МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки
Факультет Комп'ютерних наук
Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

другий (магістерський)

(рівень вищої освіти)

Дослідження та аналіз розробки архітектури ігрових серверів у жанрі «First-person Shooter»

Виконав:
студент 2 курсу групи ІПЗм-20-4

Семеренко В. С.

(прізвище, ініціали)

Спеціальність	121 – Інженерія програмного забезпечення
Тип програми	Освітньо-наукова
Керівник	проф. Власенко Л.А.

(посада, прізвище, ініціали)

Допускається до захисту
Зав. Кафедри _____ З.В. Дудар
2022 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121– Інженерія програмного забезпечення
(код і повна назва)

Тип програми освітньо-наукова програма

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«__»_____202_ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Семеренка Володимира Сергійовича
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження та аналіз розробки архітектури ігрових серверів у жанрі «First-person Shooter»»

затверджена наказом університету від «__»_____202___р. №____

2. Термін подання студентом роботи до екзаменаційної комісії «__» _____202__р.

3. Вихідні дані до роботи *архітектура ігрових серверів, First-person Shooter, одноранговa архітектура, архітектура клієнт-сервер, пояснювальна записка.*

4. Перелік питань, що потрібно опрацювати в роботі *аналіз предметної галузі та постановка задачі, дослідження та аналіз загальних принципів, дослідження та аналіз архітектур, порівняння результатів дослідження.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	17.01.2022	виконано
2	Дослідження та аналіз загальних принципів проектування	01.02.2022	виконано
3	Дослідження та аналіз архітектур	21.02.2022	виконано
4	Порівняння результатів дослідження	15.03.2022	виконано
5	Підготовка пояснювальної записки	29.03.2022	виконано
6	Підготовка презентації та доповіді	26.04.2022	виконано
7	Перевірка на академічний плагіат	16.05.2022	виконано
8	Нормоконтроль		
9	Рецензування		
10	Занесення диплома в електронний архів		
11	Попередній захист		
12	Допуск до захисту		

Дата видачі завдання 17 січня 2022 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Власенко Л.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 66 с., 35 рис., 10 джер.

ARCHITECTURE, CLIENT-SERVER, FIRST-PERSON SHOOTER, GAME, P2P, SERVER

Об’єктом дослідження є засоби розробки архітектури ігрових серверів у жанрі «First-person Shooter»

Метою роботи є аналіз засобів розробки архітектури ігрових серверів у жанрі «First-person Shooter» та архітектур в цілому.

Результатом кваліфікаційної роботи є проаналізована предметна область та існуючи аналоги, проаналізовані основні критерії для розробки архітектур серверів для шутерів від першої особи та архітектур в цілому, визначені переваги та недоліки основних архітектур, зроблено порівняльну характеристику та оцінку кожної архітектури для серверів у жанрі «First-person Shooter» та в цілому.

ARCHITECTURE, CLIENT-SERVER, FIRST-PERSON SHOOTER, GAME, P2P, SERVER

The object of research is the means of developing the architecture of game servers in the genre of «First-person Shooter».

The aim of the practice is to analyze the means of developing the architecture of game servers in the genre of «First-person Shooter» and in general.

The result of the qualification work is the analyzed subject area and existing analogues, analyzed the main criteria for developing server architectures for first-person shooters and architectures in general, identified the advantages and disadvantages of the main architectures, made a comparative description and evaluation of each architecture for servers in the genre of «First-person Shooter» and in general.

Я, Семеренко Володимир Сергійович, студент групи ІПЗм-20-4, здобувач вищої освіти на другому (магістерському) рівні кафедри програмної інженерії, заявляю: моя кваліфікаційна робота на тему «Дослідження та аналіз розробки архітектури ігрових серверів у жанрі «First-person Shooter»», що буде представлена до ЕК для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання. Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ 7
1 Аналіз предметної галузі та постановка задачі 9
1.1 Аналіз предметної галузі 9
1.2 Аналіз існуючих аналогів 15
1.3 Постановка задачі 20
2 Дослідження та аналіз загальних принципів проектування 21
2.1 Типи об’єктів та взаємодія між ними 21
2.2 Реплікація та допустима затримка 23
2.3 Пропускна здатність 24
2.3 Механізм інтересів 25
2.4 Узгодженість 29
3 Дослідження та аналіз архітектур 32
3.1 Дослідження однорангової архітектури 32
3.2 Дослідження клієнт-серверної архітектури 40
4 Порівняння результатів дослідження 47
4.1 Порівняння архітектур 47
4.2 Оцінка архітектур 48
Висновки 51
Перелік джерел посилання посилань 52
Перелік джерел посилань за науковими напрямами науковців кафедри програмної інженерії 53
ДОДАТОК А Апробація результатів роботи 54
ДОДАТОК Б Звіт результатів перевірки на унікальність тексту в мережі інтернет та базі ХНУРЕ 56
ДОДАТОК В Слайди презентації 57

ВСТУП

З розвитком ІТ-індустрії та процвітанням індустрії ігор, ігри стають все більш популярними. У 2021 році ігровий ринок оцінювався в 198,40 мільярда доларів США, і очікується, що до 2027 року він досягне 339,95 мільярда доларів США, що в середньому складе 8,94% протягом 2022-2027 років. Через загальнонаціональні карантини, введені через пандемію COVID-19, деякі люди звернулися до ігрових платформ, щоб скоротати час. Таким чином, ці платформи залучили сотні і тисячі нових відвідувачів до онлайн-трафіку. Останнім часом тенденції в області відеоігор пережили масовий сплеск гравців і доходів.

Світовий ринок цифрових медіа неухильно зростає, причому найбільша частка доходів ринку припадає на онлайн-ігри. У 2020 році світовий ринок онлайн-ігор приніс приблизно 21,1 мільярда доларів доходу, що означає рекордне зростання на 21,9 відсотка в порівнянні з попереднім роком. Це зростання, звичайно, було викликане глобальною спалахом COVID-19, яка змусила багатьох людей залишитися вдома і звернутися до цифрових розваг і знайти нові канали для спілкування з іншими людьми. В даний час в світі налічується близько 1 мільярда онлайн-геймерів, причому Китай, Південна Корея і Японія мають найбільше охоплення онлайн-ігор серед населення. За прогнозами, в 2025 році аудиторія онлайн-ігор перевищить 1,3 мільярда чоловік.

Мережеві багатокористувацькі ігри сьогодні є величезною частиною ігрової індустрії. Сегмент шутерів від першої особи приніс найбільшу частку виручки на світовому кіберспортивному ринку від 2020 року. Залежно від типу гри глобальний ринок кіберспорту підрозділяється на шутер від першої особи (FPS), багатокористувацьку онлайн-арену битв (MOBA), стратегію в реальному часі (RTS) і «гравець проти гравця» (PvP). Очікується, що сегмент шутерів від першої особи буде лідирувати за обсягом виручки на світовому ринку кіберспорту протягом прогнозованого періоду. FPS в даний час є найпопулярнішим і затребуваним ігровим жанром. У цьому жанрі гравець може керувати одним

аватаром в одній грі. Counter-Strike: Go, Call of Duty і ряд інших ігор є одними з найвідоміших в даний час. Одним з основних факторів популярності є великий вибір зброї і кількість карт і оточень, пропонованих в цих іграх.

Розрахована на багато користувачів відеогра — це відеогра, в якій одна чи більше однієї людини можуть одночасно грати в одному й тому ж ігровому середовищі через локальну або глобальну мережу, найчастіше через Інтернет. Геймери можуть легко знайти противників аналогічного рівня майстерності, граючи в очну гру через Інтернет. Гравці також можуть конкурувати в масових багатокористувацьких іграх, де десятки гравців грають у безперервну гру у віртуальному світі. Часто учасники можуть спілкуватися з іншими гравцями через сеанси текстового чату під час онлайн-ігор, а іноді гравці дійсно можуть розмовляти з іншими гравцями, використовуючи спеціальне звукове обладнання. Також може бути запропонована щомісячна плата за надання гравцям доступу до мережі, яка допомагає їм віддалено знаходити інших гравців і підключатися до них. Функцій та можливостей може бути багато, а отже, однією із найважливіших складових для гарної онлайн-гри є добре спроектована, високонавантажена, стабільна та легко розширювана серверна система.

Метою роботи є дослідження та аналіз архітектури ігрових серверів у жанрі «First-person Shooter». Також будуть розглянуті архітектури та моделі ігрових серверів в цілому.

В ході атестаційної роботи магістра було: проаналізовано предметну область та існуючи аналоги, проаналізовано основні критерії для розробки архітектур серверів для шутерів від першої особи та архітектур в цілому, визначені переваги та недоліки основних архітектур, зроблено порівняльну характеристику та оцінку кожної архітектури.

За результатами атестаційної роботи магістра було розроблено презентацію, досліджено та проаналізовано можливі архітектури ігрових серверів для жанру FPS та різного роду ігрових серверів в цілому.

Результати роботи можуть бути корисні розробникам архітектур як ігор FPS так і будь-яких онлайн ігор в цілому.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі

Безперервні технологічні досягнення в ігровій індустрії значно стимулюють зростання галузі. Вони покращують спосіб створення ігор і покращують загальний ігровий досвід користувачів.

Зростаюче підключення до Інтернету, а також зростаюче поширення смартфонів і поява мереж з високою пропускною здатністю, таких як 5G, ще більше збільшили попит на ігровому ринку по всьому світу. Крім того, згідно зі звітом асоціації операторів мобільного зв’язку GSMA, станом на листопад 2021 року 170 операторів мобільного зв'язку запустили комерційні послуги 5G з проникненням 7% населення до кінця 2021 року, тим самим відкриваючи нові можливості для постачальників мобільного зв'язку по впровадженню смартфонів 5G на ринок.

Також, згідно зі статистикою, опублікованою DataReporal, число користувачів Інтернету збільшилося на 7,7% в 2021 році в порівнянні з 2020 роком. Більше того, вони збільшилися на 4% до 4950 мільйонів у січні 2022 року порівняно з січнем 2021 року, коли загальна кількість користувачів склала 4758 мільйонів.

Розробники ігор в країнах з економікою, що розвивається постійно прагнуть поліпшити ігровий досвід, запускаючи і переписуючи коди для різних консолей та платформ, таких як PlayStation, Xbox і Windows PC, включені в окремий продукт, що надається геймерам через хмарну платформу. Хмарні ігрові сервіси зосереджені на використанні гіпермасштабованих хмарних можливостей, сервісів потокового мультимедіа та глобальних мереж доставки контенту для створення наступного покоління платформ соціальних розваг. Крім того, використання хмарних технологій на ігровому ринку, ймовірно, стимулюватиме попит і залучення багатокористувацьких гравців до різних ігор, стимулюючи зростання ринку.

Родоначальник сучасної мережевої багатокористувацької гри почався на університетських мейнфреймах у 1970-х роках. Однак цей тип ігор не вибухнув, поки доступ до Інтернету не став поширеним у середині-кінці 1990-х років.

Деякі з найперших відеоігор містили локальну багатокористувацьку гру, тобто вони були розроблені для двох або більше гравців, щоб грати в гру на одному комп’ютері.

Перші мережеві багатокористувацькі ігри запускалися в невеликих мережах, що складалися з мейнфреймів. Що відрізняє мережеву багатокористувацьку гру від локальної багатокористувацької гри, так це те, що мережеві ігри мають два або більше комп’ютерів, під’єднаних один до одного під час активної ігрової сесії.

Персональні комп’ютери почали набувати певного поширення в другій половині 1970-х років, розробники придумали способи зв’язку двох комп’ютерів один з одним через послідовні порти. Послідовний порт дозволяє передавати дані по одному біту за раз, і його типове призначення полягало в зв’язку із зовнішніми пристроями, такими як принтери або модеми. Однак також можна було з’єднати два комп’ютери один з одним і дозволити їм спілкуватися через це з’єднання. Це дозволило створити ігровий сеанс, який зберігався на кількох персональних комп’ютерах, і призвів до деяких з перших мережевих комп’ютерних ігор.

Одним із великих недоліків використання послідовних портів було те, що комп’ютери зазвичай не мали більше двох послідовних портів. Це означало, що для того, щоб підключити більше двох комп’ютерів через послідовний порт, потрібно було використовувати схему послідовного ланцюга, де кілька комп’ютерів з’єднані один з одним у кільце. Це можна вважати різновидом топології мережі.

Локальна мережа [1] – це термін, який використовується для опису кількох комп’ютерів, під’єднаних один до одного на відносно невеликій території. Механізм, який використовується для локального підключення, може відрізнятися — наприклад, з’єднання з послідовним портом можуть бути одним із прикладів локальної мережі. Однак локальні мережі дійсно почали розвиватися з поширенням Ethernet.

Незважаючи на те, що деякі мережеві багатокористувацькі ігри все ще випускаються з локальною мережею, тенденція останніх років, схоже, привела до того, що розробники відмовляються від гри в локальній мережі заради виключно багатокористувацької онлайн-гри.

Оскільки Інтернет почав вибухати наприкінці 1990-х, разом з ним почали розвиватися онлайн-ігри. Хоча може здатися, що онлайн-гра може бути реалізована майже так само, як і гра в локальній мережі, основним фактором є затримка або кількість часу, необхідного для передачі даних по мережі. Навіть сьогодні більшість онлайн-ігор для кількох гравців обмежені невеликою кількістю гравців за сесію гри — десь від 4 до 32 — це зазвичай кількість підтримуваних гравців. Однак у багатокористувацькій онлайн-грі (MMO) сотні, якщо не тисячі гравців можуть брати участь в одній ігровій сесії. Більшість ігор MMO є рольовими іграми, тому їх називають MMORPG. Проте існують інші стилі MMO-ігор, наприклад First-person Shooter (MMOFPS).

Starsiege: Tribes — науково-фантастичний шутер від першої особи, який був випущений наприкінці 1998 року. На момент випуску його вважали як гру, що містить як швидкі бойові дії, так і порівняно велику кількість гравців. Деякі режими гри підтримували 128 гравців через локальну мережу або Інтернет. Щоб отримати деяку уявлення про масштаби труднощів, пов’язаних із впровадженням такої гри, треба зауважити, що протягом цього часу переважна більшість гравців, які мають під’єднання до Інтернету, користувалися послугою комутованого зв’язку. У кращому випадку ці користувачі комутованого зв’язку мали модем зі швидкістю до 56,6 кбіт/с. У випадку Tribes він фактично підтримував користувачів зі швидкістю модему лише 28,8 кбіт/с. За сучасними мірками це надзвичайно низькі швидкості підключення. Іншим фактором було те, що комутовані підключення також мали відносно високу затримку — затримка в кілька сотень мілісекунд була досить поширеною. Може здатися, що мережева модель, розроблена для гри з низькими обмеженнями пропускної здатності, не має значення в сучасному світі. Однак це не так.

Зрештою, розробники Tribes розділили свої вимоги до даних на такі чотири категорії:

– негарантовані дані. Як і можна було очікувати, це дані, які гра визначає як несуттєві для гри. При обмеженій пропускній здатності гра може спочатку видалити ці дані.

– Гарантовані дані. Ці дані гарантують як надходження, так і впорядкування даних. Це використовується для даних, які можно визнати критичними, наприклад, подія, що означає, що гравець вистрілив зі зброї.

– Дані «Останній стан». Цей тип даних призначений для випадків, коли важлива лише остання версія даних. Одним із прикладів є хіти певного гравця. Очки здоров'я гравця 5 секунд тому не дуже важливі, якщо гра знає, які його очки здоров'я зараз.

– Гарантовано найшвидші дані. Цим даним надається найвищий пріоритет, щоб їх передавати якомога швидше з гарантованою доставкою. Прикладом такого типу даних є інформація про рух гравця, яка, як правило, актуальна протягом дуже короткого періоду часу, і тому повинна передаватися швидко.

Багато рішень щодо впровадження, прийняті в моделі Tribes Networking Model, зосереджені на забезпеченні цих чотирьох типів передачі даних.

Як було сказано до цього, спочатку використовувалась однорангова архітектура (P2P) [2]. На рисунку 1.1 зображено однорангову архітектуру.

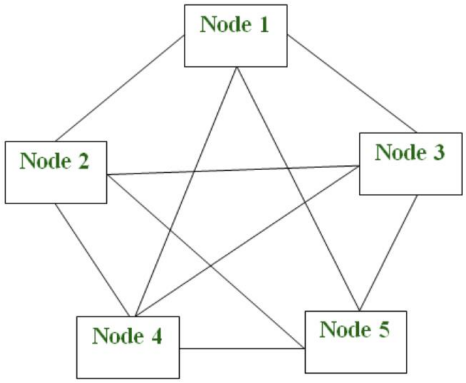

Рисунок 1.1 – Приклад однорангової моделі P2P

Вона складається з децентралізованої мережі однорангових вузлів, які є одночасно клієнтами і серверами. Мережі P2P розподіляють робоче навантаження між одноранговими вузлами, і всі однорангові вузли вносять свій внесок і споживають ресурси всередині мережі без необхідності у централізованому сервері.

Ще одним важливим дизайнерським рішенням було використання моделі клієнт-сервер [3] замість однорангової моделі.

На рисунку 1.2 зображено модель клієнт-серверної архітектури.

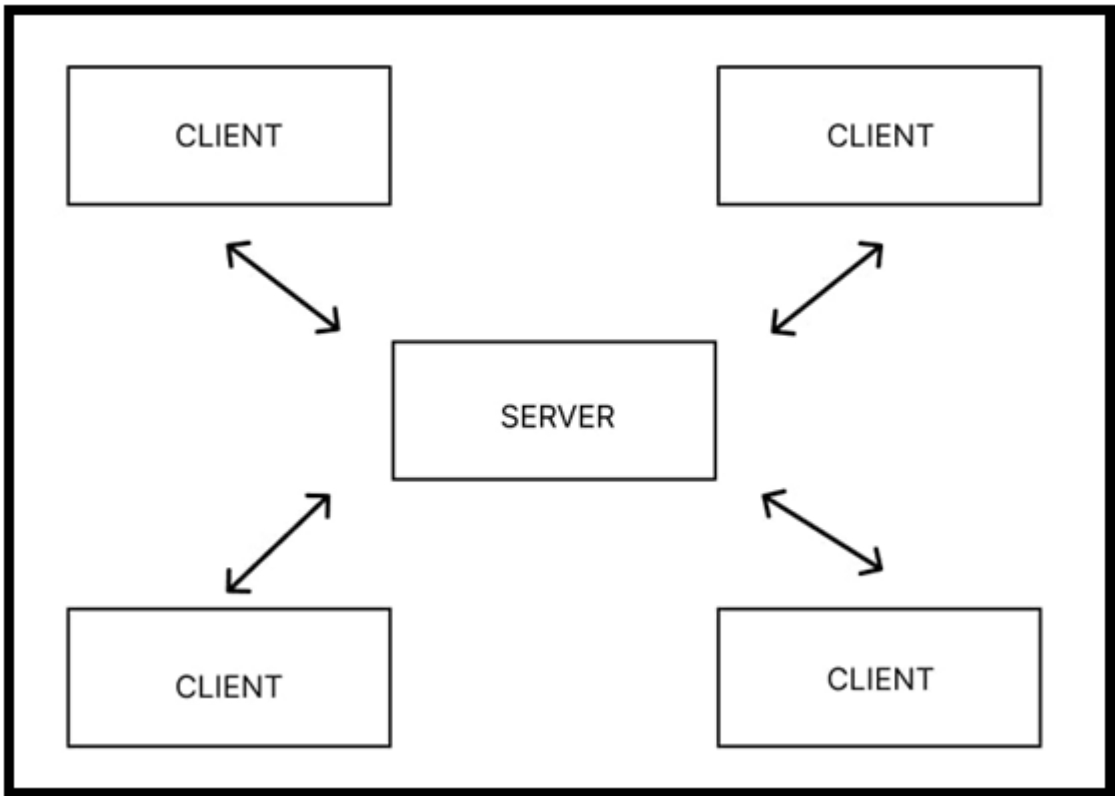

Рисунок 1.2 – Приклад архітектури клієнт-сервер

У моделі клієнт-сервер усі гравці підключаються до центрального сервера, тоді як у моделі однорангового зв’язку кожен гравець підключається до кожного іншого гравця. Для однорангової моделі потрібна пропускна здатність O (n^2). Це означає, що пропускна здатність зростає квадратично в залежності від кількості користувачів. У цьому випадку, коли n дорівнює 128, використання однорангового зв’язку призведе до дуже малої пропускної здатності на одного гравця. Щоб уникнути цієї проблеми, Tribes замість цього впровадили модель клієнт-сервер. У цій конфігурації вимоги до пропускної здатності кожного гравця залишаються незмінними, тоді як сервер повинен обробляти лише пропускну здатність O(n). Однак це означало, що сервер повинен бути в мережі, яка дозволяла б мати кілька вхідних з’єднань. Starsiege Tribes реалізував мережеву архітектуру, все ще

актуальну для сучасної екшн-гри. Він використовує модель клієнт-сервер, тому кожен гравець в грі підключений до сервера, який координує гру.

Також згодом почали використовувати багатосерверну архітектуру [4].

На рисунку 1.3 зображено модель багатосерверної архітектури.

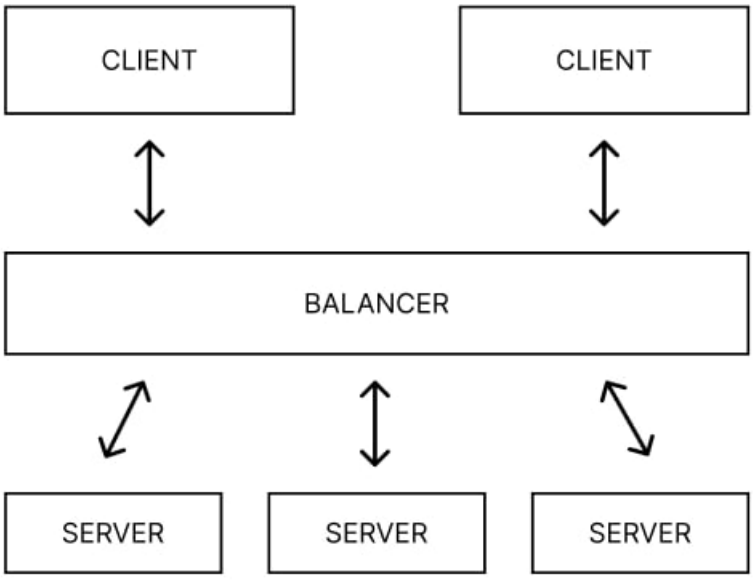

Рисунок 1.3 – Приклад багатосерверної архітектури

По суті, аргумент, висунутий на користь багатосерверної архітектури, полягає в тому, що сама архітектура веде до ефективності. Тобто, дозволяючи одному серверу зосередитися тільки на одній із можливих задач, ми можемо налаштувати його так, щоб максимізувати продуктивність для цієї задачі. Дозволяючи другому серверу зосередитися на обслуговуванні іншої задачі, ми також можемо оптимізувати його. Крім цього, багатосерверний підхід має ще одну перевагу перед односерверним підходом: простота ефективного масштабування. Оскільки ми відокремили проблеми одну від одної, це дозволяє дуже легко та ефективно масштабуватися систему, додаючи потужності саме там, де це необхідно.

Існують також так звані гібридні архітектури, які можуть поєднувати, на приклад, архітектуру P2P з архітектурою клієнт-сервер, а також можливо використовувати такі архітектури, як сервіс-орієнтована архітектура (SOA) [5].

1.2 Аналіз існуючих аналогів

На даний момент існує кілька існуючих варіантів мережевих серверів для розробки за допомогою ігрових двигуні Unity та Unreal Engine.

Ігровий двигун – це середовище розробки програмного забезпечення, яке використовується розробниками ігор для створення інтерактивних відеоігор. Розробники створюють ігри для консолей, комп'ютерів і мобільних пристроїв, використовуючи ігровий двигун, оскільки двигун – це повторно використовувані компоненти, використовувані для створення фреймворку гри.

Кожен середньостатистичний ігровий двигун дозволяє розробникам додавати загальні функції, такі як фізика, введення, рендерінг, сценарії, штучний інтелект для виявлення зіткнень і багато іншого, без необхідності їх кодування.

На рисунку 1.4 зображено логотипи ігрових двигунів Unity та Unreal Engine.

Рисунок 1.4 – Логотипи ігрових двигунів Unity та Unreal Engine

Розробку ігор було б важко уявити без цих інструментів. Обидва двигуни мають свої переваги і недоліки, тому правильний вибір залежить від вимог проекту. Unity славиться своєю величезною базою користувачів і підтримкою розробки

користувальницьких додатків, а також створенням 2D і 3D симуляцій. Движок Unreal Engine, з іншого боку, кращий для створення великих ігор, оскільки він пропонує красивішу графіку.

Unreal Engine [6] має свій власний інструмент для створення серверів. У мережевій багатокористувацькій грі Unreal Engine використовує модель клієнт-сервер. Один комп'ютер в мережі діє як сервер і проводить сеанс багатокористувацької гри, в той час як всі комп'ютери інших гравців підключаються до сервера в якості клієнтів. Потім сервер ділиться інформацією про стан гри з кожним підключеним клієнтом і надає їм засоби зв'язку один з одним. Сервер, як господар гри, зберігає єдиний справжній, авторитетний ігровий стан (репліку). Іншими словами, сервер – це місце, де насправді відбувається багатокористувацька гра. Кожен клієнт дистанційно керує персонажами, якими він володіє на сервері, відправляючи їм виклики процедур, щоб змусити їх виконувати різні ігрові дії. Однак сервер не транслює візуальні ефекти безпосередньо на монітори клієнтів. Замість цього сервер повторює інформацію про стан гри кожному клієнту, повідомляючи їм, які дії повинні існувати, як ці дійові особи повинні поводитися і які значення повинні мати різні змінні. Потім кожен клієнт використовує цю інформацію для імітації дуже близького наближення того, що відбувається на сервері.

Двигун Unity [7] має декілька ширший набір варіантів:

– UNet. Застаріла мережна технологія Unity. На даний момент deprecated і підтримка закінчиться в найближчі кілька років.

– NetCode. Пакет Unity NetCode надає модель виділеного сервера з передбаченням клієнта, яку можна використовувати для створення багатокористувацьких ігор. Це потенційно гарна технологія, яка добре працюватиме у зв'язці з Entity Component System. Але на даний момент дуже повільно розвивається, так як за останні кілька років її існування вийшло шість версій різного ступеня неопрацьованості, API постійно змінюється і робити щось серйозне на ньому поки що зарано.

– MLAPI. Альтернатива UNet із широким спектром можливостей. MLAPI (Mid-level API) — це мережева бібліотека з відкритим кодом, яка пропонує широкий спектр функцій середнього рівня, таких як NetworkedVars, керування сценою, віддалені виклики процедур (RPC), обмін повідомленнями тощо. Це рішення пропонує рівень абстракції, щоб уможливити заміну різних транспортів залежно від топології та платформ, з якими ви плануєте поставляти. Рішення передбачає певну форму топології клієнт-сервер – або виділений ігровий сервер (DGS, де клієнти підключаються і взаємодіють з грою на центральному сервері), або Listen Server (де один клієнт розміщує сервер для матчу). За замовчуванням MLAPI припускає, що ви використовуєте середовище виконання Unity без заголовка для свого сервера, що дозволяє вам написати фізику або моделювання лише один раз, а потім використовувати його як для клієнта, так і для сервера.

У мережі API поділяються на середні, високорівневі та низькорівневі. Низькорівневий API займається такими речами, як серіалізація даних, взаємодія з мережевим транспортом та управління з'єднаннями. Він має справу з найдрібнішими деталями програмування та мереж. Високорівневий API знаходиться поверх низькорівневого API і робить складніші процеси більш зрозумілими та простими у використанні. Програмувати із високорівневим API набагато простіше, ніж із низькорівневим API. У цьому використання високорівневого API знижує продуктивність. Якщо вам потрібна мережна програма з ефективним використанням пам'яті, використання високорівневого API - не завжди найкращий вибір. Unity MLAPI – це API середнього рівня означає, що він має доступ до основних процесів низькорівневого API, але відчувається як високорівневий API. MLAPI розроблений так, щоб бути найкращим інструментом з обох боків. Він надає доступ до центральних мережевих процесів та підтримує рівень абстракції для простоти використання.

Наявність такої системи не тільки дає вам безліч можливостей для аутентифікації з'єднань, але і підвищує безпеку. Будь-яке з'єднання, яке не відповідає вашим вимогам (наприклад, ввід правильного пароля), просто

відхилюється. Це робить MLAPI серйозним вибором для розробників, яким потрібен рівень безпеки та налаштування.

– DarkRift 2. DarkRift 2 — це швидке та високопродуктивне мережеве рішення низького рівня — ключові функції включають двоканальний TCP та UDP, серіалізацію та настроюваний журнал. Рішення передбачає топологію виділеного сервера і включає багатопотоковий сервер, який можна розширити за допомогою плагінів. Можна використовувати DarkRift із сервером на базі Unity і при цьому використовувати систему плагінів. За замовчуванням усе рішення є багатопоточним, що робить його одним із найбільш масштабованих. Однак він не використовує нову систему завдань C# від Unity для багатопотокової роботи, тому для їх використання разом потрібні додаткові знання.

– Photon PUN. PUN (Photon Unity Networking) – це клон оригінального мережевого API Unity, що працює на основі надійної інфраструктури Photon. Окрім повсюдного підбору партнерів, основними будівельними блоками PUN є: серіалізація станів ігрових об’єктів (з вбудованою підтримкою трансформацій тощо); віддалені виклики процедур (RPC). PUN дає розробнику прямий і повний контроль над тим, що надсилати/отримувати, і, у поєднанні з його гнучкою моделлю комунікаційного зв’язку, подібною до багатоадресної передачі, є потужною робочим рішенням для мережевих ігор. Photon PUN, по суті, є рішенням сітчастої топології (він же прямий P2P), в якому кожен клієнт синхронізує дані всіх інших.

Перевага сітчастої топології полягає в тому, що немає єдиного хоста/сервера для керування. Однак недоліком є те, що кожен клієнт повинен керувати логікою синхронізації кожного іншого гравця, що обмежує масштаб гри, і немає повноважень сервера для того щоб запобігти, на приклад, шахрайству. Це означає, що це рішення має деякі ризики і прийняття цієї топології може ускладнити згодом перехід до моделі, що користується повноваженнями сервера.

– Photon Quantum 2.0. Photon Quantum – це високопродуктивна детермінована система ECS (Entity Component System) для багатокористувацьких онлайн-ігор, створених за допомогою Unity. Він заснований на підході прогнозування/відкату,

який ідеально підходить для онлайн-ігор, чутливих до затримок, таких як екшн-рольові ігри, спортивні ігри, файтинги, FPS тощо. Quantum також допомагає розробнику писати чистий код, повністю відокремлюючи логіку моделювання (Quantum ECS) від перегляду/презентації (Unity), а також піклується про особливості мережевих реалізацій.

– Mirror. Mirror простий у використанні і має відмінне ком'юніті. Це одна з найбільших переваг Mirror. Також він відносно простий у використанні, з великою кількістю прикладів та непоганою документацією. В цілому, багатьом користувачам легко почати роботу з Mirror. Але за відзивами користувачів йому необхідне поліпшення управління пам'яттю для досягнення високої продуктивності і масштабування.

– DOTS-Netcode. DOTS NetCode Unity – це виділений сервер із мережевою моделлю прогнозування клієнта.

На рисунку 1.5 зображено порівняння між представленими вище інструментами за п'ятибальною шкалою, включаючи вартість в цілому.

	Безкоштовність	Стабільність та Підтримка	Простота використання	Продуктивність	Масштабованість	Функціональність
MLAPI	Так	5	4	5	4	5
Mirror	Так	5	5	3	4	4
Photon Quantum 2.0	Ні	4	4	5	2	4
DarkRift	Ні	5	3	5	5	3
Photon PUN	Ні	4	5	3	1	4

Рисунок 1.5 – Порівняння інструментів Unity

Слід зазначити, що для розробки ігор категорії AAA (найбільш високобюджетних комп'ютерних ігор, розрахованих на масову аудиторію і вимагаючих величезних витрат як на саму розробку гри, так і на її маркетинг), в більшості випадків, створюють сервери з нуля

1.3 Постановка задачі

Метою роботи є дослідження та аналіз засобів розробки архітектури ігрових серверів у жанрі «First-person Shooter» та архітектур в цілому.

Три основні архітектури, як правило, використовуються в іграх з великою кількістю гравців:

а) архітектура клієнт-сервер;

б) багатосерверна архітектура;

в) однорангова (P2P) архітектура.

В роботі будуть досліджені та проаналізовані основні принципи для архітектур багатокористувацьких ігор.

Загалом, будуть розглянута архітектури клієнт-сервер та однорангова архітектура, з точки зору їх компонентів та їх типової поведінки. Багатосерверна архітектура детально розглядатися не буде, так як вона сама по собі є комбінацією декількох архітектур клієнт-сервер, але коротко буде розглянута в зрівнянні з іншими. З кожною архітектурою будуть розглянуті її основні проблеми та причини, які спонукають до вибору такої конкретної архітектури, а також детальні механізми, які використовуються в їхніх проектах для реалізації.

В довершені, буде розглянута порівняльна характери за головними критеріями для побудови архітектур.

2 ДОСЛІДЖЕННЯ ТА АНАЛІЗ ЗАГАЛЬНИХ ПРИНЦИПІВ ПРОЕКТУВАННЯ

2.1 Типи об’єктів та взаємодія між ними

Багатокористувацькі онлайн ігри приносять міль’ярди доларів доходу в основному через те, що вони розраховані на велику кількість гравців, які можуть одночасно знаходитись в одному місці ігрового світу. Адже чим більше гравців буде в ігровому світі, тим більш привабливою, інтерактивною і складнішою стане ігрове середовище. Таким чином, для того щоб гра стала успішною, вона повинна забезпечити по-справжньому масштабований ігровий світ, зберігаючи швидкість відгуку.

Багатокористувацька гра – це гра, де гравці можуть грати разом, бути незалежними супротивниками або ж можуть грати разом проти супротивників, які управляються за допомогою штучного інтелекту. Такі ігри можуть генерувати великий сітьовий трафік та велике обчислювальне навантаження. Таким чином основною задачею перед розробником стають забезпечити гравців масштабованістю, узгодженістю, безпекою і швидким часом відгуку, і все це, як правило, повинно працювати одночасно.

Для того щоб перейти безпосередньо до аналізу найпопулярніших архітектур, треба розібратися з деякими принципами та концепціями, зв’язаними з проектуванням ігрових серверів в цілому, яка б архітектура не була обрана.

Більшість ігор мають мінімум три типи об’єктів:

– незмінні об’єкти, тобто, ті об’єкти які зазвичай проектуються і створюються один раз на етапі завантаження гри і ніколи не міняють свого стану під час неї;

– об’єкти якими керує гравець (ігровий персонаж), зазвичай має своє положення в ігровому світі і допускає ряд дій, такі як взаємодія з другими об’єктами, оновлення свого стану, взаємодію з іншими персонажами гравців та інші дії;

– об’єкти стан яких може змінитись при взаємодії с гравцями чи з приходом якихось внутрішньо-ігрових івентів.

Саму взаємодію гравців можна також поділити на декілька категорій:

– оновлення персонажу (графічне оновлення персонажу чи оновлення його позиції);

– взаємодія персонажу с іншими персонажами інших гравців (будь-який варіант атаки гравцем іншого гравця зі збільшенням бонусів для переможця) та персонажа з об’єктами навколишнього середовища (знаходження предмету і додавання його в інвентар).

На рисунку 2.1 зображено можливі типи об’єктів та взаємодія між ними.

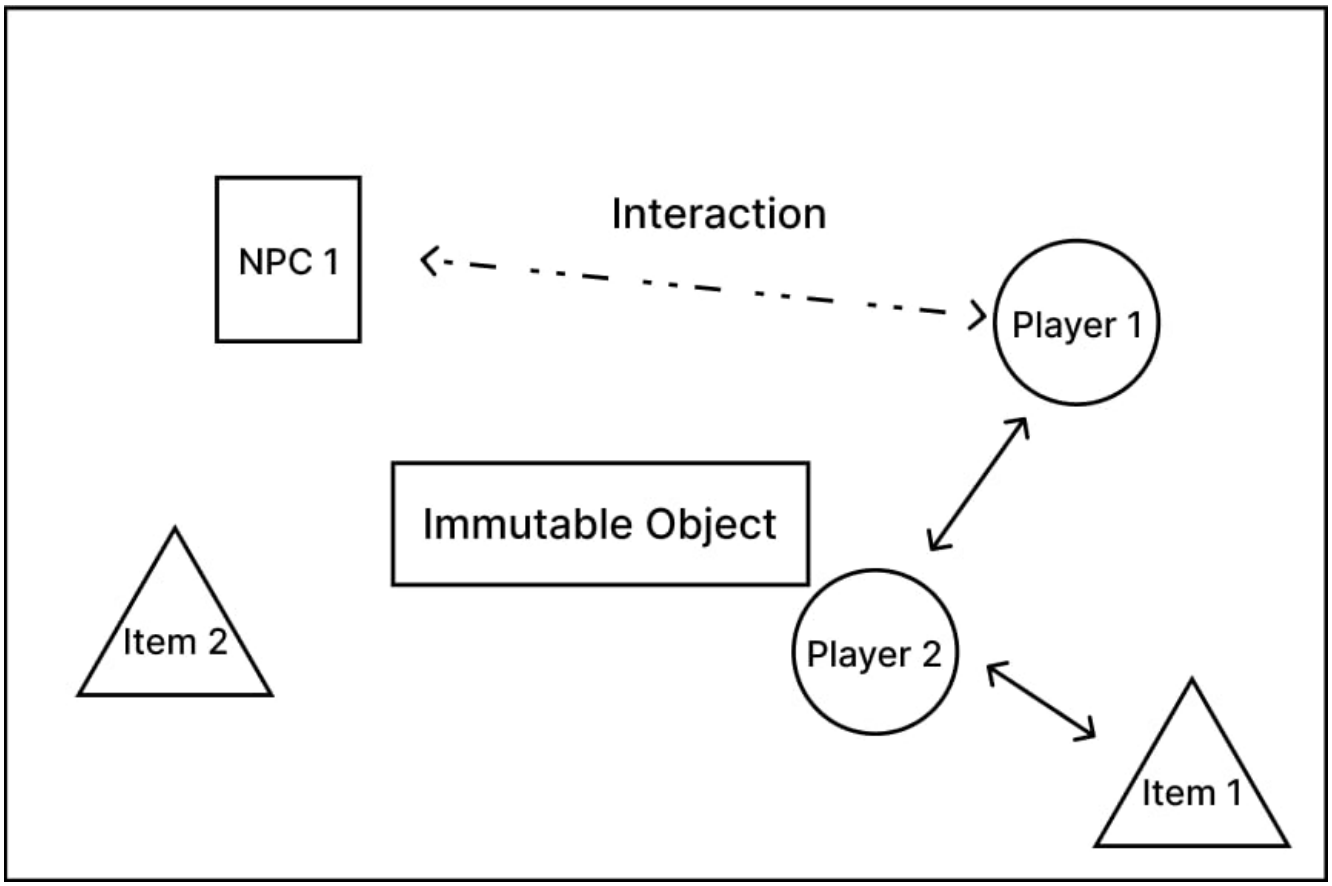

Рисунок 2.1 – Можливі типи об’єктів та взаємодія між ними

Виявлення типів взаємодії дуже важлива складова при вирішені таких проблем як узгодженість, які виникають під час однакових дій над одним і тим же

предметом для оновлення його стану, а також для запобігання багатьох типів шахрайства застосованих безпосередньо до типів взаємодії.

2.2 Реплікація та допустима затримка

Кожен гравець, приєднавшись до гри, отримує свій екземпляр ігрового світу (репліку), який включає в себе різні типи ігрових об'єктів. Тобто для кожного об'єкта є авторитетна копія, яка називається основною копією. Всі інші копії є вторинними. Кожен гравець зберігає на комп'ютері копії ігрових об'єктів, які представляють інтерес для нього. Будь-яке оновлення об'єкта має бути спочатку виконано на первинній копії. Те, як розподіляються первинні та вторинні копії (наприклад, первинні копії можуть завжди перебувати на сервері або також можуть зберігатися клієнтами), залежить від архітектури гри. Якщо гравець хоче виконати оновлення об'єкта, для якого у нього немає основної копії, він повинен відправити оновлення основної копії. Власник первинної копії вирішує, приймати оновлення чи ні, а потім відправляє оновлений об'єкт всім, у кого є вторинна копія, де застосовуються зміни. Даний механізм розповсюдження оновлень дуже схожий на патерн проектування Підписник (Observer). Таким чином кожна репліка стає підписником на первинну копію об'єкта і отримує публікації (оновлення) від неї.

Під час проектування механізму реплікації, основну увагу слід приділити допустимій затримці між оновленням первинної копії та реплікою. Сама затримка залежить від мережевих затримок та архітектурного дизайну.

Різні типи багатокористувацьких ігор мають свої допустимі значення затримки. У рольових іграх (RPG) чи стратегіях у реальному часі (RTS) дозволена затримка може бути більшою, ніж у шутерах від першої особи (FPS), так як у типі FPS гравець виконує короткочасні і менш складні дії, тому потрібна більш висока швидкість реакції. В результаті потрібна більш висока швидкість відгуку. В залежності від вимог гри, допустима затримка може бути від 100 до 300 мілісекунд.

Оскільки затримка в мережі для кожного клієнта різна, і зазвичай основна копія отримує різні запити на оновлення одночасно від різних клієнтів, більшість ігор навмисно відстають у виконанні подій. Це забезпечує справедливість, незважаючи на відмінності в затримках, і більший контроль над витратами на поширення оновлень.

Зазвичай реалізують цикл дискретних подій (також може називатися фреймом), в якому всі дії (події), відправлені з моменту останнього виконання циклу, приймаються, буферизуються і потім виконуються. Потім оновлення відправляються в кінці циклу. Ігровий цикл зазвичай виконується від 10 до 20 разів кожну секунду; цю частоту іноді називають частотою кадрів. Затримка повинна бути обрана таким чином, щоб залишалося достатньо часу для отримання оновлень від різних клієнтів. Це допомагає забезпечити справедливість для клієнтів, у яких може бути погане підключення, і контролювати кількість оновлень, які необхідно відправляти в секунду. У той же час відставання має бути досить невеликим, щоб його не помітили гравці. По суті, необхідно знайти компроміс.

2.3 Пропускна здатність

Вимоги до пропускної здатності можуть бути визначені за допомогою кількості одержувачів, частоти оновлення та середнього розміру повідомлень. Ігри з мільйонами гравців мають високі вимоги до пропускної здатності через їх динамічне середовище або високу частоту оновлень. Крім того, задля забезпечення прийнятного ігрового процесу, потрібно враховувати випадкові сплески трафіку, які можуть у багато разів перевищувати розраховані середні вимоги. Такі сплески можуть відбуваються на фоні раптових змін навколишнього середовища або битв в процесі гри.

2.3 Механізм інтересів

Ще одним необхідним механізмом є механізм інтересів. Ідея цього механізму у тому що гравці багатокористувацької гри мають лише обмежені можливості пересування і зору. Тобто гравці можуть переміщатися тільки на невелику відстань в ігровому світі за заданий проміжок часу. В результаті доступ до даних в іграх показує просторову і тимчасову локальність. Цей варіант гарний тим, що кожному гравцю потрібно буде отримати репліку тільки с тими даними, які знаходяться у його зоні сприйняття, що в свою чергу спрощує масштабованість.

Механізм інтересів на основі простору заснований на моделі aura-nimbus, де aura це обмежувальна область навколо гравця, а nimbus визначає саму область в якій ігровий персонаж може сприймати об’єкти.

На рисунку 2.2 зображено дану модель.

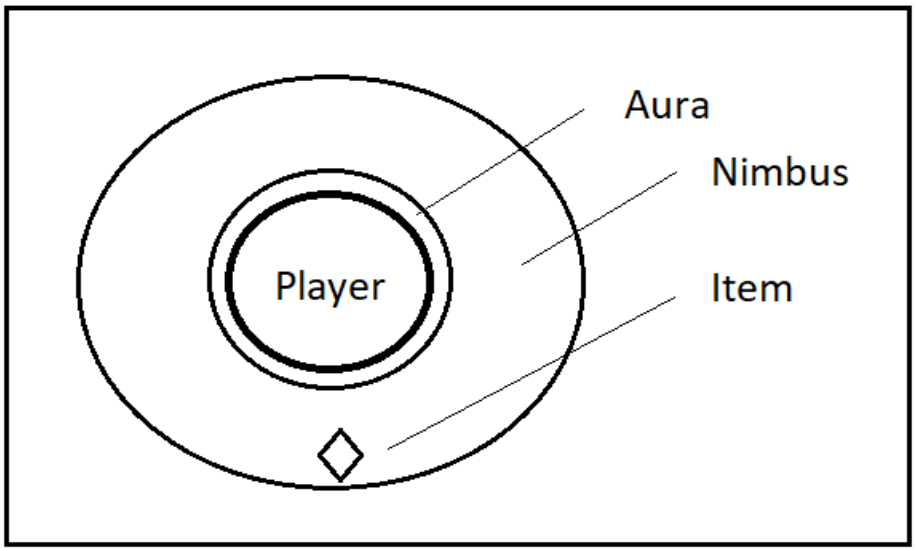

Рисунок 2.2 – Приклад моделі aura-nimbus

Його також можуть називати областю інтересів. Тож зазвичай гравець може взаємодіяти з об’єктами тільки своєї зони інтересів і тому йому потрібно мати репліки тільки цих об’єктів. В результаті необхідні обчислювальні та мережеві вимоги істотно знижуються в порівнянні з підтриманням копій всіх об'єктів.

Зонування є найбільш поширеним механізмом управління, при якому вся ігрова мапа ділиться на зони. Різні підходи цього механізму можуть мати різну форму своїх зон. Так у самому простому випадку, зона інтересу гравця знаходиться в тій же зоні, в якій знаходиться сам гравець, а в деяких системах вся зона може дорівнювати зоні інтересу персонажа і змінюватись зі зміною зони на мапі. У інших випадках зона інтересу, далі ЗІ, буде являти підобласть зони на мапі (часто коло або сфера) і коли гравець буде переміщуватись по мапі, його ЗІ буде переміщуватись разом з ним. Тому ЗІ повинен мати кожен ігровий об’єкт для того щоб визначити зіткнення с ЗІ гравця.

Більш продвинуті схеми управління дозволяють ЗІ охоплювати більше однієї зони. Це важливо для безперервних світів. Гравці на межах зони повинні мати можливість бачити об'єкти, що знаходяться безпосередньо за кордоном зони в сусідній зоні, і взаємодіяти з ними.

Що стосується форми зон, ігровий світ можна просто статично розділити на декілька типів.

Grid сітки. Оскільки це простий підхід, багато рішень використовують його.

На рисунку 2.3 зображено приклад Grid сітки.

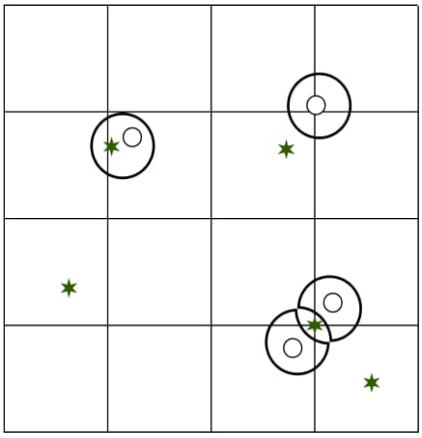

Рисунок 2.3 – Приклад Grid сітки

Гексагональне зонування. Шестикутники мають однакову орієнтацію і однакову суміжність, що означає, що гравці завжди будуть переміщатися в сусідню зону.

На рисунку 2.4 зображено приклад Гексагонального зонування.

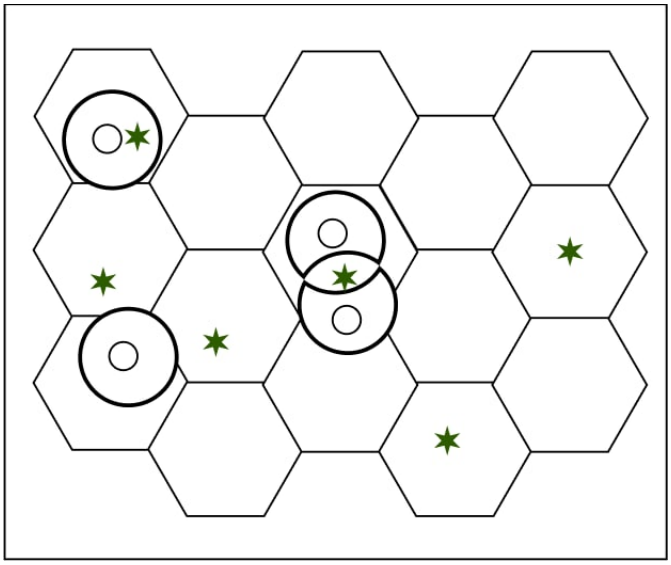

Рисунок 2.4 – Приклад Гексагонального зонування

Тріангуляційне зонування. Воно дозволяє враховувати перешкоди в ігровому світі.

На рисунку 2.5 зображено приклад Тріангуляційного зонування.

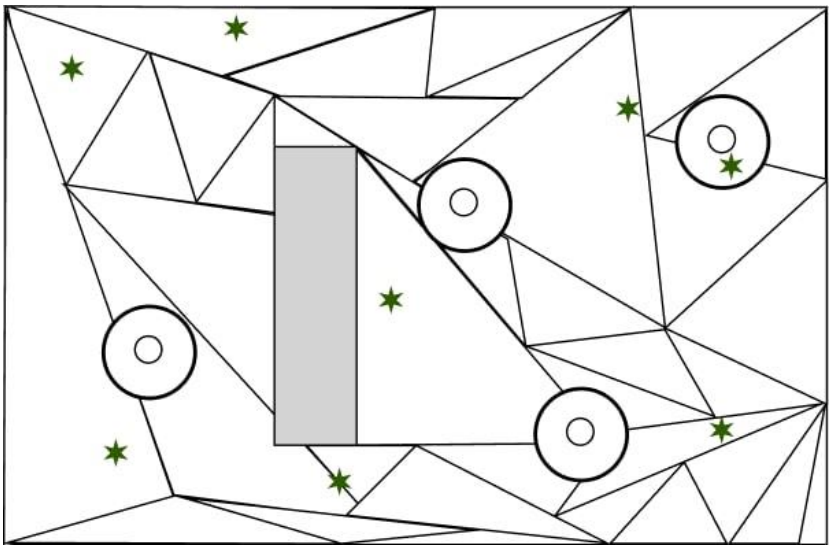

Рисунок 2.5 – Приклад Тріангуляційного зонування

Це може допомогти зменшити ЗІ, отже, кількість об'єктів, які необхідно враховувати для управління інтересами, що призводить до меншої кількості реплік

об'єктів, які повинні підтримуватися гравцями, і меншої кількості повідомлень про оновлення, які необхідно надсилати. Такі методи, як тріангуляція Делоне, були широко вивчені. Вони можуть бути використані для тріангуляції області всередині або навколо перешкоди у формі багатокутника, так що трикутники повторюють межі перешкод.

Визначення правильного розміру зон є складним завданням. Різні ігри мають різні характеристики, і навіть всередині гри різні частини можуть вимагати різних механізмів зонування. Отже, оптимального механізму зонування для всіх випадків не існує. Дуже велика зона може призвести до занадто великої кількості об'єктів в одній зоні, що робить управління інтересами менш ефективним, оскільки будь-який гравець може бути зацікавлений тільки в невеликому наборі цих об'єктів. З іншого боку, занадто мала зона може бути набагато меншою, ніж ЗІ гравців, що знову призводить до складного управління інтересами між кількома зонами. Цей компроміс може бути вирішеним за допомогою динамічних схем управління інтересами.

ЗІ гравця може страждати від групування (коли багато гравців переміщуються в одну зону). Це може негативно позначитися на управлінні реплікацією. Такі зони формуються бо деякі області стають цікавішими або прибутковішими для гравців з точки різного роду цікавих об’єктів або запрошень інших гравців. Це призводить до того, що велика кількість гравців приходить в одну і ту ж область, в той час як інші області стають менш населеними. Такі зони можуть швидко змінюватися, коли гравці переміщуються в нові цікаві області, тому це часто ускладнює підготовку до таких змін. Популяції в реальних іграх часто слідують степеневому закону розподілу, і збільшення кількості гравців може призвести до квадратичного збільшення мережевого трафіку, що генерується через збільшення кількості взаємодій між гравцями

Групування може бути частково усунуто за допомогою динамічних зон. У міру того, як все більше гравців переходять в популярну зону, зона може бути розділена. Однак, якщо зони стають значно менше, ніж типовий ЗІ, поділ насправді більше не допомагає, оскільки вузьким місцем стають міжзонні комунікації. Тоді

один з варіантів полягає в тому, щоб зменшити ЗІ гравців. Однак це може погіршити ігровий досвід. Крім того, в залежності від дизайну гри, ЗІ може бути додатково розбитий на різні типи.

2.4 Узгодженість

У розподіленій архітектурі, такій як багатокористувацька гра, одночасні і, можливо, конфліктуючі оновлення можуть призводити до неузгоджених станів. Невідповідності виникають через виконання паралельних і конфліктуючих оновлень, і механізми узгодженості повинні уникати або виправляти їх. Наприклад, якщо два гравці стріляють у третього гравця майже одночасно, всі гравці повинні бачити оновлення в тому ж порядку, і тільки перший постріл повинен бути успішним у всіх репліках.

Як правило, системи побудовані таким чином, що якщо всі репліки виконують всі оновлення в одному і тому ж порядку, то всі репліки будуть мати однаковий стан. Однак, якщо повідомлення приймаються не по порядку, їх виконання може призвести до неузгодженого стану, якщо повідомлення з порушенням порядку є причинно-наслідковими.

Дуже сильна форма узгодженості досягається, якщо кожна взаємодія розглядається як транзакція, яка забезпечує транзакційні властивості, такі як ізоляція та атомарність. Однак надання транзакційних властивостей часто є дорогим і може виявитися нездійсненним для всіх ігрових дій. Наприклад, запуск двофазного протоколу фіксації для забезпечення атомарності призводить до великих затримок, які можуть значно знизити інтерактивність гри. Кінцева узгодженість є слабшою формою узгодженості. Це дозволяє окремим копіям об'єкта бути тимчасово неузгодженими, але в кінцевому підсумку узгодженими, що означає, що якщо дія оновлення припиниться на досить тривалий час, всі копії об'єкта в кінцевому підсумку матимуть однаковий стан. Оскільки інтерактивність

має важливе значення для успіху більшості ігор, вони часто жертвують узгодженістю заради інших цілей і в результаті гри зазвичай забезпечують вирішення невідповідностей замість запобігання невідповідностей, забезпечуючи не більше ніж можливу узгодженість.

Вище у розділі було описано модель реплікації первинної копії, яка вимагає, щоб всі оновлення спочатку виконувалися в первинній копії. Це спрощує управління узгодженістю, оскільки неможливо, щоб конфліктуючі оновлення виконувалися в різних копіях. Замість цього всі оновлення серіалізуються в основній копії. Однак репліки отримають оновлення тільки через деякий час після того, як вони відбудуться на основній. Протягом цього періоду копії застарівають. Гравці спостерігають за цими значеннями стану і можуть ініціювати неприпустимі запити на оновлення основної копії на основі цих застарілих значень. Таким чином, дії, вжиті гравцями на основі їх локального стану, можуть привести до результатів, яких гравці не очікували. В ідеалі первинна копія безпосередньо надсилає оновлення всім реплікам, щоб звести до мінімуму старіння. Однак це дорого, і можуть бути використані інші методи, які можуть збільшити затримку і старіння, що випробовуються репліками.

Існує кілька методів, щоб приховати старіння і невідповідності через затримку оновлень. Ці методи також можуть бути використані для приховування втрати повідомлень. Зазвичай вони підпадають під дві категорії механізмів прогнозування контрактів (Predictive Contract Mechanisms) і моделювання з декількома рішеннями (Multiresolution Simulation), часто використовуються разом.

Мертвий рахунок (Dead reckoning [8]) є поширеною формою РСМ. Спочатку він був розроблений в області навігації і авіації для обчислення поточного положення літака на основі попереднього положення і вектора руху. В іграх використовується для обчислення положення гравця в майбутньому кадрі на основі його попереднього місця розташування, а також швидкості і напрямку руху. Він також може бути використаний для виявлення зіткнень, які відбудуться в майбутньому. Мертвий рахунок використовується, якщо повідомлення не надходять вчасно через затримки або втрати. Однак також можливо, що всі репліки

безперервно виконують мертвий підрахунок, і первинна копія відправляє оновлення стану реплік тільки в тому випадку, якщо виявляє, що стан, обчислений за допомогою мертвого підрахунку, відрізняється від істинного стану більше, ніж певний поріг. В якості предикторів були запропоновані нейронні мережі для PCM, а також гібридний предиктор мертвого рахунку/найкоротшого шляху.

Ступінь неузгодженості може бути визначена шляхом порівняння (потенційно неузгодженого) стану кожного клієнта з віртуальним ідеальним станом, який отримує і виконує всі взаємодії без затримок і в правильному порядку. В цілому об'єктивну систему оцінки якості досвіду в іграх можна визначити за трьома критеріями: оперативність, точність і справедливість. Оперативність – це час, необхідний системі для реагування на подію, точність – це ступінь точності, необхідна для успішного виконання дії, а справедливість – це ступінь відмінності між ігровими середовищами всіх гравців.

3 ДОСЛІДЖЕННЯ ТА АНАЛІЗ АРХІТЕКТУР

3.1 Дослідження однорангової архітектури

Однорангові системи зазвичай створюються шляхом спеціального додавання вузлів і побудови оверлейної мережі прикладного рівня поверх мережевого рівня. Фундаментальною вимогою для дуже високої масштабованості є те, що жоден одноранговий вузол не може знати всіх інших однорангових вузлів в системі. Замість цього він завжди підключений лише до невеликої частини однорангових вузлів. Системи P2P широко використовуються в системах поширення контенту (файлів). Замість того, щоб мати один сервер, який надає контент всім клієнтам, однорангові вузли можуть завантажувати контент з інших однорангових вузлів і, в свою чергу, пропонувати свій контент іншим. Для того щоб одноранговий вузол міг знайти шуканий вміст, повинні бути створені ефективні протоколи маршрутизації запитів.

Залежно від того, як вузли з'єднані один з одним, накладання P2P, як правило, поділяються на два типи: структуровані та неструктуровані. У структурованих системах P2P детермінований протокол використовується для формування певної структури графа (шляхом визначення того, які однорангові вузли будують з'єднання один з одним), який гарантує, що будь-який вузол може направити повідомлення будь-якому іншому вузлу (або знайти об'єкт) в мережевому оверлеї (шляхом обміну O(log(N)) повідомлень, де n-кількість вузлів). Механізми маршрутизації на основі ключів використовуються для надання служби пошуку, аналогічної хеш-таблиці, але відрізняється тим, що пари (ключ, значення) розподілені по вузлах. Вони можуть додавати або видаляти вузли з оверлея з низькими накладними витратами. Прикладами таких p2p-субстратів є Pastry, Chord, Tapestry і Mercury. Розподілені хеш-таблиці (DHT [9]) можуть використовувати ці субстрати в якості базового механізму для забезпечення функціональності хеш-таблиці, розподіленої по багатьох вузлах.

У неструктурованих p2p-системах не існує детермінованого алгоритму організації та оптимізації мережевих підключень між одноранговими вузлами. Мережеві з'єднання зазвичай встановлюються випадковим чином або з використанням імовірнісних механізмів, метою яких є конвергенція в відповідне накладення (наприклад, шляхом підключення з більш високою ймовірністю до вузлів, які семантично близькі). Пошук в таких мережах часто виконується імовірнісним способом, і для прискорення пошуку використовуються механізми надмірності, такі як розсилання або реплікація контенту.

Основна ідея багатьох ігор на базі P2P полягає в тому, щоб розподіляти стан гри між одноранговими вузлами, а разом з ним і завдання обробки, мережі та зберігання. У схемі реплікації на основі первинної копії це означає розподіл первинних копій об'єктів між одноранговими вузлами. Крім того, накладення P2P можна використовувати для поширення оновлень.

Існують структуровані, неструктуровані та гібридні архітектури P2P.

Структуровані архітектури P2P зазвичай використовують DHT (розподілена система на основі хеш-таблиць) як основний механізм для розподілу стану гри та розповсюдження оновлень. Для прикладу можна взяти Pastry (оверлейна маршрутизаційна сітка) для розповсюдження ігрових об'єктів і Scribe (примітив багатоадресної розсилки) для поширення оновлень.

На рисунку 3.1 зображено накладення та маршрутизацію повідомлень Pastry.

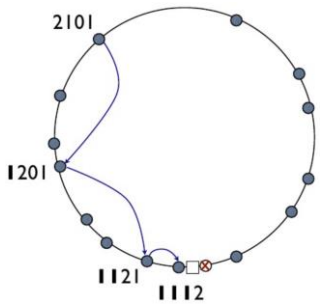

Рисунок 3.1 – Приклад маршрутизації повідомлень Pastry

У Pastry кожен вузол має 128-бітний ідентифікатор вузла. Ідентифікатори вузлів можуть бути рівномірно розподілені з використанням хеш-коду IP-адресів

вузлів або їх відкритих ключів. Загалом, для поширення об'єктів з використанням Pastry (тут об’єкт відноситься до будь-якого об'єкта серверу) кожному об'єкту присвоюється ObjectId (хеш-код об'єкта), а його первинна копія розміщується на вузлі Pastry, ідентифікатор вузла якого знаходиться найближче до ObjectId. Об'єкт можна пізніше знайти, надіславши повідомлення з ObjectId в якості ключа. Повідомлення буде автоматично перенаправлено на вузол, на якому знаходиться об'єкт. Таким чином, коли вузол з реплікою хоче оновити об'єкт, він повинен надіслати повідомлення про оновлення з ідентифікатором об'єкта в якості ключа, і воно буде автоматично отримано вузлом з первинною копією, який потім може виконати оновлення. Метою розробки в Pastry є можливість ефективної маршрутизації повідомлення із заданим ключем на вузол з найближчим ідентифікатором до цього ключа.

Scribe – це примітив багатоадресної розсилки, побудований поверх Pastry. Для кожної групи багатоадресної розсилки Scribe будує дерево багатоадресної розсилки, вузли якого є одноранговими вузлами накладення. Вузли можуть приєднуватися до груп багатоадресної розсилки і залишати їх. Будь-яке багатоадресне повідомлення групі відправляється всім поточним членам групи, звідки воно відправляється на кореневий вузол групи і потім пересилається по деревоподібної структурі багатоадресної розсилки. Ідея полягає в тому, що кореневий вузол - це вузол, ідентифікатор вузла якого найближче до ідентифікатора групи. Тому, щоб знайти кореневий вузол, Scribe використовує протокол маршрутизації запитів Pastry. Scribe забезпечує максимально ефективну доставку багатоадресних повідомлень і повністю децентралізований.

Багато архітектур використовують ті ж принципи, однак вони можуть використовувати різні DHT, управління інтересами та системи публікацій та підписки.

Неструктуровані архітектури – це архітектури в яких відсутній глобальний механізм, такий як DHT, який управляв би з'єднаннями і виходами, підтримував накладення або надавав глобальні протоколи маршрутизації.

Більшість пропонованих неструктурованих архітектур P2P засновані на системі взаємного повідомлення. У цих системах немає необхідності ділити світ на регіони; замість цього гравці використовують механізм для виявлення інших гравців у своїх зонах інтересу і безпосередньо відправляють свої оновлення цим гравцям. Щоб виявити гравців, які переміщуються в ЗІ гравця, гравці залежать від повідомлень своїх сусідніх гравців. Для прикладу можна взяти pSense систему. pSense призначений для управління оновленнями позицій гравців і не бере до уваги інші типи дій. Його дизайн заснований на тому факті, що однорангові вузли в ЗІ гравця повинні швидко отримувати оновлення, в той час як іншим одноранговим вузлам взагалі не потрібно отримувати оновлення. Тому всякий раз, коли гравець переміщається, оновлення відправляються на всі вузли в ЗІ гравця. Для того, щоб зробити це, підтримується список вузлів в ЗІ, званий списком сусідів. Кожного разу, коли вузол залишає ЗІ, він видаляється зі списку.

На рисунку 3.2 зображено приклад роботи pSense системи.

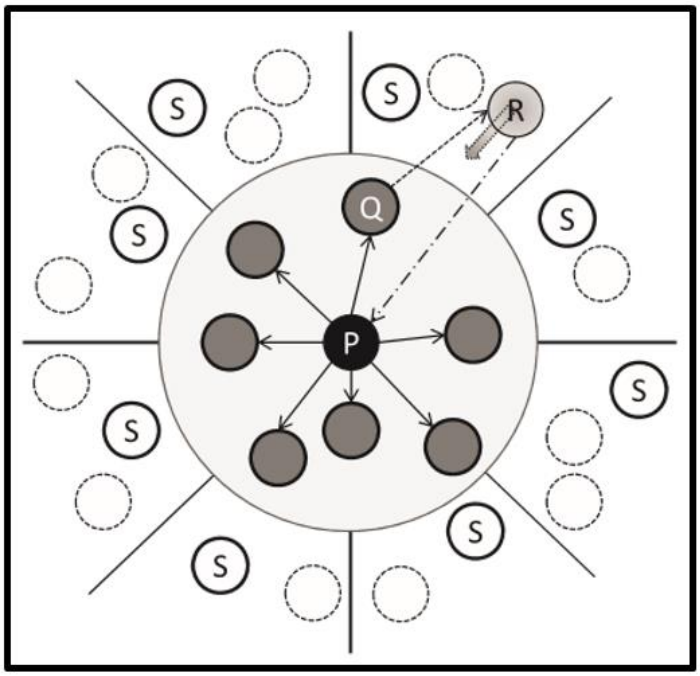

Рисунок 3.2 – Приклад системи pSense

All-to-all – це ще одна форма неструктурованої архітектури P2P, в якій всі гравці підтримують повний стан гри і використовують зв'язок точка-точка для

поширення оновлень. У таких системах необхідні спеціальні методи для зниження вимог до пропускної здатності.

Гібридні архітектури – це архітектури які можуть бути використані як у поєднанні P2P з сервером, так і в повністю безсерверному варіанті.

У комбінуванні з сервером, сервер відповідає за зберігання конфіденційних даних користувача, таких як імена користувачів і паролі, а також ділить ігровий світ на сегменти і виконує управління інтересами. Також сервер відповідає за поширення повідомлень між одноранговими вузлами і виконання перевірок безпеки оновлень.

У варіції типу P2P-P2P, система являє собою особливу форму гібридної системи, оскільки вона поєднує в собі два типи архітектур P2P: структуровану і неструктуровану та ділить ігровий світ на зони, контрольовані координаторами, і використовує Pastry і Scribe для поширення стану та оновлення серед колег. Однак координатори також періодично обмінюються інформацією про свій район.

Як вже було сказано до цього в більшості багатокористувацьких ігор використовується первинна реплікація. Різні архітектури пропонують різні місця для зберігання основної копії та способи надсилання оновлень до реплік. Крім того, вони надають різні методи для підтримки стану гри узгодженим.

Розташування первинних копій. У системах, де суперпіри є координаторами для регіонів, первинні копії всіх об'єктів в регіоні знаходяться на координаторі. Сюди входять гравці, неігрові персонажі та інші типи об'єктів. Координатор отримує всі запити на оновлення для цих об'єктів, виконує їх і поширює оновлення, як правило, через багатоадресну розсилку. У цих системах балансування навантаження може бути досягнуте шляхом зменшення розмірів областей для перевантажених вузлів, тим самим скоротивши кількість об'єктів, що обслуговуються координатором. Особливою проблемою можуть бути NPC, оскільки вони повинні виконати функцію мислення, щоб визначити свої наступні дії. Враховуючи, що обчислення штучного інтелекту можуть бути трудомісткими, це може призвести до перевантаження координатора. Координатор може прийняти рішення делегувати це завдання іншим учасникам в регіоні.

З іншого боку, зазвичай в системах взаємного повідомлення і загального доступу кожен гравець є власником основної копії свого аватара і може виконувати оновлення локально. Поширення оновлень здійснюється безпосередньо серед інших однорангових вузлів.

Як правило, гравець підтримує лише копії об'єктів у своєму поточному ЗІ. Коли ЗІ змінюється, одноранговому вузлу, можливо, доведеться отримувати нові репліки. Це може зайняти багато часу і привести до зниження якості гри. Щоб обійти цю проблему, об'єкти можуть бути попередньо обрані. Часто напрямок, в якому рухається гравець, можна передбачити. Таким чином, копії відповідних об'єктів можуть бути передані до того, як вони фактично потраплять в ЗІ. Можна також просто мати область підписки, яка більше, ніж ЗІ. Всякий раз, коли об'єкт знаходиться в цій області підписки, гравець підписується на нього і отримує репліку. Аналогічно, репліки об'єктів видаляються тільки тоді, коли вони більше не знаходяться в цьому ЗІ протягом деякого часу. Це запобігає поведінці, при якій гравець постійно створює і видаляє репліки через невеликі переміщень.

Механізм реплікації тісно пов’язаний з поняттям узгодженості. Насправді механізми узгодженості є різні і багато з низ можуть використовуватися як у архітектурі P2P так і в інших. . Модель первинної копії забезпечує суворий контроль узгодженості об'єкта (це означає, що одночасні і конфліктуючі оновлення не відправляються реплікам), проте репліки можуть містити застарілу інформацію.

Щоб приховати застарівання даних, можна використовувати кілька механізмів. У розділі 2 згадували про механізми мертвого рахунку і множинного дозволу. Вони також можуть бути застосовані в архітектурах P2P. Dead reckoning можна використовувати для гравців, які не часто відправляють свої оновлення. Однак, незважаючи на те, що було запропоновано багато підходів для підвищення точності, Dead reckoning, як правило, здатний витримувати затримки всього в кілька сотень мілісекунд і обмежений рухами гравця.

Особлива проблема виникає, якщо дія оновлює більше одного об'єкта, оскільки це вимагає координації між різними первинними копіями. В ідеалі таке виконання є транзакційним, забезпечуючи атомарність і ізоляцію. Однак добре

відомі протоколи, такі як методи блокування або кворуму, є дорогими, і більшість ігор допускають більш низькі рівні узгодженості.

Низький рівень узгодженості визначає ступінь старіння даних. Прогресивні рухи гравців можуть потрапляти в цю категорію. Мертвий розрахунок, наприклад, можна вважати механізмом з низькою узгодженістю. Три наступних рівня призначені для дій з декількома об'єктами. Вони відрізняються тим, як вони забезпечують ізоляцію транзакційних властивостей. Середній рівень узгодженості не забезпечує ізоляції, зокрема, він допускає застаріле читання, таким чином, дії можуть привести до результатів, які гравець не очікує (наприклад, підняття порожньої пляшки, яку гравець вважав повною). Високий рівень узгодженості забезпечує ізоляцію тільки щодо деяких спеціально зазначених атрибутів, таких як ціна об'єкта. Нарешті, рівень точної узгодженості забезпечує повну ізоляцію, уникаючи будь-якої форми застарілого читання. Очевидно, що чим вище рівень узгодженості, тим дорожче механізм, який його забезпечує.

Не у всіх іграх використовується класичний підхід до первинного копіювання. Деякі дозволяють відправляти оновлення в будь-яку копію (наприклад, для кращої чуйності клієнта). У цьому випадку необхідний протокол обробки невідповідностей.

У багатьох іграх оновлення виконуються на репліках з оптимізмом під час локальної відправки оновлення. Це допомагає приховати затримку, що виникає при відправці оновлень на основний сервер. Якщо дія призводить до різних змін в репліці і первинному сервері, необхідний протокол вирішення невідповідностей. Одним з простих механізмів вирішення, що використовуються, наприклад, в мобільних іграх, може бути вибір одного з станів як авторитетного стану, відкидаючи всі інші неузгоджені стану. У архітектурі Colyseus, якщо є одночасні оновлення, останнє отримане оновлення виграє без будь-якого вирішення конфлікту. Альтернативною формою вирішення конфліктів є повернення до старих, послідовних станів. Timewarp – це механізм на основі відкату, запропонований для розподіленого моделювання та баз даних, де всім копіям дозволено виконувати оновлення. Вони відстежують старі стани і відкочуються

при виникненні невідповідностей. Такі невідповідності можуть виникати, наприклад, якщо вузли отримують оновлення, що вийшли з ладу. Timewarp також періодично обчислює Глобальний віртуальний час (GVT), який являє собою момент часу в системі, коли гарантується, що жоден учасник не повернеться до попереднього часу. Стани з моментом часу до GVT можуть бути видалені, оскільки вони ніколи не будуть використовуватися. Оскільки відкати можуть викликати невдоволення гравців, в Timewarp було запропоновано кілька оптимізацій для поліпшення його продуктивності в ігровому середовищі. Hydra пропонує нову модель програмування для забезпечення гарантій узгодженості при збої вузлів в ігровій архітектурі P2P. Це досягається за допомогою системи контрольних точок і відновлення стану гри, а також накладення умов на обробку повідомлень і доставку повідомлень.

Також слід зауважити що одною з причин, чому не вибирають цю архітектуру є те, що архітектури P2P, в результаті їх проектування, більш схильні до атак і шахрайства, ніж централізовані систем. Наприклад, якщо для розповсюдження оновлень використовується накладення P2P, гравці можуть шахраювати, приховуючи або отримуючи доступ до повідомлень про оновлення, що дає їм несправедливу перевагу. Якщо Архітектура P2P забезпечує розподілене виконання оновлень і дозволяє гравцям зберігати первинні копії ігрових об'єктів, шахраї можуть маніпулювати їх сховищами об'єктів і виконувати зміни стану гри, які порушують правила гри.

Архітектури P2P використовуються для розподілу станів, а також для поширення оновлень серед однорангових вузлів. Ці архітектури можна розділити на три категорії: структуровані, неструктуровані та гібридні. Хоча архітектури P2P можуть забезпечити більш високу масштабованість і нижчу вартість, вони повинні вирішувати багато проблем, такі як обмежена пропускна здатність або вразливість для шахрайства.

3.2 Дослідження клієнт-серверної архітектури

Зазвичай для достатньо великої кількості ігрових проектів використовують архітектуру клієнт-сервер де один сервер підтримує стан ігрового світу або розрізненої частини ігрового світу.

На рисунку 3.3 зображено типову клієнт-серверну архітектуру.

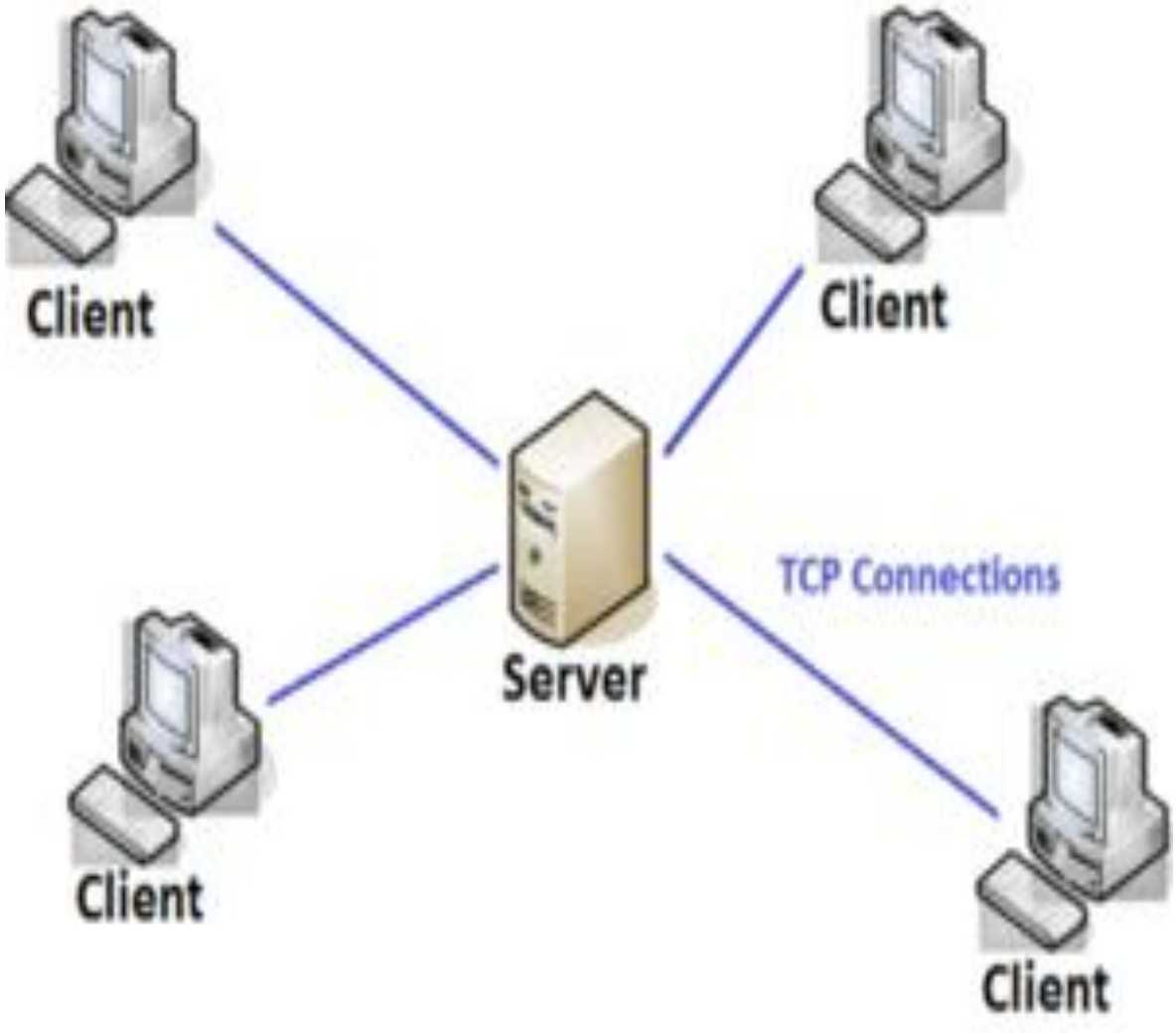

Рисунок 3.3 – Приклад типової клієнт-серверної архітектури

Ігровий стан зазвичай структурується як набір об'єктів, кожен з яких представляє частина ігрового світу, таких як територія ігрового світу, аватари гравців, керовані комп'ютером гравці (боти), предмети і так далі. Кожен об'єкт

пов'язаний з фрагментом коду, званою функцією мислення, яка визначає дії об'єкта. Типові функції мислення перевіряють і оновлюють стан як пов'язаного об'єкта, так і інших об'єктів в грі. Стан і виконання гри складаються з комбінації цих об'єктів і пов'язаних з ними функцій мислення.

Сервер запускає цикл дискретних подій. У кожній ітерації (або фреймі на ігровій мові) сервер викликає функцію think для кожного об'єкта в грі і відправляє нове уявлення стану гри кожному гравцеві. В іграх з частотою кадрів в секунду виконується від 10 до 20 ітерацій кожну секунду; ця частота (фрейм-рейт [10]) зазвичай нижче в інших жанрах.

Вимоги до пропускної здатності ігрового сервера в основному визначаються трьома ігровими параметрами: кількістю об'єктів в грі N, середнім розміром цих об'єктів S і частотою кадрів гри F. На приклад, якщо розглянути тільки об'єкти, що представляють гравців. N дорівнює 100 гравців (всі головні об’єкти інтересу), S становить близько 200 байт, а F дорівнюватиме 10 оновлень в секунду. Наївна реалізація сервера, яка просто транслює оновлення всіх об'єктів всім ігровим клієнтам C, зажадає витрат на пропускну здатність в C * N * S * F, або близько 160 Мбіт/с в іграх за участю 100 гравців. Для зниження цієї вартості використовуються дві поширені оптимізації: фільтрація по області, яка вже була описана вище та дельта-кодування.

Дельта-кодування надає змогу зменшувати розмір повідомлення, відправляючи тільки відмінності (дельту) в повідомленнях оновлень. Наприклад, відправляються тільки ті атрибути об'єкта, які змінилися з моменту останнього поширення оновлення. В теорії за допомогою цього методу повідомлення про оновлення об'єкта можна зменшити з 200 байт до 24 байт. Тому у більшості випадків надсилають різницю (тобто. дельту) між оновленнями. Ці оптимізації (фільтрація по області та дельта-кодування) зменшують N і S відповідно. Так під час ігор вимоги до пропускної здатності сервера можуть знижуватись приблизно до 120 Кбайт/с при умові, що у ЗІ гравця знаходиться 5 об’єктів інтересу.

Далі за основу буде взята архітектура Colyseus, так як на мою думку ця архітектура є одним із непоганих варіантів для розробки FPS ігор.

На рисунку 3.4 Зображено компоненти архітектури Colyseus

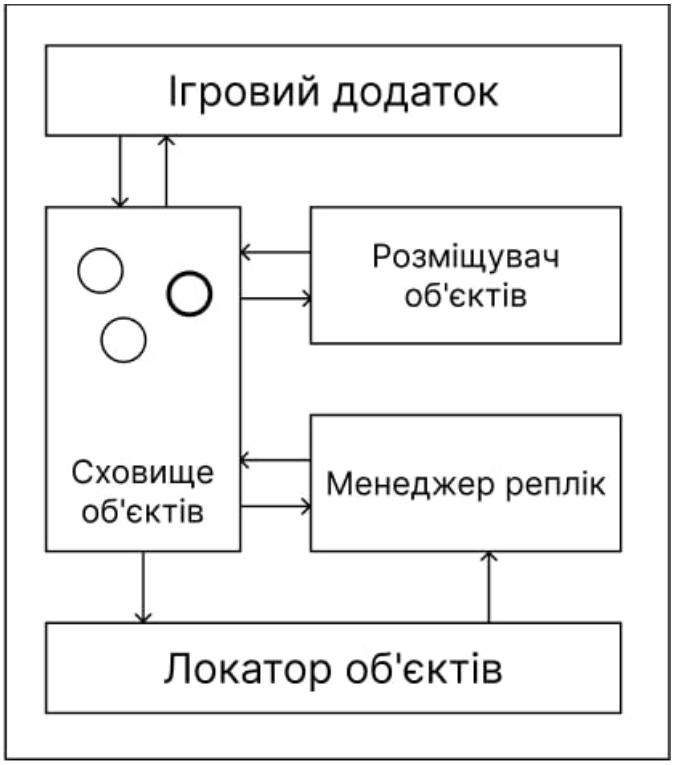

Рисунок 3.4 – Компоненти архітектури Colyseus

Існує два типи ігрових об'єктів: незмінні і змінювані. Ми припускаємо, що незмінні об'єкти (наприклад, геометрія карти, код гри та графіка) реплікуються глобально (тобто кожен вузол у системі має копію), оскільки вони оновлюються дуже рідко, якщо взагалі оновлюються. Colyseus управляє колекцією змінюваних об'єктів (наприклад, аватари гравців, керовані комп'ютером персонажі, двері, предмети), який можна назвати глобальним сховищем об'єктів.

Кожен об'єкт в глобальному сховищі об'єктів має первинну (авторитетну) копію, яка знаходиться рівно на одному вузлі. Оновлення об'єкта, що виконуються на будь-якому вузлі в системі, передаються основному власнику, який забезпечує порядок серіалізації оновлень. На додаток до первинної копії кожен вузол в системі може створити вторинну репліку. Ці репліки дозволяють віддаленим вузлам виконувати код, який звертається до об'єкта. Репліки слабо узгоджені і синхронізуються з первинним джерелом в залежності від програми. На практиці

вузол, що містить первину копію може синхронізувати репліки так само, як видимі об'єкти синхронізуються на ігрових клієнтах в архітектурі клієнт-сервер.

Таким чином, кожен вузол має локальне сховище об'єктів, яке являє собою набір первинних і реплік, засіб управління репліками, який синхронізує первинні і вторинні репліки, і засіб розміщення об'єктів, який вирішує, де розміщувати і переносити первинні репліки.

Кожен первинний об'єкт передбачає набір об'єктів, які він очікує прочитати або записати в найближчому майбутньому, і Colyseus попередньо витягує копії цих об'єктів. Це передбачення задається як вибірковий фільтр за атрибутами об'єкта, який по суті є областю інтересу об'єкта. Вважається, що більшість ігор можуть коротко висловити свої області інтересів, використовуючи предикати діапазону по кілька атрибутів об'єктів, які особливо добре працюють для опису просторових областей в ігровому світі.

Colyseus підтримує репліки, які знаходяться в межах об'єднання областей, що представляють інтерес для його основних вузлів, в локальному сховищі об'єктів кожного вузла. Colyseus може використовувати як локатор об'єкта традиційний рандомізований DHT, або DHT з можливістю запиту діапазону. Цей механізм реалізовують за принципом публікації-підписки, який вже було розглянуто у розділі архітектури P2P.

Особливою проблемою при застосуванні DHT для визначення місця розташування об'єкта в режимі реального часу є подолання затримки між відправкою підписки і отриманням відповідних публікацій.

Компонент управління репліками управляє синхронізацією реплік, відповідає на запити про реплікацію даних на інших вузлах і видаляє репліки, які більше не потрібні. Первинні копії синхронізують репліки ідентичним чином, як виділені ігрові сервери синхронізують клієнтів: кожен кадр, якщо основний об'єкт змінено, оновлення в дельта-кодуванні відправляється всім реплікам. Аналогічно, при зміні вторинної репліки оновлення з дельта-кодуванням відправляється у первинну репліку для серіалізації. Ця модель проста і відображає ту ж слабку узгодженість в існуючих клієнт-серверних архітектурах.

Важливим аспектом менеджера реплік Colyseus є поділ виявлення об'єктів та синхронізації реплік. Як тільки вузол виявляє цікаву йому репліку, з цього моменту він синхронізує репліку безпосередньо з первинною. Вузол періодично реєструє інтерес до вузла, на якому розміщений основний вузол, щоб продовжувати отримувати оновлену інформацію про репліку.

Щоб швидше знаходити недовговічні об'єкти, такі як, на приклад, стріли, Colyseus використовує спостереження про те, що більшість об'єктів походять з місць, близьких до їх творця, тому вузли, зацікавлені в творці, ймовірно, будуть зацікавлені в нових об'єктах. Наприклад, стріла з'являється в тому ж місці, що і гравець, який її випустив.

У colyseus записи в репліки є попередніми і відправляються на первинний сервер для серіалізації. Іншими словами, окремі об'єкти слідують простої моделі первинного резервного копіювання з оптимістичною послідовність. Стан резервної копії відстає від основного на невелике тимчасове вікно і в кінцевому підсумку узгоджені після цього тимчасового вікна.

Будемо вважати, що більшість ігор з швидким темпом швидше потерплять тимчасову неузгодженість, ніж матимуть наслідки затримки запису (тобто дій гравця). Це повинно гарантувати, що час очікування репліки залишається нижче 100 мс майже весь час. Обмежена застарілість зазвичай допустима в іграх, оскільки існує фундаментальна межа людського сприйняття в коротких часових масштабах і ігрових клієнтах може екстраполювати або інтерполювати зміни об'єктів, щоб надати гравцям плавний огляд гри.

Для визначення місця розташування об'єктів Colyseus реалізує розподілену службу визначення місця розташування на DHT. На відміну від інших сервісів публікації та підписки, локатор об'єктів в Colyseus повинен мати можливість знаходити об'єкти за допомогою запитів діапазону, а не точних збігів. Крім того, елементи даних (тобто. інформація про місцезнаходження об'єкта) часто змінюються, і відповіді на запити повинні доставлятися швидко, щоб уникнути погіршення узгодженості уявлень на різних вузлах системи.

DHT забезпечують масштабоване зберігання метаданих і їх розміщення на великій кількості вузлів, зазвичай забезпечуючи логарифмічну прив'язку до кількості переходів, які повинні пройти пошукові запити. При використанні традиційного DHT локатор об'єктів розбиває карту на дискретну кількість регіонів, а потім зберігає кожну публікацію в DHT під своїм (випадковий) регіональний ключ. Аналогічно, підписки об'єднуються в пошукові запити DHT для кожного регіону, на який накладається запит діапазону. Коли кожен пошук DHT досягає вузла, що зберігає метадані для цього регіону, він повторно повертає публікації, що відповідає вихідному запиту, назад на вихідний вузол.

DHT з можливістю запиту діапазону можуть краще підходити для архітектури розподіленої гри. На відміну від традиційних DHT, які зберігають публікації під дискретними випадковими ключами для досягнення балансу навантаження, DHT з можливістю запиту діапазону організовує вузли в круговому накладенні, де сусідні вузли відповідають за безперервний діапазон ключів. Запит діапазону зазвичай маршрутизується шляхом доставки його вузлу, відповідальному за крайнє ліве значення в діапазоні. Потім цей вузол пересилає запит іншим вузлам в діапазоні. Наприклад, використовуючи DHT з можливістю запиту діапазону, об'єкт розміщення може використовувати атрибут вимірювання безпосередньо в якості ключа. Оскільки значення ключів зберігаються в оверлеї безперервно (а не випадковим чином), запити діапазону можуть бути виражені безпосередньо, замість того, щоб розбиватися на кілька пошукових запитів DHT. Більш того, метадані про місцезнаходження об'єкта і запити, ймовірно, будуть демонструвати просторову локальність, яка відображається безпосередньо на оверлей, дозволяючи відправнику об'єкта обходити шляхи маршрутизації і доставляти повідомлення безпосередньо на місце зустрічі шляхом кешування останніх маршрутів. Нарешті, оскільки вузли динамічно балансують навантаження відповідно з розподілом публікацій і підписок, вони можуть краще справлятися з розподілом популярності регіонів.

У більшості систем публікації та підписки, у DHT реєструються і підтримуються тільки підписки, а публікації – ні. Локатор об'єктів зберігає як

публікації, так і підписки в м'якому стані, термін дії якого закінчується після TTL, що переноситься кожним елементом. Коли надходить підписка, вона зіставляється з усіма збереженими в даний момент публікаціями, на додаток до публікацій, які надходять після неї.

Цей дизайн досягає двох цілей: по-перше, якби зберігалися тільки підзаголовки, слухачам довелося б чекати наступної публікації цікавого об'єкта, перш ніж він буде підібраний на рандеву. Зберігаючи публікації, підписка може бути негайно зіставлена з останніми публікаціями. Цього достатньо для інформування вузла про відповідні об'єкти через просторову локалізацію оновлень об'єктів. По-друге, різні типи об'єктів змінюють свої атрибути іменування з різною частотою (наприклад, предмети змінюють місце розташування тільки в тому випадку, якщо їх підбирає гравець), тому було б марнотратно публікувати їх всі з однаковою швидкістю.

Після розгляду основних принципів клієнт-серверної архітектури можна зробити висновок, що це гарна архітектура для багатьох варіантів ігор, але в неї теж є свої недоліки, котрі підсуються у наступному розділі. В даному розділи не будуть розглядатися принципи таких архітектур як сервер-сервер, так як подібні архітектури є комбінаціями розглянутих клієнт-сервер та P2P. Їх короткий огляд буде представлено у наступному розділи, де між основними архітектурами проведеться огляд за основними критеріями при їх обранні для створення багатокористувацьких онлайн ігор, включаючи шутери від першої особи (FPS).

4 ПОРІВНЯННЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

4.1 Порівняння архітектур

В онлайн іграх, які розраховані на багато користувачів, зазвичай використовуються три типи архітектур:

а) клієнт-серверна архітектура;

б) багатосерверна архітектура;

в) P2P архітектура (однорангова).

На рисунку 4.1 зображено типові архітектури для багатокористувацьких ігор.

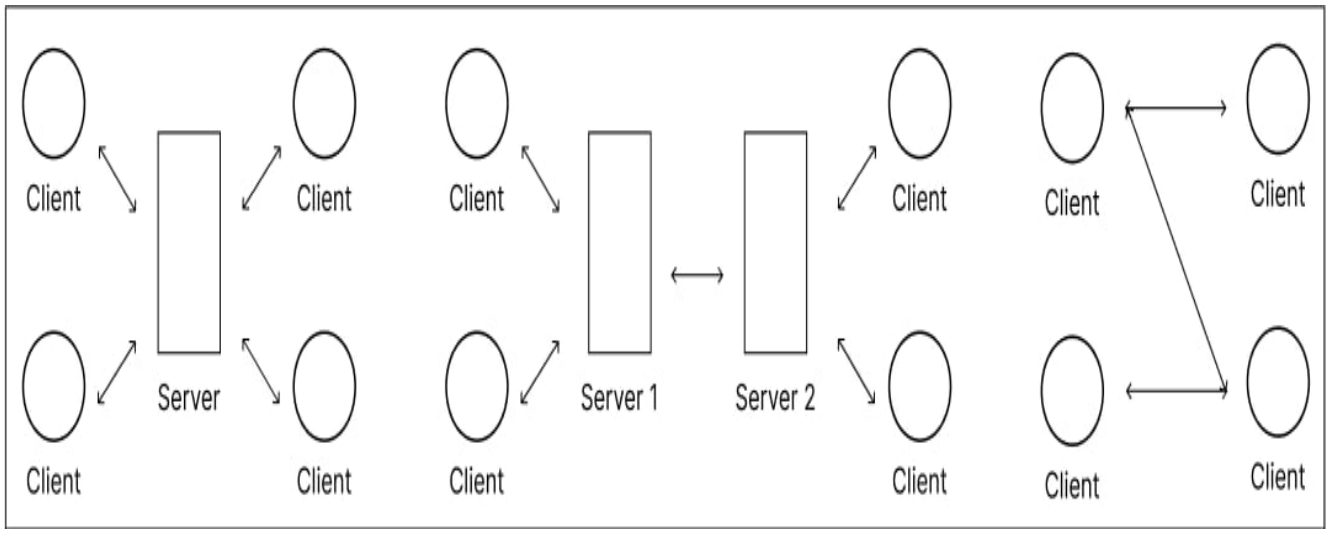

Рисунок 4.1 – Типові архітектури для багатокористувацьких ігор

В роботі було представлено архітектури клієнт-сервер та P2P, так як багатосерверна архітектура це по суті комбінація двох або більше архітектур клієнт-сервер. Також в роботі були представлені основні моменти під час їх розробки з точки зору їх компонентів і типової поведінки.

В архітектурі клієнт-сервер, сервер являється центральним контролером: він зберігає всі копії персонажів і об'єктів, він з'єднується з кожним клієнтом і підтримує глобальний стан гри. Оновлення всіх гравців і взаємодії відбуваються на сервері. Клієнти можуть отримувати необхідну інформацію про глобальний світ з

сервера. Ця архітектура дуже проста в розгортання, обслуговування та модифікації, оскільки сервер є єдиним контролером.

Архітектура з декількома серверами розширює архітектуру клієнт-сервер, зазвичай через велику кількість користувачів. Як правило, ця архітектура ділить глобальний ігровий світ на кілька регіонів, і кожен регіон працює як одна архітектура клієнт-сервер. Це означає, що кожен сервер відповідає за свій власний набір клієнтів і має свій власний глобальний світ. Кожен регіон з'єднується один з одним за допомогою зв'язку між серверами, тому сервери можуть обмінюватися один з одним своїм власним глобальним світом і підтримувати узгоджений глобальний ігровий стан. Однак у більшості багатокористувацьких ігор, гравці з одного регіону не можуть зв'язатися з гравцями з інших регіон. Існує механізм передачі даних між серверами, що дозволяє здійснювати цей зв'язок в різних регіонах.

Однорангова Архітектура (P2P) пропонується для поліпшення масштабованості. Кожен вузол працює як клієнт, так і сервер, і він буде обробляти деякі частини копій персонажів і об'єктів, а також підтримувати деякі частини глобального стану гри. Ця архітектура може бути високо масштабованою, оскільки збільшена навантаження буде розподілена на різні вузли, і за рахунок додавання нових вузлів буде додано більше ресурсів.

Існують і інші архітектури, наприклад, можна об'єднати архітектуру P2P з архітектурою клієнт-сервер яка буде розглядатися як гібридна архітектура.

4.2 Оцінка архітектур

Різні архітектури мають свої плюси та мінуси, які, доречі, можна також спробувати перекрити за допомогою різних архітектурних рішень. Зараз же буде спроба зробити об’єктивну оцінку на кожну архітектуру в цілому згідно з проведених досліджень.

На рисунку 4.1 буде представлена оцінка архітектур за п’ятибальною шкалою.

Характеристика	Клієнт-серверна	Багатосерверна	Однорангова
Узгодженість	4	5	3
Масштабованість	2	4	5
Вартість	2	2	5
Керованість	5	3	3
Пропускна здатність	4	2	4
Безпека	5	5	2

Рисунок 4.1 – Оцінка архітектур за основними критеріями

В залежності від обраних архітектурних механізмів, клієнт-серверна архітектура надає непоганий контроль узгодженості, високий рівень простоти розробки та простоти управління. Велика кількість ігор засновані на клієнт-серверній архітектурі через її простоту. Програміст простіше писати код в цій архітектурі, ніж в одноранговій. Роль сервера забезпечує найвищий рівень контролю над ігровим світом. Творці ігор можуть легко змінювати і оновлювати стан гри і мати контроль над необхідними оновленнями в такій архітектура. Простота також дає ще одну перевагу для клієнт-серверної архітектури, яка проста в управлінні система. Більше того, оскільки сервер виконує всі оновлення та вирішення конфліктів, контроль узгодженості простіше, ніж у багатосерверних архітектурах, для яких потрібні механізми передачі даних, або в архітектурах P2P, які мають ще більш високий розподіл.

Якщо ж потрібна проста в управлінні і експлуатації з більшою масштабованістю, відмовостійкістю і узгодженістю, то можна обрати багатосерверну архітектуру. Оскільки багатосерверна архітектура об'єднує кілька

односерверних архітектур разом, вона добре успадковує переваги останніх, тому нею також легко управляти. Крім того, оскільки ігровий світ розділений на кілька регіонів і один сервер обслуговує кожен з них окремо, одночасно може підтримуватися більша кількість гравців, так що він добре задовольняє умову масштабованості. Причина, за якою відмовостійкість була б головною проблемою, полягає в тому, що ці сервери можуть служити резервним копіюванням для інших серверів, але у разі одного сервера цей механізм резервного копіювання не може бути наданий. Як згадувалося раніше, контроль узгодженості є проблемою в багатосерверній архітектурі, але він простіше, ніж попередня архітектура клієнт-сервер.

Головними перевагами однорангової архітектури є її вартість та масштабованість. Як правило, архітектури P2P мають найбільший потенціал масштабованості серед усіх архітектур, оскільки кожен одноранговий вузол привносить в систему нові ресурси. Ще одною перевагою P2P є те, що ці ресурси додаються без будь-яких додаткових витрат для постачальника ігор, тому що однорангові вузли керують навантаженням, і в принципі немає необхідності додавати дорогі сервери.

В залежності від вимог до гри, потрібно досить серйозно відноситися до вибору архітектури та кожного архітектурного рішення.

ВИСНОВКИ

У ході виконання магістерської роботи було досліджено та проаналізовано архітектури, які можна використовувати для розробки ігор FPS, а також можливі архітектури в цілому.

Було розглянуто та проаналізовано предметну область. В результаті аналізу були визначені та проаналізовані основні аналоги двигунів Unity та Unreal Engine.

Далі були досліджені та проаналізовані такі основні критерії, які необхідно оцінити та обрати для майбутньої архітектури, як:

– типи об’єктів та взаємодію з ними – це допомагає продумати такі механізми як узгодженість для вирішення проблем, які можуть виникнути під час однакових подій користувачів над одним і тим самим предметом;

– реплікацію та допустиму затримку, які впливають на то, чи бачать користувачі однакову картину світу, коли виконують якісь дії і чи немає між ними видимої для людського ока затримки;

– пропускну здатність, враховування якої допомагає при різких всплесках трафіку у грі;

– узгодженість, яка відповідає за несуперечність даних та станів гри, а конфліктів між ними;

– механізм інтересів, одну із найголовніших складових, яка впливає на деякі інші механізми такі як реплікація, пропускна здатність та узгодженість.

Та в останньому розділі було детально проаналізовано такі архітектури як однорангова та клієнт-сервер, а також був зачеплений вид архітектури сервер-сервер. Його не було розглянуто детально так як він є симбіозом кількох архітектур клієнт-сервер.

В роботі не було приділено достатньо уваги конкретній реалізації так як для створення масштабного проекту бракувало достатньо часу. Є деякі напрацювання, які, базуючись на результатах роботи та багатьох опрацьованих джерелах, дають змогу покращити подальше проектування та розробки гри жанру FPS.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ПОСИЛАНЬ

1. Лунтовський А.О., Мельник І.В. Проектування та дослідження комп'ютерних мереж. К.: Університет «Україна», 2012. – 362 с.

2. Чередниченко І.С., Д.І. Матвєєв Peer To Peer–еволюція інформаційних систем. // ΛΌГОΣ. МИСТЕЦТВО НАУКОВОЇ ДУМКИ. 2020. С. 45-47

3. Gerardus Blokdyk Client Server Architecture A Complete Guide - 2020 Edition. S.: 5STARCooks, 2020. – 308 p.

4. René J. Chevance Server Architectures: Multiprocessors, Clusters, Parallel Systems, Web Servers, Storage Solutions. I.: Digital Press; 1st edition, 2004. – 784 p.

5. Philip Wik Service-Oriented Architecture: Principles and Applications. W.: A Blue Kitten Book, 2015. – 431 p.

6. Gonçalo Marques, Devin Sherry, David Pereira, Hammad Fozi Elevating Game Experiences with Unreal Engine 5: Bring your game ideas to life using the new Unreal Engine 5 and C++. S.: Packt Publishing, 2022. – 253 p.

7. Paris Buttfield-Addison, Jon Manning, Tim Nugent Unity Game Development Cookbook: Essentials for Every Game. W.: O'Reilly Media, 2019. – 647 p.

8. Wikipedia URL: https://www.wikiwand.com/en/Dead_reckoning (дата звернення: 01.03.2022)

9. Wikipedia URL: https://www.wikiwand.com/en/Distributed_hash_table (дата звернення: 01.05.2022)

10. Wikipedia URL: https://www.wikiwand.com/en/Frame_rate (дата звернення: 01.02.2022)

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ ЗА НАУКОВИМИ НАПРЯМАМИ НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

2. Чередниченко І.С., Д.І. Матвєєв Peer To Peer–еволюція інформаційних систем. // ΛΌГОΣ. МИСТЕЦТВО НАУКОВОЇ ДУМКИ. 2020. С. 45-47