БАКАЛАВРСЬКА РОБОТА

БР. ІП - 48.00.00.000 ПЗ

Група ІП-21-2

Панас Віталій

2025

Івано-Франківський національний технічний університет нафти і газу
Інститут інформаційних технологій
Кафедра інженерії програмного забезпечення

Панас Віталій Романович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Методики мережевих імплементацій ігрових стратегій

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____Панас В. Р._____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____Храбатин Роман Ігорович, к.т.н., доцент_____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.__
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Панас Віталій Романович

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Методики мережевих імплементацій ігрових стратегій”.

керівник проекту (роботи) к.ф-м.н., доц. Храбатин Р.І.

затверджені наказом вищого навчального закладу від “ 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Вступ до мережевих імплементацій ігрових стратегій
2. Передумови та пов’язані технології
3. Мережеве програмування windows sockets
4. Проектування системи vpng

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

1. Діаграма станів протоколу запуску peer-to-peer (рис.2.1,ст.22)
2. Моделі клієнт-серверного програмування (рис.2.2,ст.26)
3. Класи сокетів Windows (рис.3.1,ст.35)
4. Налаштування зв'язку між сервером і клієнтом (рис.3.2,ст.41)
5. Діаграма архітектури VPNG (рис.4.1,ст.46)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання **2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обґрунтування теми роботи	15.02.2025	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	25.02.2025	виконано
3	Побудова моделі або алгоритму власного рішення	15.03.2025	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	25.04.2025	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	10.06.2025	виконано

Студент _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 55 сторінок, 11 рисунків, 2 таблиці, список використаних джерел із 30 найменування,

Метою роботи є аналіз технологій і підходів до програмування мережевих ігор, зокрема використання Windows Sockets і розробки системи VPNG, а також надання рекомендацій для забезпечення низької затримки та високої масштабованості.

Об'єкт дослідження: процес створення та функціонування багатокористувацьких комп'ютерних ігор у мережевому середовищі.

Предмет дослідження: Методики реалізації ігрових стратегій у мережевих багатокористувацьких системах, зокрема засоби синхронізації, архітектура клієнт-серверної взаємодії та застосування Windows Sockets у контексті розробки гри типу відеопокеру.

Результати дослідження: зосереджується на теоретичних основах, технологічних передумовах і практичних аспектах реалізації.

У першому розділі - розглядаються вступні аспекти мережевих імплементацій ігрових стратегій, включаючи історію комп'ютерних ігор, сучасний стан індустрії та огляд багатокористувацьких ігор

Другий розділ - присвячений передумовам і пов'язаним технологіям, зокрема моделям і функціям програмування мережевих ігор

Третій розділ - аналізує мережеве програмування з використанням Windows Sockets, охоплюючи концепції, моделі програмування та роботу з архівами.**Четвертий розділ** - фокусується на проектуванні системи VPNG, включаючи її опис, архітектуру, модель структури, класи та поведінкові моделі.

Висновок: узагальнено ключові результати та окреслено перспективи розвитку методик мережевих імплементацій.

КЛЮЧОВІ СЛОВА: МЕРЕЖЕВЕ ПРОГРАМУВАННЯ, ІГРОВІ СТРАТЕГІЇ, БАГАТОКОРИСТУВАЦЬКІ ІГРИ, WINDOWS SOCKETS, VPNG, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, СИНХРОНІЗАЦІЯ, МАСШТАБОВІСТЬ.

ANNOTATION

The bachelor's thesis contains 55 pages, 11 figures, 2 tables, a list of used sources with 30 names,

The purpose of the work is to analyze technologies and approaches to programming network games, in particular the use of Windows Sockets and the development of the VPNG system, as well as to provide recommendations for ensuring low latency and high scalability.

Object of research: the process of creating and operating multi-user computer games in a network environment.

Subject of research: Methods for implementing game strategies in network multi-user systems, in particular synchronization tools, client-server interaction architecture and the use of Windows Sockets in the context of developing a game such as video poker.

Research results: focuses on theoretical foundations, technological prerequisites and practical aspects of implementation.

The first section - considers introductory aspects of network implementations of game strategies, including the history of computer games, the current state of the industry and an overview of multiplayer games

The second section - is devoted to the prerequisites and related technologies, in particular, models and functions of programming network games

The third section - analyzes network programming using Windows Sockets, covering concepts, programming models and working with archives. The fourth section - focuses on the design of the VPNG system, including its description, architecture, structure model, classes and behavioral models.

Conclusion: the key results are summarized and the prospects for the development of network implementation techniques are outlined.

KEYWORDS: NETWORK PROGRAMMING, GAME STRATEGIES, MULTI-USER GAMES, WINDOWS SOCKETS, VPNG, CLIENT-SERVER ARCHITECTURE, SYNCHRONIZATION, SCALABILITY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

RTS - стратегії в реальному часі

MMO - масові багатокористувацькі ігри

VPNG - Virtual Private Network Game

NTP - протокол мережевого часу

IPS - пакета протоколів Інтернету

BSD - Berkeley UNIX Berkeley Software Distribution

ВСТУП

Методики мережевих імплементацій ігрових стратегій є ключовим напрямом сучасної розробки комп'ютерних ігор, що забезпечує створення багатокористувацьких ігрових середовищ із високим рівнем інтерактивності та синхронізації. Зі зростанням популярності онлайн-ігор, таких як стратегії в реальному часі (RTS) і масові багатокористувацькі ігри (MMO), потреба в ефективних мережевих технологіях, зокрема програмуванні сокетів і оптимізації клієнт-серверної взаємодії, стала критично важливою. Використання інструментів, таких як Windows Sockets, і розробка спеціалізованих систем, наприклад дозволяють забезпечити стабільну та швидку комунікацію між гравцями в реальному часі.

Актуальність теми зумовлена швидким розвитком ігрової індустрії, де багатокористувацькі ігри займають значну частку ринку. За даними аналітичних звітів, глобальний ринок онлайн-ігор у 2025 році продовжує зростати, що підкреслює необхідність розробки надійних мережевих рішень для забезпечення низької затримки, високої масштабованості та безпеки. Водночас складність реалізації мережевих ігрових стратегій, пов'язана з синхронізацією даних, обробкою великих обсягів запитів і підтримкою різних архітектур, вказує на потребу в систематичному дослідженні методик програмування та проектування таких систем.

Метою цього дослідження є аналіз методик мережевих імплементацій ігрових стратегій, включаючи їх теоретичні основи, технологічні передумови та практичні аспекти створення багатокористувацьких ігрових систем. Робота спрямована на систематизацію знань про моделі програмування мережевих ігор, використання Windows Sockets і проектування спеціалізованих систем, таких як VPNG, а також на надання рекомендацій для розробників. Дослідження охоплює як теоретичні, так і прикладні аспекти, з акцентом на інженерні підходи до розробки.

Ця робота має на меті надати комплексне уявлення про методики

мережевих імплементацій ігрових стратегій, а також практичні рекомендації для розробників, інженерів і дослідників, які прагнуть створювати ефективні та масштабовані багатокористувацькі ігрові системи в умовах швидкого технологічного прогресу.

У роботі розглянуто історію та сучасний стан комп'ютерних ігор, огляд багатокористувацьких ігор, моделі та функції програмування мережевих ігор. Проаналізовано мережеве програмування з використанням Windows Sockets, включаючи концепції, моделі програмування та роботу з архівами. Окремо досліджено проектування системи VPNG, її архітектуру, модель структури, класи та поведінкові моделі.

Практична частина дослідження включає розробку прототипу багатокористувацької ігрової системи на базі Windows Sockets, наприклад, для стратегії в реальному часі. Оцінюються такі аспекти, як синхронізація даних, продуктивність і користувацький досвід. Проведено порівняльний аналіз різних моделей програмування за критеріями ефективності та гнучкості.

Завданнями дослідження: Провести огляд історичного розвитку комп'ютерних ігор та сучасного стану індустрії, проаналізувати підходи до програмування багатокористувацьких ігор і технології, пов'язані з мережевою взаємодією, дослідити можливості Windows Sockets для реалізації комунікації між клієнтом і сервером, розробити клієнт-серверну архітектуру відеопокеру з базовим функціоналом, виявити основні проблеми під час реалізації системи та запропонувати шляхи їх усунення, запропонувати напрями вдосконалення гри, включно з інтерфейсом, додатковими функціями (чат, ІІ) та масштабованістю системи.

Результати дослідження: побудовано архітектуру клієнт-серверної системи багатокористувацької гри типу відеопокеру, розроблено первинний інтерфейс гри на основі бібліотеки MFC

Об'єкт дослідження: Процес створення та функціонування багатокористувацьких комп'ютерних ігор у мережевому середовищі**.**

Предмет дослідження: Методики реалізації ігрових стратегій у мережевих багатокористувацьких системах, зокрема засоби синхронізації, архітектура клієнт-серверної взаємодії та застосування Windows Sockets у контексті розробки гри типу відеопокеру.

Методи дослідження: використали такі методи аналіз та синтез, моделювання, б'єктно-орієнтованого проєктування, експериментальний, програмної реалізації

Бакалаврська робота містить 55 сторінок, 11 рисунків, 2 таблиці, чотири розділи, список використаних джерел із 26 найменуванням.

РОЗДІЛ 1. ВСТУП ДО МЕРЕЖЕВИХ ІМПЛЕМЕНТАЦІЙ ІГРОВИХ СТРАТЕГІЙ

1.1 Коротка історія комп'ютерних ігор

Грати в ігри на комп'ютерах стало можливо завдяки появі мінікомп'ютерів наприкінці 1950-х років. Звільнившись від бюрократії перфокарт IBM, програмісти вперше змогли дослідити можливості, що відкриваються завдяки практичній взаємодії з комп'ютерами. Ігри були серед перших програм, які спробували створити перші «хакери», студенти-члени клубу технічних моделей залізниць Массачусетського технологічного інституту. Результатом, у 1962 році, стала спільна розробка першої комп'ютерної гри: Spacewar, базової версії того, що згодом стане аркадною грою Asteroids, у яку грали на DEC PDP-1 вартістю 120 000 доларів. [3] Комп'ютерний дизайнер Бренда Лорел вказує на це раннє визнання центральної ролі комп'ютерних ігор як моделей взаємодії людини з комп'ютером.

Чому Spacewar було «природним» рішенням для створення за допомогою цієї нової технології? Чому б не кругову діаграму, автоматизований калейдоскоп чи робочий стіл? Її розробники визначили дію як ключовий інгредієнт і задумали Spacewar як гру, яка могла б забезпечити хороший баланс між мисленням та діями для своїх гравців. Вони розглядали комп'ютер як машину, природно пристосовану для представлення речей, які можна бачити, контролювати та з якими можна грати [23]. Його цікавий потенціал полягав не в його здатності виконувати обчислення, а в його здатності представляти дії, в яких можуть брати участь люди [5].

Оскільки комп'ютери стали доступнішими для університетських дослідників протягом 1960-х років, з'явилося кілька жанрів комп'ютерних ігор. Програмісти розробили шахові програми, достатньо складні, щоб перемагати людей. Перша комп'ютерна рольова гра, Adventure, була написана в Стенфорді в 1960-х роках: вводячи короткі фрази, можна було керувати пригодами

персонажа, який подорожував чарівним ландшафтом, розв'язуючи головоломки. А в 1970 році колумніст Scientific American Мартін Гарднер познайомив американців з LIFE[21], симуляцією моделей росту клітин, написаною британським математиком Джоном Конвеєм. LIFE була першою «програмною іграшкою», захоплюючою та відкритою моделлю системного розвитку, розробленою для того, щоб її можна було нескінченно змінювати та насолоджуватися нею [1].

Звичайно ж, 1970-ті роки стали днем народження відеоаркад, домашніх ігрових систем та персональних комп'ютерів. До початку 1980-х років виробництво програмного забезпечення для комп'ютерних ігор стало цілою галуззю [1]. А за останні п'ятнадцять років, оскільки можливості персональних комп'ютерів продовжували експоненціально зростати, комп'ютерні ігри продовжували розвиватися, пропонуючи дедалі деталізованішу графіку та звуки, зростаючі можливості для взаємодії кількох гравців через модеми та онлайн-сервіси, а також дедалі складніші алгоритми моделювання.

Світ комп'ютерних ігор сьогодні варіюється від аркадних ігор, що наголошують на координації рук і очей, до рольових ігор, що додають звук і відео до формули пригод, та симуляторів, у яких гравці контролюють зростання та розвиток систем, починаючи від міст і закінчуючи галактиками та альтернативними формами життя. Видання про комп'ютерні ігри поділяють сучасну сферу на сім жанрів: екшн/аркада, пригоди, рольові пригоди, симулятори, спорт, стратегія та війна. Звичайно, в цих категоріях залишається багато спільного. Наприклад, гра про будівництво імперії, така як Civilization, знаходиться десь між військовою грою та симулятором, тоді як багато пригодницьких ігор містять інтерлюдії в аркадному стилі [11].

1.2 Статус світу комп'ютерних ігор

Комп'ютерні ігри швидко стали значною та зростаючою галуззю індустрії розваг та сучасної культури. Створюються різноманітні концептуальні та

теоретичні моделі для розуміння ігор та їхньої роботи, водночас самі ігри набувають нових вимірів завдяки своїм онлайн- та багатокористувацьким можливостям. Перехід у світ мобільних ігор створює ще більше викликів та додаткових можливостей.

Комп'ютерні ігри – це відносно нове нововведення в загальній схемі речей. Вони існували в різних формах з самого початку появи комп'ютерів і багато в чому були важливими на шляху, яким комп'ютери стали частиною нашого повсякденного життя.

Відсутність досліджень мережевих ігор ніколи не була проблемою. До виходу Doom [8] у 1993 році майже всі мережеві ігри були текстовими та використовували telnet або подібні протоколи для передачі даних від гравця до сервера і назад. Але навіть з появою Doom мережеві ігри все ще були доступні лише невеликій частині населення. Однак за останні 5 років, з розвитком Інтернету, це докорінно змінилося. В інтернет-середовищі переважна більшість мережевих гравців грають у карткові ігри, шахи, шашки та подібні ігри. Жанри ігор, які мають найбільшу кількість гравців, після салонних ігор, - це шутери від першої особи (FPS) та масові багатокористувацькі онлайн-рольові ігри (MMORPG), за якими одразу йдуть стратегії в реальному часі (RTS).

Починаючи з Doom, FPS становили значну частину мережевих ігор. У цих іграх гравець дивиться на світ крізь призму свого персонажа (від першої особи) і зазвичай повинен переміщатися по різних локаціях, вбиваючи монстрів та інших гравців, використовуючи різноманітну зброю дальнього бою, яку можна знайти на своєму шляху (стрілялка). В середньому вночі існує понад 10 000 серверів для ігор, що використовують двигун Half-Life, які підтримують понад 40 000 гравців. Інші FPS підтримують дещо меншу кількість користувачів. MMORPG – це галузь, що швидко розвивається з моменту виходу Ultima Online7 у 1996 році. MMORPG можна сміливо розглядати як графічне багатокористувацьке підземелля. 8 Усі MMORPG, випущені досі, пропонують певний механізм для просування персонажів, великі території суші для подорожей та інших гравців для взаємодії. «Велика трійка» – Asheron's Call9,

Ultima Online10 та Everquest11 – стверджує, що має майже 1 мільйон підписників разом, і хоча лише п'ята частина з них входить у систему щодня,12 ці гравці споживають значну кількість пропускної здатності. Крім того, за останні місяці було випущено ще кілька MMORPG,13 що збільшило цю загальну кількість [20].

Першою грою в жанрі RTS була Dune 2,14, яка була вільно заснована на світі із серії [2]. Ігри в жанрі RTS зазвичай характеризуються збором ресурсів, будівництвом юнітів та битвами, що складаються з великої кількості анімованих солдатів, що стоять на відстані кількох футів один від одного, і знову і знову виконують один і той самий анімований атакувальний рух. Всі ці дії відбуваються безперервно, на відміну від попередніх стратегічних ігор (найбільш помітних у Civilization15 та різних військових іграх від SSI та інших), в яких гравець міг витратити стільки часу, скільки йому було потрібно, на планування свого ходу, перш ніж натиснути кнопку процесу ходу. З моменту Dune 2 було випущено ще кілька ігор, кожна зі своєю варіацією на цю тему. Наразі кількість шанувальників RTS, які грають у Starcraft1 7 щоночі, становить щонайменше 20 000 гравців[10].

1.3 Огляд багатокористувацької гри

Ще п'ять років тому багатокористувацькі ігри були радше винятком, ніж нормою у світі відеоігор. Ідея гри онлайн з друзями або проти них обговорювалася чи читалася лише в комп'ютерних журналах. Сьогодні практично кожна нова комп'ютерна відеогра підтримує певну можливість гри для кількох гравців. Якщо ні, будьте обережні. Фанати будуть обурені тим, що в їхню нову гру не можна грати онлайн з випадковими людьми з найвіддаленіших куточків земної кулі [4].

Світ багатокористувацьких ігор вибухнув за останні кілька років, зробивши такі сайти, як The Zone від Microsoft, широко популярними серед тисяч потенційних гравців, готових і чекаючи, щоб зіграти в ту саму гру, в яку хочете ви. Важко визначити, чи матиме нова гра, яку ви купуєте, таку ж, кращу чи гіршу

ігрову спроможність між автономним та багатокористувацьким всесвітами. Така гра, як Age of Empires 2: Age of Kings, чудова для самостійної гри, але часто штучний інтелект буває або занадто складним, або занадто легким, що робить можливості повторного проходження гри дуже обмеженими. Погодьтеся, ніхто не любить отримувати шльопання щоразу, коли грає, і так само гра, в якій виграно занадто легко, стає нудною та сумною. Взяти цю гру та випустити її в Зону відкриває цілий новий світ [7].

Не встигнете й озирнутися, як вже 3:00 ранку, і ви вже п'яту гру намагаєтеся відправити своїх перських бойових слонів на турецьку базу, поки їхні яничари та пікіари повільно розбирають ваше село. Гра за реальну людину, будь то хтось, хто сидить поруч з вами, чи хтось у Європі чи Азії, робить гру цікавішою. Знання того, що є хтось на іншому кінці вашого завоювання (незалежно від того, наскільки віртуальним і незначним це може бути), має вирішальне значення. Майже кожен великий веб-портал пропонує онлайн-ігри, від Yahoo до MSN, від BBC до AOL. Усі ці сайти пропонують безліч ігор, які є абсолютно безкоштовними : просто завантажте невеликий файл, і ви граєте в шашки, шахи чи якусь іншу гру-головоломку проти когось.

Щоб грати в справді чудові багатокористувацькі ігри, такі як Age of Empires, Star Wars Galactic Battlegrounds, Ashron's Call або Ultima Online, вам доведеться придбати гру, а потім ви зможете грати [12]. Однак Ashron's Call стягує щомісячну абонентську плату за онлайн-гру з тисячами інших фентезійних гравців. Незалежно від того, чи хочете ви зіграти швидку партію в шашки, 2-годинну гру в Age of Empires чи повну кампанію, на завершення якої можуть знадобитися тижні або досягнення вашої мети, онлайн-ігри, ймовірно, доступні для всіх них. У будь-якій пошуковій системі Інтернету просто введіть «онлайн-ігри» або «багатокористувацькі ігри»,[9] і вона обов’язково видасть сотні, якщо не більше, ігор. У найближчі тижні обов’язково перевіряйте огляди, посилання, найкращі пропозиції, скріншоти та додаткові статті, що містять екшн-ігри з можливістю багатокористувацького режиму.

Щоб дві гри мали однакову популярність, вкрай важливо, щоб гравці погоджувалися щодо поточного стану гри. В іншому випадку, ви можете поговорити з людиною, якої вже немає поруч, або підібрати предмет, який уже взяв опонент. Якщо стан змінюється дуже швидко, як-от у авіасимуляторах, іграх з м'ячем або бойових сценах, це ставить високі вимоги до зв'язку між комп'ютерами двох гравців. Сучасні інтернет-протоколи погано адаптовані до цього типу зв'язку, який повинен бути одночасно безпечним і надзвичайно швидким. Варіації пропускної здатності мережі та затримки також впливатимуть на враження від гри. Щоб гра була добре сприйнята, вона повинна пропонувати гравцям відчуття справедливого ставлення, де кожен має однакову ймовірність успіху, незалежно від того, наскільки змінюється їхнє з'єднання.

Існує кілька різних різновидів багатокористувацьких ігор, кожен з яких вимагає різних підходів. З одного боку, є ігри, написані для багатьох гравців, але розроблені для гри на одній машині. З іншого боку, є ігри, написані для локальної гри, або загалом, гри на кількох машинах з локальним підключенням, яке не залежить від мережі (наприклад, гра з нуль-модемним кабелем або модемним з'єднанням). Нарешті, є ігри, які відбуваються у великих мережах. Далі я називатиму їх відповідно одномашинними, локальними та мережевими багатокористувацькими іграми.

У багатокористувацьких іграх на одному комп'ютері основний ігровий цикл повинен викликати движок для обробки кожного гравця, який бере участь у грі; це включає отримання від них вхідних даних, їх переміщення, рендеринг їхнього вигляду та обчислення будь-якої іншої логіки, пов'язаної з ними в грі. У локальній багатокористувацькій грі ігровий цикл повинен обробляти лише одного гравця, гравця на комп'ютері, на якому запущено гру, потім надсилати інформацію про гравця на інші комп'ютери та приймати інформацію про інших гравців (за потреби) у відповідь. Мережеві багатокористувацькі ігри схожі на локальні багатокористувацькі ігри, але зазвичай використовується модель клієнт/сервер, де кожен комп'ютер передає свої дані на один сервер, а потім отримує інформацію про інших гравців назад від сервера.

Багатокористувацькі ігри для одного автомата

Під час розробки однокористувацьких багатокористувацьких ігор, які сьогодні не дуже поширені на ПК, гравець повинен враховувати кілька питань. Найважливішими з них будуть: «Як кожен гравець отримуватиме інформацію в грі?» та «Як кожен гравець бачитиме рухи свого персонажа, маючи лише один екран?».

Деякі поширені підходи включають використання двох пристроїв введення, таких як джойстик і клавіатура, або спільне використання одного пристрою введення, наприклад, двоє людей використовують одну клавіатуру (для кожної дії кожного гравця повинні бути окремі клавіші). Було кілька більш гідних підходів до проблеми відображення. Більшість ігор на одному комп'ютері з багатьма гравцями одночасно просто дозволяють гравцям використовувати один і той самий вид. Як правило, це добре працюватиме лише за умови, що вид не від першої особи, і лише за умови, що (для 3D-ігор) керування камерою не залежить від позицій гравців. У старі часи сайд-скролерів ми іноді бачили розділені екрани, де вид кожного гравця відображався на різній половині екрана. Розділені екрани значно затримують час рендерингу для будь-якої гри, і вони, безумовно, можуть зробити гру громіздкою, але вони працюють.

Чи не найефективніший підхід до проблем відображення та введення, пов'язаних з багатокористувацькими іграми на одному комп'ютері, принаймні для ПК, полягає у використанні покрокової системи. Враховуючи, що вони досі використовуються, покрокові системи мали найбільше успіху протягом століть. Покрокові системи працюють, перемикаючись між гравцями та дозволяючи грати лише одному гравцеві одночасно.

Будь-хто, хто може написати однокористувацьку гру та розуміє масштабований дизайн, знає достатньо, щоб написати багатокористувацьку гру для однієї машини. Багатокористувацькі одномашові ПК-ігри дедалі більше зникають, але варто зазначити, що старі одномашові методи не є марними. Консолі все ще дуже популярні в одномашовому багатокористувацькому режимі.

Локальні багатокористувацькі ігри

Локальні багатокористувацькі ігри існують вже досить давно. Підтримка гри без модему все ще дуже поширена в сучасних іграх. Навпаки, прямі модемні з'єднання між машинами набагато менш популярні, ніж колись, через те, що їх дещо обмежують телефонні витрати, пов'язані з міжміськими модемними дзвінками.

Архітектура локальної багатокористувацької гри досить проста. Кожна машина відповідає за оновлення гри для одного гравця. Потім кожна машина зберігає зміни стану гравця в пакеті (пакет - це дані із заголовком, які передаються через з'єднання), який надсилається на інші машини. Інші машини в грі використовують ці дані для оновлення свого стану гри та надсилання інформації про рухи своїх гравців, і цикл продовжується. Очевидно, що конкретна реалізація цієї схеми відрізняється залежно від гри: в іграх у реальному часі асинхронна передача пакетів є обов'язковою, і гра повинна залишатися синхронізованою, навіть якщо пакети даних пропущені. В іграх не в реальному часі гравці можуть покладатися на надсилання та отримання пакетів у певний час, і гравцям також не потрібно турбуватися про перевантаження свого потоку даних.

Отже, ось базовий рецепт локальної багатокористувацької гри. Спочатку гравці встановлюють з'єднання з іншими машинами в грі. Це не повинно бути надто складно, враховуючи, що фізичне з'єднання (наприклад, локальна мережа, нуль-модемний кабель або пряма телефонна лінія [модем] має існувати, щоб гра вважалася "локальною"[26]. Потім гравці якимось чином домовляються про початкові налаштування гри. Нарешті, гра починається, і гравці починають щойно описаний процес передачі даних. Потім гравці розривають "з'єднання". Концептуально це просто.

Мережеві багатокористувацькі ігри

Мережеві багатокористувацькі ігри концептуально такі ж, як і локальні багатокористувацькі ігри, за кількома винятками. По-перше, кожна машина в грі все ще обробляє дані для гравця, який її використовує. Однак, замість того, щоб

надсилати пакети всім іншим гравцям у грі (або лише іншому гравцеві), у мережевій грі машини надсилають пакети на один центральний сервер і отримують пакети від сервера у відповідь. Сервер знаходиться в центрі гри, зберігаючи всю ігрову інформацію та забезпечуючи її роботу. Якщо якимось чином використовуються покрокові системи, то сервер відповідає за їх керування. Усі машини, підключені до ігрового сервера, називаються клієнтами. Ця модель загалом називається клієнт-серверною моделлю зв'язку. Клієнт-сервер - це модель зв'язку, яка використовується в реальному світі.

Мережі реалізують комунікаційні протоколи, щоб надати сенсу та структури комунікаціям, що відбуваються в них. Усі комунікації в мережі здійснюються відповідно до стандартного набору цих протоколів; якщо ви хочете ефективно використовувати мережу, то ви повинні використовувати її комунікаційні протоколи. Інтернет має TCP/IP (Протокол керування передачею/Інтернет-протокол) для цієї мети. TCP/IP – це сімейство протоколів як прикладного рівня (тобто високого рівня, як-от FTP, HTTP, Gopher), так і мережевого рівня (таких як IP, ARP, ICMP). У TCP/IP пакети створюються та маршрутизуються через мережу до відповідної машини на основі заголовків цих пакетів. Область даних цих пакетів – це прийнятне місце для розміщення даних вашої гри (вам не потрібно обробляти пакети так, як це було зроблено вище). Інтернет гарантує, [22]що ваші пакети будуть надіслані на правильну машину, тобто на машину, на якій працює сервер.

Клієнтські програми або версія гри, яку ви розповсюджуєте серед кінцевих користувачів (цю програму часто називають «клієнтом»), надсилають пакети TCP/IP на сервер, програму, яка їх аналізує та обробляє; сервер - це програма, що працює на машині, ідентифікованій IP-адресою. Сервер прослуховує «порт» TCP/IP, через який надходять дані, і надсилає їх назад своїм клієнтам через TCP/IP.

Для аматорів ігри на базі клієнт/сервер можуть створювати багато проблем, якщо спробувати розпочати їх з нуля: Перш за все, вам потрібна виділена машина з виділеною IP-адресою для запуску сервера. Далі у списку

машина повинна мати можливість фактично запускати сервер: Багато комерційних операційних систем для кінцевих користувачів, таких як Windows 95, не постачаються з реалізаціями сервера TCP/IP.

У мережевих іграх надсилання та обробка одного пакета даних пов'язані з багатьма процесами. На щастя, у більшості операційних систем ми маємо доступ до реалізацій клієнтів TCP/IP, які дозволяють нам уникнути технічних труднощів низькорівневого зв'язку з Інтернетом. Наприклад, за допомогою Winsock можна створювати програми, які виконують такі дії, як отримання веб-сторінок з порту 80 на будь-якій машині, приблизно за сторінку коду. Високорівневі API операційної системи для функцій Інтернету – це благословення, а не перешкода, незважаючи на будь-які скарги. Зрештою, чи справді практично самостійно впроваджувати близько тридцяти років Інтернету?

1.4 Висновки до розділу

Комп’ютерні ігри виникли як форма експерименту з новими технологіями й швидко перетворилися на окрему індустрію, що поєднує інтелектуальну взаємодію, розваги та складне моделювання. Від перших простих програм до сучасних багатожанрових ігор, вони стали важливою частиною культури та цифрової еволюції.

Комп’ютерні ігри перетворилися з простих текстових програм на потужну галузь із глобальною аудиторією, охоплюючи безліч жанрів – від мережевих шутерів до масштабних онлайн-рольових ігор і стратегій у реальному часі. Розвиток інтернету та мобільних технологій значно розширив можливості геймерів і вплив ігор на сучасну культуру.

Багатокористувацькі ігри пройшли шлях від рідкісного нововведення до основи сучасної індустрії, запропонувавши гравцям глобальну взаємодію, складну технічну архітектуру та різноманіття форматів – від локальних мереж до потужних клієнт-серверних систем. Вони стали не лише розвагою, а й важливою частиною цифрової культури та технологічного прогресу.

РОЗДІЛ 2. ПЕРЕДУМОВИ ТА ПОВ'ЯЗАНІ ТЕХНОЛОГІЇ

2.1 Моделі програмування мережевих ігор для багатьох гравців

У нашому сценарії два або більше гравців беруть участь у грі через мережу. Ця мережа може бути локальною мережею або Інтернетом. Гравці в грі повинні відчувати по суті одну й ту саму реальність, тому, якщо гравець А стріляє в гравця В, гравець В повинен бачити постріл, що надходить з позиції гравця А. Таким чином, ігрові світи повинні бути синхронізовані таким чином, щоб усі гравці бачили дії один одного абсолютно однаково. Таким чином, зберігається узгодженість нашої ігрової реальності. Існують дві популярні моделі синхронізації ігрових світів. У моделі peer-to-peer усі залучені комп'ютери спілкуються безпосередньо з усіма іншими комп'ютерами. У моделі клієнт/сервер кожен комп'ютер спілкується лише із сервером.

Коли цей протокол активний, усі клієнти взаємодіють виключно за принципом peer-to-peer. Тобто, немає виділеного сервера (а також немає клієнта, який би також виступав у ролі сервера). Під час запуску клієнт надсилає запит на підключення, щоб знайти інших клієнтів, які вже грають. Якщо під час цієї фази запуску не знайдено інших клієнтів, клієнт вирішує, що він сам по собі, і входить у гру самостійно. Інші можуть підключитися будь-коли пізніше. Однак в одній грі одночасно не може бути більше чотирьох гравців. Пакети, які передають поточний стан гри іншим клієнтам, надсилаються всім іншим клієнтам в одноадресному режимі. Теоретично, для цієї мети було б ідеально використовувати багатоадресну розсилку, але таким чином базова мережа не зобов'язана підтримувати будь-який тип багатоадресної розсилки. Оскільки кількість клієнтів для гри peer-to-peer обмежена чотирма, це цілком можливо. Найскладнішою проблемою протоколу peer-to-peer є фаза запуску (ця частина протоколу називається протоколом запуску). Оскільки сервер відсутній, існує багато потенційних проблем із умовами гонки, коли два або більше клієнтів запускаються приблизно одночасно. Ми розробили досить складний протокол

запуску, щоб вирішити всі ці проблеми та досягти передбачуваних і послідовних результатів у всіх випадках.

По суті, в протоколі peer-to-peer клієнти знаходять один одного, транслюючи початковий запит на з'єднання (кілька разів, якщо необхідно) та очікуючи, чи отримають вони відповідь від інших клієнтів, які наразі запущені, і таким чином прослуховуючи такі запити на з'єднання. Якщо після певного часу відповіді не отримано, клієнт вирішує, що він один у своєму світі, і входить без зайвих слів.

Наведена нижче діаграма станів ілюструє протокол запуску peer-to-peer з використанням псевдокоду для опису того, що робиться в кожному стані:

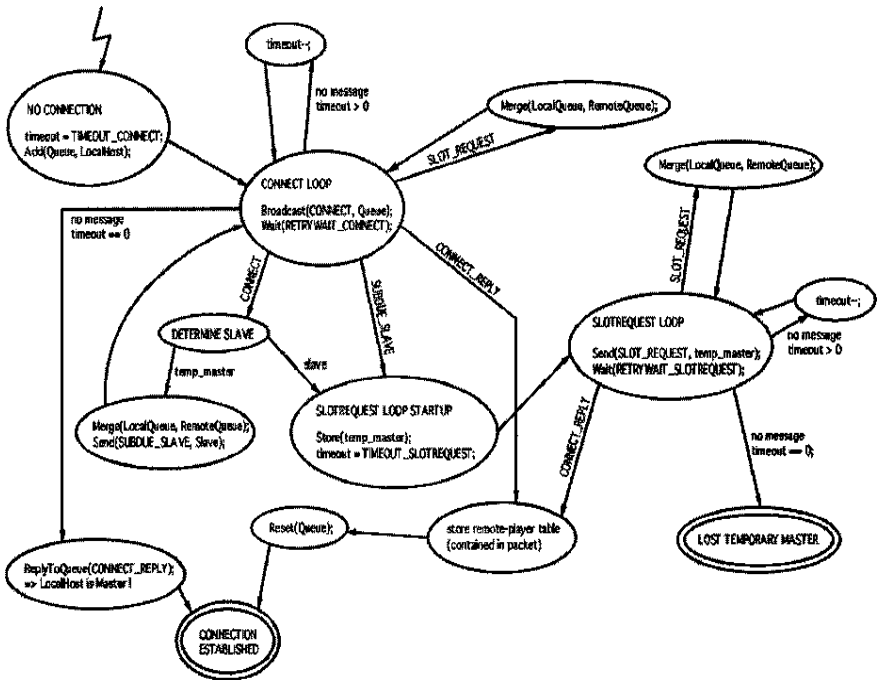

Рисунок 2.1 - Діаграма станів протоколу запуску peer-to-peer

Тут використовуються два фундаментальні підходи. По-перше, конфлікти між кількома клієнтами, що запускаються приблизно одночасно (це означає, що принаймні один клієнт запускається під час фази запуску іншого клієнта), вирішуються шляхом порівняння адрес вузлів клієнтів. Оскільки ці адреси повинні бути унікальними, у разі конфлікту перемагає клієнт з вищою адресою. По-друге, клієнти, що запускаються, завжди ставлять у чергу запити на підключення, отримані від інших клієнтів. Коли клієнт стає підлеглим іншого

клієнта, який щойно розіслав свій запит на підключення, він пересилає свою чергу цьому клієнту. Це необхідно для того, щоб жодні запити на підключення не були втрачені. Навіть якщо запити на підключення надсилаються кілька разів, якщо відповіді не отримано, цього самого недостатньо, щоб гарантувати, що весь запит не буде втрачено під час передачі іншому клієнту без постановки запитів у чергу та їх пересилання за потреби.

Тут є три основні цикли, які є важливими для нас. Два з них показано на діаграмі станів вище та є частиною протоколу запуску. По-перше, це цикл підключення. Відразу після запуску клієнт входить до цього циклу. Вихід з нього відбудеться, якщо цей клієнт або поступиться місцем іншому клієнту з вищим статусом (клієнт, який вже підключений, або клієнт, який також щойно запускається, але з вищою адресою вузла), або якщо мине певний час, протягом якого жоден клієнт з вищим статусом не ідентифікував себе. Якщо це трапляється, клієнт надсилає відповіді на підключення всім клієнтам у черзі з нижчим статусом і входить у ігровий цикл (третій цикл, див. нижче).

По-друге, існує цикл запиту слотів. Клієнт потрапляє в цей цикл, коли він або неявно вирішує для себе, що він є підлеглим (отримано запит на підключення від клієнта з вищою адресою вузла), або коли він отримує сповіщення SUBDUE_SLAVE, таким чином явно стаючи підлеглим. Клієнт, який перебуває в циклі запиту слотів, чекає там, поки не отримає повідомлення від головного сервера про виділення слота для клієнта. Зазвичай гарантується, що це завжди відбуватиметься для клієнтів, які перебувають у цьому циклі; принаймні до тих пір, поки поточний (тимчасовий) головний сервер не вийде з ладу посеред фази запуску, поки інший клієнт перебуває в циклі запиту слотів. Якщо це Однак, якщо це станеться, клієнт перерве час очікування, відобразить повідомлення про помилку, і користувачеві доведеться виконати ще одну команду підключення.

У будь-якому випадку, після фази запуску клієнт увійде в ігровий цикл. (Цей цикл і є власне грою (гра, рендеринг тощо), що стосується мережевого коду.) Існує два способи досягнення цього. Більшість клієнтів входять в ігровий цикл як підлеглі, тобто хтось, хто вже працює, призначив їм слот. Клієнт також

може увійти в ігровий цикл як тимчасовий майстер. Тимчасовий у цьому випадку означає, що фактично немає відносин майстер/підлеглий, як тільки всі клієнти знаходяться в ігровому циклі. Тим не менш, під час запуску обирається тимчасовий майстер, який призначає слот собі та кожному з підлеглих, щоб уникнути виникнення умов гонки. Після того, як усі слоти призначені, всі клієнти рівні. Існує дуже вагома причина не зберігати вже встановлені відносини майстер/підлеглий, коли всі знаходяться в ігровому циклі. Якщо майстер виходить з ігрового циклу, потрібно переобрати іншого майстера. По-перше, це не зовсім тривіально зробити безпечним способом, а по-друге, виникають всілякі проблеми, коли клієнти намагаються запуститися, поки ще не обрано нового майстера.

Клієнти в ігровому циклі реагують на отримані запити на підключення наступним чином. Спочатку вони переглядають список своїх вузлів (включаючи себе) та встановлюють унікальний порядок ранжування, використовуючи адреси вузлів. Потім вони чекають, помноживши свій ранг (починаючи з 0), на певний інтервал очікування. Якщо вони не зможуть перехопити відповідь на підключення іншого клієнта протягом цього періоду, вони самі надішлють відповідь[16]. Такий підхід гарантує, що хтось відповість на запит на підключення, навіть якщо клієнти, які все ще перебувають у списку, вже вийшли, не надіславши сповіщення (тобто, зазнали збою, або їхній хост був вимкнений без виходу). Це також гарантує відсутність конфліктів, оскільки реплікований список клієнтів може бути невідповідним лише щодо клієнтів, які зазнали збою, але не може бути клієнтів, які підключені, але не є частиною списку. Тому жодні два клієнти не намагатимуться відповісти одночасно.

Клієнт / сервер

У моделі «клієнт-сервер» є виділений комп’ютер, який виконує роль сервера. Усі клієнти підключаються до сервера. Весь зв’язок відбувається між клієнтами та сервером, оскільки клієнти ніколи не спілкуються один з одним. Сервер виконує обов’язки «хоста», інформуючи клієнтів про гравців, що приєднуються та виходять. Сервер також автентифікує ходи гравців, перш ніж

надсилати їх іншим клієнтам. Таким чином, схема функціонує наступним чином. «Гравець хоче рухатися вперед. Він надсилає команду «вперед» на сервер. Якщо сервер дозволяє хід, він транслює цей хід усім іншим гравцям. Якщо команда недійсна, сервер надсилає повідомлення клієнту та забороняє хід».

Перевагами клієнт-серверного підходу є покращена безпека та чудова масштабованість. Покращена безпека досягається завдяки тому, що сервер перевіряє всі дії клієнтів. Масштабованість є гарною, оскільки кожен клієнт бачить лише вхідний та вихідний трафік від сервера, який по суті не залежить від загальної кількості клієнтів. Подальшої масштабованості можна досягти за допомогою балансування навантаження – принципу, який вимагає наявності певної кількості серверів, кожен з яких обробляє частину загальної кількості підключених клієнтів.

Недоліками клієнт-серверної моделі є потреба у виділеному обладнанні та знижена відмовостійкість. Вимога у виділеному обладнанні є основним недоліком клієнт-серверного підходу. Серверу потрібна лінія з дуже високою пропускною здатністю, оскільки це полегшує зв'язок з усіма клієнтами. Також знижується відмовостійкість, оскільки якщо головний сервер стане недоступним, гра зависне. У контексті онлайн-ігор сервер – це сутність, яка обчислює та моделює ігрові стани на основі дій гравців, а клієнт – це сутність, яка відображає та представляє ігрові стани користувачеві.

Коли сервер запускає ігрову симуляцію або коли клієнт надає ігровий стан, вони синхронізують події або стани на основі часу їх генерації. Щоб синхронізувати ігровий процес та взаємодію між користувачами, система онлайн-ігор повинна враховувати ці затримки між серверами та клієнтами.

• «Клієнт-серверна архітектура» описує мережеву архітектуру, в якій кожен комп’ютер виконує частину обробки, але позначається як «клієнт» або «сервер» стосовно кожного процесу.

• « Сервери » – це спільні центральні комп’ютери, призначені для керування певними завданнями для кількох клієнтів. Наприклад, «файлові сервери» призначені для зберігання файлів, «сервери друку» – для друку, а

«мережеві сервери» – для маршрутизації мережевого трафіку.

• « Клієнти » – це робочі станції, на яких користувачі запускають програми та додатки. Клієнти покладаються на сервери для отримання ресурсів, таких як файли, пристрої та навіть обчислювальна потужність, але обробляють незалежно від серверів. Клієнт-серверна архітектура може використовуватися в мережі будь-якого розміру; відмінною рисою клієнт-серверної архітектури є те, що всі комп’ютери в мережі беруть участь в обробці, але певні комп’ютери призначені для певних служб або завдань і не виконують жодної іншої роботи.

• Моделі клієнт-серверного програмування [19] Клієнтсько -серверні програми розподіляють функції та процеси між клієнтськими та серверними комп'ютерами. У більшості випадків клієнтсько-серверні програми створюють три функціональні розділи "роботи":

• Зберігання даних -- зберігання даних, що використовуються програмою, зазвичай у великій базі даних.

• Бізнес-логіка – процеси, які виконують «роботу», таку як запит даних, сортування даних, повернення даних, обробка даних у звіти.

• Представлення даних -- відображення та інтерфейс користувача.

На наступній діаграмі ілюструється проблема проектування:

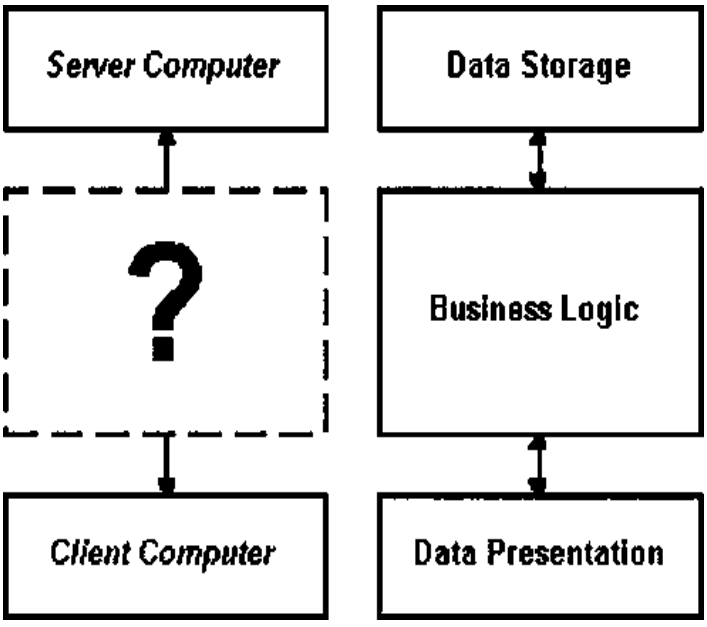

Рисунок 2.2 - Моделі клієнт-серверного програмування

Майже всі клієнт-серверні моделі розміщують сховище даних на серверному комп'ютері, а презентацію даних — на клієнтському. Різниця між клієнт-серверними моделями залежить від того, як бізнес-логіка розділена між клієнтом і сервером.

Модель між рівноправними особами

У моделі peer-to-peer усі клієнти взаємодіють з усіма іншими клієнтами. Таким чином, індивідуальні оновлення стану передаються на всі машини. Також одна машина позначається як «хост». Хост майже як сервер, оскільки він відстежує гравців та інформує інших клієнтів, коли клієнт приєднується до гри або виходить з неї. Однак на цьому подібність закінчується, оскільки хост не служить центральною точкою підключення для всіх клієнтів. Хост є обов'язковою сутністю, тому peer-to-peer ігри майже завжди підтримують міграцію хоста, яка є протоколом, що зазвичай використовується. «Міграція хоста» автоматично передає обов'язки хоста іншому комп'ютеру у випадку, якщо активний хост відключається або стає недоступним. Нижче наведено приклад типової peer-to-peer конфігурації.

Перевагою методу peer-to-peer є його відмовостійкість, простота впровадження та відсутність потреби в спеціальному обладнанні. Оскільки немає центрального сервера, який би порушував роботу, якщо щось трапиться з одним або кількома гравцями, решта не постраждають. Таким чином, ця система дуже відмовостійка. Реалізація також дуже проста, оскільки всі клієнти однакові, і немає потреби в спеціалізованому окремому сервері. Крім того, оскільки сервера немає, немає потреби в спеціальному обладнанні. Недоліками методу peer-to-peer є високий ризик безпеки та проблеми масштабованості. Проблема ризику безпеки виникає тому, що клієнти несуть відповідальність за надсилання інформації про себе всім іншим клієнтам.

Таким чином, немає зовнішнього механізму перевірки, і якщо клієнта зламають, він може надіслати помилкову інформацію решті клієнтів, яку вони сприймуть за чисту монету. Отже, в грі шахрай може розмовляти крізь стіни, отримувати нескінченне здоров'я тощо. Однак найбільшими недоліками

однорангових мереж є проблеми масштабованості.

• «Архітектура однорангової мережі» описує тип мережі, в якій кілька робочих станцій з'єднані між собою, але жоден комп'ютер не призначений як «сервер». Натомість кожна робоча станція може бути (і зазвичай є) одночасно «клієнтом» і «сервером», залежно від завдання (наприклад, робоча станція А може бути підключена до принтера та виконувати роль «серверу друку» для інших робочих станцій, тоді як робоча станція В може бути підключена до модему та виконувати роль «сервера зв'язку» для інших). Однорангові мережі зазвичай невеликі (менше 25 робочих станцій) і не забезпечують стабільної продуктивності у великих інсталяціях або за умов високого навантаження мережевого трафіку.

Порівняння цих моделей

Обидва підходи є дійсними, але в різних масштабах. Якщо ігровий світ складатиметься лише з кількох гравців, то модель peer-to-peer є прийнятним методом синхронізації ігрового світу. Наприклад, [18]Age of Empires від Microsoft використовує схему peer-to-peer мережі, але обмежує кількість активних гравців чотирма. Однак, якщо є ймовірність одночасної присутності в ігровому світі помірної або великої кількості гравців, то слід використовувати клієнт-серверну модель. Слід також зазначити, що peer-to-peer модель набагато менш безпечна, тому, якщо безпека є фактором, то модель слід вибирати відповідно. Чудовим прикладом клієнт-серверного підходу є Quake III від ID Software, який являє собою динамічний 3D-шутер, що дозволяє брати участь в одній грі до сорока клієнтів.

2.2 Функція програмування мережевих ігор для багатьох гравців

Мережеві ігри для багатьох гравців пропонують широкий спектр технологічних викликів. Для деяких із наведених нижче пунктів вже існують задовільні рішення, інші наразі є предметом дослідження, а інші знову ж таки є ймовірними кандидатами для майбутньої діяльності. Для всіх цих проблем слід

пам'ятати, що вирішення однієї проблеми не повинно ускладнювати інші, спричиняючи значні затримки, додаткову обробку чи подібні накладні витрати.

Синхронізація

Узгодження змінної часу є обов'язковим для отримання стану спільного доступу [24]. У випадках, коли протокол мережевого часу (NTP) не застосовується через брандмауери або використання інших мереж, окрім Інтернету, ігри слід розглядати як розроблені з процедурою синхронізації, яка забезпечує точність порядку системного годинника.

Масштабованість для багатьох гравців

Скільки гравців може підтримувати гра? Для гри між гравцями (peer-to-peer) вимоги до зв'язку масштабуються пропорційно квадрату кількості гравців. У випадку з сервером, вони масштабуються лінійно. В обох випадках існує верхня межа кількості гравців, які можуть підтримуватися. Як можна розширити цю межу або, в ідеалі, повністю уникнути її за допомогою розподілених обчислень?

Міцність

Оскільки багато гравців підключаються до гри за допомогою різних типів з'єднання та обладнання, неминуче, що деякі клієнтські з'єднання матимуть неякісну продуктивність через високу затримку, низьку пропускну здатність або високий рівень втрати пакетів. У такому випадку важливо, щоб якомога менше інших гравців постраждали від цього, і якщо лише одне клієнтське з'єднання буде поганим, це призведе до серйозних проблем. Ми сподіваємося зменшити проблему, щоб постраждали лише гравці, які безпосередньо взаємодіють з неякісним з'єднанням.

Міцність

Оскільки багато гравців підключаються до гри за допомогою різних типів з'єднання та обладнання, неминуче, що деякі клієнтські з'єднання матимуть неякісну продуктивність через високу затримку, низьку пропускну здатність або високий рівень втрати пакетів[13]. У такому випадку важливо, щоб якомога менше інших гравців постраждали від цього, і якщо лише одне клієнтське

з'єднання буде поганим, це призведе до серйозних проблем. Ми сподіваємося зменшити проблему, щоб постраждали лише гравці, які безпосередньо взаємодіють з неякісним з'єднанням.

Мертвий відлік

Облік на місці займається тим, як передбачити поведінку в ситуаціях, коли фактичні оновлення інформації не надійшли через затримку мережі або обмеження пропускної здатності [6]. Облік на місці є важливим для того, щоб забезпечити клієнтам відчуття негайної реакції на їхні дії.

Масштабованість об'єктів

Світ охоплює величезний діапазон масштабів довжини. Від величезних планетарних систем до поверхні планети, до ландшафтних елементів, міст, будинків, аж до деталей меблів і навіть далі. Якби вся система була такою ж деталізованою... як найкраща роздільна здатність, вона включатиме величезні обсяги даних. Динамічна генерація ландшафту за допомогою фрактальної процедури з відомим початковим значенням є однією частковою відповіддю на це питання, можливо, є й інші.

Управління пропускною здатністю

Пропускна здатність є обмежувальним фактором у багатьох відношеннях. Через неоднорідну природу Інтернету не можна очікувати, [14]що клієнти відповідатимуть будь-яким гарантованим вимогам щодо пропускної здатності. Якщо гра може динамічно адаптуватися до доступної пропускної здатності, гравці можуть отримати кращий ігровий досвід, не виключаючи тих, у кого поганий зв'язок.

Управління пропускною здатністю

Пропускна здатність є обмежувальним фактором у багатьох відношеннях. Через неоднорідну природу Інтернету не можна очікувати [17], що клієнти відповідатимуть будь-яким гарантованим вимогам щодо пропускної здатності. Якщо гра може динамічно адаптуватися до доступної пропускної здатності, гравці можуть отримати кращий ігровий досвід, не виключаючи тих, у кого поганий зв'язок.

Наполегливість

Ніхто ніколи не згадував про Всесвітню павутину. Це було б неможливо, але через те, як створена павутина, це не є необхідним. При створенні справді масштабного багатокористувацького світу, де люди можуть виходити з неї та виходити, коли їм заманеться, потрібна така ж гнучкість [25]. Код потрібно завантажувати під час роботи гри, щоб мати змогу впроваджувати нові функції, не вимагаючи від усіх гравців виходу з системи та перезавантаження системи.

Швидкий доступ до бази даних. Для величезних світів база даних може бути таким же вузьким місцем, як і пропускна здатність мережі. Тому швидкий, безпечний та гнучкий доступ до бази даних є першочерговим завданням при розробці великих ігрових світів. Надійні, але швидкі протоколи. В Інтернеті найпоширенішими транспортними протоколами є UDP для швидкого та ненадійного зв'язку та TCP для надійного зв'язку практично без верхньої межі затримки. Це означає, що вам доведеться створити власний протокол, якщо вам потрібні всі повідомлення, але ви не можете дозволити собі чекати на них. Протокол Direct Play від Microsoft спробував вирішити деякі з цих проблем.

2.3 Висновки до розділу

Моделі клієнт-сервер і peer-to-peer мають свої переваги та недоліки: peer-to-peer підходить для невеликих ігор з кількома гравцями завдяки простоті та відмовостійкості, тоді як клієнт-сервер забезпечує кращу масштабованість і безпеку, що робить його оптимальним для великих онлайн-ігор.

Програмування мережевих ігор для багатьох гравців передбачає вирішення складного комплексу технічних задач - від синхронізації та масштабованості до забезпечення стійкості до затримок і адаптації до пропускної здатності. Для створення динамічного, масштабного й стабільного ігрового середовища потрібні гнучкі підходи, власні протоколи й ефективна робота з базами даних.

РОЗДІЛ 3. МЕРЕЖЕВЕ ПРОГРАМУВАННЯ WINDOWS SOCKETS

3.1 Концепції Windows Sockets

Базовим структурним блоком для комунікації є сокет. Сокет – це кінцева точка комунікації, до якої може бути прив'язане ім'я. Кожен сокет, що використовується, має тип і пов'язаний з ним процес. Сокети існують в межах доменів комунікації. Домен комунікації – це абстракція, введена для об'єднання загальних властивостей потоків, що взаємодіють через сокети. Сокети зазвичай обмінюються даними лише з сокетами в одному домені (можливо перетинати межі доменів, але лише за умови виконання певного процесу трансляції). Засоби Windows Sockets підтримують один домен комунікації: домен Інтернету, який використовується процесами, що взаємодіють за допомогою пакета протоколів Інтернету (IPS). (Майбутні версії цієї специфікації можуть містити додаткові домени.)

Сокети типізуються відповідно до властивостей зв'язку, видимих користувачеві. Вважається, що програми взаємодіють лише між сокетами одного типу, хоча ніщо не перешкоджає зв'язку між сокетами різних типів, якщо базові протоколи зв'язку це підтримують. Наразі користувачеві доступні два типи сокетів. Потоковий сокет забезпечує двонаправлений, надійний, послідовний та недубльований потік даних без меж записів.

Сокет дейтаграми підтримує двонаправлений потік даних, який не гарантується послідовним , надійним або недубльованим. Тобто, процес, який отримує повідомлення через сокет дейтаграми, може виявити, що повідомлення дублюються, і, можливо, в порядку, відмінному від порядку, в якому вони були надіслані. Важливою характеристикою сокета дейтаграми є збереження меж записів у даних. Сокети дейтаграми точно моделюють можливості, що зустрічаються в багатьох сучасних мережах з комутацією пакетів, таких як Ethernet.

Що таке сокет-застосунок? Сокетний інтерфейс вперше з'явився в

Berkeley UNIX (BSD) у вісімдесятих роках. Він був розроблений як механізм мережевого міжпроцесного зв'язку (IPC) для вбудованого TCP/IP. Сокет визначає двонаправлену кінцеву точку для комбінації між процесами.

Сокет має три основні компоненти:

- Інтерфейс, до якого він прив'язаний
- Номер порту, на який він надсилатиме або отримуватиме дані
- Тип сокета - потік або дейтаграма

У протоколі TCP/IP інтерфейс – це IP-адреса хоста. Номер порту – це адреса програмного процесу. У протоколі IPX/SPX інтерфейс – це комбінація ідентифікатора мережі IPX та MAC-адреси мережевого інтерфейсу. Номер порту – це адреса програмного процесу (номер сокета IPX). Серверна програма прослуховує добре відомий порт через усі встановлені мережеві інтерфейси. Клієнт зазвичай ініціює зв'язок з певного інтерфейсу з будь-якого доступного порту.

Що таке Windows Socket

Сокет — це кінцева точка зв'язку — об'єкт, через який програма Windows Sockets надсилає або отримує пакети даних через мережу. Сокет має тип і пов'язаний із запущеним процесом, а також може мати назву. Наразі сокети зазвичай обмінюються даними лише з іншими сокетами в тому ж «комунікаційному домені», який використовує пакет протоколів Інтернету (Internet Protocol Suite). Обидва типи сокетів є двонаправленими; це потоки даних, які можуть передаватися в обох напрямках одночасно (повний дуплекс).

Доступні два типи розеток:

- Потокові сокети

Потокові сокети забезпечують потік даних без меж записів: потік байтів. Гарантовано, що потоки будуть доставлені, правильно упорядковані та не будуть дублюватися.

- Сокети дейтаграм

Сокети дейтаграм підтримують потік даних, орієнтований на записи, доставка якого не гарантується, і який може не бути відсортований за

послідовністю відправлення або не дублюватися.

Windows Sockets 2 (Winsock) дозволяє програмістам створювати розширені інтернет-, інтрамережеві та інші мережеві програми для передачі даних програм через дротовий зв'язок, незалежно від використовуваного мережевого протоколу. Завдяки Winsock програмісти отримують доступ до розширених мережевих можливостей Microsoft® Windows®, таких як багатоадресна розсилка та якість обслуговування (QOS).

Winsock дотримується моделі архітектури відкритої системи Windows (WOSA); він визначає стандартний інтерфейс постачальника послуг (SPI) між інтерфейсом прикладного програмування (API) з його експортованими функціями та стеками протоколів. Він використовує парадигму сокетів, яка вперше була популяризована Berkeley Software Distribution (BSD) UNIX. Пізніше вона була адаптована для Windows у Windows Sockets 1.1, з якою програми Windows Sockets 2 мають зворотну сумісність. Програмування Winsock раніше було зосереджено навколо TCP/IP. Деякі методи програмування, які працювали з TCP/IP, не працюють з кожним протоколом. Як результат, API Windows Sockets 2 додає функції, де це необхідно, для обробки кількох протоколів.

Що таке Windows Sockets Специфікація Windows Sockets визначає мережевий програмний інтерфейс для Microsoft Windows, який базується на парадигмі "сокетів", популяризованій у Berkeley Software Distribution (BSD) Каліфорнійського університету в Берклі. Вона охоплює як знайомі процедури в стилі сокетів Берклі, так і набір специфічних для Windows розширень, розроблених для того, щоб програміст міг скористатися перевагами керованої повідомленнями природи Windows. Специфікація Windows Sockets призначена для забезпечення єдиного API, за яким розробники додатків можуть програмувати, і який можуть використовувати різні постачальники мережевого програмного забезпечення. Крім того, в контексті певної версії Microsoft Windows вона визначає двійковий інтерфейс (ABI) таким чином, що додаток, написаний за допомогою Windows Sockets API, може працювати з сумісною реалізацією протоколу від будь-якого постачальника мережевого програмного

забезпечення. Таким чином, ця специфікація визначає виклики бібліотек та пов'язану з ними семантику, за якою розробник додатків може програмувати та яку може реалізувати постачальник мережевого програмного забезпечення.

Мережеве програмне забезпечення, яке відповідає цій специфікації Windows Sockets, вважатиметься «сумісним з Windows Sockets». Постачальники інтерфейсів, які є «сумісними з Windows Sockets», називатимуться «постачальниками Windows Sockets». Щоб бути сумісним з Windows Sockets, постачальник повинен реалізувати 100% цієї специфікації Windows Sockets.

Програми, які здатні працювати з будь-якою реалізацією протоколу, сумісного з Windows Sockets, вважатимуться такими, що мають інтерфейс Windows Sockets, і називатимуться «Програмами Windows Sockets». Ця версія специфікації Windows Sockets визначає та документує використання API разом із пакетом інтернет-протоколів (IPS, зазвичай званим TCP/IP). Зокрема, всі реалізації Windows Sockets підтримують як потокові (TCP), так і дейтаграмні (UDP) сокети. Хоча використання цього API з альтернативними стеками протоколів не заборонено (і очікується, що воно буде предметом майбутніх переглядів специфікації), таке використання виходить за рамки цієї версії специфікації.

Тип даних Socket

Кожен об'єкт сокета MFC інкапсулює дескриптор об'єкта Windows Sockets. Тип даних цього дескриптора - SOCKET. Дескриптор SOCKET аналогічний HWND для вікна. Класи сокетів MFC забезпечують операції над інкапсульованим дескриптором. Тип даних SOCKET детально описано в Win32 SDK. Див. розділ "Тип даних Socket та значення помилок" у розділі "Сокети Windows".

Використання Socket

Сокети дуже корисні щонайменше у трьох комунікаційних контекстах:

- Моделі клієнт/сервер
- Сценарії однорангової взаємодії, такі як програми чату
- Здійснення викликів віддалених процедур (RPC) шляхом

інтерпретації повідомлення застосунком-отримувачем як виклику функції

3.2 Моделі програмування сокетів

Дві моделі програмування MFC Windows Sockets підтримуються такими класами: MFC надає два класи для підтримки програмування мережевих застосунків за допомогою Windows Sockets API. Клас CAsyncSocket інкапсулює Windows Sockets API один до одного, надаючи досвідченим мережевим програмістам максимальну потужність та гнучкість. Клас CSocket надає спрощений інтерфейс для серіалізації даних до об'єкта CArchive та з нього.

Класи сокетів Windows

• CAsyncSocket Цей клас інкапсулює API Windows Sockets. CAsyncSocket призначений для програмістів, які знаються на мережевому програмуванні та бажають гнучкості програмування безпосередньо в API сокетів, а також зручності функцій зворотного виклику для сповіщення про мережеві події. Окрім пакування сокетів в об'єктно-орієнтованій формі для використання в C++, єдиною додатковою абстракцією, яку надає цей клас, є перетворення певних повідомлень Windows, пов'язаних із сокетами, у зворотні виклики.

• CSocket (див. рис. 3.1) Цей клас, похідний від CAsyncSocket, надає абстракцію вищого рівня для роботи з сокетами через об'єкт MFC CArchive. Використання сокета з архівом дуже нагадує використання протоколу серіалізації файлів MFC. Це спрощує його використання порівняно з моделлю CAsyncSocket . CSocket успадковує багато функцій-членів від CAsyncSocket , які інкапсулюють API Windows Sockets; вам доведеться використовувати деякі з цих функцій та розуміти програмування сокетів загалом. Але CSocket керує багатьма аспектами зв'язку, які вам довелося б виконувати самостійно, використовуючи або необроблений API, або клас CAsyncSocket. Найголовніше, що CSocket забезпечує блокування (з фоновою обробкою повідомлень Windows), що є важливим для синхронної роботи CArchive.

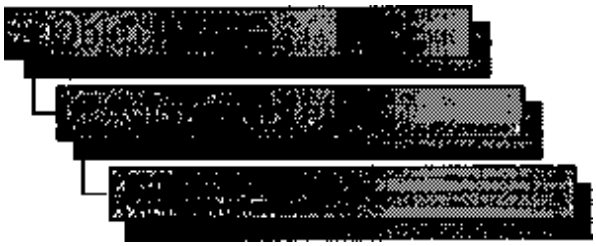

Рисунок 3.1 - Класи сокетів Windows

Клас CSocket походить від CAsyncSocket та успадковує його інкапсуляцію Windows Sockets API. Об'єкт CSocket представляє вищий рівень абстракції Windows Sockets API, ніж об'єкт CAsyncSocket . CSocket працює з класами CSocketFile та CArchive для керування надсиланням та отриманням даних.

Об'єкт CSocket також забезпечує блокування, що є важливим для синхронної роботи CArchive. Блокувальні функції, такі як Receive, Send, ReceiveFrom, SendTo та Accept (усі успадковані від CAsyncSocket), не повертають помилку WSAEWOULDBLOCK у CSocket. Натомість ці функції чекають завершення операції. Крім того, початковий виклик завершиться з помилкою WSAEINTR , якщо CancelBlockingCall викликається, поки одна з цих функцій блокує. Щоб використовувати об'єкт CSocket , викличте конструктор, а потім викличте Create для створення базового дескриптора SOCKET (тип SOCKET). Параметри Create за замовчуванням створюють потоковий сокет, але якщо ви не використовуєте сокет з об'єктом CArchive , ви можете вказати параметр для створення сокета дейтаграми або прив'язати його до певного порту для створення сокета сервера. Підключіться до клієнтського сокета за допомогою Connect на стороні клієнта та Accept на стороні сервера. Потім створіть об'єкт CSocketFile та пов'яжіть його з об'єктом CSocket у конструкторі CSocketFile . Далі створіть об'єкт CArchive для надсилання та один для отримання даних (за потреби), а потім пов'яжіть їх з об'єктом CSocketFile у конструкторі CArchive . Після завершення зв'язку знищіть об'єкти CArchive, CSocketFile та CSocket . Тип даних SOCKET описано у статті "Сокети Windows: передісторія в посібнику програміста Visual C++".

Модель програмування CSocket

Використання об'єкта CSocket передбачає створення та асоціювання кількох об'єктів класу MFC. У загальній процедурі, наведеній нижче, кожен крок виконується як сокетом сервера, так і сокетом клієнта, за винятком кроку 3, на якому кожен тип сокета вимагає різних дій. Під час виконання серверна програма зазвичай запускається першою, щоб бути готовою та «слухати», коли клієнтська програма шукає з’єднання. Якщо сервер не готовий, коли клієнт намагається підключитися, зазвичай потрібно, щоб користувацька програма спробувала підключитися пізніше.

Щоб налаштувати зв'язок між сокетом сервера та сокетом клієнта Створіть об'єкт CSocket. Використайте об'єкт для створення базового дескриптора SOCKET . Для клієнтського об'єкта CSocket зазвичай слід використовувати параметри за замовчуванням для Create , якщо вам не потрібен сокет дейтаграми. Для серверного об'єкта CSocket необхідно вказати порт у виклику Create . Якщо сокет є клієнтом, викличте CAsyncSocket::Connect, щоб підключити об'єкт сокета до сокета сервера. -або- Якщо сокет є сервером, викличте CAsyncSocket::Listen, щоб розпочати прослуховування спроб підключення від клієнта. Після отримання запиту на підключення прийміть його, викликавши CAsyncSocket::Accept. Створіть об'єкт CSocketFile, пов'язавши з ним об'єкт CSocket . Створіть об'єкт CArchive для завантаження (отримання) або зберігання (надсилання) даних. Архів пов'язаний з об'єктом CSocketFile . Майте на увазі, що CArchive не працює з сокетами дейтаграм. Використовуйте об'єкт CArchive для передачі даних між сокетами клієнта та сервера. Майте на увазі, що даний об'єкт CArchive переміщує дані лише в одному напрямку: або для завантаження (отримання), або для зберігання (надсилання). У деяких випадках ви використовуватимете два об'єкти CArchive : один для надсилання даних, інший для отримання підтверджень. Після прийняття з’єднання та налаштування архіву можна виконувати такі завдання, як перевірка паролів. Знищити архів, файл сокета та об'єкти сокета.

3.3 Використання сокетів з архівами

Клас CSocket забезпечує підтримку сокетів на вищому рівні абстракції, ніж клас CAsyncSocket. CSocket використовує версію протоколу серіалізації MFC для передачі даних до об'єкта сокета та з нього через об'єкт MFC CArchive. CSocket забезпечує блокування (керуючи фоновою обробкою повідомлень Windows) та надає доступ до CArchive, який керує багатьма аспектами зв'язку, які вам довелося б виконувати самостійно, використовуючи або необроблений API, або клас CAsyncSocket. Якщо ви пишете клієнтську програму MFC для зв'язку з усталеними (не MFC) серверами, не надсилайте об'єкти C++ через архів. Якщо сервер не є MFC- застосунком, який розуміє типи об'єктів, які ви хочете надсилати, він не зможе отримувати та десеріалізувати ваші об'єкти.

Як працюють сокети з архівами

У цьому розділі пояснюється, як об'єкти CSocket , CSocketFile та CArchive поєднуються для спрощення надсилання та отримання даних через сокет Windows. У статті «Сокети Windows: приклад сокетів з використанням архівів» представлено функцію PacketSerialize. Архівний об'єкт у прикладі PacketSerialize працює подібно до архівного об'єкта, що передається до функції MFC Serialize. Істотна відмінність полягає в тому, що для сокетів архів приєднується не до стандартного об'єкта CFile (зазвичай пов'язаного з файлом на диску), а до об'єкта CSocketFile . Замість підключення до файлу на диску, об'єкт CSocketFile підключається до об'єкта CSocket .

Об'єкт CArchive керує буфером. Коли буфер архіву, що зберігається (надсилається), заповнюється, пов'язаний з ним об'єкт CFile записує вміст буфера. Очищення буфера архіву, підключеного до сокета, еквівалентне надсиланню повідомлення. Коли буфер архіву, що завантажується (отримується), заповнюється, об'єкт CFile припиняє читання, доки буфер знову не стане доступним.

Клас CSocketFile походить від CFile, але він не підтримує функції-члени CFile, такі як функції позиціонування (Seek, GetLength, SetLength тощо), функції

блокування (LockRange, UnlockRange) або функцію GetPosition . Все, що повинен робити об'єкт CSocketFile, це записувати або зчитувати послідовності байтів до або з пов'язаного об'єкта CSocket . Оскільки файл не задіяний, такі операції, як Seek та GetPosition, не мають сенсу. CSocketFile походить від CFile, тому зазвичай він успадковує всі ці функції-члени. Щоб запобігти цьому, непідтримувані функції-члени CFile перевизначаються в CSocketFile , щоб викликати виняток CNotSupportedException. Об'єкт CSocketFile викликає функції-члени свого об'єкта CSocket для надсилання або отримання даних. На наступному рисунку показано зв'язки між цими об'єктами з обох сторін зв'язку. До моменту створення об'єкта CSocketFile наступна послідовність є точною (з кількома відмінностями в параметрах) як для CAsyncSocket , так і для CSocket. З цього моменту послідовність застосовується виключно до CSocket. У наступному рисунку показано послідовність операцій для налаштування зв'язку між клієнтом і сервером (див. рис. 3.2)

Server	Client
// construct a socket CSocket sockSrvr;	// construct a socket CSocket sockClient;
// create the SOCKET sockSrvr.Create(nPort); [1,2]	// create the SOCKET sockClient.Create( ); [2]

// start listening
sockSrvr.Listen( );	

	// seek a connection sockClient.Connect(strAddr, nPort); [3,4]

// construct a new, empty socket
CSocket sockRecv;
// accept connection
sockSrvr.Accept(sockRecv); [5]	

// construct file object CSocketFile file(&sockRecv);	// construct file object CSocketFile file(&sockClient);
// construct an archive	// construct an archive

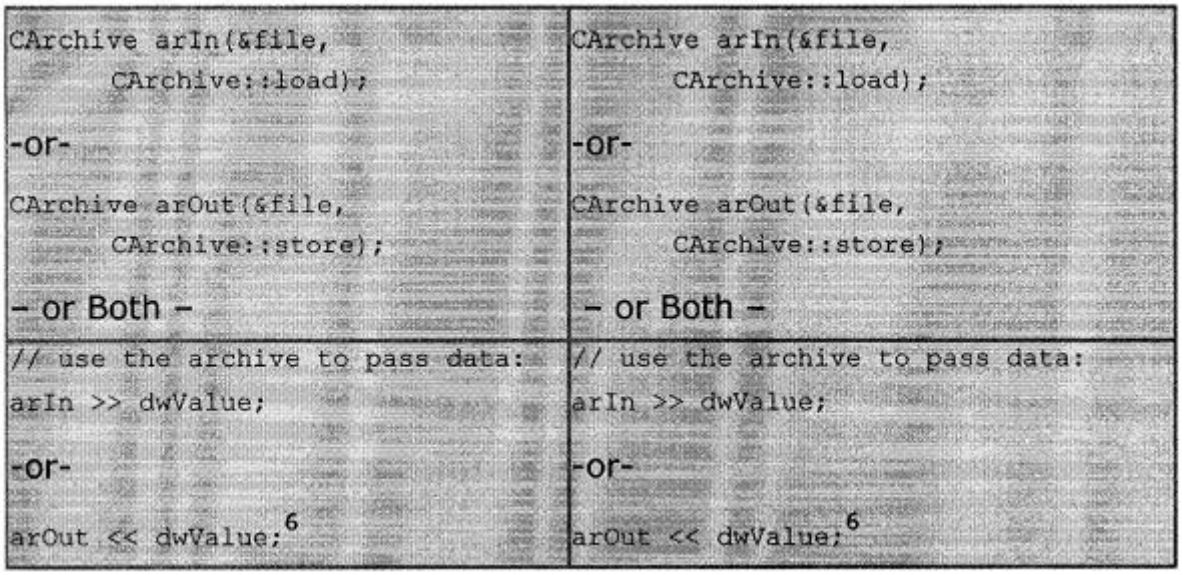

Рисунок 3.2 - Налаштування зв'язку між сервером і клієнтом

1. Де nPort – це номер порту.

2. Сервер завжди повинен вказувати порт, щоб клієнти могли підключатися. Виклик Create іноді також вказує адресу. На стороні клієнта використовуйте параметри за замовчуванням, які запитують у MFC використання будь-якого доступного порту.

3. Де nPort – це номер порту, а strAddr – це адреса машини або IP-адреса.

4. Адреси машин можуть мати кілька форм: " ftp.microsoft.com ", "microsoft.com". IP-адреси використовують "число з крапкою" формату "127.54.67.32". Функція Connect перевіряє, чи є адреса числом з крапкою (хоча вона не перевіряє, чи є номер дійсним комп'ютером у мережі). Якщо ні, Connect приймає ім'я комп'ютера в одній з інших форм.

5. Коли ви викликаєте Accept на стороні сервера, ви передаєте посилання на новий об'єкт socket. Спочатку потрібно створити цей об'єкт, але не викликати для нього Create . Майте на увазі, що якщо цей об'єкт сокета виходить за межі області видимості, з'єднання закривається. MFC підключає новий об'єкт до дескриптора SOCKET . Ви можете створити сокет на стеку, як показано, або на купі.

6. Архів та файл сокета закриваються, коли вони виходять за межі

області видимості. Деструктор об'єкта сокета також викликає функцію-член Close для об'єкта сокета, коли об'єкт виходить за межі області видимості або видаляється.

Додаткові примітки щодо послідовності. Послідовність викликів, показана в попередній таблиці, стосується потокового сокета. Сокети дейтаграм, які не потребують встановлення з'єднання, не потребують викликів CAsyncSocket::Connect, Listen та Accept (хоча за потреби можна використовувати Connect). Натомість, якщо ви використовуєте клас CAsyncSocket, сокети дейтаграм використовують функції-члени CAsyncSocket::SendTo та ReceiveFrom . (Якщо ви використовуєте Connect із сокетом дейтаграм, ви використовуєте Send та Receive.) Оскільки CArchive не працює з дейтаграмами, не використовуйте CSocket з архівом, якщо сокет є дейтаграмою.

CSocketFile не підтримує всі функції CFile ; такі члени CFile , як Seek, які не мають сенсу для зв'язку через сокет, недоступні. Через це деякі стандартні функції MFC Serialize несумісні з CSocketFile. Це особливо стосується класу CEditView. Не слід намагатися серіалізувати дані CEditView через об'єкт CArchive, приєднаний до об'єкта CSocketFile, використовуючи CEditView::SerializeRaw; використовуйте CEditView:Serialize. замість цього (не задокументовано). Функція SerializeRaw очікує, що об'єкт файлу матиме функції, такі як Seek, які CSocketFile не підтримує.

3.4 Висновок до розділу

Сокети в Windows Sockets є універсальним інтерфейсом для мережевої взаємодії, що підтримує надійну двосторонню комунікацію між додатками та дозволяє створювати масштабовані клієнт-серверні та однорангові мережеві рішення.

Моделі програмування сокетів у MFC, представлені класами

CAsyncSocket і CSocket, забезпечують гнучкий вибір між низькорівневим контролем і високорівневою зручністю для розробки мережевих застосунків у Windows.

Використання сокетів з архівами у MFC через класи CSocket, CSocketFile та CArchive забезпечує зручну і гнучку модель обміну даними у потоковому з’єднанні, але не підходить для роботи з дейтаграмами чи несумісними типами об'єктів.

РОЗДІЛ 4 . ПРОЕКТУВАННЯ СИСТЕМИ VPNG

4.1 Основні параметри та функціональність системи VPNG

Вхідні дані для цієї системи:

1. Сума грошей, яку гравці хочуть поставити,
2. Ставки або паси на хід гравцями

Вихідні дані системи:

1. Баланс гравців.
2. Статус карт усіх гравців, окрім першого ходу.
3. В останній хід усі карти з'являються перед усіма гравцями.

Система VPNG надає такі функції:

- Відображати історичну інформацію для кожного гравця
- Відображати ігрові записи для кожного гравця
- Відображення кількох ігрових груп
- Обчисліть результат для кожного ходу та відобразіть результати.

Характеристики системи

Через загальну кількість карт 52 та 5 карт для кожного гравця, кількість гравців у грі обмежена максимум 10. Один гравець може грати в цю гру самостійно, не роблячи ставок, і вигравати. Щонайменше два гравці можуть грати зі ставками та вигравати.

Таблиця 4.1

Символи системи VPNG

Персонажі	**Мінімум**	**Максимум**	**Примітка**
Кількість гравців	1	10	
№ Ігрові групи	1	Безліміт	Насправді, це залежить від системних обмежень
Сума грошей для ставок	0,00 дол. США	1000,00 доларів США	

Кількість ігрових груп означає, скільки груп гравців може працювати на цьому сервері. Отже, кількість гравців на цьому сервері дорівнює сумі (кількість гравців у кожній групі). Наприклад, є 5 груп, і в кожній групі є різні гравці, як зазначено нижче.

Таблиця 4.2

Приклад загальної кількості гравців на одному сервері

Група	Кількість гравців
0	5
1	3
2	4
3	6
4	10

4.2 Архітектура та структура класів системи VPNG

Система VPNG базується на моделі клієнт/сервер. Система VPNG поділена на дві підсистеми: клієнтську підсистему та серверну підсистему.

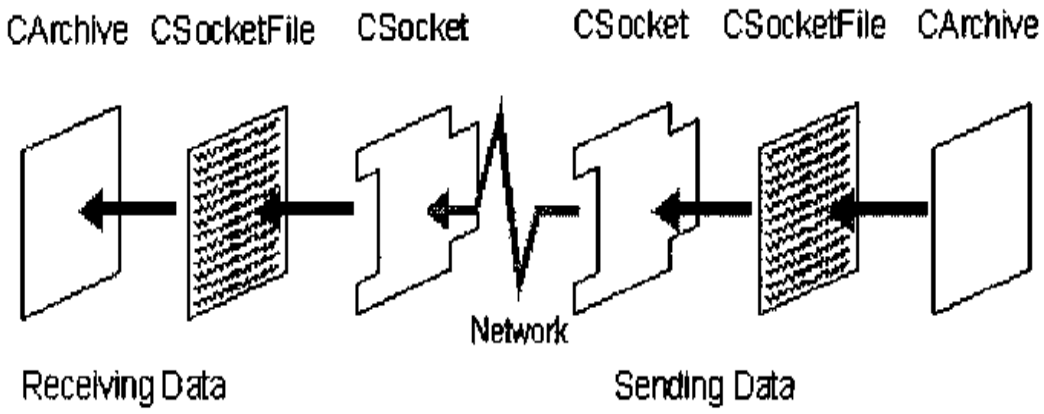

Рисунок 4.1 - Діаграма архітектури VPNG

Модель структури

Клас CDocument забезпечує базову функціональність для класів документів, визначених користувачем. Документ являє собою одиницю даних, яку користувач зазвичай відкриває за допомогою команди «Відкрити файл» та зберігає за допомогою команди «Зберегти файл». Користувачі взаємодіють з документом через пов'язані з ним об'єкти CView. Представлення відображає зображення документа у вікні-фреймі та інтерпретує введені користувачем дані як операції над документом. З документом може бути пов'язано кілька представлень. Коли користувач відкриває вікно документа, фреймворк створює представлення та приєднує його до документа. Шаблон документа визначає, який тип представлення та вікна-фрейма використовуються для відображення кожного типу документа.

Об'єкт CEditView - це тип класу перегляду, який забезпечує функціональність елемента керування редагуванням Windows і може бути використаний для реалізації простих функцій текстового редактора. Клас CEditView надає такі додаткові функції: друк, пошук та заміну. Об'єкт CString складається з послідовності символів змінної довжини. CString надає функції та оператори, використовуючи синтаксис, подібний до Basic. Оператори конкатенації та порівняння, а також спрощене керування пам'яттю.

Клас CStringList підтримує списки об'єктів CString. Усі порівняння виконуються за значенням, тобто порівнюються символи в рядку, а не адреси рядків.

Діаграма класів підсистеми сервера

Клас CServerDoc походить від Cdocument. Діаграма класів, показана на рисунку 4.2, ілюструє класи та їх зв'язки в контексті VPNG.

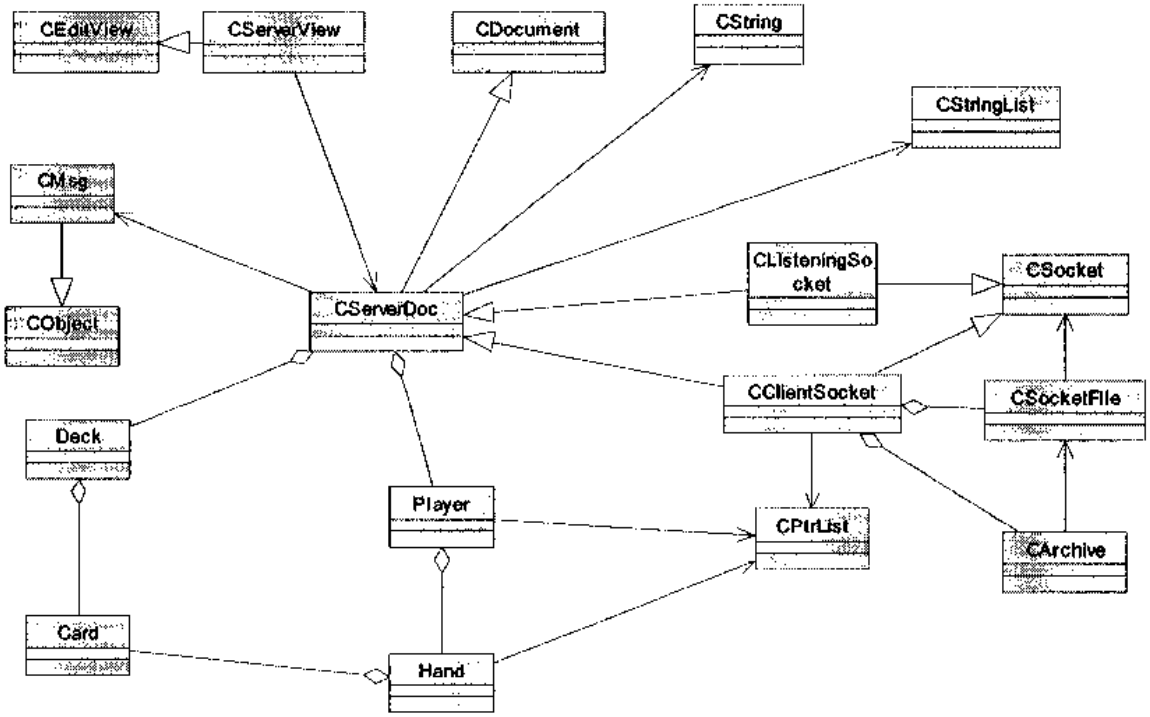

Рисунок 4.2 - Діаграма класів підсистеми сервера

Діаграма класів клієнтської підсистеми (див рис. 4.3)

Метою реалізованої ієрархії класів є створення принаймні одного унікального класу для кожного клієнта. Це забезпечує високий рівень налаштування.

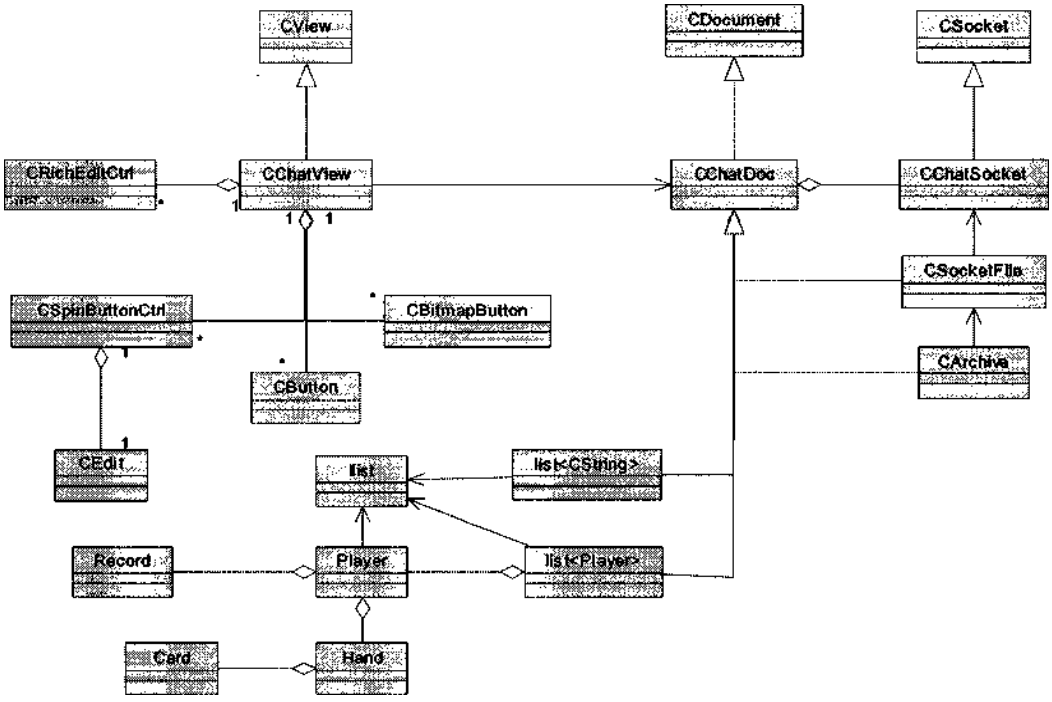

Рисунок 4.3 - Діаграма класів клієнтської підсистеми

4.4 Розробка клієнт-серверної архітектури відеопокеру: проблеми та перспективи

Card(){}
Card(const CardFace &f, const CardSuit &s);
~Card();
CardFace GetFace() const;
CardSuit GetSuitO const; void SetFace(const
CardFace &f); void SetSuit(const CardSuit &s);
bool isNextFaceOf(const Card &c);
//bool isNextS C String getStringOutput();
private:
CardFace Face;
CardSuit Suit;
};
bool operator==(const Card &cl, const Card
&c2); bool operator<(const Card &cl, const
Card &c2);

class deck
#include "card.h" class Deck
{
public:
Deck(); void Shuffle(); void Sort(); void Arrange();
Card GetCard(int i)const;
void SetCard(int i, const Card &c);
private:
Card Cards[52];
class hand
#include "card.h"

```
enum Rank { nothing, Onepair, TwoPair, ThreeOfAKind,
Straight, Flush, FullHouse, FouIOfAkind, StraightFlush};
class Hand{
public:
Hand(){}
Card GetCard(int i) const {return Cards [i-1];} void SetCard(int i, const Card &c){Cards[i-l]=c;} void Sort();
void SwapCards(int i, int j);
Rank QualityO const;
CardFace getOnePairFace() const;
CardFace gettwoPairFaceO const;
CardFace getThreeofAKindFace() const; CardFace getFullHouseFace() const;
CardFace getFourOfAkindFace() const;
bool HasOnePair() const;
bool HasTwoPair() const;
bool HasThreeofAKind() const;
bool HasStraight() const;
bool HasFlush() const;
bool HasFulMouse() const;
bool HasF ourOfAKind() const; bool HasStraightFlush() const;
private:
Card Cards[5];
};
//auxiliary operators
bool operator==(const Hand &a, const Hand &b); bool operator<(const Hand &a, const Hand &b); ostream& operator«(ostream &sout, const Hand &b); CString outputRankStrName(Rank r);
class playingRecord
{
public:
```

```
CString winner; CString suit; int win; int money;
class Record
{
public:
int turn; CString borP;
class Player
{
Player(); virtual ~Player();
list<Record> resultList; int betUnit; int money;
CString Name;
CString cards; bool hasPut;
Hand hand;
};
class CChatDoc : public CDocument
{
protected: // create from serialization only CChatDocO;
DECLAREDYNCREATE(CChatDoc)
// Attributes public:
BOOL mbAutoChat; CString mstrHandle; CChatSocket* m_pSocket; CSocketFile*
m_pFile; CArchive* m_pArchiveIn;
CArchive* m_pArchiveOut; bool firstTime; bool secondtum;
int betUnit; int total; int balance; int turn;
list<CString> dispInfoList; list<playingRecord> playingRecordList; list<Player>
myPlayerList;
CString result;
// Operations public:
BOOL ConnectSocket(LPCTSTR IpszHandle, LPCTSTR IpszAddress, UINT nPort);
void ProcessPendingRead();
void SendMsg(CString& strText);
void ReceiveMsg();
```

```
void DisplayMsg(LPCTSTR IpszText);
BOOL isDeplicated(list<Player> plist, CString str);
// Overrides
// ClassWizard generated virtual function overrides
// { {AFXVIRTUAL(CChatDoc)
public:
virtual BOOL OnNewDocument(); virtual void DeleteContents();
//} } AFXVIRTUAL
// Implementation public:
CString getPlayingRecord();
void restMessage();
BOOL repeatRecord(list<Record> reList,int turn); void changeDeck(CString &str); int
getTum(CString &str); int toInteger(CString &string); void getPlayerInfo(CString &
str);
CString toString(int num);
bool isStringDeplicated(list<CString> plist, CString str); virtual ~CChatDoc();
virtual void Serialize(CArchive& ar); // overridden for document i/o #ifdef DEBUG
virtual void AssertValid() const;
virtual void Dump(CDumpContext& dc) const;
#endif
protected:
// Generated message map functions protected:
// {{AFX_MSG(CChatDoc)
//afxmsg void OnGameAnothergame();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};
class CChatView : public CView
{
protected: // create from serialization only CChatView();
```

```
11 Attributes public:
CChatDoc* GetDocument();
CSpinButtonCtrl mupDown;
CEdit mbuddyEdit;
CEdit mbalance;
CListCtrl mlistView;
CImageList msmalllmageList;
CImageList mlargelmageList;
CButton msmallButton;
CButton m_largeButton;
CButton m_listButton;
CButton mreportButton;
//CBitmapButton bl, Ь2,Ь3,Ь4, b5;
CRichEditCtrl m_richEdit; int index;
UINT muiTimer; list<Player> playerList;
// Operations public:
void Message(LPCTSTR IpszMessage);
// Overrides
// ClassWizard generated virtual function overrides
// {{AFX_YIRTUAL(CChatV iew)
public:
virtual void OnDraw(CDC* pDC); // overridden to draw this view
66
protected:
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
//}} AFXVERTUAL
11 Implementation public:
int getIndex(CString& str);
void displayCard(CString& cardNameStr, int n);
void creatButton();
```

BOOL OnNotify(WPARAM wParam, LPARAM IParam, LRESULT* pResult);
virtual ~CChatView();
#ifdef DEBUG
virtual void AssertValid() const;
virtual void Dump(CDumpContext& dc) const;
#endif
protected:
void CreateUpDownCtrl(); void CreateListViewO;
//void CreateTreeView(); void CreateRichEdit();
CString toString(int num);
// Generated message map functions protected:
//{{AFXMSG(CChatView)
afxmsg int OnCreate(LPCREATESTRUCT lpCreateStruct); afxmsg void OnGameBetting(); afx msg void OnGamePassingQ;
afx_msg void OnGameAnothergame(); afx_msg void OnGameChangebetunit(); afxmsg void OnTimer(UINT rJDEvent); //}}AFX_MSG
DECLAREMESSAGEMAPQ
#ifndef_DEBUG 11 debug version in chatvw.cpp inline CChatDoc* CChatView::GetDocument()
{return (CChatDoc*)m_pDocument; }
#endif
Клас на стороні сервера:
Карта, колода, рука однакові, нижче показано документацію сервера та фототип класу сервера.
class CServerDoc : public CDocument
{
protected: // create from serialization only CServerDoc();
DECLAREDYNCREATE(CServerDoc)
// Attributes public:
CListeningSocket* m_pSocket;

CStringList mjmsgList;

CPtrList mconnectionList; int max;//total player number int num; int step; int total;

Deck deck; bool ready; bool sdisp; bool isBegin; bool isfinished; bool afterleft;

CString cardlist;

CString displnfo; list<Player> myPlayerList; list<CString> dispInfoList; list<CString> playerList_a;

11 Operations public:

void UpdateClients();

void ProcessPendingAccept();

void ProcessPendingRead(CClientSocket* pSocket);

CMsg* AssembleMsg(CClientSocket* pSocket);

CMsg* ReadMsg(CClientSocket* pSocket); void SendMsg(CClientSocket* pSocket, CMsg* pMsg); void CloseSocket(CClientSocket* pSocket); void Message(LPCTSTR IpszMessage);

CString toString(int num);

CString getPlayerName(CString msg);

CString getCommand(CString msg);

bool isRepeat(CStringList& csl, CString cstr);

int to!nteger(CString string);

// Overrides

// ClassWizard generated virtual function overrides

//{ {AFX_VIRTUAL(CServerDoc)

public:

virtual BOOL OnNewDocument(); virtual void DeleteContents();

//} } AFX_VIRTUAL

// Implementation public:

bool isDeplicated(list<Player> plist, CString str); bool isStringDeplicated(list<CString> plist, CString str); void cardlni(); virtual ~CServerDoc();

virtual void Serialize(CArchive& ar); // overridden for document i/o #ifdef JDEBUG

```
virtual void AssertValid() const;
virtual void Dump(CDumpContext& dc) const;
#endif protected:
// Generated message map functions protected:
//{ (AFX_MSG(CServerDoc) afx_msg void OnGameAnothergame();
//}}AFX_MSG
afx_msg void OnUpdateMessages(CCmdUI* pCmdUI); afxmsg void
OnUpdateConnections(CCmdUI* pCmdUI); DECLARE_MESSAGE_MAP()
};
class CServerView : public CEditView
{
protected: // create from serialization only CServerView();
DECLAREDYNCREATE(CServerView)
// Attributes public:
CServerDoc* GetDocument();
// Operations public:
void Message(LPCTSTR IpszMessage);
// Overrides
// ClassWizard generated virtual function overrides
// { {AFX_VIRTUAL(CServerView)
public:
virtual void OnDraw(CDC* pDC); // overridden to draw this view protected:
//} } AFX_VIRTUAL
// Implementation public:
virtual ~CServerView();
#ifdef DEBUG
virtual void AssertValid() const;
virtual void Dump(CDumpContext& dc) const;
#endif
protected:
```

// Generated message map functions protected:

//{ {AFX_MSG(CServerView)

// NOTE - the ClassWizard will add and remove member functions here. // DO NOT EDIT what you see in these blocks of generated code !

//}}AFX_MSG

DECLARE MESSAGE MAP()

#ifndef _DEBUG 11 debug version in srvrvw.cpp inline CServerDoc* CServerView::GetDocument() {return (CServerDoc*)m_pDocument;}

#endif

Поведінкові моделі (див .рис. 4.4, 4.5,4.6, 4.7)

Діаграма послідовності

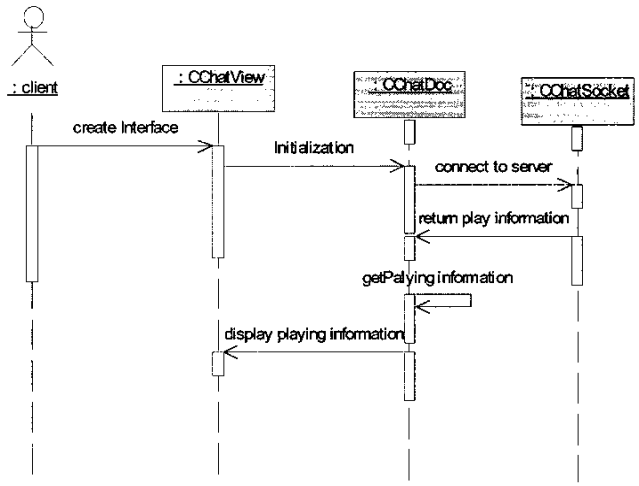

Рисунок 4.4 -Діаграма послідовності для клієнтської підсистеми

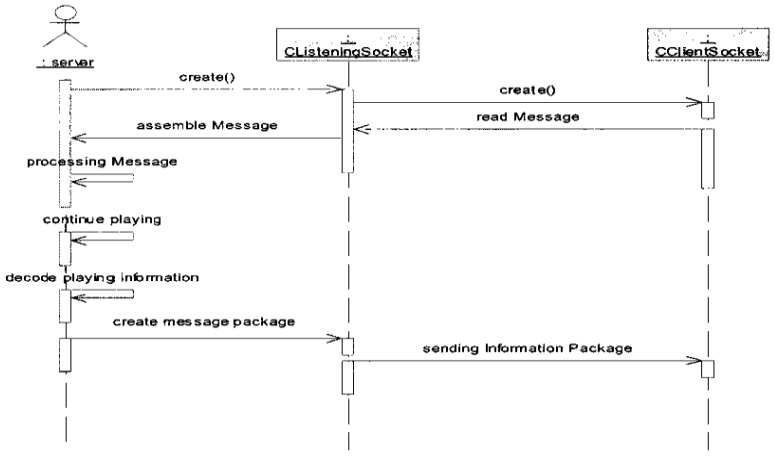

Рисунок 4.5 -Діаграма послідовності для серверної підсистеми

Activity Diagram

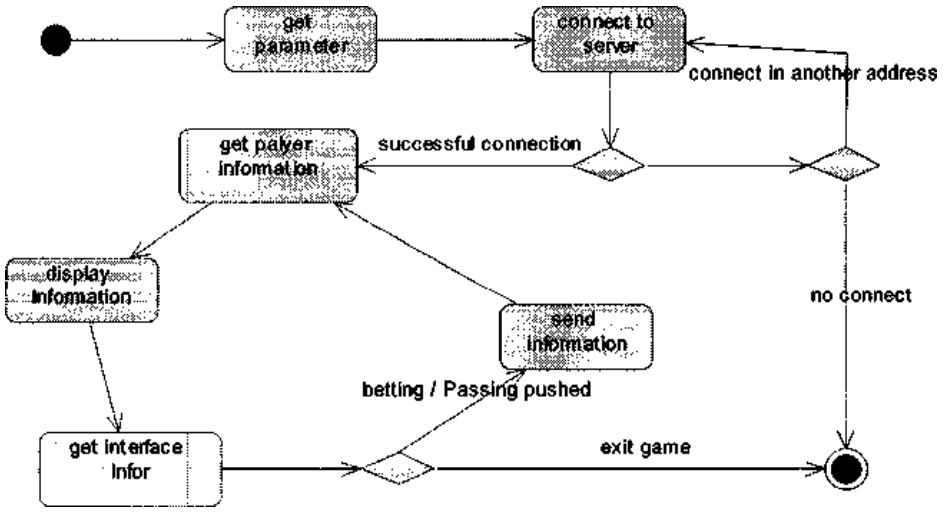

Рисунок 4.6 -Діаграма діяльності клієнтської підсистеми

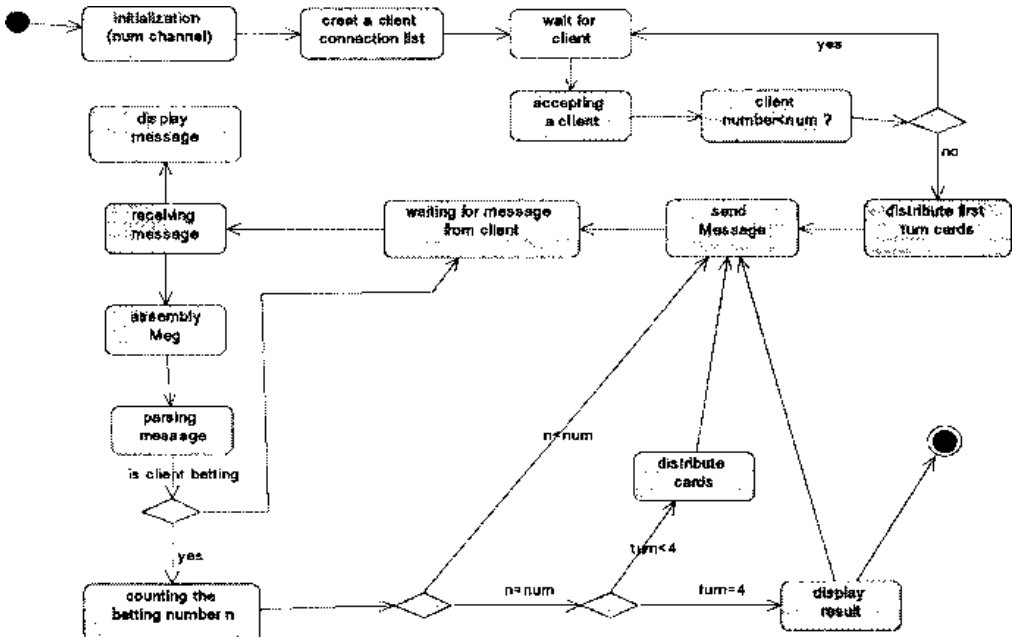

Рисунок 4.7 - Діаграма діяльності серверної підсистеми

Обсяг досліджень, присвячених мережевим іграм, недостатній, навіть попри те, що їхня популярність зростає. Ми мали на меті заповнити частину цієї прогалини в знаннях за допомогою дослідження того, як ігри поводяться в Інтернеті, та забезпечити засоби для сприяння майбутнім дослідженням. У звіті представлені концепції, пов'язані з багатокористувацькою мережевою грою та програмуванням багатокористувацьких мережевих ігор. У ньому представлено

відеопокер, заснований на клієнт-серверній архітектурі. Ця архітектура використовує переваги, які, як було показано, забезпечують кращі послуги для клієнтів, і поширює їх на систему, яка змінюється з дуже високою швидкістю. Також описано новий механізм синхронізації для забезпечення ефективної узгодженості між клієнтами та сервером.

Наше дослідження також корисне для розробки застосунків, не пов'язаних з іграми, таких як веб-трансляції та онлайн-заходи з великою кількістю глядачів, веб-сайти спільнот.

Незважаючи на кількість виконаних нами завдань, завжди є низка можливостей для покращення. Протягом проєкту було зроблено вибір, який вимагав відкидання варіантів. У цьому розділі представлено кілька можливих робіт. Нарешті, був досить великий перелік тем, які, хоча й цікаві та корисні, виходили за рамки нашого конкретного проєкту. Залишилося виконати три завдання. Ми хотіли трохи більше над ними попрацювати, щоб вони були такими ж відшліфованими, як і решта проєкту. По-перше, ми могли б покращити інтерфейс гри, оскільки він має бути простим у вивченні та використанні. Однак, можливо, знадобиться багато роботи з MFC, оскільки MFC зараз не є популярним інструментом у розробці ігор.

По-друге, нам вдалося розробити базову структуру для клієнт-серверної гри та частково її реалізувати. Однак функціональність для гри в Інтернеті ще не повністю реалізована. Вона може стати повноцінно функціональною.

По-третє, нам слід провести більше тестової роботи для налагодження системи. У цій системі є деякі помилки, які потрібно виправити.

Було кілька областей, на які, хоча й були в рамках проекту, нам бракувало часу чи ресурсів. Наприклад, ми можемо додати функцію чату до гри, щоб гравці могли спілкуватися один з одним під час гри. Також, коли лише один гравець отримує доступ до сервера та надсилає запит на налаштування гри для себе, він/вона може грати самостійно, не виграючи чи програючи. Ніхто не любить грати в ігри без результату. Якби можна було додати систему VPNG функція, за допомогою якої будь-який гравець може грати на сервері, могла б бути

цікавішою. Однак, на серверній підсистемі потрібно багато кодування, оскільки для деяких алгоритмів можуть знадобитися навіть знання та алгоритми штучного інтелекту.

4.4 Висновок до розділу

Система VPNG моделює гру з можливістю ставок, підтримкою до 10 гравців у групі та численних груп одночасно, забезпечуючи облік результатів, історії та статусу кожного гравця.

Система VPNG реалізована на основі клієнт-серверної архітектури, де сервер і клієнт мають власні підсистеми з об’єктно-орієнтованою структурою класів, що дозволяє гнучко керувати документами, представленнями та взаємодією користувача.

Проєкт продемонстрував базову реалізацію мережевої гри на основі клієнт-серверної архітектури з використанням MFC, однак залишилося кілька невирішених технічних завдань, зокрема покращення інтерфейсу, повна реалізація онлайн-функціоналу та виправлення помилок. Дослідження підкреслює потенціал такого підходу як для ігрових застосунків, так і для інших інтерактивних мережевих сервісів.

ВИСНОВКИ

Дослідження, проведене в рамках цієї роботи, підкреслює ключову роль методик мережевих імплементацій у створенні ефективних і масштабованих багатокористувацьких ігрових стратегій, які відповідають сучасним вимогам ігрової індустрії. Аналіз вступних аспектів показав, що еволюція комп'ютерних ігор від одиночних до багатокористувацьких платформ зумовила потребу в надійних мережевих технологіях. Сучасний стан індустрії, з її акцентом на онлайн-ігри, такі як стратегії в реальному часі, підтверджує важливість оптимізації мережевої взаємодії для забезпечення низької затримки та високої синхронізації.

Другий розділ, присвячений передумовам і пов'язаним технологіям, продемонстрував, що моделі програмування мережевих ігор, такі як клієнт-серверна та однорангова (peer-to-peer), відіграють вирішальну роль у забезпеченні масштабованості та стабільності. Функції програмування, зокрема обробка пакетів даних і керування станом гри, дозволяють створювати динамічні ігрові середовища, які підтримують одночасну взаємодію багатьох гравців. Ці технології є основою для розробки систем, здатних витримувати високі навантаження.

Третій розділ, що аналізує мережеве програмування з використанням Windows Sockets, показав, що ця технологія залишається стандартом для реалізації мережевих ігор завдяки своїй гнучкості та підтримці різних моделей програмування, таких як блокуючі та неблокуючі сокети. Використання сокетів з архівами забезпечує ефективне збереження та передачу ігрових даних, що є критично важливим для стратегій із складними сценаріями. Практичне застосування Windows Sockets підтвердило їхню надійність у реальних проєктах.

Четвертий розділ, присвячений проектуванню системи VPNG, продемонстрував, що чітко визначена архітектура, модульна модель структури та продумані класи дозволяють створювати гнучкі та масштабовані ігрові системи. Поведінкові моделі, реалізовані в системі, сприяють ефективній синхронізації між клієнтами та сервером, забезпечуючи стабільний ігровий

досвід. Практична реалізація системи VPNG показала, що інтеграція мережевих технологій із об’єктно-орієнтованим підходом значно полегшує розробку багатокористувацьких ігор.

Узагальнюючи, методики мережевих імплементацій ігрових стратегій є важливим інструментом для створення сучасних багатокористувацьких ігор, які відповідають високим стандартам продуктивності та користувацького досвіду. Результати дослідження можуть бути використані розробниками ігор, інженерами та дослідниками для створення інноваційних мережевих рішень. Перспективи подальших досліджень включають інтеграцію штучного інтелекту для оптимізації мережевої синхронізації, розробку кросплатформних ігрових систем і вдосконалення протоколів для зменшення затримок у реальному часі. Ці напрямки сприятимуть розвитку більш ефективних і доступних ігрових технологій у майбутньому.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. **Blow, J.** Networking for Games: Fundamentals and Practices. *Game Developer Magazine*, 2004, 11(3), 34–40.

2. **Bettner, P., & Terrano, M.** 1500 Archers on a 28.8: Network Programming in Age of Empires II. *GDC Vault*, 2001. — Режим доступу: https://www.gdcvault.com/play/1015319/1500-Archers-on-a-28

3. **Caltagirone, S., Keys, M., Schlief, B., & Willman, J.** Architecture for a Massively Multiplayer Online Role-Playing Game Engine. *Journal of Computing Sciences in Colleges*, 2002, 18(2), 105–116.

4. **Claypool, M., & Claypool, K.** Latency and Player Actions in Online Games. *Communications of the ACM*, 2006, 49(11), 40–45. — Режим доступу: https://doi.org/10.1145/1167838.1167860

5. **Epic Games.** Unreal Engine Networking Guide. *unrealengine.com*, 2024. — Режим доступу: https://docs.unrealengine.com/5.4/en-US/networking-overview

6. **Fiedler, G.** What Every Programmer Needs to Know About Game Networking. *gafferongames.com*, 2010. — Режим доступу: https://gafferongames.com/post/what_every_programmer_needs_to_know_about_game_networking

7. **Funkhouser, T. A.** RING: A Client-Server System for Multi-User Virtual Environments. *ACM SIGGRAPH Computer Graphics*, 1995, 29(2), 85–92. — Режим доступу: https://doi.org/10.1145/218994.218999

8. **Glinka, F., Ploss, A., Gorlatch, S., & Müller-Iden, J.** High-Level Development of Multiplayer Online Games. *Proceedings of NetGames 2007*, 2007, 21–26. — Режим доступу: https://doi.org/10.1109/NETGAMES.2007.11

9. **Gregory, J.** *Game Engine Architecture.* — 3rd ed. — Boca Raton, FL: CRC Press, 2018. — 1240 с. — Режим доступу: https://www.gameenginearchitecture.com

10. **Hall, C.** *Windows Sockets: An Open Interface for Network Programming under Microsoft Windows.* — Redmond, WA: Microsoft Press, 1993. — 208 с.

11. **Hampel, T., Bopp, T., & Hinn, R.** A Peer-to-Peer Architecture for Massive

Multiplayer Online Games. *Proceedings of NetGames 2006*, 2006, Article No. 7. — Режим доступу: https://doi.org/10.1145/1230040.1230065

12. **Jacobson, J., & Hwang, Z.** Unreal Tournament for Immersive Interactive Theater. *Communications of the ACM*, 2002, 45(1), 39–42. — Режим доступу: https://doi.org/10.1145/502269.502294

13. **Kushner, D.** *Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture.* — New York, NY: Random House, 2003. — 352 с.

14. **Microsoft.** Windows Sockets 2 API Reference. *docs.microsoft.com*, 2024. — Режим доступу: https://learn.microsoft.com/en-us/windows/win32/winsock/windows-sockets-start-page-2

15. Distributed Nash Equilibrium Seeking Algorithm Design для Multi-Cluster Games with High-Order Players– Zhenhua Deng, Yangyang Liu (2021)

16. Distributed strategy-updating rules for aggregative games of multi-integrator systems with coupled constraints – Xin Cai, Feng Xiao, Bo Wei (2021)

17. . SDN-based Resource Allocation in Edge and Cloud Systems: Evolutionary Stackelberg Differential Game- Jun Du et al. (2021)

18. Distributed Machine Learning with Strategic Network Design: A Game-Theoretic Perspective– Shutian Liu, Tao Li, Quanyan Zhu (2020, ревізія 2022)

19. **Mulligan, J., & Patrovsky, B.** *Developing Online Games: An Insider's Guide.* — Indianapolis, IN: New Riders, 2003. — 528 с.

20. **Nystrom, R.** *Game Programming Patterns.* — San Francisco, CA: Genever Benning, 2014. — 354 с. — Режим доступу: https://gameprogrammingpatterns.com

21. **OnTu.** Комп'ютерні ігри та мультимедіа як інноваційний підхід до освіти. *ontu.edu.ua*, 2024. — Режим доступу: https://ontu.edu.ua/comp-games-multimedia

22. **Pano.** Методичні рекомендації щодо організації освітнього процесу та викладання навчальних предметів. *pano.pl.ua*, 2024. — Режим доступу: https://pano.pl.ua/methodical-recommendations-2024

23. **Rak, S., & Van der Ster, J.** Multiplayer Game Programming: Architecting Networked Games. *Game Developer Magazine*, 2015, 22(7), 28–34.

24. **Smed, J., & Hakonen, H.** *Algorithms and Networking for Computer Games.* — 2nd ed. — Chichester, UK: Wiley, 2017. — 416 с. — Режим доступу: https://www.wiley.com/en-us/Algorithms+and+Networking+for+Computer+Games-p-9781119259763

25. **Steinmetz, R., & Wehrle, K.** *Peer-to-Peer Systems and Applications.* — Berlin, Germany: Springer, 2005. — 626 с. — Режим доступу: https://doi.org/10.1007/11530657

26. **Unity Technologies.** Multiplayer Networking in Unity. *docs.unity3d.com*, 2024. — Режим доступу: https://docs.unity3d.com/Manual/UNet.html

27. **Valve Corporation.** Source Engine Networking Overview. *developer.valvesoftware.com*, 2024. — Режим доступу: https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking

28. **White, W., Koch, C., Gehrke, J., & Demers, A.** Better Scripts, Better Games. *Communications of the ACM*, 2008, 51(12), 42–47. — Режим доступу: https://doi.org/10.1145/1409360.1409375

29. **Wikipedia.** Мережеве програмування. *uk.wikipedia.org*, 2024. — Режим доступу: https://uk.wikipedia.org/wiki/Мережеве_програмування

30. **X Post by @gamedev_net.** Optimizing network latency in multiplayer strategy games using UDP and prediction algorithms. *X*, 2025. — Режим доступу: https://t.co/9mXzQwRtYp

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: "Методики мережевих імплементацій ігрових стратегій"

Обсяг пояснювальної записки: 55 аркушів

Дата закінчення дипломної роботи 10 червня 2025р.

Підпис студента _____