації клієнт-серверної взаємодії, створення лобі, матчмейкингу та інших сервісних функцій через плагін GdSteam.

Мета роботи полягає у проєктуванні та реалізації багатокористувацького режиму для гри у жанрі стратегії в реальному часі з використанням рушія Godot Engine та Steam API.

Для досягнення поставленої мети в роботі було окреслено низку завдань. Насамперед здійснено дослідження особливостей реалізації багатокористувацького режиму в середовищі Godot Engine, що включало вивчення його мережевої інфраструктури, можливостей синхронізації об’єктів та роботи з віддаленими процедурами виклику (RPC). Особливу увагу приділено ознайомленню з принципами функціонування Steam API, а також з бібліотекою

GdSteam, що забезпечує взаємодію між Godot та сервісами платформи Steam, зокрема у контексті автентифікації, створення лобі та запрошення гравців.

У межах проєкту розроблено архітектуру клієнт-серверної взаємодії, яка забезпечує обмін даними між учасниками гри в реальному часі. Реалізовано базові механізми синхронізації стану гри між кількома клієнтами, що включає оновлення позицій персонажів, дій гравців та загального стану ігрового середовища.

Як практичний результат створено прототип гри в жанрі RTS (стратегія в реальному часі), що підтримує багатокористувацький режим. У завершальному етапі виконано аналіз функціональних можливостей реалізованого рішення, а також визначено потенційні напрями подальшого вдосконалення проєкту, зокрема щодо масштабованості, поліпшення продуктивності та розширення ігрових механік.

Практична значущість роботи полягає в тому, що створений прототип може бути використаний як база для подальших проєктів у жанрі RTS, розроблених у Godot Engine. Крім того, він дає змогу на практиці перевірити доцільність застосування Steam API для вирішення задач мультиплеєрної взаємодії в іграх, що розробляються незалежними командами або індивідуальними розробниками.

РОЗДІЛ 1
ПРОЄКТУВАННЯ АРХІТЕКТУРИ БАГАТОКОРИСТУВАЦЬКОГО РЕЖИМУ ГРИ ТА ОСНОВ СИСТЕМИ ДЛЯ МАСШТАБОВАНОСТІ

1.1 Порівняльний аналіз рушіїв

У сучасній ігровій розробці рушій відіграє фундаментальну роль, оскільки саме він визначає технічні можливості проєкту [5], гнучкість реалізації механік та подальшу підтримку. Для створення багатокористувацьких ігор, особливо в жанрі RTS, обирають рушії, які забезпечують достатній рівень продуктивності, підтримку мережевої взаємодії та спільноту розробників. У цьому підрозділі буде розглянуто та порівняно три найпопулярніші рушії: Godot Engine [1], Unity та Unreal Engine, з особливим акцентом на їхню придатність до розробки мультиплеєрних систем.

Godot Engine — це відкритий рушій із акцентом на простоту, швидкий прототипінг і гнучку структуру сцени. Основною перевагою Godot є його повністю відкритий код, що дозволяє розробникам змінювати рушій під власні потреби. Також він не вимагає ліцензійних виплат, що критично важливо для незалежних проєктів.

У контексті мультиплеєрних ігор Godot пропонує вбудовану підтримку RPC, функції network_master, rpc_id, та можливість обрати як peer-to-peer, так і client-server модель. Хоча його мережевий API вважається менш розвиненим, ніж у конкурентів, завдяки додатковим рішенням, розширення функціоналу не викликає труднощів. Простота GDScript дозволяє швидко створювати і налагоджувати ігрову логіку без значного технічного боргу.

Unity — це один із найпоширеніших рушіїв для розробки 2D та 3D ігор, який підтримує C# як основну мову програмування та має розвинуту систему компонентів. Unity активно використовується як у мобільній, так і в AAA-розробці, а його екосистема включає тисячі плагінів на Unity Asset Store.

Для створення мультиплеєру Unity історично мав власне рішення UNet, яке було згодом застарілим. Сьогодні рекомендується використовувати Netcode for GameObjects або сторонні рішення — Photon, Mirror, FishNet. Їхня інтеграція дозволяє реалізовувати як хостований P2P, так і повністю серверний підхід. Unity також пропонує Matchmaking, Relay і Lobby сервіси через Unity Gaming Services, проте більшість із них є платними при масштабуванні, що може бути критично для інді-проєктів.

Unreal Engine — це високопродуктивний рушій із C++ як основною мовою та системою візуального програмування Blueprint. Його використовують у AAA-проєктах, VR/AR та кінематографічному виробництві. Unreal пропонує розвинуту мережеву модель, підтримку реплікації, вбудовані компоненти реплікації станів, авторитарну архітектуру та передові засоби відлагодження.

У випадку мультиплеєру Unreal дозволяє створювати dedicated-сервери, використовує чітко визначені правила reparenting і володіння об’єктами (Owner, NetOwner, RemoteRole). Однак крива навчання дуже стрімка, а використання рушія вимагає знань C++ і глибокого розуміння низькорівневої архітектури. Через це Unreal не завжди підходить для швидких або малобюджетних інді-проєктів.

Враховуючи обмежений бюджет, відсутність необхідності у фотореалізмі, потребу в кастомному мультиплеєрі та швидкому прототипуванні, Godot Engine став найбільш доцільним вибором для реалізації проєкту Legends of Draxxor. Відкритий код, гнучка структура та можливість інтеграції Steam API через GdSteam забезпечили не лише комфорт розробки, а й потенціал для масштабування в майбутньому.

1.2 Огляд використаного інструментарію

Godot Engine [1] є потужним кросплатформним рушієм з відкритим вихідним кодом, що ідеально підходить для створення як 2D, так і 3D ігор. Його популярність серед інді-розробників зумовлена низкою ключових переваг, які роблять процес розробки ефективним та приємним. Головною особливістю

Godot є його модульна структура, яка дозволяє розробникам з легкістю комбінувати різні компоненти ігрової сцени: об'єкти, логіку, анімації та ресурси. Ця архітектура сприяє високій гнучкості та перевикористанню коду, що є критично важливим для великих і складних проєктів.

Рушій відомий своєю невеликою вагою, що забезпечує швидке завантаження та плавну роботу навіть на менш потужних системах. Інтуїтивно зрозумілий інтерфейс редактора Godot значно знижує поріг входу для нових розробників, дозволяючи їм швидко освоїти інструментарій та зосередитися на творчості. Завдяки активній та зростаючій спільноті розробників, Godot постійно вдосконалюється, а користувачі завжди можуть знайти підтримку та ресурси для вирішення будь-яких питань.

Історія Godot бере свій початок у 2014 році, коли він був вперше випущений як відкритий проєкт. Спочатку створений для внутрішніх потреб анімаційної студії, рушій швидко еволюціонував, перетворившись на повноцінного конкурента таким гігантам індустрії, як Unity та Unreal Engine. Відкритість Godot не просто означає безкоштовну ліцензію; вона надає розробникам повну свободу змінювати та адаптувати код під власні потреби. Це є ключовою перевагою для проєктів, які вимагають високого рівня кастомізації або мають унікальні вимоги до функціоналу. Можливість глибокого втручання в архітектуру рушія дозволяє розробникам створювати унікальні рішення, які не завжди можливі з комерційними рушіями.

Особливе місце в Godot займає його система сцен та вузлів (node system). Це фундаментальна концепція, яка дозволяє будувати складні ієрархії об'єктів гри. Кожен об'єкт у Godot є компонентом, який може мати дочірні вузли, формуючи таким чином дерево сцен. Цей підхід не лише впорядковує структуру проєкту, але й значно зменшує необхідність дублювання коду. Наприклад, якщо у вас є персонаж, що складається з графічної моделі, колізії, камери та скриптів, ви можете об'єднати їх в одну сцену-вузол і використовувати її багаторазово в різних частинах гри. Це робить розробку більш модульною та легкою для підтримки.

Для реалізації ігрової логіки Godot пропонує два основні варіанти: власну мову сценаріїв GDScript або широко поширену мову C#. Ця гнучкість дозволяє розробникам обирати інструмент, який найкраще відповідає їхнім навичкам та вимогам проєкту. GDScript, будучи тісно інтегрованим з рушієм, надає швидкий та ефективний спосіб написання ігрової логіки, тоді як C# дозволяє використовувати потужні можливості .NET фреймворку.

Godot також відзначається своєю мультиплатформною підтримкою, дозволяючи розробникам збирати проєкти для широкого спектру платформ, включаючи Windows, Linux, macOS, Android, iOS та HTML5. Ця можливість значно розширює потенційну аудиторію для ігор, створених на Godot, оскільки розробникам не потрібно переписувати код для кожної нової платформи.

Вбудована підтримка мережевої взаємодії є ще однією значною перевагою Godot. Рушій надає як низькорівневу взаємодію через UDP/TCP сокети для тонкого контролю над мережевим трафіком, так і високорівневий механізм Remote Procedure Calls (RPC) [7]. RPC значно спрощує реалізацію мультиплеєрної логіки, дозволяючи викликати функції на віддалених клієнтах або серверах так, ніби вони виконуються локально. Завдяки своїй прозорій архітектурі, Godot ідеально підходить для реалізації як однокористувацьких, так і колективних проєктів зі складною внутрішньою логікою, забезпечуючи надійну основу для онлайн-ігор.

GDScript є ключовим елементом екосистеми Godot Engine, розробленим спеціально для максимальної ефективності та зручності у контексті ігрової розробки. Це високорівнева, динамічно типізована мова програмування, синтаксис якої максимально наближений до Python. Така схожість робить GDScript надзвичайно легкою у засвоєнні, особливо для розробників, які вже знайомі з Python, і дозволяє швидко прототипувати та реалізовувати ігрову логіку. Простота синтаксису GDScript дозволяє зосередитися на самій логіці гри, а не на складностях мови програмування.

Однією з головних особливостей GDScript є її глибока інтеграція з усіма компонентами рушія. Це означає, що GDScript визначає об'єкти сцени, сигнали,

ресурси, інтерфейси та інші елементи Godot на системному рівні. Така інтеграція дозволяє програмістам безпосередньо взаємодіяти з рушієм, створюючи складну поведінку об'єктів з мінімальними зусиллями. Наприклад, для керування анімацією персонажа, рухом камери або взаємодією з інтерфейсом користувача достатньо лише кількох рядків коду, що значно прискорює процес розробки. Така інтеграція забезпечує високу продуктивність та зменшує обсяг коду, необхідного для інтеграції різних компонентів, який часто потрібен при використанні інших мов програмування в ігрових рушіях.

Історично GDScript була розроблена з метою максимально спростити взаємодію з внутрішньою логікою Godot. Це дозволило розробникам зосередитися на геймплеї та ігрових механіках, замість того, щоб витрачати час на складні налаштування або обхідні шляхи. Мова надає зручний спосіб реалізувати події, обробляти вхідні дані від гравців, керувати об'єктами на сцені та створювати складні механіки, такі як системи інвентарю, бойові системи або системи квестів.

У контексті мультиплеєрної розробки, GDScript демонструє свої потужні можливості. Вона дозволяє легко викликати RPC-функції, маркувати змінні для реплікації (автоматичної синхронізації між клієнтами) та обробляти мережеві повідомлення без необхідності використання сторонніх бібліотек. Ця вбудована функціональність значно спрощує створення мережевих ігор, оскільки розробник може зосередитися на логіці гри, а не на низькорівневих аспектах мережевого програмування. Наприклад, синхронізація положення гравців або стану ігрових об'єктів може бути реалізована за допомогою декількох рядків GDScript, що робить процес розробки мультиплеєра доступним навіть для менш досвідчених розробників.

Крім того, GDScript дозволяє активно використовувати об'єктно-орієнтовану модель програмування (ООП). Це сприяє масштабованості та повторному використанню коду, що є вкрай важливим для розробки великих проєктів. Використання класів, успадкування та поліморфізму дозволяє створювати гнучку та розширювану архітектуру гри. Це особливо актуально

при розробці мультиплеєрних проєктів, де необхідно чітко розділяти локальну та мережеву логіку, забезпечуючи при цьому чистоту та порядок у кодовій базі. Можливість створювати власні класи та розширювати функціонал існуючих вузлів робить GDScript надзвичайно потужним інструментом для складних ігрових систем.

Steam є найбільшою у світі цифровою платформою для розповсюдження комп'ютерних ігор, розробленою компанією Valve. Її роль у сучасній ігровій індустрії важко переоцінити, оскільки вона не лише забезпечує можливість публікації та продажу ігор, але й надає широкий набір інструментів для взаємодії з користувачами. Цей комплексний підхід робить Steam незамінним партнером для розробників, які прагнуть досягти широкої аудиторії та інтегрувати сучасні онлайн-функції у свої проєкти. Серед ключових можливостей, що надаються платформою, є система друзів, ігрові лобі, досягнення, хмарні збереження, а також потужна мережева інфраструктура.

Платформа була запущена у 2003 році як засіб автоматичного оновлення ігор компанії Valve, таких як культові Half-Life та Counter-Strike. Згодом Steam трансформувався у повноцінний майданчик для поширення ігор, який щодня підтримує мільйони активних користувачів. Для незалежних розробників Steam став одним із головних каналів розповсюдження продуктів завдяки сервісу Steam Direct, який значно спростив процес публікації ігор. Це дозволило безлічі невеликих студій та інді-розробників вийти на світовий ринок та представити свої твори мільйонам гравців.

Steam активно використовується незалежними розробниками, оскільки дозволяє не лише значно розширити аудиторію, а й інтегрувати сучасні онлайн-сервіси у свої ігрові продукти. Через Steamworks SDK розробники отримують доступ до Steam API — потужного набору функцій, що полегшує створення онлайн-функціоналу в іграх. Це включає в себе все: від автентифікації користувачів до складної мережевої взаємодії та монетизації.

Крім технічних можливостей, Steam також забезпечує комплексну економічну інфраструктуру. Це включає систему ліцензування, детальну

статистику продажів, маркетингові інструменти для просування ігор та потужні інструменти захисту авторських прав. Ці переваги роблять платформу надзвичайно привабливою як для новачків, що тільки починають свій шлях у розробці ігор, так і для великих студій з багаторічним досвідом. Завдяки Steam, розробники можуть зосередитися на створенні якісного контенту, довіряючи питання розповсюдження, монетизації та підтримки користувачів надійній та перевіреній платформі.

Steam API [3] (Application Programming Interface) є фундаментальним набором інструментів, що дозволяє розробникам взаємодіяти з сервісами Steam з метою реалізації онлайн-функцій у грі. Використання цього API є невід'ємною частиною створення сучасних мультиплеєрних ігор, оскільки воно забезпечує доступ до широкого спектру функціоналу, який інакше довелося б розробляти з нуля. Серед ключових можливостей Steam API: аутентифікація користувача, робота з лобі (створення, приєднання, отримання списку гравців), обмін повідомленнями між гравцями, збереження ігрової статистики та досягнень, а також обробка хмарних збережень. Це дозволяє розробникам зосередитися на унікальних аспектах геймплею, делегуючи складні завдання, пов'язані з онлайн-сервісами, надійній інфраструктурі Steam.

API складається з декількох підсистем, кожна з яких відповідає за окрему групу функціональностей. Наприклад, ISteamUser відповідає за автентифікацію гравця та отримання інформації про нього, ISteamMatchmaking керує пошуком лобі та матчів, а ISteamNetworking відповідає за встановлення мережевих з'єднань та обмін даними. Цей модульний підхід дозволяє розробникам використовувати лише ті компоненти, які необхідні для конкретного проєкту, уникаючи зайвого навантаження та складності. Така архітектура забезпечує гнучкість та ефективність у використанні ресурсів.

Використання Steam API дозволяє значно спростити процес реалізації мультиплеєрного режиму, оскільки більшість завдань із синхронізації та передачі даних делегується готовим сервісам Steam. Це не тільки економить час та ресурси розробників, але й забезпечує високу стабільність, надійність та

масштабованість, що є критично важливим для онлайн-ігор з великою кількістю користувачів. Steam має перевірену часом інфраструктуру, здатну витримувати значні навантаження, що робить його ідеальним вибором для мультиплеєрних проєктів.

Однак пряма взаємодія зі Steam API часто вимагає використання C++ та ручної компіляції нативних бібліотек, що може бути складним завданням для розробників, які працюють переважно з GDScript або іншими мовами високого рівня. Тут на допомогу приходить GdSteam [2] — сторонній плагін для Godot Engine, що реалізує обгортку над Steam API. Цей плагін є надзвичайно цінним інструментом, оскільки він дозволяє напряму використовувати Steam-функціональність у Godot без необхідності писати C++ код або компілювати нативні бібліотеки вручну.

GdSteam надає розробникам Godot доступ до всіх ключових функцій Steam API через зручні класи та методи, доступні безпосередньо у GDScript. Серед можливостей, які надає плагін: ініціалізація Steam-клієнта в грі, робота з лобі, відправка та отримання P2P-повідомлень, отримання Steam ID та імені користувача, а також управління списком друзів у Steam. Це значно спрощує інтеграцію Steam-функціоналу та дозволяє зосередитися на ігровій логіці.

Плагін GdSteam підтримується активною спільнотою, має відкритий вихідний код і регулярно оновлюється відповідно до змін у Steam SDK. Він активно використовується в інді-розробці як простий та ефективний спосіб забезпечити базову мультиплеєрну функціональність без потреби створювати власні сервери чи складну мережеву інфраструктуру.

Завдяки GdSteam, розробник отримує повноцінний доступ до мережевої інфраструктури Steam без необхідності виходити за межі Godot Engine [9], що робить реалізацію мультиплеєру значно швидшою та зручнішою. Цей плагін фактично абстрагує складні процеси взаємодії із Steam API, надаючи розробнику доступ до високорівневих функцій у вигляді простих методів і сигналів у GDScript. Таким чином, комбінація Godot Engine, GDScript, Steam та

GdSteam утворює потужний та доступний інструментарій для створення сучасних, функціональних та успішних ігор з онлайн-можливостями.

1.3 Обґрунтування архітектури мультиплеєрної системи гри

При розробці багатокористувацьких ігор ключовим фактором успішної реалізації є вибір відповідної мережевої архітектури. Існує кілька класичних підходів до побудови мультиплеєрної взаємодії, кожен із яких має свої переваги та недоліки залежно від вимог до гри, кількості учасників, рівня безпеки та складності реалізації. У цьому підрозділі буде розглянуто основні архітектури: Peer-to-Peer (P2P), Client-Server, Dedicated Server, а також Authoritative Server [4].

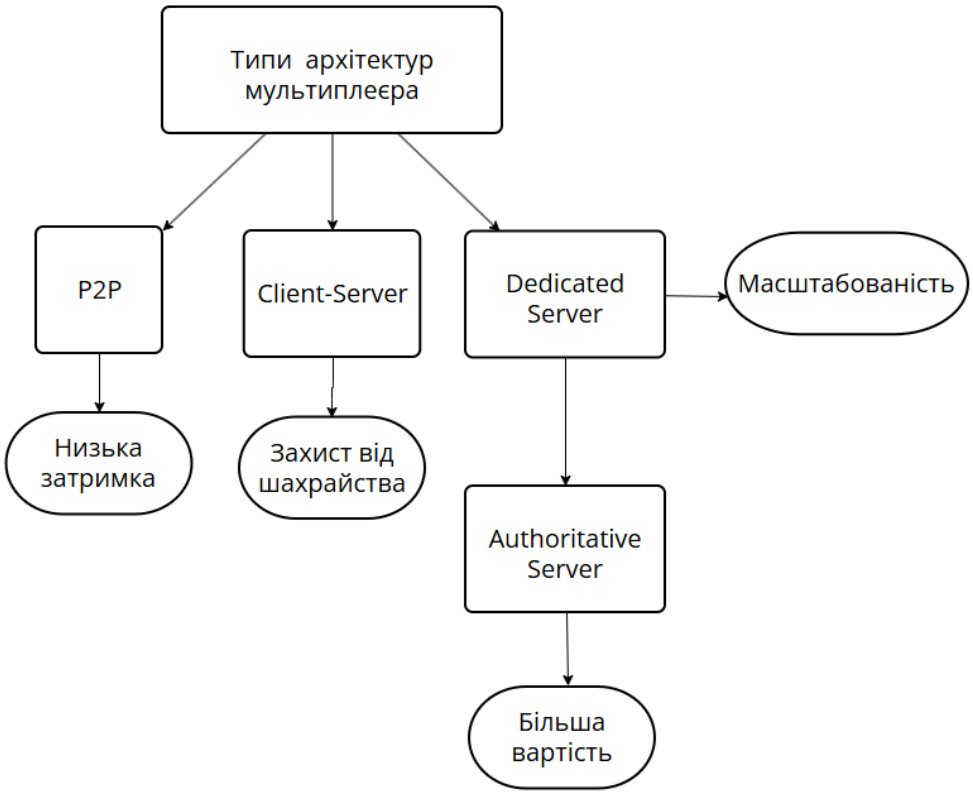

Рис. 1 Класифікація архітектур мультиплеєра та їх характеристики

Модель Peer-to-Peer передбачає, що всі учасники гри під'єднуються один до одного без централізованого хоста. Кожен клієнт має рівні повноваження і

самостійно обробляє частину логіки гри. Цей підхід є простим у реалізації і не вимагає окремого сервера, що знижує витрати на інфраструктуру.

Проте P2P модель має суттєві обмеження. Основна проблема — це синхронізація даних між усіма клієнтами та безпека: оскільки кожен гравець має доступ до повної логіки гри, можливе шахрайство. Також складно реалізувати масштабованість, оскільки зі збільшенням кількості гравців росте складність обміну повідомленнями.

У проєкті Legends of Draxxor використовується покращена P2P модель з одним визначеним «хостом» серед гравців, що частково компенсує недоліки класичного P2P.

У цій архітектурі один гравець виступає як сервер (хост), а інші під'єднуються до нього як клієнти. Сервер відповідає за збереження глобального стану гри, а клієнти надсилають запити на зміну цього стану.

Цей підхід дозволяє уникнути багатьох проблем P2P — забезпечується цілісність ігрових даних, знижується ризик читів, централізована обробка подій дозволяє уникнути колізій. Проте стабільність гри повністю залежить від роботи хоста, і у разі його виходу з гри — сесія переривається.

У випадку Legends of Draxxor, ця модель реалізована через Steam API, де хост отримує пріоритет у прийнятті рішень, а всі дії клієнтів реплікуються через RPC.

Dedicated Server — це окрема машина або процес, який працює як незалежний сервер без участі гравця. Усі клієнти під'єднуються до цього сервера, а він обробляє всю логіку гри. Це найнадійніша модель, яка використовується в AAA-іграх (CS:GO, Dota 2, Valorant) [12].

Переваги — це висока стабільність, можливість розподілу навантаження, розширення за допомогою кластеризації. Проте вона вимагає додаткових витрат на інфраструктуру, адміністрування та захист. Для невеликих інді-проєктів це часто є економічно недоцільним.

Ця архітектура є підвидом Client-Server або Dedicated Server, але з жорсткою централізацією логіки: сервер не довіряє клієнтам і перевіряє всі їхні

дії. Навіть якщо клієнт надсилає запит на рух або атаку — сервер сам вирішує, чи можливо виконати цю дію відповідно до правил гри.

Такий підхід майже повністю виключає можливість шахрайства, але вимагає додаткових перевірок, що збільшує затримку (latency). У складних RTS-іграх authoritative сервер дозволяє уникати ситуацій, коли різні клієнти мають різний стан гри.

У межах реалізації Legends of Draxxor було обрано гібридну модель P2P з визначеним хостом. Такий підхід дозволяє уникнути витрат на сторонні сервери, забезпечити достатню стабільність у невеликих сесіях (до 6 гравців), і водночас делегувати більшість рішень на одну авторитетну сторону. Хост виконує перевірку команд, обробляє синхронізацію юнітів, споруд, бойових дій та ресурсів.

У перспективі архітектура дозволяє масштабуватися до моделі Dedicated Server без необхідності суттєвого переписування логіки, що забезпечує гнучкість та еволюційність у майбутніх версіях гри.

1.4 Реалізація архітектури в проєкті.

Проєктування архітектури багатокористувацького режиму гри є одним із основних етапів у розробці сучасних ігрових продуктів, які передбачають взаємодію кількох користувачів у спільному ігровому середовищі. На відміну від однокористувацьких проєктів, багатокористувацькі ігри вимагають синхронізації станів, обробки одночасних запитів, стабільного з'єднання та гнучкої реакції на втрату доступу до мережі. Саме тому обґрунтування обраної архітектури є критично важливим для досягнення цілісного та масштабованого рішення.

У межах цієї роботи поставлено завдання реалізувати багатокористувацький режим із застосуванням Godot Engine у поєднанні з GdSteam — плагіном, який надає доступ до Steamworks API. Основою комунікаційної логіки є P2P-з'єднання (peer-to-peer), яке дозволяє гравцям підключатися один до одного напряму без необхідності централізованого

сервера. Такий підхід є особливо ефективним у незалежних проєктах, де важлива мінімізація витрат на серверну інфраструктуру.

Для синхронізації ігрового стану в середовищі Godot застосовано механізм Remote Procedure Call (Виклик віддалених процедур), що дає змогу ініціювати певні дії або змінювати стани об'єктів на стороні іншого клієнта або хоста. За допомогою GdSteam RPC-методи інтегруються з P2P-каналами Steam, що дозволяє досягти низької затримки та високої частоти оновлень.

Рис. 2. Відображення головного меню з інтеграцією Steam: автоматичне завантаження аватара та імені користувача з профілю Steam

Обрана архітектура також враховує принцип масштабованості, тобто можливість адаптації системи до більшої кількості користувачів, змін у транспортному протоколі, або переходу до серверно-клієнтської моделі.

Крім того, при проєктуванні враховано також можливість повторного використання архітектури в інших проєктах. Кодова база побудована за модульним принципом, із чітким розділенням відповідальності: лобі, підключення, синхронізація, обробка команд — кожен з цих аспектів має власну реалізацію і може бути легко адаптований або замінений.

1.5 Особливості мережевої моделі з використанням Steam API та P2P-з’єднань

У межах реалізації багатокористувацької взаємодії в грі Legends of Draxxor обрано підхід, заснований на використанні Steam API через спеціалізований плагін GdSteam для рушія Godot Engine. Такий вибір зумовлений потребою в мінімізації витрат на розгортання серверної інфраструктури, забезпеченні зручного підключення гравців та використанні перевіреного стеку технологій, підтримуваного однією з провідних ігрових платформ — Steam.

Steam API пропонує декілька способів реалізації багатокористувацької взаємодії, серед яких особливу увагу привертає однорангова модель (peer-to-peer), що реалізується засобами SteamNetworkingMessages та SteamNetworkingSockets. У цій моделі з'єднання встановлюються напряму між клієнтами, без залучення централізованого ігрового сервера. Такий підхід дозволяє досягти низьких затримок, зменшити витрати на підтримку серверної інфраструктури, а також забезпечити гнучке управління лобі та сесіями гравців.

Фактична реалізація підключення в Legends of Draxxor побудована на основі лобі-системи Steam, у якій один із гравців виступає ініціатором (host), створюючи лобі та отримуючи унікальний ідентифікатор SteamLobbyID. Інші гравці приєднуються до цього лобі через Steam API або отримуючи запрошення напряму. Таким чином, хост автоматично стає авторитетним вузлом, відповідальним за управління станом гри, обробку команд та передачу реплікованих даних до інших учасників сесії.

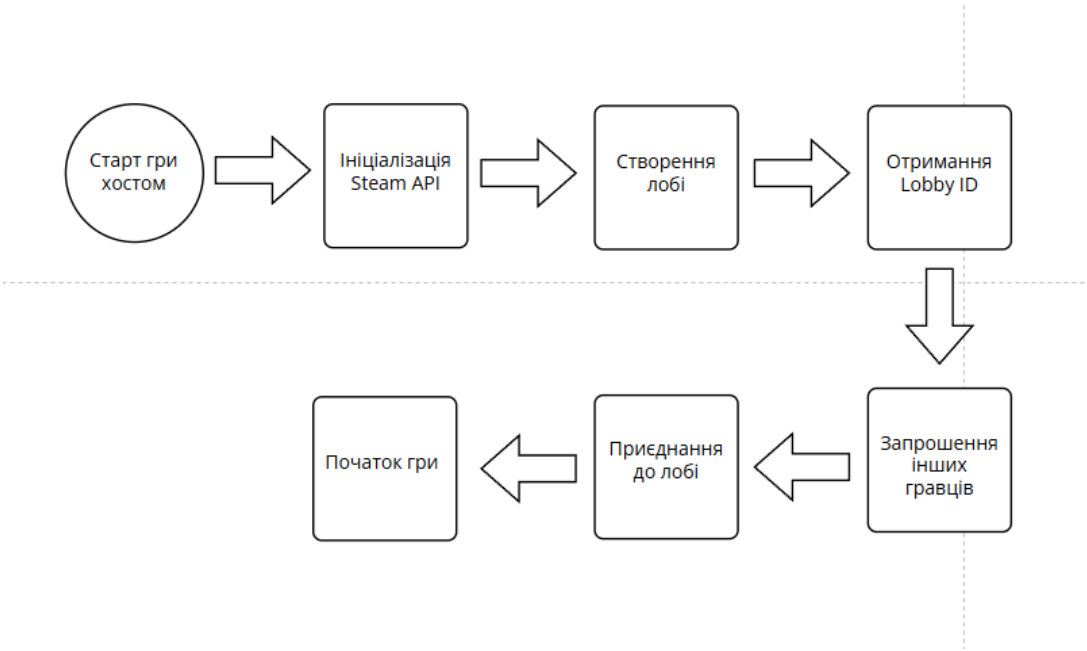

Рис. 3. Схема процесу створення багатокористувацької сесії через Steam API

У межах цієї моделі спостерігається чітке розділення обов’язків між хостом та клієнтами. Хост ініціює гру, зберігає поточний стан сесії, відповідає за перевірку коректності вхідних команд і здійснює розсилку змін іншим учасникам за допомогою викликів RPC. Клієнти, у свою чергу, підключаються до хоста, отримують оновлення стану гри та мають обмежену можливість надсилати власні дії, які проходять обов’язкову валідацію на стороні хоста. Такий підхід дозволяє забезпечити контроль цілісності даних та зменшити ризик шахрайських дій з боку клієнтів.

Однією з переваг Steam API є автоматичне встановлення P2P-з'єднань навіть у складних мережевих умовах — наприклад, за наявності NAT [8] або обмежень з боку провайдера. Steam використовує власний транспортний протокол, який забезпечує шифрування трафіку, виявлення втрат пакетів та повторну передачу при необхідності.

Для обміну даними в Godot використано механізми RPC (Remote Procedure Call) та RPC Unreliable, які дозволяють викликати методи об’єктів на інших вузлах у мережі. У реалізованій моделі статичні події, такі як спавн об’єкта, передаються за допомогою методів rpc() або rpc_id(). Натомість події,

що не є критичними до втрати, наприклад оновлення позицій, передаються через rpc_unreliable() з метою зменшення мережевого навантаження.

Усі мережеві дії централізовано обробляються в модулі Network, який ізольовано від ігрової логіки та реалізує Steam-специфічні функції. Такий підхід забезпечує гнучкість архітектури, дозволяючи у майбутньому безболісно замінити Steam API на інший транспортний рівень без внесення істотних змін до бізнес-логіки гри.

Також при розробці враховано підтримку повторного підключення клієнтів після втрати з'єднання, обробку помилок мережі — зокрема виявлення втрати пакетів та ініціацію повторної синхронізації — а також можливість пізнього підключення (late join), коли новий гравець приєднується до сесії та отримує повний поточний стан гри.

Описана модель дозволяє створити гнучку, масштабовану й ефективну архітектуру для реалізації мультиплеєрного ігрового процесу в середовищі Godot, із повною інтеграцією технологій Steam. Використання GdSteam та P2P-з'єднань забезпечує не лише функціональність, а й доступність рішення для широкого кола незалежних розробників, що робить його перспективним вибором у контексті сучасної ігрової інді-індустрії.

1.6 Реплікація стану гри та синхронізація через RPC у мультиплеєрі

Основною проблемою при створенні багатокористувацьких ігор є забезпечення синхронного бачення ігрового світу усіма учасниками. У ситуації, коли кожен гравець взаємодіє зі світом у реальному часі, будь-яка подія (будівництво, атака, переміщення, спавн юнітів) має бути передана іншим гравцям із мінімальною затримкою та гарантованою послідовністю. Саме тому у мультиплеєрі Legends of Draxxor реалізовано механізм реплікації ігрового стану з використанням віддалених процедурних викликів (RPC), доступних у рушії Godot з використанням розширення GDSteam.

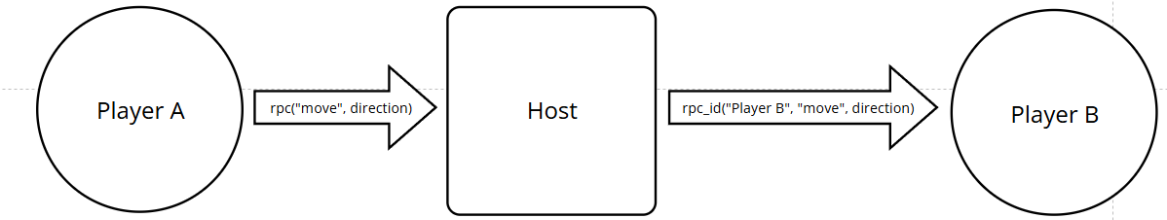

Рис. 4. Послідовність обміну командами руху між гравцями через хоста

У межах цієї роботи було використано модель із авторитетним вузлом (host-based authoritative model), за якої саме хост виконує остаточне рішення щодо зміни ігрового стану. Клієнти надсилають запити на виконання дій, однак ці дії не застосовуються до локального світу без підтвердження з боку хоста. Такий підхід дозволяє уникнути конфліктів, запобігти несанкціонованим змінам гри та забезпечити єдину, узгоджену логіку для всіх учасників.

Для передачі даних у Godot використовуються два основних методи: rpc() — виклик віддаленого методу з гарантією доставки та збереження порядку, і rpc_unreliable() — виклик без гарантії доставки, що доцільний для часто повторюваних подій, зокрема оновлення позицій ігрових об’єктів.

У практичній реалізації ці механізми розмежовуються за критичністю передаваних подій. Зокрема, створення та знищення ігрових об’єктів, таких як будівлі чи юніти, виконуються з боку хоста та передаються через rpc() для забезпечення надійності. Натомість переміщення об’єктів або зміна їхніх станів здійснюються за допомогою rpc_unreliable(), частота викликів якого залежить від типу об’єкта: для юнітів — приблизно 5 разів на секунду, для будівель — лише в момент зміни стану. Важливі зміни ігрового процесу, такі як початок гри, її завершення чи активація паузи, також передаються виключно через rpc() як надійний канал зв’язку.

Усередині клієнтської логіки реалізовано чергу очікування підтвердження .дій, що дозволяють відображати клієнту попередній візуальний результат (наприклад, початок будівництва) ще до отримання офіційного підтвердження з боку хоста. Такий підхід належить до категорії предикативного рендерингу, коли клієнт прогнозує успішне виконання дії та завчасно відображає її результат, а вже після отримання підтвердження або фіксує цю дію, або скасовує її у разі відхилення з боку серверної логіки. відхиляє (наприклад, якщо не вистачає ресурсів або таймінг порушено).

Реплікація також враховує можливість затримок та втрати пакетів, характерних для P2P-з'єднання. Для критично важливих об'єктів реалізовано повторне надсилання стану у випадку тривалого мовчання з боку клієнта, або коли підтвердження не було отримано в межах заданого тайм-ауту. У структурі модуля Network ведеться реєстр усіх активних об'єктів, які повинні бути репліковані, разом із їхніми мережевими ідентифікаторами та останнім підтвердженим станом.

Для забезпечення масштабованості реплікаційної системи було реалізовано механізм фільтрації оновлень, згідно з яким кожен клієнт отримує лише ті дані, що стосуються об'єктів, розташованих у межах його зони видимості або активної ігрової області. Такий підхід дозволяє істотно зменшити обсяг переданої інформації, оптимізувати використання мережевих ресурсів і водночас підтримувати релевантність даних, що потрапляють до клієнта. У результаті забезпечується ефективна робота системи навіть за великої кількості об'єктів на сцені або значного числа гравців у сесії.

Загалом, використання RPC як основного інструменту синхронізації у поєднанні з авторитетною моделлю, предикативним клієнтом і механізмами фільтрації реплік дозволило створити стабільну, узгоджену та ефективну систему обміну ігровими станами, придатну як для локальних матчів із невеликою кількістю учасників, так і для перспективних розширень до повноцінних матчмейкінг-систем.

1.7 Архітектурні рішення для масштабованості мультиплеєрної системи

Поняття масштабованості в контексті мультиплеєрних ігор можна розглядати як здатність системи підтримувати зростання складності або навантаження без значних змін у структурі проєкту. У грі Legends of Draxxor масштабованість розглядається як можливість розширення мережевої логіки на більшу кількість гравців, більші ігрові карти та більшу кількість об'єктів, що взаємодіють у реальному часі. Архітектура гри розроблена з урахуванням цих вимог.

Насамперед, мережева логіка винесена в окремий модуль, що дозволяє централізовано керувати всіма аспектами синхронізації, обміну даними та логікою підключення. Така декомпозиція забезпечує слабке зв’язування між мережевим рівнем і бізнес-логікою гри, що є передумовою для гнучкого масштабування без втручання у ключові геймплейні системи.

Одним із базових механізмів масштабування є система вибіркової реплікації, за якої кожен клієнт отримує лише ті дані, які безпосередньо впливають на його ігровий досвід. Це особливо актуально в умовах великих ігрових сесій, де кількість активних об’єктів (юнітів, споруд, ефектів) може сягати кількох сотень.

Ще одним критичним аспектом реалізації мультиплеєра є оптимізація частоти оновлення, яка реалізована на рівні RPC-викликів. Замість використання фіксованої частоти синхронізації для всіх об’єктів було впроваджено адаптивну схему, що враховує важливість кожного типу даних. Зокрема, юніти передають свої позиції з частотою 5–10 разів на секунду, що забезпечує плавність анімації руху без надмірного навантаження на мережу. Будівлі надсилають оновлення лише в разі зміни свого стану, тоді як незначущі події, як-от анімації чи візуальні ефекти, обробляються виключно локально без реплікації в мережі.

Крім того, у систему інтегровано менеджер ідентифікаторів, який забезпечує унікальність кожного мережевого об'єкта. Ідентифікатори генеруються авторитетним вузлом (хостом) та розповсюджуються серед інших учасників сеансу. Такий підхід дозволяє уникнути колізій ідентифікаторів при створенні об’єктів у розподіленому середовищі, водночас знімаючи потребу в додаткових перевірках на стороні клієнтів.

Особливу увагу приділено уніфікації ролей у мережевій логіці: усі клієнти реалізують один і той самий API взаємодії з мережею, незалежно від того, є вони хостом чи звичайним учасником. Перевірка ролі (is_network_authority()) здійснюється динамічно, що спрощує масштабування архітектури за рахунок уніфікованої логіки обробки подій.

Усе це дозволяє забезпечити стабільну роботу гри навіть при значному навантаженні — зростанні кількості юнітів, гравців або тривалості матчів — без погіршення продуктивності чи порушення послідовності ігрового процесу.

Таким чином, реалізована мережева архітектура в Legends of Draxxor не лише відповідає поточним вимогам проєкту, а й містить необхідну основу для масштабування, що забезпечує її життєздатність у подальших розширеннях та доповненнях до гри.

РОЗДІЛ 2
АДАПТАЦІЯ ІГРОВИХ МЕХАНІК ДЛЯ МУЛЬТИПЛЕЄРНОЇ АРХІТЕКТУРИ В УМОВАХ P2P-З'ЄДНАННЯ

2.1 Реалізація керування генералом у мультиплеєрному середовищі

Основним елементом геймплею Legends of Draxxor є керування генералом — унікальним ігровим персонажем, який виконує роль бойового юніта та командира одночасно. Із геймдизайнерської точки зору, механіка генерала повинна забезпечити гравцю постійну залученість у процес гри, дозволяючи здійснювати як стратегічні, так і тактичні дії. У контексті мультиплеєру ця система вимагає особливої уваги до питань синхронізації, реплікації та уніфікації обробки команд.

Керування генералом реалізоване за принципом point-and-click: гравець за допомогою натискання миші або відповідної кнопки надсилає команду на переміщення, після чого об'єкт починає рух до заданої позиції з урахуванням доступності шляху. У локальному середовищі ця логіка є тривіальною, однак у мультиплеєрній системі з архітектурою P2P необхідно чітко визначити, хто є авторитетом у прийнятті рішень щодо позиції генерала.

У межах реалізації гри було обрано класичну authoritative host model, за якої саме хост здійснює остаточне схвалення руху генерала. Алгоритм роботи наступний: клієнт формує запит на переміщення, який надсилається хосту через rpc_id(), хост виконує перевірку валідності дій (наприклад, чи не заблоковано шлях) та транслює фінальну позицію всім клієнтам. Завдяки такому підходу забезпечується єдиний джерело істини щодо стану генералів, що особливо важливо при наявності кількох гравців у сесії.

Схожа логіка використовується для реалізації атаки генералом. Гравець вказує ціль для атаки, після чого відправляється запит хосту. Якщо умови для атаки виконуються (наприклад, ціль у зоні досяжності, генерал не оглушений тощо), хост виконує відповідний rpc() виклик на всі клієнти, запускаючи потрібні процедури. Такий підхід унеможливлює зловживання з боку клієнтів і підвищує

чесність взаємодії між учасниками.

З технічного боку було реалізовано механізм буферизації команд, який дозволяє обробляти множинні запити від гравця в межах одного мережевого тіку. Це запобігає виникненню ситуацій, коли повторне натискання призводить до порушення безперервності руху або асинхронного положення генерала на різних клієнтах. Також у системі використовується rpc_unreliable() для проміжних позицій — наприклад, коли генерал ще в дорозі, але хост не надсилає підтвердження на кожен піксель переміщення.

Для уникнення конфліктів у мультиплеєрному середовищі реалізовано механізм ідентифікації генералів через Steam ID, який дозволяє точно визначити, який об’єкт належить якому гравцю. Таким чином, кожен клієнт має доступ лише до власного генерала і не може безпосередньо взаємодіяти з чужим об'єктом без обробки на рівні хоста.

Загалом, адаптація системи керування генералом до мультиплеєрного середовища потребувала повної декомпозиції локальної логіки та її повторного побудування з урахуванням затримок, авторитетності та принципів мережевої безпеки. Результатом стала стабільна й масштабована система, яка коректно функціонує при підключенні кількох клієнтів, забезпечуючи одночасно комфортний контроль для гравця та коректну синхронізацію в межах P2P-архітектури.

2.2 Реплікація унікальних споруд генерала в мультиплеєрному середовищі

Важливою особливістю геймплею Legends of Draxxor є наявність унікального набору споруд, доступного кожному генералу залежно від раси та вибраної гілки розвитку. Ці споруди виконують критичні функції в межах бою: спавн юнітів, оборонні дії або активацію особливих властивостей. У мультиплеєрному контексті ключовим завданням є правильна реплікація створення, стану та взаємодії споруд між усіма учасниками гри, причому з

урахуванням того, що кожен гравець має власний набір доступних споруд і самостійно приймає рішення щодо їх розміщення в межах дозволеної зони.

Архітектурно споруди реалізовані як об'єкти, які мають унікальний мережевий ідентифікатор, прив'язаний до Steam ID власника. Це дозволяє хосту точно визначити, яка будівля належить якому гравцю та які правила до неї мають бути застосовані. При побудові споруди клієнт надсилає хосту rpc_id() [2] запит, який містить тип будівлі, координати, вибрані параметри та додаткові характеристики (наприклад, модифікатори раси або прокачки). Хост здійснює валідацію запиту: перевіряє наявність ресурсів, допустимість позиції, відсутність конфліктів з іншими об’єктами. У разі успішної перевірки хост викликає rpc() [2] на всі клієнти, ініціюючи створення будівлі на клієнтах усіх гравців.

Особливу увагу приділено часу активації споруд. Через різницю в затримках між клієнтами було реалізовано механізм маркера часу створення, який уніфікує момент появи споруди для всіх учасників. Завдяки цьому всі властивості, пов’язані зі спорудою (наприклад, спавн хвиль, генерація аури або зона впливу), починають діяти одночасно для кожного гравця, попри можливі мережеві коливання.

Унікальні ефекти споруд (наприклад, постійне посилення союзників у радіусі) не розраховуються на клієнтах — вся логіка залишається на хості, а результати передаються лише у вигляді наслідків (наприклад, зміна параметрів). Це дозволяє знизити об’єм трафіку та запобігти можливості маніпулювання локальними параметрами зі сторони клієнту.

Для побудови споруди також передбачено попередній локальний перегляд (ghost-режим) — коли гравець бачить прозору модель споруди в обраній позиції ще до підтвердження від хоста. Це реалізовано локально і не має жодного впливу на фактичний стан гри до моменту підтвердження, що забезпечує плавність інтерфейсу та зручність для користувача.

Реплікація споруд у мультиплеєрному режимі охоплює кілька ключових аспектів, зокрема синхронізацію стану пошкодження, знищення об’єкта та його

прокачку. Зміни в здоров'ї будівель передаються за допомогою rpc_unreliable() із частотою не більше ніж чотири рази на секунду для зберігання балансу між точністю та економією мережевих ресурсів. Видалення будівлі ініціюється виключно хостом шляхом виклику rpc("destroy_structure", id), що гарантує узгоджене знищення об'єкта на всіх клієнтських копіях гри. У випадку покращення споруди відповідна дія проходить перевірку на сервері, після чого викликається rpc() із оновленими параметрами, що забезпечує коректне застосування змін для всіх учасників сесії.

Також реалізовано локальну реєстрацію об'єктів, коли при приєднанні нового гравця (late join) хост передає йому поточний список усіх споруд на мапі, їхні типи, позиції та стани. Це дозволяє забезпечити повну синхронізацію навіть у ситуаціях, коли гравець підключається в активну фазу гри.

Таким чином, реалізована система реплікації споруд забезпечує не лише точність і надійність мережевої синхронізації, але й враховує особливості ігрової логіки, пов'язаної з расовою унікальністю та прокачуванням генерала. Побудова відбувається під повним контролем хоста, що дозволяє уникнути потенційних проблем із розсинхронізацією та підтримати чесність гри в умовах P2P-архітектури.

2.3 Система ресурсів та її синхронізація в мультиплеєрному середовищі

Ресурсна система є фундаментальною складовою будь-якої стратегії, оскільки саме вона визначає темп гри, економічну модель і можливість розвитку гравця. У грі Legends of Draxxor реалізовано три основні типи ресурсів: золото, дерево та ядро. Кожен із них має свою специфіку отримання, призначення та правила накопичення, що обумовлює потребу в точній синхронізації цих даних між усіма учасниками багатокористувацької сесії.

Нарахування ресурсів у мультиплеєрі не може здійснюватися локально на кожному клієнті, оскільки це створює ризик розбіжностей, шахрайства або розсинхронізації. У зв'язку з цим було впроваджено централізовану модель

обліку, в межах якої лише авторитетний вузол (хост) має право змінювати стан ресурсів кожного гравця. Клієнти, у свою чергу, надсилають запити на зміну (наприклад, після вбивства юніта), але фактичне оновлення балансу відбувається лише після підтвердження з боку хоста.

Золото — це основний бойовий ресурс, що нараховується за знищення ворожих юнітів. У момент смерті ворожого об'єкта хост визначає, хто є ініціатором атаки, і викликає rpc_id() до цього гравця з вказанням кількості золота для зарахування. Щоб зменшити мережеве навантаження, подібні транзакції можуть групуватися в пакет оновлень, який надсилається кожні кілька секунд, а не після кожного вбивства.

Дерево є ресурсом будівельної економіки, що нараховується гравцю при спорудженні певних типів будівель або виконанні дій, пов’язаних із розвитком інфраструктури. Оскільки саме гравець ініціює ці дії, система має забезпечити обов’язкову перевірку валідності кожної транзакції. Хост перевіряє, чи справді гравець здійснив легальне будівництво, і тільки після цього надсилає оновлення стану дерева через rpc().

Ядро — це унікальний, елітний ресурс, що видається лише за перемогу над спеціальним босом. У межах мережевої архітектури реалізовано індивідуальний контроль кожного боса для кожного гравця. Коли гравець перемагає власного боса, ця подія фіксується лише локально, однак валідується на хості. Лише після верифікації до балансу додається один елемент ресурсу «ядро», що синхронізується зі всіма гравцями для відображення лічильника, але доступ до використання має лише власник.

Для зберігання даних про ресурси використовується окремий ResourceManager, який прив’язаний до Steam ID кожного гравця. Цей менеджер веде журнал змін, дозволяє швидко оновити інформацію по запиту (наприклад, при reconnect або late join) та забезпечує стійкість до розривів з’єднання.

Крім того, реалізовано механізм періодичного пінгу ресурсу, який дозволяє клієнту звірити свій локальний відображений баланс із хостовим. Якщо відбувається розбіжність через затримки чи втрату пакету, клієнт запитує

оновлення через rpc_id() та отримує актуальний стан. Це особливо важливо в динамічних ситуаціях — наприклад, при миттєвому накопиченні ресурсу під час хвилі атак.

Таким чином, вся система ресурсів побудована з урахуванням мережевої цілісності, централізованого контролю та захисту від розсинхронізації. Обрані підходи забезпечують стабільність гри, навіть при високій частоті транзакцій або у складних мережевих умовах, і дозволяють надалі масштабувати ресурсну економіку гри з мінімальними змінами в архітектурі.

2.4 Система розвитку генерала та її мережеве відображення

У грі Legends of Draxxor кожен генерал має можливість поступової еволюції за допомогою гілок прокачки, які відкривають нові здібності, покращують характеристики або розблоковують доступ до унікальних споруд. Попри відносну простоту реалізації з боку геймплею, дана система вимагає чіткої координації між клієнтом та хостом у мультиплеєрному середовищі для забезпечення коректності та однозначності вибору прокачки.

Прокачка ініціюється гравцем через інтерфейс, але сама дія не виконується локально. Замість цього клієнт надсилає запит хосту із зазначенням вибраної гілки, пункту та поточного рівня. Хост перевіряє валідність запиту (наявність доступного рівня, правильність порядку прокачки, ресурсну можливість) і лише після цього підтверджує оновлення через виклик rpc() до всіх клієнтів.

Внутрішньо, система розвитку реалізована як серія унікальних ідентифікаторів, що відповідають вузлам дерева прокачки. Це дозволяє безпомилково ідентифікувати обрані гравцем опції, незалежно від локалізації чи порядку.

Оскільки прокачка генерала передбачає не поступову зміну його характеристик, а повну заміну об’єкта на новий, який відповідає обраній гілці розвитку, обробка такої події має відбуватися централізовано. Після вибору

гравцем напрямку прокачки клієнт надсилає запит хосту із вказаним типом нового генерала. Хост перевіряє коректність запиту, після чого здійснює заміну об’єкта генерала на новий екземпляр, який відповідає тій самій расі, але має змінений набір здібностей або споруд. Процес заміни реплікується на всі клієнти за допомогою rpc(), що забезпечує підтримку узгодженого стану гри для всіх учасників.

Таким чином, система розвитку генерала інтегрована в мережеву архітектуру [7] через запити та підтвердження, що унеможливлює розсинхронізацію станів або використання несанкціонованих покращень. Вона зберігає баланс між швидкою реакцією гравця та необхідною мережею валідацією, що забезпечує стабільність ігрового процесу.

2.5 Обробка ігрових подій у мультиплеєрному середовищі: принципи синхронізації та централізації

У процесі функціонування багатокористувацької гри виникає низка ігрових подій, які, хоча й не належать до базових механік наприклад руху, атаки чи будівництва, але мають суттєве значення для узгодженості ігрового процесу. Йдеться про події, пов’язані з фазами гри (початок і кінець, перехід між режимами), глобальними подіями (наприклад, поява боса чи завершення матчу), також про тригери, що ініціюють важливі логічні процеси (наприклад, старт або появу босів), а в окремих випадках — супроводжуються візуальними ефектами на клієнтській стороні. Усі такі події потребують координованої обробки з боку хоста, щоб забезпечити узгоджене та непротирічне сприйняття поточного стану сесії.

У мультиплеєрному контексті, особливо в умовах однорангової архітектури з авторитетним хостом, критично важливо, щоб події мали єдине джерело активації. З цією метою в Legends of Draxxor було реалізовано систему централізованої обробки подій, в якій виняткове право на ініціацію має хост, тоді як інші клієнти виступають лише отримувачами цієї інформації. Така модель

дозволяє уникнути розбіжностей між клієнтами та забезпечує послідовність у відображенні динамічних змін.

З технічної точки зору, кожна подія має власну унікальну ідентифікацію та параметри, які передаються через RPC. Наприклад, поява глобального боса ініціюється хостом, який надсилає rpc("spawn_boss") всім клієнтам. Це є сигналом для запуску відповідних візуальних ефектів, логіки бою та оновлення інтерфейсу. У разі індивідуальних подій, як-от отримання нагороди за вбивство боса, використовується адресна передача через rpc_id(), що дозволяє надіслати інформацію виключно відповідному гравцеві.

Особливу увагу приділено синхронізації глобальних фінальних подій, таких як завершення гри або колективна перемога. У цих випадках хост здійснює одночасну розсилку через rpc("end_game", status), що забезпечує одномоментне оновлення інтерфейсу, завершення сесії та блокування введення на всіх клієнтах.

З метою підвищення стійкості системи реалізовано також підтримку відкладених подій для гравців, які підключаються пізніше. Під час приєднання нового клієнта хост автоматично надсилає йому актуальний список активних глобальних подій, які перебувають у статусі виконаних або таких, що тривають. Це дає змогу новому учаснику миттєво синхронізуватися з поточним станом гри, забезпечуючи цілісність і послідовність ігрового процесу без втрати логічного контексту.

Таким чином, обробка ігрових подій у Legends of Draxxor здійснюється на основі принципів централізації, однозначної авторитетності та гарантованої доставки, що забезпечує узгоджене функціонування всіх систем гри незалежно від кількості підключених користувачів, поточної фази сесії або якості мережевого з’єднання. Такий підхід дозволяє досягти високого рівня стабільності, надійності та масштабованості в межах реалізованої архітектури.

РОЗДІЛ 3
ОПТИМІЗАЦІЯ МУЛЬТИПЛЕЄРНОЇ ВЗАЄМОДІЇ ТА ПІДГОТОВКА ДО МАСШТАБУВАННЯ

3.1 Методика тестування мультиплеєрної частини гри

Ефективність та стабільність мультиплеєрної взаємодії [6] є критичними факторами для успішного функціонування будь-якої багатокористувацької гри. Під час реалізації мережевої архітектури Legends of Draxxor, яка базується на P2P-з'єднанні з використанням Steam API та GdSteam, було розроблено спеціалізовану методику тестування, орієнтовану на виявлення потенційних проблем синхронізації, перевірку коректності обробки RPC-викликів та оцінку стабільності з'єднання між учасниками.

Тестування проводилося в умовах локальної емуляції багатокористувацького режиму, а також через реальні Steam-акаунти з підключенням із різних пристроїв у межах локальної мережі та інтернету. Це дозволило моделювати ситуації, наближені до реального користувацького досвіду, і врахувати типові мережеві умови: різні затримки, втрати пакетів, перепідключення тощо.

Процедура тестування включала в себе кілька послідовних етапів, спрямованих на перевірку стабільності, коректності логіки та витривалості мережевої взаємодії. На першому етапі здійснювалась перевірка початкового підключення: моделювались сценарії створення лобі, приєднання клієнтів, затримки у відповіді Steam API, а також відмова одного з клієнтів на етапі запуску сесії. Далі проводилось тестування базових дій у грі — зокрема, перевірялась коректність руху генерала, обробка атак, будівництво споруд, прокачка та отримання ресурсів у процесі активної гри з двома або трьома клієнтами.

Окрему увагу приділено моделюванню втрати зв'язку. Було вручну відтворено короткострокові обриви з'єднання одного з клієнтів із подальшим перепідключенням, а також змодельовано повне відключення хоста з аналізом

реакції клієнтів на втрату авторитетного вузла. На завершальному етапі проведено навантажувальне тестування, під час якого на сцені створювались ситуації з великою кількістю активних об’єктів (понад 100 юнітів). Це дозволило оцінити стабільність виконання RPC-викликів, частоту реплікацій і відповідність ігрової логіки на всіх клієнтах у стресових умовах.

Для фіксації результатів використовувалась внутрішня система логування, яка зберігала ключові події (початок гри, отримання команд, зміна ресурсів, помилки з’єднання тощо) у вигляді журналів. Це дозволило аналізувати причини некоректної поведінки гри у складних мережевих умовах.

Особливу увагу було приділено перевірці цілісності ігрового стану, тобто відповідності параметрів на хості та клієнтах після проходження певного періоду гри. Було визначено набір критичних точок синхронізації, які відслідковувалися вручну або через вбудовані assert-умови (наприклад, відповідність кількості споруд, кількості золота або статусу прокачки генерала).

Таким чином, створена методика тестування дозволила комплексно перевірити працездатність основних механік мультиплеєра та забезпечити необхідну впевненість у стабільності та коректності їх функціонування в умовах, наближених до реального використання. Результати тестування стали основою для подальшої оптимізації, описаної в наступних підпунктах.

3.2 Виявлені проблеми та оптимізація мережевої взаємодії в умовах P2P-архітектури

У процесі практичного тестування мультиплеєрної частини гри Legends of Draxxor було зафіксовано низку особливостей та проблемних аспектів, які мали безпосередній вплив на якість ігрового процесу, стабільність з’єднання та коректність синхронізації станів між клієнтами. Урахування цих моментів стало основою для подальшої оптимізації мережевої архітектури та адаптації ключових механік до умов роботи з реальними користувачами в межах P2P-моделі з авторитетним хостом.

Однією з найпоширеніших проблем, яка виникала під час взаємодії між клієнтами, була затримка оновлення стану об'єктів на стороні гравців. У деяких випадках (особливо при високому навантаженні) клієнти отримували сповіщення про події (рух генерала, побудову споруди, отримання ресурсу) із помітною затримкою, що негативно позначалося на суб’єктивному відчутті плавності гри. Проведений аналіз показав, що основною причиною цього є надмірна частота викликів rpc() для подій, які не потребували миттєвої реплікації.

З метою оптимізації було впроваджено динамічне регулювання частоти оновлень, яке дозволяє змінювати інтервали надсилання мережевих подій залежно від поточної ситуації. Зокрема, у стані спокою (коли об’єкти не рухаються і не взаємодіють) частота rpc_unreliable() знижується, а під час бою — збільшується до допустимих меж. Такий підхід дозволив зменшити загальне навантаження на мережу без втрати ігрової точності.

Ще однією виявленою проблемою стала вразливість до втрати порядку пакетів у деяких сценаріях, зокрема при швидкій послідовності дій, що супроводжуються кількома запитами на реплікацію. У відповідь на це було реалізовано систему контролю послідовності подій, яка відстежує порядкові номери критичних дій (наприклад, прокачка генерала, отримання ядра) та ігнорує дублікати або застарілі повідомлення.

Крім того, під час тривалих сесій було зафіксовано накопичення непотрібних об’єктів у сцені, що створювали додаткове навантаження на клієнтську сторону. Це стосувалося як невикористаних ефектів, так і застарілих референсів до знищених об’єктів. Для вирішення цієї проблеми було впроваджено механізм автоматизованого очищення ресурсів, що більше не беруть участі в активній взаємодії, з обов’язковим попереднім підтвердженням з боку хоста.

Рис. 5. Фільтрація оновлень на основі зони видимості

Також у ході аналізу мережевого трафіку було виявлено надмірне дублювання запитів на спавн однакових об'єктів, що в окремих ситуаціях призводило до некоректної реплікації. Це було зумовлено тим, що окремі клієнти повторно надсилали запити в разі затримки отримання підтвердження від хоста, що призводило до дублювання дій. Для усунення цієї проблеми було запропоновано та реалізовано механізм кешування очікуваних подій. Його суть полягає в тимчасовому блокуванні повторної ініціації однієї й тієї самої дії протягом визначеного інтервалу часу, поки клієнт очікує на відповідь від хоста. Такий підхід дозволяє зменшити кількість зайвих запитів і підвищити узгодженість взаємодії в умовах нестабільного мережевого з'єднання. Таким чином було значно зменшено кількість зайвих повідомлень у мережі та покращено стабільність реплікації.

Особливу увагу було приділено реалізації механізмів обробки помилок підключення, зокрема втрати з'єднання клієнта з хостом. Було запроваджено просту, але ефективну схему виявлення таймауту, яка дозволяє хосту ідентифікувати клієнтів, що втратили з'єднання, й ініціювати механізм повідомлення інших учасників або навіть перепідключення. Це рішення підвищило стійкість системи до типових збоїв у мережевій інфраструктурі.

Загалом, проведена оптимізація суттєво покращила роботу мультиплеєрного компонента. Вдалося зменшити затримки під час реплікації об'єктів, знизити мережеве навантаження в умовах пікової активності, а також забезпечити надійніший облік ігрових подій та розподілу ресурсів між клієнтами. Крім того, було підвищено стійкість гри до мережевих збоїв, що дозволяє зберігати цілісність ігрового процесу навіть у разі тимчасових втрат з'єднання.

Таким чином, на основі виявлених у процесі тестування проблем було реалізовано низку технічних рішень, які суттєво покращили якість мультиплеєрної взаємодії та створили базу для подальшого розширення функціоналу в межах існуючої архітектури.

3.3 Підготовка до масштабування мультиплеєра в майбутніх ітераціях проєкту

Масштабованість є одним із ключових принципів, який повинен закладатися в архітектуру будь-якої багатокористувацької гри ще на етапі її початкового проєктування. Незалежно від того, чи йдеться про невеликий PvP-проєкт для кількох гравців, чи про потенційно масову мережеву гру, що підтримує десятки й сотні одночасних учасників, структурна готовність до масштабування дозволяє уникнути необхідності повного переписування коду з часом. У випадку Legends of Draxxor, підготовка до масштабування розглядалася не як окремий етап, а як невід'ємна частина побудови мультиплеєрної системи з самого початку розробки.

Передусім, одним із основних рішень, що забезпечують потенціал до масштабування, є ізоляція мережевої логіки в окремих модулях. Уся робота з мережею — ініціація з’єднання, реплікація об’єктів, передача ресурсів, синхронізація дій — винесена до спеціалізованого NetworkManager. Це дозволяє реалізовувати інші моделі з’єднання (наприклад, перехід від P2P до клієнт-серверної моделі) без суттєвого втручання в основну ігрову логіку. Більшість ігрових об’єктів та механік взаємодіють із мережею через чітко визначені публічні інтерфейси, що дає змогу легко підмінювати реалізацію згодом.

Крім того, усі мультиплеєрні дії реалізовано через єдину точку входу — виклики rpc() та rpc_id() проходять попередню обробку через проміжну логіку в модулі Network, що дозволяє з часом впровадити додаткові механізми, такі як шифрування, верифікація підписів або навіть черги повідомлень для асинхронної обробки. Такий рівень гнучкості є критично важливим, коли йдеться про

масштабування гри на більшу кількість гравців, складніші топології або зміну способу доставки даних (наприклад, перехід на dedicated-сервери).

З технічної точки зору, кожен тип об’єкта у грі має унікальний мережевий ідентифікатор, який дозволяє підтримувати однозначне адресування навіть у великих сесіях. Ідентифікатори генеруються централізовано хостом, а потім передаються всім клієнтам, що унеможливлює конфлікти при створенні об’єктів у розподіленому середовищі. При зростанні кількості об'єктів — юнітів, будівель, ефектів — така модель дозволяє підтримувати логічну цілісність без перевірки на конфлікти кожного разу.

Також було реалізовано систему просторової фільтрації реплікацій, що є необхідним кроком для масштабування. Вона дозволяє кожному клієнту отримувати лише ті оновлення, які стосуються об’єктів, розташованих у межах його поля зору або зони впливу.

Наприклад, якщо гравець не бачить ворожу базу, то зміни, які там відбуваються, не надсилаються його клієнту. Це дозволяє суттєво зменшити обсяг трафіку при великій кількості учасників і зробити масштабні матчі реально досяжними без критичного просідання продуктивності.

У межах внутрішньої логіки гри також реалізовано низку рішень, що сприяють масштабованості системи. Зокрема, кожен гравець має власну ізольовану логіку керування генералом, що дає змогу обробляти десятки паралельних процесів прокачки без виникнення конфліктів між гравцями. Уся інформація, пов’язана з ресурсами, подіями та спорудами, зберігається у структурованих словниках, ключами в яких виступають Steam ID користувачів. Такий підхід дозволяє масштабувати систему до практично необмеженої кількості учасників без істотного ускладнення архітектури коду. Крім того, система логічних подій, таких як спавн босів або завершення матчу, реалізована через подієво-орієнтовану модель з використанням черг і відкладених викликів, що забезпечує її гнучкість та адаптивність до зростання навантаження.

Варто також відзначити, що реалізовано набір внутрішніх діагностичних засобів, які дозволяють відслідковувати навантаження на клієнта в режимі

реального часу. Наприклад, підрахунок активних RPC-викликів, трекінг використаної пам’яті об'єктів, затримки між отриманням та обробкою повідомлень. У майбутньому ці інструменти можуть стати основою для побудови автоматизованої системи балансування навантаження у великомасштабних матчах.

Підготовка до масштабування також враховує майбутню можливість використання спеціалізованого серверного рішення. Попри те, що на даний момент реалізація базується на P2P-моделі, всі виклики побудовані таким чином, щоб їх можна було легко перенести на dedicated-сервер, який обслуговуватиме логіку сесій, зберігатиме статистику, виконуватиме автентифікацію гравців тощо. Це дозволяє розглядати Legends of Draxxor не лише як локальний мультиплеєрний проєкт, а як основу для розгортання повноцінного онлайнового середовища.

Таким чином, архітектура проєкту не тільки відповідає поточним потребам, але й має вбудовану здатність до еволюції, що забезпечує її життєздатність у довгостроковій перспективі. За рахунок структурної гнучкості, централізованого контролю, просторового поділу обов’язків і правильно організованих мережевих шарів, система мультиплеєра Legends of Draxxor підготовлена до подальшого масштабування як у кількісному, так і в якісному вимірі.

3.4 Архітектура та логіка багатокористувацького режиму: підключення, взаємодія, синхронізація

Для ознайомлення з функціоналом мультиплеєрного режиму гри Legends of Draxxor було створено базову інструкцію користувача, яка покликана допомогти гравцям швидко орієнтуватися в інтерфейсі, механіках і послідовності дій у межах ігрової сесії. У цьому розділі розглянуто порядок підключення до гри, взаємодію з основними мережевими функціями, а також особливості мультиплеєрної логіки, реалізованої в межах цієї кваліфікаційної роботи.

Після запуску гри користувачеві доступний інтерфейс, що дозволяє створити нову сесію або приєднатися до вже існуючої. Створення сесії автоматично призначає гравцеві роль хоста, який відповідає за централізовану синхронізацію стану гри, обробку подій та поширення інформації до всіх учасників. Усі підключення здійснюються через Steam API з використанням GdSteam: гравці можуть приєднуватися до гри, обираючи друга зі списку контактів або використовуючи інвайт-посилання. У обох випадках клієнт автоматично встановлює з’єднання з активною сесією.

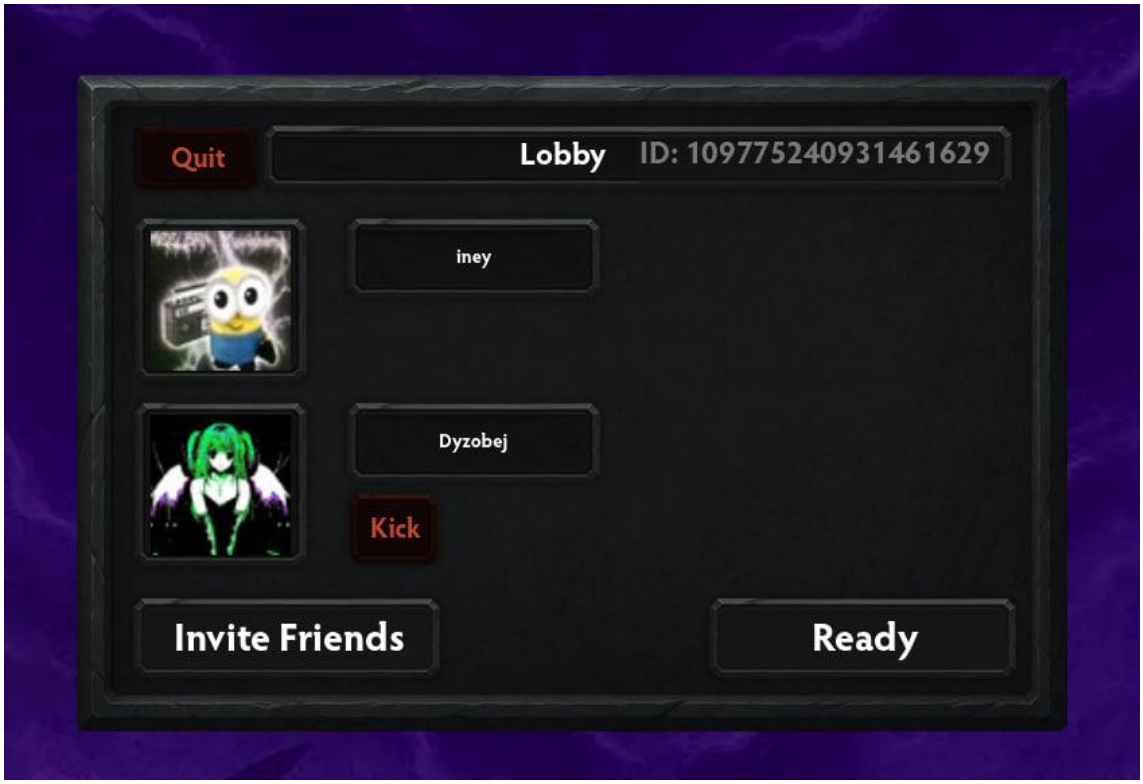

Рис. 6. Інтерфейс лобі мультиплеєрної сесії з підтримкою Steam API

Після підключення кожен гравець переходить до етапу вибору раси, що визначає доступний набір споруд, бойових одиниць і здібностей генерала. Вибір здійснюється через інтуїтивне меню, після чого результат фіксується системою і не підлягає зміні в межах поточної сесії, що сприяє дотриманню ігрового балансу.

Після підтвердження вибору всі гравці синхронно переміщуються до стартової зони карти, де розпочинається основна фаза гри. Серед базових

механік реалізовано будівництво споруд, керування генералом, збір ресурсів та прокачку. Будівництво передбачає вибір типу споруди (спавнери, оборонні структури, посилювальні або послаблювальні ефекти) з подальшим розміщенням на доступному полі; відповідні дії передаються через RPC-хост, який створює об’єкт у всіх клієнтів. Керування генералом реалізовано за принципом point-and-click, аналогічно до ігор типу Dota, із можливістю автоматичних або ручних атак. Ресурси гравець отримує за знищення ворожих юнітів (золото), спорудження будівель (дерево) та перемоги над босами (ядро). Коли гравець накопичує достатню кількість ресурсів, стає доступною прокачка генерала — процес, у межах якого поточний об’єкт замінюється на вдосконалену версію з новими здібностями.

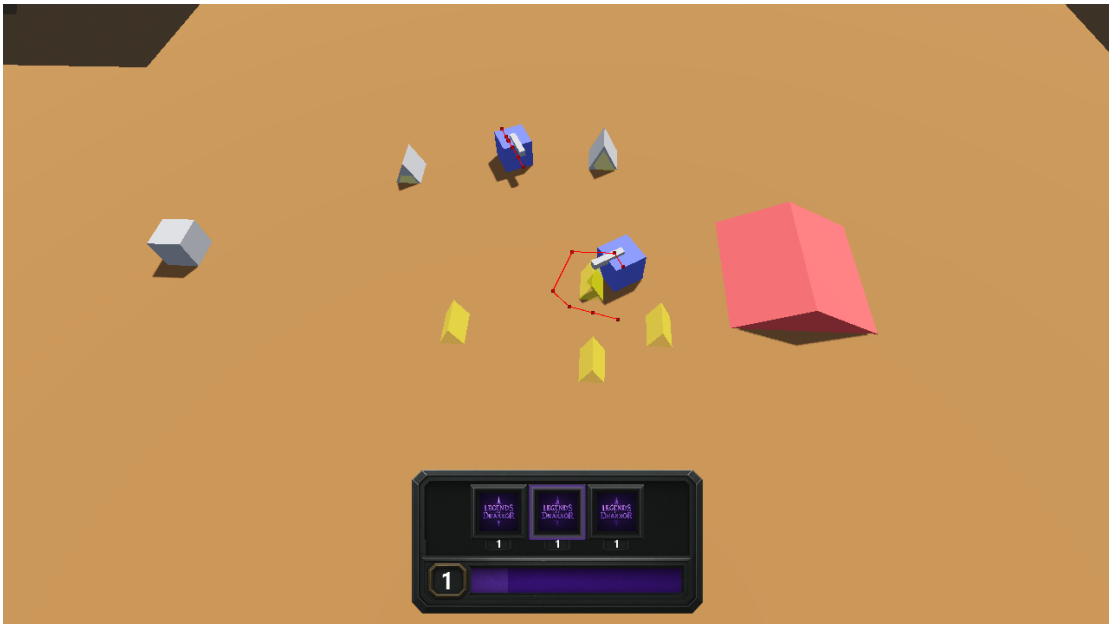

Рис. 7. Синхронізація дій двох гравців на спільній сцені з відображенням генералів, юнітів та інтерфейсу прокачки

У разі втрати з’єднання з хостом гравець отримує відповідне повідомлення з можливими діями. Поточна реалізація підтримує автоматичне повторне підключення до сесії, якщо вона залишилася активною. У майбутньому планується впровадження механізмів резервного збереження та відновлення стану гри з бекапу.

Завершення сесії відбувається після досягнення однієї з умов перемоги: знищення головної бази супротивника або повна поразка всіх ворожих гравців. У такому випадку хост надсилає команду завершення, після чого клієнти автоматично переходять у головне меню та отримують зведену статистику матчу. У перспективі передбачається реалізація збереження результатів та можливість перегляду історії ігор у профілі користувача.

РОЗДІЛ 4
ПЕРСПЕКТИВИ РОЗВИТКУ ГРИ LEGENDS OF DRAXXOR.

Розробка гри Legends of Draxxor у мультиплеєрному форматі не обмежується лише базовим функціоналом, реалізованим у межах кваліфікаційної роботи. Важливою особливістю сучасного підходу до створення ігор є поступове розширення можливостей, впровадження нових механік, вдосконалення мережевої архітектури та адаптація до нових платформ. У цьому розділі висвітлюються основні напрямки майбутнього розвитку проєкту, які дозволять суттєво підвищити його якість, привабливість для гравців та технічну стійкість у масштабованому середовищі.

4.1 Додаткові ігрові режими

У перспективі передбачається розширення функціоналу гри шляхом впровадження нових режимів, які виходять за межі класичного сценарію знищення бази супротивника. Серед потенційних режимів варто відзначити кооперативну гру проти штучного інтелекту, у якій гравці об’єднуються для відбиття хвиль ворожих ботів і захисту спільного ядра. Також планується реалізація арени формату 1v1v1 — три гравці одночасно змагаються за домінування на полі бою, а переможцем стає той, чия база залишиться неушкодженою. Окремо розглядається можливість впровадження марафонного режиму — нескінченної сесії з поступовим ускладненням ворогів, яка дозволяє оцінити витривалість обраної стратегії та гнучкість тактичних рішень.

Кожен із цих режимів вимагає адаптації мережевої архітектури до нових типів подій та механік, включно з повторним використанням RPC, збереженням прогресу та балансуванням AI.

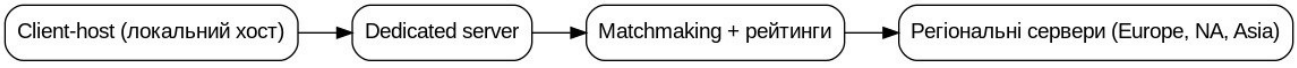

Рис. 8. Очікувана еволюція архітектури мережевої інфраструктури гри

4.2 Розширення інтеграції зі Steam

Хоча основна інтеграція зі Steam API [3] вже реалізована за допомогою бібліотеки GdSteam, у подальшому передбачається розширення функціональних можливостей шляхом додавання нових сервісів платформи. Зокрема, планується впровадження системи досягнень (Achievements), яка дозволить створити додаткову мотивацію для гравців шляхом відкриття внутрішніх нагород за виконання певних дій. Хмарні збереження (Cloud Saves) забезпечать можливість зберігати прогрес і продовжувати гру з іншого пристрою без втрат. Також передбачається реалізація лідербордів (Leaderboards), що додасть грі змагального елементу через публічні рейтинги.

У перспективі розглядається підтримка Steam Workshop як засобу для розміщення модифікацій, кастомних генералів або унікальних наборів споруд, створених спільнотою. Такий підхід сприятиме не лише утриманню гравців у межах проєкту, а й активному залученню користувачів до його розвитку через створення та обмін власним контентом.

4.3 Візуальне вдосконалення та оптимізація

Окремим напрямком подальшого розвитку проєкту є вдосконалення графічної складової гри, що безпосередньо впливає на якість користувацького досвіду. Серед ключових покращень передбачається впровадження візуальних ефектів, які супроводжуватимуть атаки, прокачку генерала та події, що відбуваються у реальному часі. Також важливим є удосконалення інтерфейсу користувача з урахуванням різних розширень екранів, що дозволить адаптувати гру для ширшого кола пристроїв. Крім цього, планується додавання анімацій, плавних переходів і реактивних елементів UI, які підвищують динамічність і загальну привабливість візуального представлення гри.

Поліпшення візуального контенту є особливо важливим у мультиплеєрному режимі, де поряд із геймплейною механікою значну роль відіграє зручність сприйняття подій, швидке орієнтування в інтерфейсі та загальний естетичний рівень проєкту.

4.4 Підтримка мобільних пристроїв та кросплатформеність

У довгостроковій перспективі планується адаптація гри для Android та iOS-платформ. Завдяки кросплатформеній природі Godot Engine це можливо без повного переписування логіки. Також передбачається підтримка гри між мобільними та десктопними гравцями (cross-play), що значно розширює потенційну аудиторію.

У майбутньому може бути реалізована й підтримка контролерів, що дозволить гравцям грати з геймпадами, підвищуючи комфорт та доступність гри.

4.5 Турнірна та рейтингова система

Для підтримки конкурентного духу гравців у планах — реалізація системи рейтингу з автоматичним підбором супротивників (matchmaking). Передбачається створення сезонної таблиці, де гравці отримують ранги за перемоги. Додатково можуть проводитися онлайн-турніри зі збереженням статистики в профілі Steam.

4.6 Підвищення масштабованості мережевої архітектури

Поточна реалізація побудована на P2P із визначеним хостом. У подальшому можливе масштабування до dedicated-серверів, що дозволить уникнути втрат сесій при виході хоста. Також розглядається впровадження відкладених сесій, резервного хоста, логування боїв та аналітики дій гравця для вдосконалення балансу гри.

Таким чином, Legends of Draxxor має значний потенціал для масштабування як з технічного, так і з контентного боку, що робить проєкт перспективним з точки зору розвитку незалежної гри із мультиплеєрним ядром.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було успішно досягнуто поставленої мети — спроєктовано та реалізовано багатокористувацький режим для гри «Legends of Draxxor» у жанрі стратегії в реальному часі з використанням ігрового рушія Godot Engine. Розроблена архітектура забезпечує можливість взаємодії кількох гравців у спільному ігровому середовищі, відповідаючи вимогам до стабільності з'єднання, синхронізації стану гри та продуктивності системи.

Практична значущість отриманого результату полягає в тому, що запропонований підхід та впроваджені технічні рішення демонструють ефективний шаблон для розробки багатокористувацьких RTS-проєктів на Godot Engine з інтеграцією Steam-сервісів. Накопичений досвід та сформовані рішення можуть бути використані як базовий шаблон для майбутніх інді-ініціатив, орієнтованих на мультиплеєрну взаємодію в реальному часі.

У результаті проведеної роботи було окреслено теоретичні засади проєктування мережевих архітектур у контексті багатокористувацьких ігор. Детально проаналізовано принципи функціонування P2P-з'єднань, механізм синхронізації через Remote Procedure Calls (RPC), а також проблематику NAT. Окрему увагу приділено вивченню Steam API та його інтеграції з рушієм Godot Engine за допомогою плагіна GdSteam, що дозволило забезпечити базову інфраструктуру для побудови мультиплеєрного середовища.

На основі отриманих теоретичних засад було реалізовано базові ігрові механіки з урахуванням особливостей мультиплеєрної взаємодії. Зокрема, впроваджено логіку синхронізації генералів, обробки бойових дій, будівництва споруд і системи прокачки персонажів. Особливий акцент зроблено на забезпеченні надійного обміну даними між клієнтами та хостом із урахуванням можливих затримок і втрат мережевого з'єднання, що дозволило підтримувати цілісність ігрового процесу.

Проведено комплексне тестування реалізованої системи, яке включало моделювання різноманітних мережевих сценаріїв, навантажувальне тестування та профілювання. Це дало змогу виявити вузькі місця архітектури й оптимізувати окремі компоненти, що позитивно вплинуло на стабільність та продуктивність гри.

Окремо розроблено рекомендації щодо майбутнього масштабування проєкту. Запропоновано модульний підхід до побудови логіки, уніфікацію мережевих запитів і відокремлення бізнес-логіки від клієнтської візуалізації. Також окреслено перспективи розвитку, зокрема впровадження матчмейкінгу, рейтингової системи, турнірного режиму та інших елементів соціальної взаємодії, що можуть значно розширити потенціал гри.

Загалом результати виконаної роботи підтверджують практичну ефективність використання GdSteam у поєднанні з Godot Engine для розробки багатокористувацьких RTS-проєктів. Запропонована архітектура не лише відповідає вимогам початкового рівня складності, а й створює надійне підґрунтя для подальшого розвитку та масштабування системи, що є критично важливим для життєвого циклу сучасних онлайн-ігор.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Godot Engine – Documentation [Електронний ресурс]. – Режим доступу: https://docs.godotengine.org – Дата звернення: 12.06.2025.
2. GdSteam – Godot Steam Integration Plugin [Електронний ресурс]. – Режим доступу: https://github.com/Gramps/GodotSteam – Дата звернення: 12.06.2025.
3. Steamworks Documentation [Електронний ресурс]. – Режим доступу: https://partner.steamgames.com/doc/home – Дата звернення: 12.06.2025.
4. Valente M. Multiplayer Game Programming: Architecting Networked Games. – New Riders, 2020. – 384 с.
5. Gregory J. Game Engine Architecture. – 4th ed. – Boca Raton : CRC Press, 2021. – 1248 с.
6. Ullmann G. C., Politowski C., Guéhéneuc Y.-G., Petrillo F. Game Engine Comparative Anatomy [Електронний ресурс] // ArXiv. – 2022. – Режим доступу: https://arxiv.org/abs/2207.06473 – Дата звернення: 12.06.2025.
7. Godot Engine – High-Level Multiplayer API Overview [Електронний ресурс]. – Режим доступу: https://docs.godotengine.org/en/stable/tutorials/networking/high_level_multiplayer.html – Дата звернення: 12.06.2025.
8. Cloudflare. NAT traversal explained [Електронний ресурс]. – Режим доступу: https://www.cloudflare.com/learning/network-layer/what-is-nat/ – Дата звернення: 12.06.2025.
9. GodotSteam Wiki – Getting Started [Електронний ресурс]. – Режим доступу: https://godotsteam.com/getting_started/introduction/ – Дата звернення: 12.06.2025.

10. Beeching E., Debangoye J., Simonin O., Wolf C. Godot Reinforcement Learning Agents [Електронний ресурс] // ArXiv. – 2021. – Режим доступу: https://arxiv.org/abs/2112.03636 – Дата звернення: 12.06.2025.

11. Richardson R. Unity vs Godot vs Unreal: Game Engine Comparison 2024 [Електронний ресурс] // GameDevBeginner. – Режим доступу: https://gamedevbeginner.com/unity-vs-godot-vs-unreal/ – Дата звернення: 12.06.2025.

12. Dota 2 [Електронний ресурс] // Steam. – Режим доступу: https://store.steampowered.com/app/570/Dota_2 – Дата звернення: 12.06.2025.

13. Warcraft III: Reforged [Електронний ресурс] // Battle.net. – Режим доступу: https://shop.battle.net/product/warcraft-iii-reforged – Дата звернення: 12.06.2025.

14. World of Warcraft [Електронний ресурс] // Blizzard Entertainment. – Режим доступу: https://worldofwarcraft.blizzard.com – Дата звернення: 12.06.2025.

15. League of Legends [Електронний ресурс] // Riot Games. – Режим доступу: https://www.leagueoflegends.com – Дата звернення: 12.06.2025.

16. Supervive [Електронний ресурс] // Steam. – Режим доступу: https://store.steampowered.com/app/2651760/Supervive – Дата звернення: 12.06.2025.

17. Stormgate [Електронний ресурс] // Steam. – Режим доступу: https://store.steampowered.com/app/2012510/Stormgate – Дата звернення: 12.06.2025.