

Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра
на тему: «Проект захищеного мережного протоколу об-
міну даними для комп'ютерної гри»

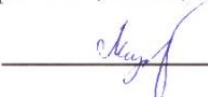
Виконав: студент 4-го курсу групи КІ-21-1
спеціальності 123 «Комп'ютерна інженерія»

 Владислав КОРНІЙЧУК

Керівник: доцент кафедри КІ та КБ, к.пед.н., доцент

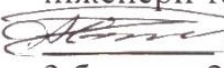
 Олена ГОЛОВНЯ

Рецензент: завідувач кафедри КН, PhD, доцент

 Марина ГРАФ

Житомир – 2025

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач
кафедри комп'ютерної
інженерії та кібербезпеки
 Андрій Єфіменко
3 березня 2025 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу бакалавра

Студент: Корнійчук Владислав Володимирович

Тема роботи: «Проект захищеного мережного протоколу обміну даними для комп'ютерної гри»

затверджена Наказом університету від 3 березня 2025 р. № 79/с

Термін здачі студентом закінченої роботи 21 червня 2025 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):
Об'єктом дослідження є мережеві протоколи транспортного та прикладного рівня для обміну даними. Предметом дослідження є засоби реалізації захисту мережевого протоколу для комп'ютерної гри.

Обладнання та програмне забезпечення:

1. Операційна система Windows 10.
2. Середовище розробки Visual Studio 2022.
3. Системні бібліотеки Win32Api та WinSock2.
4. Відеокарта сумісна з OpenGL 4.0+.
5. Ноутбук HP ProBook 650 G5.
6. Оперативна пам'ять, мінімум 512 мб.
7. HEX-редактор.

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Володимир ВОРОТНІКОВ	 3.03.25	 03.03.25
2	Володимир ВОРОТНІКОВ	 3.03.25	 03.03.25
3	Володимир ВОРОТНІКОВ	 3.03.25	 03.03.25

Студент  Владислав КОРНІЙЧУК

Керівник  Олена ГОЛОВНЯ

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	25.02.2025	
2	Створення рушія гри	03.03.2025 05.04.2025	
3	Створення консолі гри	16.03.2025 18.03.2025	
4	Створення гри, інтеграція фізики	20.03.2025 01.05.2025	
5	Створення ігрової сесії	05.04.2025 04.05.2025	
6	Створення протоколу	10.03.2025	
7	Тестування роботи протоколу	12.03.2025	
8	Створення мережевої бібліотеки	16.03.2025	
9	Прив'язка гри до мережі	15.05.2025 20.05.2025	
10	Реалізація шифрування даних	27.05.2025	
11	Підготовка презентації до захисту	20.06.2025	

Студент  Владислав Корнійчук

Керівник  Олена Головна

АНОТАЦІЯ

Пояснювальна записка до випускної кваліфікаційної роботи бакалавра на тему «Проект захищеного мережевого протоколу обміну даними для комп'ютерної гри» складається зі вступу, трьох розділів, висновків, списку використаної літератури та додатків. Текстова частина – 79 сторінок, загальний обсяг 108 сторінок. Пояснювальна записка містить 30 рисунків та 4 таблиці. Список використаних джерел налічує 15 найменувань.

Метою даної роботи є розробка захищеного мережевого протоколу для комп'ютерної гри.

У процесі роботи було розглянуто теоретичні відомості проектування протоколів, особливості мережевої архітектури, а також базові аспекти створення комп'ютерної гри. Було розроблено протокол, архітектуру мережі, підтримка захисту конфіденційності даних та відеогру. Було також проведено тестування відеогри у локальній мережі та тестування роботи створеного протоколу.

Результати тестування показали, що наш протокол та його впровадження в мережеву архітектуру відеогри дали позитивний результат, що може використатись як демонстрація базових принципів реалізації таких проєктів на практиці.

Ключові слова: МЕРЕЖА, ПРОТОКОЛ, ДАНІ, БАЙТИ, КЛІЄНТ, СЕРВЕР, ШИФРУВАННЯ, БІБЛІОТЕКИ, КОМУНІКАЦІЯ.

					ККІКБ.КР.Б – 123 – 06 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Проект захищеного мережного протоколу обміну даними для комп'ютерної гри	Літ.	Арк.	Аркушів
Розроб.		Корнійчук В.В.		16.06.25				
Керівник		Головня О.С.		16.06.25			4	108
Реценз.		Граф М.С.		16.06.25		ФІКТ, Гр. КІ-21-1		
Н. контр.		Головня О.С.		16.06.25				
Затверд.		Сфіменко А.І.		16.06.25				

ANNOTATION

The explanatory note for the bachelor's graduation qualification work on the topic «Project of a secure network data exchange protocol for computer games» consists of an introduction, three chapters, conclusions, a list of references and appendices. The text part is 79 pages, the total volume is 108 pages. The explanatory note contains 30 figures and 4 tables. The list of references includes 15 items.

The purpose of this work is to develop a secure network protocol for a computer game.

In the course of the work, theoretical information on the design of protocols, features of network architecture, and basic aspects of creating a computer game were considered. We developed a protocol, network architecture, data privacy support, and a video game. We also tested the video game in a local network and tested the operation of the developed protocol.

The test results showed that our protocol and its implementation in the network architecture of the video game gave a positive result, which can be used as a demonstration of the basic principles of implementing such projects in practice.

Keywords: NETWORK, PROTOCOL, DATA, BYTES, CLIENT, SERVER, ENCRYPTION, LIBRARIES, COMMUNICATION.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ВСТУП	8
РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ПРОЕКТУВАННЯ МЕРЕЖЕВОГО ПРОТОКОЛУ ТА СТВОРЕННЯ ГРИ.....	9
1.1 Мережевий протокол. Основні моделі мережі.....	9
1.2 Проектування мережевих протоколів	11
1.3 Мережева архітектура.....	19
1.4 Базові аспекти створення комп'ютерної гри.....	23
Висновки до першого розділу	25
РОЗДІЛ 2 РОЗРОБКА МЕРЕЖЕВОГО ПРОТОКОЛУ ТА ІГРОВОГО РУШІЯ...	28
2.1 Розробка мережевого потоку	28
2.2 Розробка мережевого коду	34
2.3 Розробка гри та впровадження мережевого коду у гру.....	41
2.4 Захист даних.....	61
Висновки до другого розділу	68
РОЗДІЛ 3 ТЕСТУВАННЯ РОБОТИ ПРОТОКОЛУ	70
3.1 Консоль додатка	70
3.2 Комп'ютерна гра.....	72
3.3 Тестування роботи протоколу.....	74
Висновки з третього розділу	77
ВИСНОВКИ.....	79
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	81
ДОДАТКИ.....	83

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – операційна система

ПЗ – програмне забезпечення

ООП – об'єктно-орієнтоване програмування

WINSOCK – Windows Socket, сокет ОС Windows

SDL – Simple DirectMedia Layer, мультимедійна бібліотека

ACK – acknowledgement, підтвердження

P2P – Peer to peer, однорангова мережа

TCP – Transmission Control Protocol, протокол управління передачею

UDP – User Datagram Protocol, протокол датаграм користувача

MTU – Maximum Transmission Unit, максимальна одиниця передачі

WLAN – Wireless Local Area Network, бездротова локальна мережа

ITNP – In The Network Protocol, протокол у мережі

DLL – Dynamic Link Library, динамічно зв'язувальна бібліотека

EXE – Executable, виконувальний файл

PDU – Protocol Data Unit, протокольний блок даних

CRC – Cyclic redundancy check, циклічний надлишковий код

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. На сьогоднішній день, онлайн ігри домінують у ігровій індустрії. Можливість грати з друзями чи з незнайомими людьми вже не є чимось таким рідкісним та особливим як, наприклад 20-30 років тому. З часом технології змінились, онлайн відеоігри все частіше удосконалювались, як з точки зору графіки, так і точки зору мережевої архітектури. Але щоб забезпечити надійну онлайн гру, зі стабільною швидкістю, враховуючи затримку та безпеку, потрібно розробити надійний мережевий код для гри, чи ігрового рушія. Для цього потрібно визначити як будуть надсилатися дані між користувачами, враховуючи певні обмеження у плані швидкості та надійності. Тут знадобиться мережевий протокол, який допоможе нам розв'язати певні проблеми.

Ця робота націлена на те, щоб показати з якими проблемами може зіткнутись розробник при реалізації власних протоколів, та допоможе узагальнити вивчення базових принципів роботи мережних додатків таких як онлайн ігри, та основи роботи комунікаційних протоколів.

Метою кваліфікаційної роботи є розробка захищеного мережевого протоколу для комп'ютерної гри. Визначена мета обумовлює наступні завдання.

1. Аналіз теоретичних відомостей та проблем проєктування мережевого протоколу для мережевої гри.
2. Проєктування мережевого протоколу, мережевої архітектури гри.
3. Розробка захищеного мережевого протоколу та впровадження його у гру.
4. Тестування роботи мережевого протоколу у локальній мережі.

Об'єктом дослідження є мережеві протоколи транспортного та прикладного рівня для обміну даними.

Предметом дослідження є засоби реалізації захисту мережевого протоколу для комп'ютерної гри.

У процесі виконання кваліфікаційної роботи було розроблено проєкт мережевої комп'ютерної гри, з забезпеченням захисту передачі даних по власному мережевому протоколу на архітектурі клієнт-сервер.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ПРОЕКТУВАННЯ МЕРЕЖЕВОГО ПРОТОКОЛУ ТА СТВОРЕННЯ ГРИ

1.1 Мережевий протокол. Основні моделі мережі

Насамперед потрібно визначитись з тим що таке мережевий (комунікаційний) протокол та для чого він використовується у мережевих системах та інтернеті.

Розглянемо наступну ситуацію – відправник надіслав дані отримувачу, отримувач їх прийняв, але сирі дані прочитати та зрозуміти не зможе через те, що він не знає що з ними робити. Аналогію можна провести навіть з бінарними файлами, де для того щоб відобразити правильно файл, його спочатку потрібно прочитати та обробити. Якщо для файлів використовується заголовок та інші елементи структури файлу, то для мереж використовується **комунікаційний протокол**.

З точки зору опису **комунікаційний**, частіше **мережевий**, **протокол** – це набір певних правил про те, як повідомлення та дані будуть формуватися та оброблятися. Порухення протоколу створить труднощі в обміні даних, а може і зовсім зробити обмін неможливим [2]. Залежно від задач протоколи бувають різних типів та рівнів мережі.

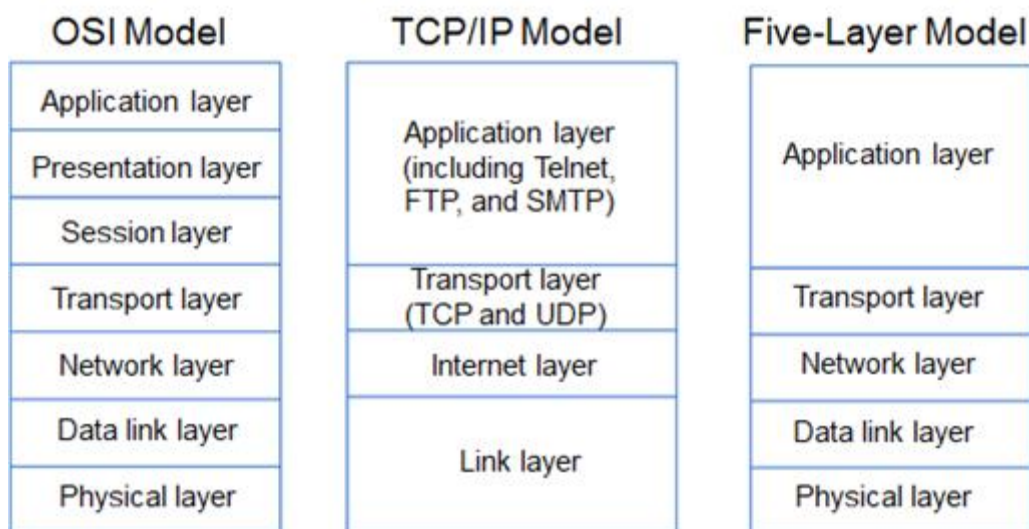


Рисунок 1.1.0 – Мережеві моделі

Всього виділяють три основні моделі мережі – модель OSI, TCP/IP модель та п'ятирівневу модель мережі (Рис. 1.1.0). Для визначення протоколів певних рівнів, спочатку розглянемо ці мережеві моделі.

Модель OSI – перша модель розшарування мережевих рівнів, заснована на ранніх роботах Міжнародної організації зі стандартизації (International Organization for Standardization, ISO) і відома як Еталонна модель взаємозв'язку відкритих систем ISO. Її часто називають моделлю ISO. На жаль, модель ISO передувала роботі в Інтернеті і не дуже добре описує протоколи Інтернету. Крім того, замість рівня, присвяченого «Інтернету», модель ISO була розроблена для однієї мережі і має «мережевий» рівень. Сьогодні модель OSI використовується передусім як теоретична основа для розуміння комп'ютерних мереж. Водночас, на базі даної моделі складно реалізовувати практичні проєкти та контролювати обробку помилок у них [4].

TCP/IP модель – друга основна модель багаторівневості. Одна з основних відмінностей між моделями ISO та Інтернет-рівня виникає з того, як вони були визначені. Модель ISO була побудована до використання мережевих протоколів, в той час як мережева модель TCP/IP була вже зконструйована з появою та форматуванням основних протоколів [4]. Має 4 рівні мережі, на відмінно від OSI з 7-ма рівнями. З недоліків цієї моделі можна виділити наступне:

1. У цій моделі немає чіткого розмежування концепцій служб, інтерфейсів і протоколів.
2. Модель TCP/IP не є загальною і досить погано описує будь-який стек протоколів, крім TCP/IP.
3. Канальний рівень більше представляється як інтерфейс між мережею і рівнями передачі даних аніж тим що є у описі рівневих протоколів.

П'ятирівнева модель мережі – модель яка найчастіше використовується. По суті це гібридна модель яка пов'язує модель OSI, без рівнів представлення та сеансу, та модель TCP/IP з використанням протоколів. Ця модель має 5 рівнів мережі, у порівнянні з попередньою моделлю [1].

Оскільки остання, 5-ти рівнева, модель найчастіше використовується при побудові протоколів та мережі, опишемо їхні основні рівні та які протоколи використовуються [9].

Фізичний рівень. Знаходиться на нижньому рівні архітектури мережі. Рівень використовується для надсилання бітів від одного комп'ютера до іншого. Фізичний

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

рівень перетворює дані в електронні імпульси. Ці дані потім подають у двійковій формі та надсилають на бездротові або дротові пристрої.

Канальний рівень. Знаходиться на другому рівні архітектури мережі. Відповідає за створення кадрів, які переміщуються мережею. Ці кадри інкапсулюють пакети та використовують MAC-адреси для ідентифікації джерела та одержувача. Приклади протоколів цього рівня – PPP, PPPoE.

Мережевий рівень. Рівень відповідає за створення пакета, який переміщується мережею. За допомогою IP-адреси мережевий рівень ідентифікує джерело та одержувача кожного пакета. Прикладами таких протоколів є IP версії 4, та IP версії 6.

Транспортний рівень. Відповідає за передачу даних по мережі з кінця в кінець, забезпечуючи надійність, керування потоком та корекцію даних. Він діє як посередник між прикладним рівнем та транспортним рівнем, з'єднуючи програми на різних хостах. Найвідоміші протоколами цього рівня є TCP та UDP, які ми розглянемо пізніше.

Рівень додатків. Знаходиться на вершині п'ятирівневої архітектури Інтернету. Рівень відповідає за обробку протоколів високого рівня та дозволяє користувачеві взаємодіяти з додатком. Протоколи прикладного рівня обслуговуються транспортним рівнем, на якому він побудований. Прикладами відомих протоколів даного рівня є HTTP/HTTPS.

1.2 Проектування мережевих протоколів

З урахуванням відомостей про мережеві моделі та визначення протоколів, буває недостатньо використати існуючий протокол для певних завдань, як наприклад для нашого додатку – комп'ютерної гри. Для створення протоколу окрім того, для чого він повинен використовуватися, потрібно врахувати кілька факторів. Наприклад який рівень буде представляти протоколу, як буде формуватися заголовок, чи буде протокол захищений і тд.

Окрім цього постають певні проблеми та питання:

- проблема надійності;

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

- проблема продуктивності;
- проблема безпеки;
- проблема сумісності.

Проблема безпеки. Зі стрімким розвитком Інтернету зростає кількість кібератак. Користувач повинен бути впевненим в тому, що дані які він надсилає, не повинні бути кимось викрадені і використовуватися для зловмисних цілей, часто всупереч інтересам. Прикладом таких даних можуть слугувати дані кредитної картки, приватні повідомлення, особиста інформація, тощо. Важливим завданням є засоби запобігання викраденню чи фальсифікацію відправника. Щоб запобігти кібератакам та кіберзагрозам потрібно використати ряд технологій для захисту мережі та впровадити у власний користувацький протокол. Наприклад, можна використовувати алгоритми шифрування [1] – симетричні (AES, Blowfish) чи асиметричні (RSA, схема Ель-Гамала) – або ж створити унікальний ідентифікатор для користувача у випадку його підробки. Водночас із проблеми безпеки впливає наступна – не менш важлива проблема.

Проблема продуктивності. Продуктивність залежить від системних вимог пристрою, налаштувань мережевого адаптеру та навантаження на мережу. Як було розглянуто вище, алгоритми шифрування можуть впливати на швидкість передачі даних, а саме на тривалість підготовки до відправлення, або підготовки до зчитування. Тут все в основному залежить від задач програми. Якщо важлива сильна безпека та надійність передавання даних – можна використовувати асиметричні алгоритми шифрування, оскільки вони більш крипто-стійкі та використовують два ключа, публічний та приватний. З іншого боку, якщо швидкість та менша обчислювальна потужність грають важливу роль, наприклад, як в онлайн відеоіграх, можна використовувати симетричні алгоритми шифрування, але такі алгоритми менш стійкі до кібератак. Те саме стосується і стиснення даних: вибір алгоритму залежить від задач, які виконуватиме протокол.

Продуктивність мережі також залежить від структурного дисбалансу ресурсів апаратного забезпечення. Так, якщо швидка, гігабітна лінія підключена до

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

комп'ютера з малою потужністю, то центральний процесор не може швидко обробляти пакети які отримує. Деякі пакети можуть губитися, але врешті решт вони знову можуть повторно надсилатися, що створить затримку в мережі та знизить загальну пропускну здатність [1].

Також слід враховувати і розмір даних які надсилаються, оскільки великий обсяг пакету може сильно навантажити проект та знизити пропускну здатність, а може й призвести до відкидання пакету маршрутизатором. Детальніше питання розміру даних розглянуто у проблемі надійності.

Проблема сумісності. Тут залежить особливості реалізації протоколу в основному залежатимуть від завдання цього протоколу та сфери його застосування. Наприклад передача даних по цьому протоколу не може гарантувати правильність роботи на інших пристроях чи навіть мережевих адаптерах. Одна із таких причин – це порядок байтів [4]. Наприклад у процесорах x86 чи x64 використовується порядок байтів від молодшого до старшого (little endian). Тобто 32-бітне ціле число 5 у порядку байтів little-endian буде 05-00-00-00₁₆. У процесорах від Motorola та у протоколі TCP/IP застосовується порядок байтів від старшого до молодшого (big endian). Наприклад те саме число 5 у порядку (big endian) еквівалентно 00-00-00-05₁₆. Також слід не забувати про те, чи зможуть брандмауери прийняти власницький протокол, або ж чи не відбудеться відкидання NAT у маршрутизаторах [2]. В такому разі потрібно визначитися, на якому мережевому рівні ми реалізуємо протокол [3], а також те, як запобігти проблемам, пов'язаним з різним порядком байтів та вирівняти ширину даних.

Проблема надійності. Ця проблема пов'язана з описаними вище проблемами з безпеки та продуктивності. Надійне з'єднання повинно гарантувати те, що під час передачі даних між користувачами ці дані не втрачуться у мережі. Тут постає також питання пропускну здатності, підтверджень з боку користувача та мінімальне навантаження на мережу. Одним із кроків вирішення цієї проблеми є реалізація механізму підтвердження - АСК (acknowledgement). Такий механізм, наприклад широко використовується у протоколі транспортного рівня TCP, де отримувач відправляє зворотній пакет зі встановленим бітом АСК, чого не реалізовано, наприклад у

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

протоколі UDP. Одним із основних моментів для надійності даних є також їхній розмір та значення MTU (Maximum Transmission Unit) [2].

MTU визначає максимальний розмір корисного навантаження пакету який можна передати на каналному рівні. Теоретично, ми зможемо надсилати пакети розмірністю до 65 535 байтів, але в такому разі відбудеться його фрагментація, тобто розподіл пакету на менші датаграми. Особливо це актуально для таких типів протоколів які не потребують підтвердження, оскільки вони часто губляться, а при фрагментуванні на менші датаграми можуть і взагалі не прийти. Варто також зазначити що MTU залежить від мережевого інтерфейсу - основні значення наведено на рисунку 1.2.0.

Технологія	MTU
DIX Ethernet	1500 байт
Ethernet 802.3	1492 байт
Token Ring (IBM, 16 Мбіт/с)	17 914 байт
Token Ring (802.5, 4 Мбіт/с)	4464 байт
FDDI	4352 байт
X.25	576 байт

Рисунок 1.2.0 – Основні значення розміру MTU

Для того, щоб запобігти втрачання даних через фрагментацію, потрібно визначитися з задачами які протокол повинен реалізувати. Одним із таких рішень є реалізація підтримки фрагментації як у протоколі TCP, або ж обмежити пакет так, аби розмір даних не перевищував значення MTU. Також слід врахувати, що значення MTU для певних пакетів рахується завжди з розміром заголовка IP, 20 байтів для IPv4 та 40 байтів для IPv6 [7]. Наприклад, для надсилання пакету за протоколом UDP за умови розміру MTU для Ethernet (1500 байтів), можна обчислити максимальний розмір пакету[8] (1500 – розмір заголовка IP(20 байтів) – розмір заголовка UDP(8 байтів) = 1472 байтів), рис. 1.2.1. Інший підхід полягає у застосуванні стиснення даних, якщо виникає потреба надіслати великий обсяг пакету за 1 кадр, але при цьому, знову ж таки, виникає проблема продуктивності мережі.

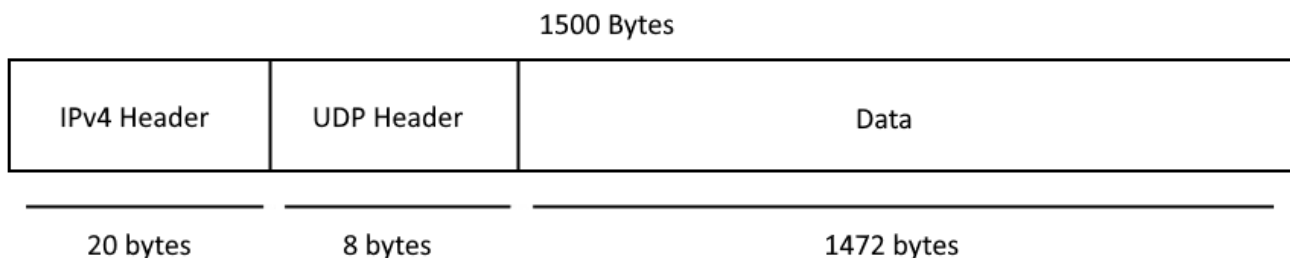


Рисунок 1.2.1 – Обрахунок розміру пакету даних для протокола UDP

Вище було розглянуто базові принципи та налаштування, які необхідні для побудови протоколів. Варто звернути увагу на те, що не всі концепції підходять для певних мережесих рівнів. Наприклад протоколи канального рівня не використовують шифрування даних, ця задача покладається на протоколи верхніх рівнів, або TCP кодує порядок байтів у little endian, в той час як UDP робить це залежно від системи та сокетів [4]. Для нашого проекту комп'ютерної гри, потрібно розробити надійний прокол, який гарантуватиме швидкість передачі даних, безпеку та можливість повторну відповідь.

Розробка протоколу з нуля теоретично можлива, якщо використати необроблені сокети (сокети буде детальніше розглянуто пізніше), але в такому разі деякі ОС будуть блокувати надсилання пакетів, якщо не налаштувати параметри мережевого з'єднання та firewall [10]. Альтернативне рішення – можна побудувати мережесий протокол на базі вже існуючого. Для такого додатка як відеогра, доцільніше використати протокол транспортного рівня. Розглянемо два відомих протоколів транспортного рівня – TCP та UDP.

Протокол UDP (User Datagram Protocol) – протокол транспортного рівня моделі OSI, з простою структурою та реалізацією [6]. UDP використовує основний інтернет-протокол для передачі повідомлень з одного комп'ютера на інший. Важливо відмітити те, що UDP надає додаткам ту ж саму семантику доставки датаграм без з'єднання, що і IP. Тобто, UDP не гарантує, що повідомлення будуть доставлені, не гарантує, що повідомлення будуть доставлені в тому ж порядку, в якому вони були відправлені, і не надає ніяких механізмів для управління швидкістю, з якою потоки інформації між парою взаємодіючих хостів. Таким чином, UDP-повідом-

лення можуть бути втрачені, продубльовані або доставлені в неправильному порядку. Крім того, пакети можуть надходити швидше, ніж одержувач може їх обробити. Обробку помилок, дублікацій та втрат реалізують протоколи вищого рівня, засновані на UDP. Попри свої недоліки, протокол забезпечує доволі швидку передачу даних. Структура заголовка протоколу показана на рисунку 1.2.2.

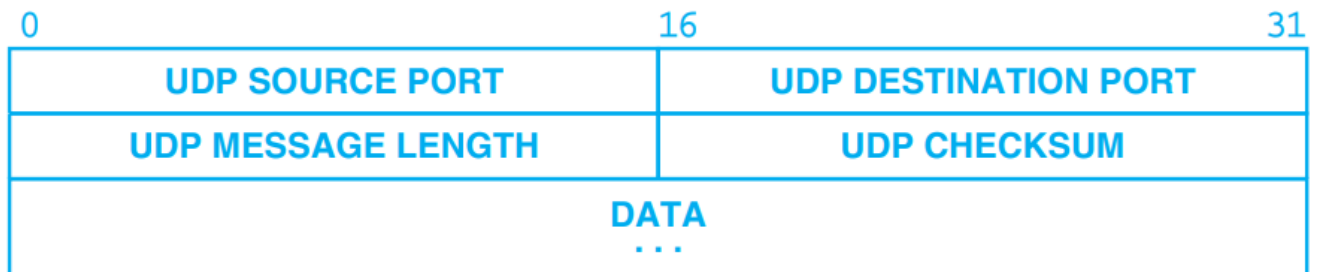


Рисунок 1.2.2 – Структура UDP

Варто зазначити що тут не всі поля є обов’язковими, як, наприклад, порт призначення та контрольна сума. Щодо поля кількість даних тут потрібно врахувати розмір датаграми UDP + розмір даних [4].

Протоколи, які використовують UDP як основу [2]:

- TFTP (Trivial File Transfer Protocol, найпростіший протокол передачі файлів);
- SNMP (Simple Network Management Protocol, простий протокол управління мережею);
- DHCP (Dynamic Host Configuration Protocol, протокол динамічної конфігурації вузла);
- DNS (Domain Name System, служба доменних імен).

Протокол TCP (Transmission Control Protocol) – протокол транспортного рівня, який забезпечує надійність передачі даних. Особливістю є те, що протокол підтримує ACK (підтвердження), фрагментування на датаграми, а також він має завжди впорядкований порядок байтів, в даному випадку це big-endian [4]. Оскільки попередній протокол UDP, а також протокол IP не гарантує порядок надсилання датаграм чи швидкість відправлення, TCP зумовлений виправити цю проблему використовуючи правильну швидкість відправлення (дотримуючись цілей ефектив-

ного використання пропускну́ї пропускну́ї здатності та запобігання перевантажен-
ням), а також стеження за інтервалами очікування, і вразі потреби відправляти да-
таграми назад відправнику. На рисунку 1.2.3 зображено поля які містить заголовок
протокол TCP. Розмір даного заголовка становить 20 байт, при повному налашту-
ванні 60 байт, що еквівалентно розміру заголовка IP версії 4 (максимальний мож-
ливий розмір IPv4 становить теж 60 байт). В основному є два спільних поля з UDP
такі як порт отримувача та порт відправника.

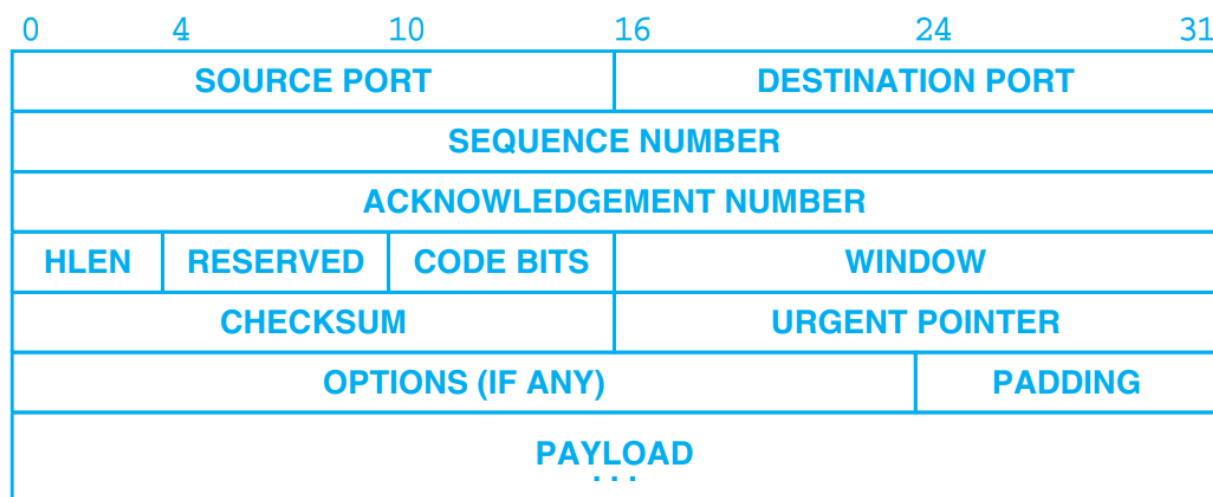


Рисунок 1.2.3 – Структура заголовка TCP

Sequence number. Поле розмірністю 32 біти, яке визначає номер першого байту в сегменті надісланих даних. Фактично використовується для того, аби запо-
бігти неправильній послідовності пакетів. Визначається як сума номеру поперед-
нього сегменту та кількості даних відправлених у ньому [5].

Acknowledgement number. Поле розмірністю 32-біти, яке визначає, чи пот-
рібно надсилати новий сегмент даних. Переважно використовується для перевірки
чи було відправлено та прийнято обома сторонами [5].

HLEN. Поле розмірністю у 4 біти. Визначає розмір заголовка TCP (20 – 60
байтів) та використовується для поля options [5].

Reserved. Розмірність поля 3 біти, та впливаючи із назви поля, зарезервовані
для майбутнього використання та містять нулі [5].

Code Bits. Розмірність поля становить 9 бітів, і кожен біт визначає те як буде
оброблятися протокол [5].

- 106 URG. Важливість (Urgent), вказує, що TCP-сегмент містить важливі дані.
- 107 ACK. Підтвердження (Acknowledge) успішності отримання TCP-сегменту.
- 108 PSH. Просування (Push), також як і прапорець URG, вказує, на пріоритетність TCP-сегменту. Хост-відправник позачергово надсилає цей сегмент даних через IP-мережу.
- 109 RST. Обривання (Reset) вказує, хосту-отримувачу негайно скинути з'єднання без подальшої взаємодії.
- 110 SYN. Синхронізація (Synchronize) використовується для встановлення з'єднання між хостами при так званому триходовому рукошесті (three-way handshake).
- 111 FIN. Фініш (Finish) вказує на завершення з'єднання.

Window. Розмірність поля 16 бітів та використовується для визначання кількості байтів даних, які відправник може надіслати до того, як отримає підтвердження (запит на новий сегмент) від хоста-отримувача [5].

Checksum. Перевірка цілісності даних, розмірність 16 бітів [5].

Urgent pointer. Поле, яке вказує, де закінчуються важливі дані у TCP сегменті. Активне тільки при встановленні прапорці URG. Розмірність Urgent pointer – 16 біт [5].

Options. Не обов'язкове поле розмірністю до 320 бітів, але тут залежить від певних значень цього поля які кратні 1 байту (8 біт). Кожна опція позначається як номер – 1 байт, довжина 1 байт та дані, розмір яких залежить від значення довжини [5].

- **MSS.** Номер 2, довжина 4, визначає максимальний розмір поля *Дані* в TCP-сегменті, інакше кажучи, це кількість даних які можуть бути поміщені в один сегмент при їх передаванні між хостами.
- **Window scale.** Номер 3, довжина 3, максимальне значення опції 14, слугує для збільшення значення TCP-вікна;

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

- **SACK.** Номер 2, довжина 4 – може і змінюватись, в залежності від кількості SACK – блоків. Функція SACK допомагає відправнику виявити «прогалини» в буфері одержувача. Якщо одержувач не отримує пакети в повному обсязі, це означає, що в буфері є дірки (пропущені точки). SACK допомагає відправнику дізнатися про ці дірки в буфері одержувача на ранній стадії.
- **Timestamps.** Номер 8, довжина 10. Містить два поля по 4 байти – мітка часу та ехо-відповідь мітки часу. Використовується для обчислення затримок у мережі.

Padding. Радше за все, це вирівнювання, для кратності заголовка TCP.

Протоколи, які використовують TCP:

- FTP (File Transfer System, протокол передавання файлів);
- HTTP/HTTPS (Hyper Text Transfer Protocol (Secure), для відображення гіпертексту сторінки);
- SSH (Secure Shell, безпечна оболонка, використовується для управління віддаленим доступом);
- BGP (Border Gateway Protocol, протокол граничного шлюзу, використовується для маршрутизації).

Оскільки більшість протоколів типу надійного з'єднання для нашого проекту знижують пропускну здатність, а наш проект це відеогра, тому було вирішено обрати протокол UDP як основу, реалізувавши при цьому певні задачі, як наприклад захист даних та надійність, оскільки TCP не підходить для онлайн відеоігор.

1.3 Мережева архітектура

У попередніх пунктах ми розглянули загальне поняття мережевого протоколу, та які пролеми виникають при конструюванні власного протоколу, з нуля або поверх іншого. Окрім визначення протоколів, важливо зрозуміти те, які технології використовуються найкраще для побудови мережі, який вид мережевої архітектури для цього використовується.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

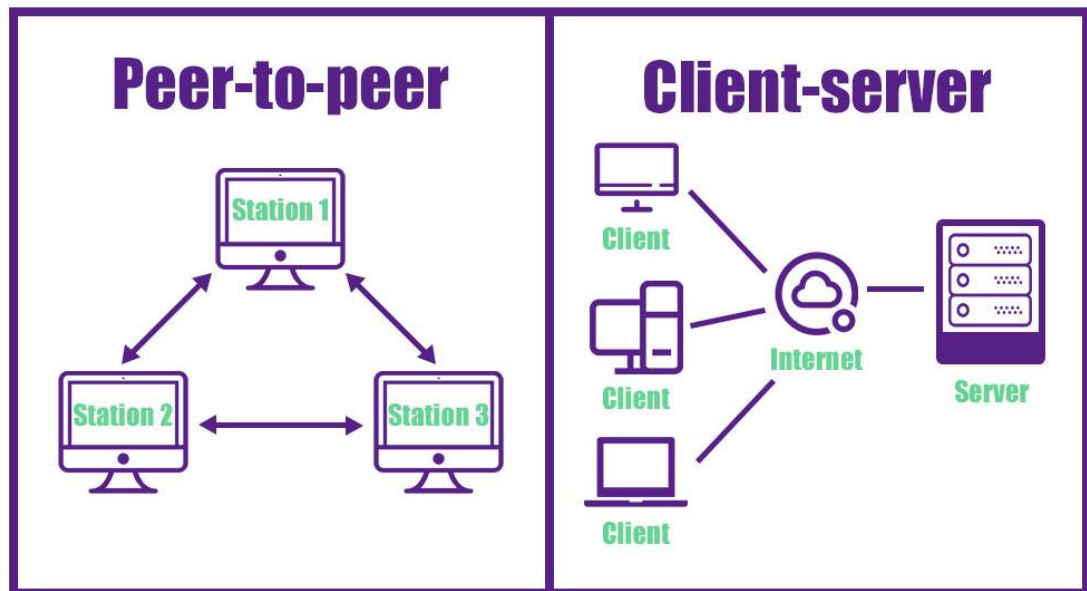


Рисунок. 1.3.1 – Види мережевої архітектури

В комп'ютерних мережах зазвичай використовуються два популярних підходів до побудови [1] – це мережа клієнт-сервер (client-server) та однорангова мережа (peer-to-peer), рис. 1.3.1. Для детальнішого розуміння, як це працює, розглянемо кожну з них та виділимо основні моменти таких підходів.

Клієнт-сервер. Найбільш вживана технологія при побудові мереж. Основне визначення базується на тому, що у нас є потужний пристрій, які обробляє необхідну інформацію – сервер та клієнти, зазвичай малопотужні, які підключаються до цього сервера [4]. Найбільш вживаним прикладом такої моделі є вебсайт, де сервер виступає в ролі веб сайту, та видає клієнтам на їхні запити веб сторінки у вигляді гіпертексту. Клієнтом в даному прикладі може слугувати комп'ютер, смартфон який має доступ до інтернету через веб браузер. Як видно на рисунку 1.3.1 завжди можна виділити мінімум два процеси – це один сервер та один клієнт [4]. Зазвичай це базова структура такого типу мережі, і якщо дивитися на комунікацію між клієнтом та сервером, по суті вони обмінюються двома типами повідомлень, це **запит** та **відповідь**. Повідомлення типу «запит» завжди відправляє клієнт на сервер та очікує відповідь від нього. Це можна навіть порівняти з тим же вебсайтом, де клієнт надсилає запит перейти по посиланню, натискаючи кнопку. Сервер цей запит прийняв, обробив та відправив відповідь клієнту – його бажану сторінку на цьому сайті.

Перевага реалізації серверів як прикладних програм полягає в тому, що сервер може виконуватися на будь-якій обчислювальній системі, яка підтримує ТСП/ІР-зв'язок [4]. Коли навантаження на сервер зростає, його можна запустити на швидшому процесорі. Існують технології, які дозволяють реплікувати сервер на декількох фізичних комп'ютерах для підвищення надійності або продуктивності - вхідні запити розподіляються між комп'ютерами, на яких працює сервер, щоб зменшити навантаження. Якщо основним призначенням комп'ютера є підтримка певної серверної програми, то термін «сервер» може застосовуватися як до комп'ютера, так і до серверної програми.

Однак тут є певні особливості такі як навантаженість сервера. Клієнт залежний від з'єднання серверу, тому повідомлення приходять з певною затримкою, якщо ще й врахувати погане з'єднання клієнта до мережі. В основному це стосується кібератак за типом DoS/DDoS, де можуть перенавантажити сервер, що призведе до погіршення доступу клієнтів до нього, а то і зовсім змусить відключитися від мережі.

Також сервери в більшості випадків енергозатратні та дорогі в обслуговуванні, особливо якщо це стосується веб серверів чи сервісів послуг. При обслуговуванні серверу потрібно мати досить високі навички адміністрування мережі, оскільки від сервера залежить робота клієнтів [4].

Варто зазначити також, що сервером не обов'язково повинен бути потужний пристрій. Це може бути і звичайний комп'ютер, як наприклад Windows Server з Active Directory, служби каталогів для користувачів організації, чи, наприклад ігровий сервер, де гравці у ролі клієнта підключаються до цього серверу. Та навіть більше, сервером може виступати і проста консольна програма, запущена на комп'ютері, та клієнти, які працюють на локальному хості.

Однорангова мережа. Ще один популярний вид архітектури мережі. Його ще називають P2P, скорочено від peer-to-peer. Основна концепція такого підходу – це відсутність централізованого серверу, де користувачі виступають у ролі як клієнта, так і серверу, безпосередньо обмінюючись ресурсами і даними з іншими вузлами [4]. Така децентралізована структура має високу відмовостійкість і підтримує

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

різні додатки, такі як обмін файлами, розповсюдження контенту та спільна робота. Це можна порівняти зі спільним доступом в Google documents, де кілька користувачів можуть додавати чи змінювати вміст документу. Як бачимо на рисунку 1.3.1, користувачі взаємодіють один з одним, надаючи спільний доступ до ресурсів.

Переваги такого підходу – це швидкість та мала енергозатратність. Мережі P2P забезпечують прямий зв'язок між одноранговими користувачами, усуваючи потребу в посередниках, як наприклад у клієнт-сервері [4]. Таке пряме спілкування сприяє швидкій доставці контенту, взаємодії в режимі реального часу та ефективному розповсюдженню даних. Оскільки користувачі надають власні ресурси для функціонування мережі, такі як пропускна здатність, сховище та обчислювальна потужність, це підвищує ресурсну швидкість, зменшуючи навантаження на учасників мережі. А це означає, що кількість учасників може бути вищою, ніж, наприклад, у клієнт-серверній моделі.

Також слід зазначити що такий тип мережі потребує менше капіталовкладень та витрат на обслуговування, оскільки вони не матимуть дорогого серверу з великим обсягом даних та споживанням ресурсів як на мережу, так і на залізо, оскільки між користувачами P2P відбувається спільний розподіл ресурсів [4].

Однак варто пам'ятати, що такий тип мережі вразливіший до атак, ніж клієнт-сервер, оскільки може скластись така ситуація, що зловмисник може передавати свої заражені дані іншим користувачам, при цьому інфікуючи мережу. У P2P складно виявити зловмисника при відсутності певної реалізації захисту, хоча і модель клієнт-сервер також може матиме таку проблему, де зловмисник виступатиме в ролі посередника між клієнтом та сервером. Хоч така мережа і витрачає не багато ресурсів, все ж таки P2P складніше розробляти та обслуговувати, ніж client-server. Це зумовлено тим, що рівноправні учасники P2P мають рівні обов'язки, завдання управління мережею, такі як адресація, безпека та оптимізація продуктивності, повинні бути розподілені між учасниками, що вимагає додаткової координації та зусиль.

Найвідомішим прикладом використання P2P мережі є протокол BitTorrent [1]. Його основна ідея наступна: обмін файлами великих розмірів базується на

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				22
Змн.	Арк.	№ докум.	Підпис	Дата		

тому, що клієнти які завантажують його, починають обмінюватися даними між собою одразу, поки файл не завантажено. Як результат, клієнти, які завантажили певну частину файлу, можуть виступати у ролі джерела його розповсюдження. Варто зазначити, що така мережа призначена для роботи не всіх протоколів, особливо – протоколів прикладного рівня. Наприклад, протокол НТТР не призначений для роботи у Р2Р, але теоретично може використовуватися у гібридних видах мережевої архітектури.

Як бачимо, кожен вид мережі має свої переваги та негаразди. Використання таких підходів залежить від певних задач. Наприклад, для нашого додатка (відеогри) ми можемо використати просту клієнт-серверну модель, оскільки за допомогою авторитарного серверу легше визначити, хто з гравців використовує взламаний значення, такі як здоров'я гравця.

1.4 Базові аспекти створення комп'ютерної гри

У попередніх пунктах ми розглянули теоретичні основи складання протоколів та комп'ютерної мережі. Варто пам'ятати, для яких задач ми будемо використовувати мережну модель, архітектуру чи який протокол ми маємо створити для таких задач. Оскільки наш проект – це комп'ютерна гра, буде дещо несправедливо не розглядати базові реалізації такого додатка. Наша задача – це створити надійний мережевий протокол та підв'язати його до відеогри. Для цього ми могли б використати будь-який ігровий двигун, але для нашого власного протоколу ми використаємо власний двигун та відеогру, з огляду на надійність та безпеку. Тут ми розглянемо базові етапи створенню технічної частини відеогри.

Комп'ютерна гра – комп'ютерна програма, електронна гра, що відбувається за допомогою електронних пристроїв [11], програмованих чи непрограмованих, які утворюють інтерактивну систему. Основний алгоритм роботи відеогри [13] зображений на рисунку 1.4.1.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				23
Змн.	Арк.	№ докум.	Підпис	Дата		

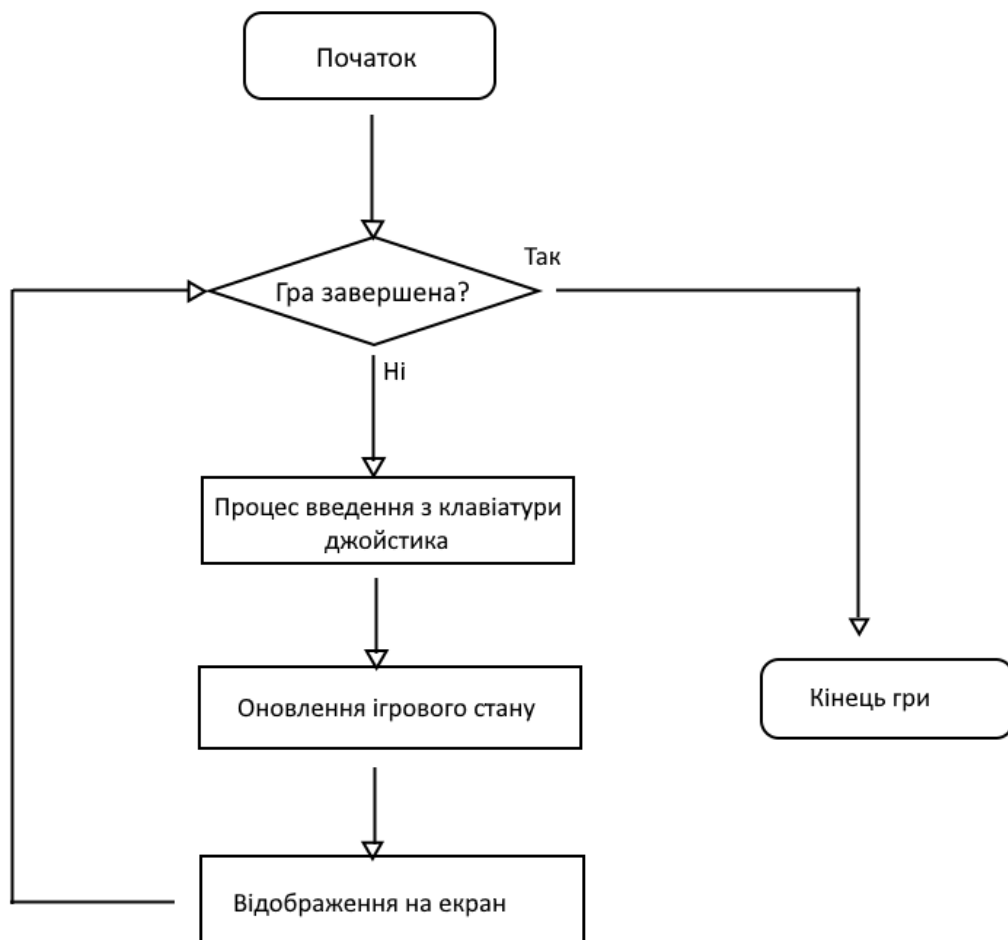


Рисунок 1.4.1 – Стандартний ігровий цикл

Для розробки таких типів продуктів враховують наступне:

- вибір мови програмування;
- вибір ОС, платформи;
- якщо гра онлайн, то вибір типу мережі – однорангова, клієнт-сервер;
- вибір бібліотек для розробки, таких як OpenGL (графіка), OpenAL (звук), Bullet Engine (симуляція фізики) тощо;
- вибір ігрового двигуна, але це за умови, якщо гра не розробляється з нуля.

Мова програмування. Теоретично, будь якою мовою програмування можна реалізувати відеогру, тут залежностей не має. Однак слід пам'ятати, що для певних жанрів чи платформ, специфіка мов програмування може не гарантувати стабільну швидкість чи правильність роботи. Наприклад, для мобільних відеоігор переважно використовують Java чи C#, для персональних комп'ютерів – C/C++ та інше.

Операційна система. Тут варто зазначити наступне – для певних операційних систем підхід буде дещо іншим, якщо відеогра буде використовувати системні функції. Якщо гра націлена на різні операційні системи, варто адаптувати код для них усіх. Те саме стосується й апаратних платформ, на яких вони працюють, особливо коли йдеться про розробку відеоігор для консолей чи смартфонів Apple. Також важливо не забувати про порядок байтів у процесорах.

Сторонні бібліотеки. Цей пункт буде актуальний в тому випадку, коли відеогра розробляється з нуля, тобто не використовує ігрові двигуни. Наприклад бібліотека графіки. На сьогоднішній день, найбільш вживаними є бібліотеки Vulkan API та DirectX. Також існує специфікація OpenGL, яка вже вважається устарілою, проте в певних цілях може використовуватися. Можна бібліотеки створювати практично з нуля, але на це необхідне велике фінансування і людина чи компанія, які підуть на пов'язані з цим ризики.

Ігрові двигуни. Ігровий рушій – це по суті власне середовище розробки для відеоігор. Іншими словами, це програмний рушій, який відповідає за функціональність гри, вирішує задачі з багатоплатформеністю, продуктивністю, звуком, графікою та основними компонентами гри. Прикладами таких рушіїв є Unreal Engine, Unity Engine, Source Engine, Serious Engine та інші.

Мультиплеєр. При розробці багатокористувацьких ігор потрібно продумати мережеву архітектуру цієї гри. Визначити мережеву архітектуру гри, чи це клієнт-серверна гра чи однорангова, оптимізувати навантаження на мережу та залізо, а також забезпечити безпеку та комунікацію між гравцями [11]. Приклади онлайн ігор: Quake III Arena, Unreal Tournament, Counter-strike, GTA Online та інші.

Висновки до першого розділу

У першому розділі було проведено загальне визначення мережевих протоколів та проблем, які виникають при розробці власного протоколу. Визначено, які є мережеві моделі та архітектури, а також розглянуто базові поняття розробки відеоігор.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

З'ясовано, що мережевий протокол – це набір правил про те, як дані будуть формуватися та оброблятися. Протоколи бувають декількох рівнів мережевої моделі OSI, від фізичного (Ethernet, наприклад), до прикладних рівнів – HTTP чи DNS. Загалом виділяють три моделі: це модель OSI, яка має 7 рівнів, модель TCP/IP (4 рівнів), та гібридна 5-ти рівнева модель. Важливо зазначити, що модель OSI так і не була повною мірою реалізована через свою будову, а модель TCP/IP досить погано описує будь-який стек протоколів, крім TCP/IP.

Також виокремлено деякі проблеми при створення користувацьких протоколів. Серед них можна виділили наступне: проблема надійності, проблема сумісності, проблема безпеки та проблема продуктивності. Також розглянуто два протоколи транспортного рівня – протокол UDP та протокол TCP. Головна відмінність полягає в тому, що UDP працює без встановлення з'єднання, в той час як TCP працює навпаки – зі встановленим з'єднанням. Оскільки наш проєкт – це відеогра, у якій важлива швидкість, за основу для розробки ми взяли протокол UDP, що він працює швидше за протокол TCP.

Також у цьому розділі ми виділили два популярних типи мережі – це тип «клієнт-сервер» та однорангова мережа (peer-to-peer). Ідея клієнт-серверу ґрунтується на тому, що клієнт відправляє запит серверу, сервер відправляє клієнту відповідь. В одноранговій мережі навпаки, замість того, щоб мати чітко визначених сервера та клієнтів, хости поділяють свої ресурси на мережу, таким чином вони можуть виступати в ролі сервера та клієнта. Перевагами однорангової моделі є економія ресурсів та висока швидкість, але для такого типу мережі важко забезпечити підтримку та безпеку. Перевагами мережі типу «клієнт-сервер» надійність, безпека та простота реалізації, але такий тип дорогий в обслуговуванні та енергозатратний. Для нашого проєкту була обрана саме клієнт-серверна модель, оскільки у ній легше знайти клієнтів, які використовують обман, а також простота в реалізації, та й для демонстраційної гри потужний сервер не потрібен.

Також ми розглянули базові принципи реалізації нашого додатку, такі як оптимізація, вибір мови програмування, вибір бібліотек та інших компонентів коду.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

Так як ми будуємо власний протокол для операційної системи Windows, ми використаємо мову програмування C++, яка підтримує ООП, та бібліотеки для графіки, введення з пристроїв, а також бібліотеку для виявлення зіткнень. Для мережі ми використаємо системну бібліотеку WinSock.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				27
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2 РОЗРОБКА МЕРЕЖЕВОГО ПРОТОКОЛУ ТА ІГРОВОГО РУШІЯ

2.1 Розробка мережевого протоколу

Проаналізувавши основні відомості щодо проектування протоколів, їх визначення ми можемо розпочинати над його реалізацією. Оскільки створення протоколів з нуля, особливо для такого додатка, як гри призведе до блокування у мережі, то потрібне додаткове налаштування системи. Тому ми реалізуємо поверх існуючих протоколів, а саме протоколу транспортного рівня UDP, обраному завдяки своїй швидкості. Протокол UDP за своєю природою ненадійний, датаграми у ньому можуть приходити не послідовно або взагалі втрачатися, і немає ніякої гарантії з боку двох хостів, чи вони отримали дані – UDP не реалізує ACK, на відміну від протоколу TCP. Для такого додатка, як гра, повна надійність може бути зайвою, окрім тих випадків, коли ми дійсно намагаємось підключитися до сервера.

Основна інформація, яка передаватиметься за розроблювальним протоколом – це команди. Команда задає те, що потрібно зробити програмі при отриманні даних. Наприклад команда «вивести текст» гарантує, що текст, який ми надішлемо буде оприлюднено на всіх хостах. Ключовою особливістю команди є те, що вона формує певний протокольний блок даних перед надсиленням даних на хост (табл. 2.0).

Таблиця 2.1.0 – Структура PDU протоколу ITNP

Назва поля	Розмірність (біт)	Опис
Одиниця команди	8	Команда, яка формує заголовок та роботу програми
Бітове поле опцій	16	Опції для оброблення та визначення типу протоколу
Розмір даних у байтах	16	Розмір записаних даних після заголовка протокола
Цілісність даних	32	Перевірка на цілісність даних
Номер послідовності	32	Використовується для упорядкування пакетів, а також для запобігання дублікатів

Таблиця 2.1.0 – Структура PDU протоколу ITNP (Продовження)

Час створення у мілі-секундах, формат UTC	64	Час створення заголовку у форматі UTC
---	----	---------------------------------------

Тепер розглянемо основні поля, які визначають роботу нашого мережевого додатка.

Одиниця команди. Основа протоколу. Визначає те, як будуть оброблятися дані перед надсиланням чи після отримання, а також виконує роботу над самою програмою. Максимальна кількість команд становить 256, де перші 6 команд зарезервовані. Додавання нових команд потрібно проводити з кінця останнього номеру команди, оскільки програма може дати збій при роботі. Нижче наведено основні команди для управління на рівні комунікації.

- **NET_CMD_BAD.** В основному для обробки помилок при втраті PDU чи їх валідації.
- **NET_CMD_NONE.** Команда нічого не задає, формує стандартний заголовок, на роботу програми не впливає.
- **NET_CMD_CONNECT.** Команда діє як запит серверу на приєднання. Формує заголовок з прапором NET_UDP_RELIABLE, що визначає PDU є надійним і клієнт буде чекати від сервера підтвердження.
- **NET_CMD_DISCONNECT_FROM_SERVER.** Команда задається, коли клієнт від'єднатися від серверу. Сервер в той час видаляє з бази відключеного клієнта.
- **NET_CMD_STOP_SERVER.** Команда для вимкнення серверу. Сервер надсилає команду усіх під'єднаним хостам, де ті в свою чергу вимикаються.
- **NET_CMD_SEND_TEXT.** Команда використовується для надсилання тексту. Варто звернути увагу, що за замовчуванням текст піддається шифруванню.

Бітове поле опцій. Тут кожен біт визначає те, яким чином дані обробляти. Чимось схоже з протоколом TCP з його полем опції. Але в даному випадку кожен

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				29
Змн.	Арк.	№ докум.	Підпис	Дата		

біт встановлює те, що потрібно зробити з даними. Зазвичай команда встановлює бітове поле перед надсиланням пакету даних. Нижче наведено список зарезервованих 8 бітів (максимальна можлива кількість становить 16).

- **NET_UDP_RELIABLE.** Встановлює, чи наш PDU надійний. Якщо так, програма, яка його надіслала, повина отримати підтвердження, що інформація надіслана успішно.
- **NET_UDP_UNRELIABLE.** Встановлює, чи PDU ненадійний. В даному випадку програмі не важливо знати чи пакет прийшов чи ні. Використовується найчастіше.
- **NET_UDP_USE_ZLIB_COMPRESSION.** Встановлює стиснення даних за допомогою ZLIB алгоритму.
- **NET_UDP_USE_ZSTD_COMPRESSION.** Встановлює стиснення даних за допомогою алгоритму ZSTD.
- **NET_UDP_USE_CRYPT_AES256.** Встановлює шифрування даних за допомогою алгоритму AES256.
- **NET_UDP_USE_CRYPT_OWN.** Встановлює шифрування за алгоритмом, визначеним у програмі. Якщо такий прапор встановлений, то потрібно, щоб клієнт та сервер мали спільний алгоритм, інакше дані не будуть належним чином прочитані.
- **NET_UDP_USE_CRC.** Використання цілісності даних. В основному воно не використовується, але його можна завжди увімкнути.
- **NET_UDP_ACK.** Встановлює біт, чи пакет надісланий успішно.

Розмір даних у байтах. Встановлює кількість даних, який слідом йде після заголовка, навідміну від UDP, де поле розмір даних охоплює і заголовок UDP. Оскільки у нас поле розмірністю 16 біт, теоретично максимальний розмір поля може становити 65535, однак UDP, та наш протокол не підтримує фрагментацію даних, якщо значення перевищить MTU. Про це ми обговорювали у першому розділі розглядаючи проблему надійності. У даному додатку розмір поля не перевищує 1441

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				30
Змн.	Арк.	№ докум.	Підпис	Дата		

байтів, не враховуючи розмір заголовка нашого протоколу (21 байт), UDP (8 байтів) та IPv4 (20 байтів).

Цілісність даних. Зазвичай це поле може використовуватися для того, чи дані не були пошкоджені в процесі комунікації між клієнтом чи сервером. У звичайних налаштуваннях воно відключене, але його можна завжди увімкнути, встановивши у налаштуваннях мережі, водночас це може негативно вплинути на швидкість роботи. Також поле використовується при підключенні клієнта до серверу, де клієнт надсилає хеш файлів і даних сесії, необхідних для перевірки того, чи клієнт не взломаний.

Номер послідовності. Тут найцікавіше. Це поле використовується для упорядкування PDU та захисту від дублювання, оскільки ми розробляємо протокол поверх UDP, який працює без встановлення з'єднання та з непередбачуваною швидкістю. На стороні сервера, при підключенні та появі першого пакету на запит приєднання, сервер зберігає номер послідовності кожного, хто йому надіслав дані. Якщо номер менший за попередній чи однаковий, ми ігноруємо даний PDU. В іншому разі ми приймаємо дані, обробляємо та зберігаємо номер послідовності для наступних порівнянь. Те саме відбувається на стороні клієнта, де клієнт зберігає останній номер послідовності серверу.

Час створення. Це поле визначає, коли сформувався основний PDU перед відправкою. Час поданий у мілісекундах за форматом UTC (Coordinated Universal Time), загальний формат часу для годинників, його також використовує протокол NTP для синхронізації часу. Поле необхідне для визначення того, наскільки довго дані надсилалися між хостами, а також для визначення затримки у мережі.

Однак варто пам'ятати, що у нашій програмі протокол використовує рівень комунікації, тобто тут відбувається процес формування даних для відправки чи прийому повідомлень. Наш додаток спілкується з хостами в основному через команди. Протокол формує дані для відправки через мережу. Але протокол призначений для передавання заданих команд, яких йому задали і важливо цю команду з даними згрупувати так, щоб протокол знав як йому поводитись з цими даними. Для

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				31
Змн.	Арк.	№ докум.	Підпис	Дата		

цього у нас є спеціальний клас між комунікацією рівнів PDU та рівнів повідомлень.

Нижче наведено фрагмент коду:

```
class CNetMessage
{
public:

    unsigned int msg_uiNumData;           // num data of network message
    unsigned int msg_uiMaxData;           // max data of network message
    unsigned int msg_uiCurrentPosition;    // current position of network message
    unsigned int msg_uiLastPosition;      // last positions (uses for rewind)

    byte* msg_data;                       // main data in bytes
    byte  msg_netCMD;                     // network command

public:
    CNetMessage(void);
    CNetMessage(size_t numData);
    ~CNetMessage(void);

    inline void Clear();

    inline void InitMessage(size_t numData);

    inline void setData(byte* data, size_t numData);

    inline void copyData(byte* data, size_t numData);

    inline void setCMD(byte cmd) { msg_netCMD = cmd; }

    /***** Writing methods *****/

    inline void write8(const byte &b);
    inline void write16(const unsigned short &s);
    inline void write32(const unsigned int &i);
    inline void write64(const unsigned long long &ll);

    inline void writeFloat(const float &f);
    inline void writeDouble(const double &d);
    inline void writeString(std::string& str);
    inline void writeBlock(byte* data, size_t num);
    inline void writeVector(Vector3D& v);

    /***** Reading methods *****/

    inline void read8(byte* b);
    inline void read16(unsigned short* s);
    inline void read32(unsigned int* i);
    inline void read64(unsigned long long* ll);
    inline void readFloat(float* f);
    inline void readDouble(double* d);
    inline void readString(std::string& str);
    inline void readBlock(byte* data, size_t num);
    inline void readVector(Vector3D* v);

    /***** I/O methods *****/

    inline void rewindMSG(void);
    inline void setPos(const unsigned int& newPos);
    inline byte* getData(void);

    inline unsigned int getLength() { return msg_uiNumData; }
```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				32
Змн.	Арк.	№ докум.	Підпис	Дата		

```
};
CNetMessage operator=(CNetMessage other);
```

Даний клас використовується для запису та читання даних, представлених у послідовності байтів, де примітивні типи даних ми формуємо у вигляді масиву байтів, все інше ми заповнюємо нулями. Розмір даних не повинен перевищувати загальний розмір для надсилання, включно з заголовком ITNP, UDP та IP. Основні поля:

- msg_uiNumData – кількість записаних даних;
- msg_uiMaxData – максимальний можливий розмір даних;
- msg_uiCurrentPosition – індекс на наступний байт для зчитування чи запис у масиву;
- msg_uiLastPosition – індекс на останню позицію байта, переважно використовується для зчитування на початок масиву;
- msg_data – байтовий масив;
- msg_netCMD – команда для формування протоколу.

Реалізацію методів цього класу можна глянути в додатках (див. Додаток «А»). Схематичне подання розподілу функцій за рівнями наведено на рисунку 2.1.0.

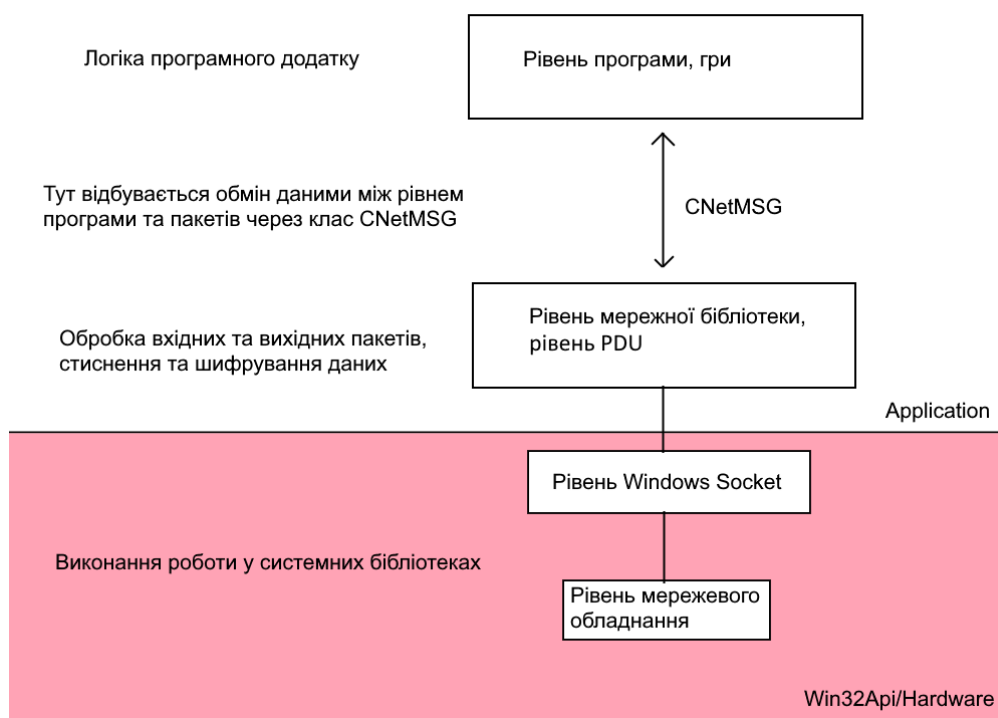


Рисунок 2.1.0 – Рівні мережевого додатка (в нашому випадку мережева гра)

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				33
Змн.	Арк.	№ докум.	Підпис	Дата		

Розглянемо простий алгоритм надсилання даних на стороні клієнта.

На початку виконується певна логіка програми – це рівень програми, потім ми створюємо клас повідомлень та команд, задаємо певну команду та збираємо всі необхідні дані у вигляді байтів, за потреби можемо їх стиснути. Це рівень повідомлень, інакше кажучи, цей рівень слугує як комунікація між рівнем PDU та рівнем програми, а рівень PDU та мережевої бібліотеки вже формує дані для відправки, на основі команди та записаних даних. Нижчі рівні уже займаються безпосередньо формуванням заголовка IPv4 та надсиланням через мережевий адаптер.

2.2 Розробка мережевого коду

Для повноцінного функціонування, нам потрібно створити підпрограму для комунікації між хостами, на основі системних мережевих бібліотек. Варто також звернути увагу на те, що ми використовуємо архітектуру клієнт-сервер, хоча теоретично наш протокол може діяти і для архітектури P2P.

Оскільки у нас операційна система сімейства Windows, нам потрібно використовувати відповідно сокети з бібліотеки WS2_32.dll. Мережевий сокет – це програмний інтерфейс (API) для забезпечення обміну даними між процесами, зазвичай для мережі [15]. Існують три типи сокетів, а саме: сокети без з'єднання (наш протокол, оскільки він створений поверх UDP), сокети, орієнтовані на з'єднання (TCP протокол), та не оброблені сокети (без протоколу транспортного рівня) [10]. Для сокетів з нуля ми могли б використати необроблені сокети, але в такому разі у нас виникає кілька проблем. Докладніше це описано у попередньому розділі.

Тепер для реалізації нашого протоколу ми створимо спеціальні структури, функції для клієнтської частини та серверної, а також забезпечимо між ними зв'язок та засоби захисту даних. На рисунку 2.2.0 зображено певні функції для сторони клієнта та сервера, а також спільні функції на обох сторонах. Функції частини серверу та клієнта дещо схожі, але мають свої певні відмінності. Зі спільних функцій у нас є комунікація, шифрування, стиснення, а також визначенням того, чи наш хост виступає у ролі серверу.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				34
Змн.	Арк.	№ докум.	Підпис	Дата		

Server

```
void startServer()
void closeServer()
void sendToClient(CNetMSG*, int)
void listenClients(CNetMSGHolder*)
```

Client

```
void startClient()
void closeClient()
void sendToServer(CNetMSG*)
void getFromServer(CNetMSG*)
```

```
bool isServer()
void sendText(const char*)
void initialiazeProtection()
bool encryptMessage(CNetMSG*)
bool decryptMessage(CNetMSG*, int i = 0)
void compressData(CNetMSG*)
void decompressData(CNetMSG*)
```

Рисунок 2.2.0 – Мережева частина для додатку

На початку ми обнуляємо дані серверу та створюємо буфер даних, який відповідає розміру, з урахуванням заголовків мережевих протоколів:

```
if (net_iNumClients > NET_MAX_CLIENTS || net_iNumClients <= 0)
{
    Throw("Trying to set clients %d, but max clients are %d", net_iNumClients,
NET_MAX_CLIENTS);
}

net_uiSequenceTick = 0;

memset(&netDataGlobals, 0, sizeof(struct NetData));

Printf("Start initialize server...\n");

pubSendingData = (byte*)malloc(NET_MAX_DATASEND);

if (pubSendingData == NULL)
{
    Throw("Cannot create buffer data for sending msg");
}

memset(pubSendingData, 0, NET_MAX_DATASEND);
```

netDataGlobals – структура, яка містить мережевий сокет дані клієнтів, адресу сервера, а також визначення чи успішно створений сокет. Якщо ми виступаємо у ролі клієнта, дані клієнтів обнуляються. Нижче наведено псевдокод структури для мережі та структури для клієнта:

```
struct NetClient
{
    sockaddr_in clientsAddr;    // address from client
    int clientsSize;           // size of client
```

```

        unsigned int lastSeqNumber; // last sequNumber of client

        unsigned long long time;    // time
    };
    struct NetData
    {
        SOCKET nd_socket;           // main socket
        sockaddr_in serverAddr;     // server address

        struct NetClient clients[NET_MAX_CLIENTS]; // client on server, active only
on server side

        unsigned int uiLastTick; // last sequence id

        bool bOpenAsServer; // check if we server
        bool bOpened;       // check if we open socket
    };

```

pubSendingData – буфер даних, байтовий масив, розміром відповідним, щоб не перевищувало MTU Ethernet. У буфер записується заголовок та оброблені дані та надсилається клієнту чи серверу. Дані, які ми не записуємо – заповнюються нулями.

Слідом за цим ми ініціалізуємо структуру сокетів Windows, необхідну для ініціалізації певних мережових структур, потрібних при підготовці до мережі, та встановлюємо встановлюємо власне сам сокет:

```

WSADATA wsaData;

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
{
    Throw("Cannot open winsock");
}

PrintF("Winsock opened\n");
netDataGlobals.nd_socket = socket(AF_INET, SOCK_DGRAM, 0);
if (netDataGlobals.nd_socket == INVALID_SOCKET)
{
    WSACleanup();
    Throw("Socket creation failed: %s\n", GetLastError());
}
u_long notBlock = 1;

if (ioctlsocket(netDataGlobals.nd_socket, FIONBIO, &notBlock) == SOCKET_ERROR)
{
    WSACleanup();
    Throw("Failed to setup nonblocking socket: %s\n", GetLastError());
}

```

Після встановлення сокету, ми налаштовуємо адресу серверу, прив'язуємо наш мережовий сокет до нашої адреси та визначаємо кількість клієнтів для поточного сеансу:

```

netDataGlobals.serverAddr.sin_family = AF_INET;
#ifdef NET_ENABLE_LAN

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				36
Змн.	Арк.	№ докум.	Підпис	Дата		

```

inet_pton(AF_INET, "127.0.0.1", &netDataGlobals.serverAddr.sin_addr);
#else
netDataGlobals.serverAddr.sin_addr.s_addr = INADDR_ANY;
#endif
netDataGlobals.serverAddr.sin_port = htons(NET_PORT);

// step 5 bind socket

if (bind(netDataGlobals.nd_socket, (sockaddr*)&netDataGlobals.serverAddr,
sizeof(netDataGlobals.serverAddr)) == SOCKET_ERROR)
{
    closesocket(netDataGlobals.nd_socket);
    WSACleanup();
    Throw("Failed bind UDP socket %s\n", lastErrorWSA());
}

PrintF("Server was started succesfull\n");

PrintF("-----\n");

netDataGlobals.bOpenAsServer = true;
netDataGlobals.bOpened = true;

memset(&netMainServer, 0, sizeof(struct NetServer));

// in default max clients always 2, where first is server (local client)
netMainServer.ns_numMaxClients = net_iNumClients;

```

З цього моменту сервер вважається запущеним та готовий приймати, обробляти та надсилати повідомлення клієнтам.

Створення клієнта майже ідентичне серверу. Відмінність полягає в тому, що ми вказуємо адресу серверу, необхідну для підключення, а також ми не прив'язуємо сокет до адреси клієнта.

Тепер переходимо до найцікавішої частини – обмін даними між сервером та клієнтами.

Кожен життєвий цикл програми серверу, викликається функція «listenClients», яка в підсумку формує оброблені дані, отримані від клієнтів. На стороні клієнта запускається аналогічна функція, «getFromServer». Алгоритм функції «listenClients» є наступним:

- Заповнити буфер даних нулями.
- Запустити цикл перевірки, чи надсилаються дані з клієнтів.
- Якщо даних немає – йдемо на наступну ітерацію.
- Якщо дані є, перевіряємо, чи підключений такий клієнт, якщо ні – надаємо йому індекс для комунікації.

- Якщо ми маємо підключеного клієнта, встановлюємо йому адресу та знаходимо індекс для подальшої обробки.
- Перевіряємо, чи дані прийшли в правильному порядку. Якщо PDU даних не правильний – ігноруємо його, в іншому випадку ми додаємо для наступних перевірок клієнта.
- Після чого на основі команд та прапорів протоколу ми обробляємо дані.

Повний код функції наведено у додатках.

На стороні клієнта операція відбувається схожим алгоритмом, але менш громіздким. Нижче наведено код для отримання відповіді від серверу:

```
void getFromServer(CNetMSG* msg)
{
    //if (msg == NULL) return;

    ITNP_t* packet = NULL;

    sockaddr_in serverAddrNew;

    int size = sizeof(serverAddrNew);

    memset(pubSendingData, 0, NET_MAX_DATASEND);

    if (recvfrom(netDataGlobals.nd_socket, (char*)pubSendingData,
NET_MAX_DATASEND, 0, (sockaddr*)&serverAddrNew, &size) == SOCKET_ERROR)
    {
        Printf("Cannot get data from server: %s\n", lastErrorWSA());
        return;
    }

    packet = (ITNP_t*)pubSendingData;

    // if we have doubled packets, just ignore it, command of stop server must
be ignored
    if (packet->uiSequenceID <= netDataGlobals.uiLastTick && packet->ubNetCMD !=
NET_CMD_STOP_SERVER)
    {
        msg->InitMessage(15);
        memset(msg->msg_data, 0, sizeof(byte) * 15);
        msg->setCMD(NET_CMD_BAD);
        return;
    }
    else
    {
        netDataGlobals.uiLastTick = packet->uiSequenceID;
    }

    msg->copyData(pubSendingData + sizeof(ITNP_t), packet->uwNumData);
    msg->msg_netCMD = packet->ubNetCMD;
}
```

Єдина відмінність полягає в тому, що визначення за типом команди, полягає відбувається, рівнем вище.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				38
Змн.	Арк.	№ докум.	Підпис	Дата		

Ми розглянули процес отримання відповіді на стороні клієнта та сервера. Як тоді клієнт та сервер надсилають повідомлення один одному? Нижче наведено реалізації коду для серверної частини, так і для клієнтської.

Лістинг коду для серверної частини:

```
void sendToClient(CNetMSG* msg, int iClient)
{
    assert(msg != NULL);
    assert(msg->getData() != NULL);

    ITNP_t* packet = NULL;

    memset(pubSendingData, 0, NET_MAX_DATASEND);
    packet = (ITNP_t*)pubSendingData;
    packet->uiSequenceID = net_uiSequenceTick;
    packet->ubNetCMD = msg->msg_netCMD;
    packet->ullTimeSend = SDL_GetTicks64();
    packet->uiCRC = 0;
    packet->uwNumData = msg->getLength();

    if (packet->ubNetCMD == NET_CMD_SEND_TEXT && net_bCryptText &&
!net_bCryptAllData)
    {
        if (net_ubCryptType == 1)
            packet->uwFlags |= NET_UDP_USE_CRYPT_AES256;
        else if (net_ubCryptType == 2)
            packet->uwFlags |= NET_UDP_USE_CRYPT_OWN;
        else
            packet->uwFlags |= 0;
    }

    if (net_bCryptAllData)
    {
        if (net_ubCryptType == 1)
            packet->uwFlags |= NET_UDP_USE_CRYPT_AES256;
        else if (net_ubCryptType == 2)
            packet->uwFlags |= NET_UDP_USE_CRYPT_OWN;
        else
            packet->uwFlags |= 0;
    }

    memcpy(pubSendingData + sizeof(ITNP_t), msg->getData(), msg->getLength());

    //PrintF("Sending to client[%d]...\n", iClient);

    if (sendto(netDataGlobals.nd_socket, (char*)pubSendingData,
NET_MAX_DATASEND, 0, (sockaddr*)&netDataGlobals.clients[iClient].clientsAddr,
netDataGlobals.clients[iClient].clientsSize) == SOCKET_ERROR)
    {
        PrintF("^2Warning: cannot send to client[%i]: %s\n", iClient,
lastErrorWSA());
    }

}
```

Лістинг коду для клієнтської частини:

```
void sendToServer(CNetMSG* msg)
{
    //if (msg == NULL) return;
    assert(msg != NULL);
```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				39
Змн.	Арк.	№ докум.	Підпис	Дата		

```

if (net_uiSequenceTick == 0xFFFFFFFF) net_uiSequenceTick = 0;

net_uiSequenceTick++;

unsigned int length = msg->getLength() + sizeof(ITNP_t);

if (length > NET_MAX_DATASEND)
{
    Printf("Cannot send data, because size bigger than maximum possible
data\n");
    return;
}

memset(pubSendingData, 0, NET_MAX_DATASEND);

ITNP_t* packet = (ITNP_t*)pubSendingData;

packet->uiSequenceID = net_uiSequenceTick;
packet->ubNetCMD      = msg->msg_netCMD;
packet->ullTimeSend   = SDL_GetTicks64();
packet->uwNumData     = msg->getLength();
packet->uiCRC         = 0;
packet->uwFlags       |= NET_UDP_UNRELIABLE;

// text always encrypted, but you can change this option
if (packet->ubNetCMD == NET_CMD_SEND_TEXT && net_bCryptText &&
!net_bCryptAllData)
{
    if (net_ubCryptType == 1)
        packet->uwFlags |= NET_UDP_USE_CRYPT_AES256;
    else if (net_ubCryptType == 2)
        packet->uwFlags |= NET_UDP_USE_CRYPT_OWN;
    else
        packet->uwFlags |= 0;
}
// you can encrypt all data
if (net_bCryptAllData)
{
    if (net_ubCryptType == 1)
        packet->uwFlags |= NET_UDP_USE_CRYPT_AES256;
    else if (net_ubCryptType == 2)
        packet->uwFlags |= NET_UDP_USE_CRYPT_OWN;
    else
        packet->uwFlags |= 0;
}
// this sends if we have packet unreliable
if (packet->ubNetCMD == NET_CMD_CONNECT)
{
    packet->uwFlags &= ~NET_UDP_UNRELIABLE;
    packet->uwFlags |= NET_UDP_RELIABLE;
}

memcpy(pubSendingData + sizeof(ITNP_t), msg->getData(), length);

if (sendto(netDataGlobals.nd_socket, (const char*)pubSendingData,
NET_MAX_DATASEND, 0, (sockaddr*)&netDataGlobals.serverAddr,
sizeof(netDataGlobals.serverAddr)) == SOCKET_ERROR)
{
    Printf("Cannot send data to server: %s\n", lastErrorWSA());
}

packet = NULL;
}

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				40
Змн.	Арк.	№ докум.	Підпис	Дата		

В цілому, реалізація схожа на обох сторонах. Тут ми на основі повідомлення формуємо заголовок протоколу та надсилаємо дані клієнту, якщо це сервер, або серверу, якщо це клієнт. Як бачимо на основі повідомлень ми формуємо бітове поле для того, щоб клієнтська сторона змогла обробити дані. Варто зазначити, що кожен порядковий номер збільшується на одиницю, у клієнті коли ми надсилаємо серверу дані, у сервері коли ми отримуємо дані. У випадку сервера, це зумовлено тим, що серверу можливо потрібно надіслати дані клієнту, тому може виникнути десинхронізація.

На закінчення нам потрібно завершити роботу клієнта та сервера, після того, як вони завершили свій сеанс. Нижче наведено лістинг коду закриття серверу:

```
void closeServer(void)
{
    if (!netDataGlobals.bOpened)
    {
        return;
    }

    Printf("Close server...\n");

    closesocket(netDataGlobals.nd_socket);

    WSACleanup();

    if (pubSendingData != NULL)
        free(pubSendingData);

    pubSendingData = NULL;

    memset(&netDataGlobals, 0, sizeof(struct NetData));

    Printf("Server closed\n");
}
```

На стороні клієнта код практично повністю ідентичний. Тут головне закрити сокет та очистити ресурси, які ми використали при роботі з Windows Socket API. Наша мережева частина програми готова до використання, залишилось впровадити її у наш додаток, а саме відеогру.

2.3 Розробка гри та впровадження мережевого коду у гру

Після розробки протоколу та мережевої частини, нам потрібно розробити вже саму відеогру.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				41
Змн.	Арк.	№ докум.	Підпис	Дата		

У цій частині буде розглянуто лише деякі ключові моменти реалізації програмного додатка, а також впровадження мережевого коду у гру, для підтримки мультиплеєру.

Для полегшення роботи з системними бібліотеками ОС Windows та швидкості програми, ми використаємо мову програмування C++. На рисунку 2.3.0 зображена архітектура нашого ігрового двигуна



Рисунок 2.3.0 – Архітектура комп'ютерної відеогри

– **Engine.dll**. Основний модуль, який відповідає за графіку, обробку фізики, кешування даних та містить віртуальну консоль для управління додатком. У ньому описуються основні класи для роботи над відеогрою, такі як моделі, текстури, структура рівня та ігрових об'єктів. Саме тут ми впровадимо мережевий код відеогри.

– **Game.dll**. Модуль, який відповідає за ігрову логіку, ігровий сеанс, та мережевий код вищого рівня, для зручності роботи над додатком. Тут описані класи, функції для управління грою. Доступ до класу здійснюється через покажчиків на функції, захованих у динамічній бібліотеці ОС Windows.

– **Power.exe.** Модуль, який відповідає за відображення гри на екран, а також виведення та управління через клавіатуру та мишку. Тут в основному налаштовується гра.

Точку входу у програму наведено нижче:

```
int main(int argc, char** argv)
{
    /***** init window *****/
    WindowInit(&ScreenPower);

    /***** init engine *****/
    InTheEngine_Start(argc, argv, GAME_NAME);

    /***** load game library *****/
    LoadGameLibrary(DLL_NAME);

    /***** parse command line *****/
    ParseCommandLine(argc, argv);

    /***** main loop game *****/
    MainLoop();

    /***** never gets here *****/
    return 0;
}
```

Спочатку ми ініціалізуємо вікно бібліотеки SDL2, яка відповідає за відображення на екран гри та певних її елементів, після чого ми ініціалізуємо модуль ігрового двигуна, з глобальними вказівниками на об'єкти та компіляції шейдерів OpenGL 4.0. Наступна функція завантажує бібліотеку ігрового модуля, в результаті чого ми отримуємо покажчик на структуру з покажчиками на ігровий модуль. Структура наведена нижче:

```
struct Game_s
{
    /* PID process of application */
    int ProcessId;
    /* keyboard key detect */
    int key;
    bool KeyDown, KeyUp;
    /* mouse coordinates */
    int mouseX, mouseY;
    /* check if enable game menu */
    bool enableMenu;
    /* check if we server */
    bool isServer;
    /* states of game */
    enum e_gameStateType gaType;
    /* last drawn time uses for lerping */
    double lastTime;

    void ( * ConMenu )(void);

    /* read key from keyboard */
    void ( * ReadKey )(const float ftime);
    /* initialiaze game instance */
    void (* Initialize )(void);
}
```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				43
Змн.	Арк.	№ докум.	Підпис	Дата		

```

/* begin game */
void ( * RunGame )(void);
/* stop game, paused or change level */
void ( * StopGame)(void);
/* end entire game */
void ( * EndGame )(void);
/* input of game */
void ( * Input)(const float time);
/* loading level data to play world */
void ( * LoadLevel)(const char* name);
/* write game */
void ( * SaveGame )(string_t strFileName);
/* read game data and run game */
bool ( * LoadGame )(const char* strFileName);
/* make screenshot */
void ( * ScreenShot)(void);
/* start new game from level */
bool ( * NewGame )(void);
bool ( * JoinGame)(void);
/* join to game network (starts as client) */
//void ( *JoinToGame )(const char* addr, unsigned short port);
};

```

Тут покажчики на функції, які знаходяться у ігровій бібліотеці. Створення такої структури нам дає суттєву перевагу – модульність. Це означає, що ми зможемо замінити на іншу бібліотеку, з іншими об'єктами, рівнями, типом ігри і тд. Іншими словами це проста реалізація модифікацій гри, що означає зміну ігрового процесу чи оформлення під власні потреби.

На стороні додатка, Power.EXE, ми завантажуюмо динамічну бібліотеку Windows у пам'ять та імпортуємо функцію, для заповнення ігрового модуля. Нижче наведена реалізація:

```

void LoadGameLibrary(const char* strDllName)
{
    PrintF("Loading game library: %s\n", strDllName);
    hGameDLL = LoadLibraryA(strDllName);

    if (hGameDLL == NULL)
    {
        /* error */
        ...
    }

    CREATE_GAME GameCreate = (CREATE_GAME)GetProcAddress(hGameDLL, "GameCreate");

    if (GameCreate == NULL)
    {
        /* error */
        ...
    }

    Game = GameCreate();

    if (Game == NULL)
    {
        /* error */
        ...
    }
}

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				44
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    Printf("Game library: \"%s\" loaded succesfull\n", strDllName);

    Game->Initialiaze();
}

```

В свою чергу, ми викликаємо системну функцію LoadLibraryA для завантаження динамічно зв'язувальної бібліотеки. Про це трохи згодом.

Викликаючи ще одну системну бібліотеку, для імпорту символів з бібліотеки GetProcAddress, ми імпортуємо нашу функцію GameCreate(). Нижче наведено фрагмент коду:

```

/* create game */
Game_t* GameCreate(void)
{
    Game = (Game_t*)malloc(sizeof(struct Game_s));

    if (Game == NULL)
        return NULL;

    Game->ProccessId = 1;
    Game->KeyDown = FALSE;

    ...

    Game->NewGame      = NewGame;
    Game->JoinGame     = JoinGame;

    return Game;
}

```

Наша гра використовує два типи динамічно зв'язувальних бібліотек [12]:

- Бібліотеки, які зв'язуються під час завантаження програми у пам'ять.
- Бібліотеки, які зв'язуються під час виконання програми.

Перший тип використовує Engine.dll, тобто модуль ігрового рушія, оскільки він має експортовані символи, такі як класи, функції чи прості типи даних, які використовує відеогра завжди. Другий тип використовує лише ігровий модуль, задля підтримки модульності.

Модуль ігрового рушія дуже великий за своєю природою, багато чого не буде включено у описи, але основу потрібно обговорити для подальшої роботи та реалізації мережевого коду.

У модулі ігрового рушія загалом було створено 23 класи для загального користування, їхні описи наведено у коментарях:

```

class CBrush;    // Пензлик, випуклий статичний об'єкт
class CConsole;  // Консоль, де зберігаються логи виведені через функцію Printf

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

```

class CConsoleDataBase; // База даних консолі, оболонка для виконання команд
class CDataBase; // База даних для завантаження об'єктів
class CEntityMain; // Головна ігрова сутність
class CFileName; // Клас для роботи з рядками типу файлів
class CFileOBJ; // Імпорт моделей у форматі Wavefront (OBJ)
class CFileStream; // Клас для операції з файлами
class CFont; // Шрифт для виведення на екран
class CGameWorld; // Ігровий світ
class CModel; // Екземпляр 3D моделі
class CModelAnimFrames; // Робота з анімаціями
class CModelObject; // полігональна 3D модель
class COBJDecoder; // Імпорт моделей, старіша версія
class CPhysicsEntity; // Сутність з колізіями та фізикою
class CPolygonWorld; // Полігон для випуклих та опуклих об'єктів
class CRoomBase; // Сектор, або опуклий об'єкт
class CShader; // Шейдер для OpenGL
class CSkyBox; // Небо
class CTexture; // Екземпляр растрового файлу
class CTextureObject; // Растровий файл для використання текстур
class CVertex; // Вершина для об'єктів
class CView; // Головний екран гри

```

Ці класи використовуються двома ігровими модулями, для полегшення роботи, а також доступу до функцій та методів рушія. Для того, щоб використати бібліотеку Engine.dll для повноціної роботи, бо створено імпортує функцію для створення глобальних об'єктів для роботи програми – InTheEngine_Start():

```

void InTheEngine_Start(int argc, char** argv, const char* strGame)
{
    initializeProtection();

    getGlobalAppPath();

    pConsoleMain = new CConsole;
    pConsoleMain->Init(strGame);

    pConsoleDataBase = new CConsoleDataBase;

    ...

    if (!gladLoadGLLoader((GLADloadproc)SDL_GL_GetProcAddress))
    {
        Error("Failed to initialize GLAD\n");
    }

    getGPUDriverInfo();

    ...

    InitGlobals();

    compileBasicShaders();

    ...
}

```

Перед завершенням програми необхідно очистити ресурси, викликаючи відповідну функцію InTheEngine_Shutdown(). Ця функція видаляє з пам'яті завантажені ресурси та звільняє пам'ять глобальних об'єктів.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

Після того, як етап ініціалізації проєктів закінчено, додаток переходить у основний ігровий цикл, де вихід можливий через клавішу «Escape» або команду Quit(). Нижче наведено реалізацію головного ігрового циклу роботи нашого додатка:

```
void MainLoop(void)
{
    // setup view of main canvas game
    pViewport->setUpView(ScreenPower.Wigth, ScreenPower.Height, 0.1f, 1000.0f, 90.f);
    ...

    // print dummy console style menu
    Printf("***** Dummy console type menu *****\n");
    Printf("to create new game, print in console: startGame()\n");
    Printf("to join local host, print in console: joinGame()\n");
    Printf("to search LAN game, print in console: searchLAN()\n");
    Printf("to help for console commands, print: helpConsole()\n");
    ...

    /* main loop game */
    while ( 1 )
    {
        // calc time
        double currentFrame = (double)SDL_GetTicks64() * 0.001;
        deltaTime = currentFrame - lastFrame;
        lastFrame = currentFrame;
        // puts time
        Game->lastTime = deltaTime;

        // input logic from keyboard / mouse
        ProcessInput(deltaTime);
        // render view
        BeginFrame();
        // swap buffers
        SDL_GL_SetSwapInterval(1);
        SDL_GL_SwapWindow(ScreenPower.window);
    }
}
```

Спочатку ми обробляємо вивід з пристроїв введення, після чого ми виконуємо основну логіку програми та рендеримо все на екран. Алгоритм циклу було розглянуто у попередньому розділі.

А тим часом нам потрібно впровадити мережевий код у створену відеогру. Для того, щоб це реалізувати, ми підключимо нашу створену у попередніх пунктах бібліотеку для роботи з нашим протоколом у модуль ігрового рушія. Після чого нам потрібно визначити експортовані символи з DLL, так як у нас це динамічна бібліотека, яка використовується на етапі завантаження програми у пам'ять [12]. При визначенні спеціальної директиви препроцесора символи, такі як функції, глобальні об'єкти, експортуються у бібліотеці, в іншому випадку ми імпортуємо їх з бібліотеки. Нижче наведена реалізація, а також приклад використання:

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

```
#ifdef ENGINE_EXPORTS
#define ENGINE_API __declspec(dllexport)
#else
#define ENGINE_API __declspec(dllimport)
#endif
```

Приклад використання:

```
ENGINE_API void sendToClient(CNetMSG* msg, int iClient);
```

Так як у нас гра має архітектуру клієнт-сервер, ми виконаємо реалізацію обох частин. По суті ми виконуємо мережеву частину поверх теї, яку ми створили у попередньому пункті. Відмінність полягає в тому, що нижній рівень використовується безпосередньо для надсилання даних, верхній рівень стосується вже ігрової сесії.

Серверна частина

Як і у мережевій частині, ігровий модуль теж поділяється на клієнтську частину та серверну. Для реалізацію серверу використаємо такий алгоритм:

- створення серверу;
- запуск серверу;
- виконання серверу;
- закриття серверу.

Меню гри ми створили через власну bash-подібну консоль, у якій ми можемо встановити значення змінних, а також виконати певні команди та функції, записавши їх у консоль. До прикладу, через консоль ми викликаємо функцію startGame():

```
static void startGame()
{
    if (prepareScene) return;

    prepareScene = Game->NewGame();

    bStartGame = prepareScene;
    if (bStartGame)
        Game->enableMenu = false;
}
```

У разі успішності ми перейдемо у ігровий цикл серверу. Game->NewGame() вказує на функцію створення серверу, визначену у DLL. Тепер розглянемо як ми це реалізували:

```
bool NewGame(void)
{
    CFileName FileName = "Data\\Levels\\Multiplayer\\DM1.wo";

    try
```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				48
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{
    serverSetUP(fileName);
}
catch (char* e)
{
    Printf("Cannot create server\n%s\n", e);
    Network::closeServer();
    return false;
}

openConsole = false;

return true;
}

```

Ми завантажуюємо світ, запускаємо сервер та налаштовуємо його. Якщо десь виникає помилка, ми закриваємо сервер на мережевій частині та повідомляємо помилку. Встановлення серверу відбувається наступним чином:

```

void serverSetUP(const CFileName& strFileName)
{
    Network::startServer();

    gameSession.LoadWorld(strFileName);

    renderPrepareWorld(gameSession.getWorld());

    /* add player to world */
    spawnPlayer();

    pConsoleDataBase->addConsoleFunction("ghostObjectsInfo()", ghostObjectsInfo);
    pConsoleDataBase->addConsoleVariable(CVT_BOOL, "sv_bGetMSGSize", &sv_bGetMSGSize);

    createChecksumSession(&net_uiCRC);

    GameHUD::setup();
}

```

Ми викликаємо відому нам функцію, яку ми реалізували у попередніх пунктах, де налаштовуються сокети, завантажуюємо ігровий світ, готуємо для візуалізації, створюємо гравця, додаємо функції та змінні у оболонку консолі, перевіряємо цілісність даних та налаштовуємо інтерфейс. В цілому ми розглядатимемо код, який стосується мережевої частини гри.

Функція createChecksumSession() використовується для того, аби перевірити цілісність даних рівня, для запобігання обману з боку клієнта, які можуть мати змінені версії рівня. Його основна реалізація наведена у додатках.

Після цього у ігровому циклі ми переходимо безпосередньо у цикл серверу. Ідея алгоритму наступна:

- отримати дані від клієнтів;

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

- провести аналіз даних на основі команд;
- оновити ігрову логіку;
- надіслати ігровий стан клієнту;
- рендер світу.

Для отримання даних клієнту скористаємось функцією яку ми реалізували у мережеві частині listenClients(), та через масив повідомлень аналізуємо вихідні дані, отримані з клієнта – це функція serverAnalyseMSG():

```
void serverAnalyseMSG( Network::CNetMSGHolder* holder )
{
    size_t stringLen;
    char strName[128];
    unsigned short s_Id = 0;
    byte color = 0;

    CPlayer* player = nullptr;

    for ( int i = 0 ; i < holder->msgNum; i++ )
    {
        Network::CNetMSG* msg = holder->getMSG(i);

        switch (msg->msg_netCMD)
        {
            case NET_CMD_ADD_PLAYER:

                player = new CPlayer;

                msg->read16(&player->pl_sID);
                msg->read8(&color);
                msg->read32(&stringLen);

                ...

                Printf("Player[%d] \"%s\" joined to us, count %d\n", i, strName,
gameSession.getClientsCount());

                break;
            case NET_CMD_DISCONNECT_FROM_SERVER:

                msg->read16(&s_Id);

                removePlayer(s_Id);

                gameSession.updatePlayerScores();

                break;
            case NET_CMD_ACTION:

                applyClientsActions(msg);

                break;
            case NET_CMD_NONE:

                break;
            default:

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        break;
    }
}

```

У нас з'явилися дві команди NET_CMD_ACTION та NET_CMD_ADD_PLAYER. Перша команда надсилається щоразу як клієнтам, так і серверу і визначає ігровий стан. Друга команда надсилається серверу для додавання підключеного гравця у гру.

Тепер розглянемо безпосередньо функцію applyClientsActions, вона є ключем до того, як гравці управляють грою. Нижче наведено псевдокод:

```

static void applyClientsActions(Network::CNetMessage* msg)
{
    unsigned short id;

    msg->read16(&id);

    auto iter = gameSession.gs_listClientsPlayers.find(id);

    if (iter == gameSession.gs_listClientsPlayers.end()) return;

    CPlayer* player = iter->second;

    player->getFromClient(msg);
}

```

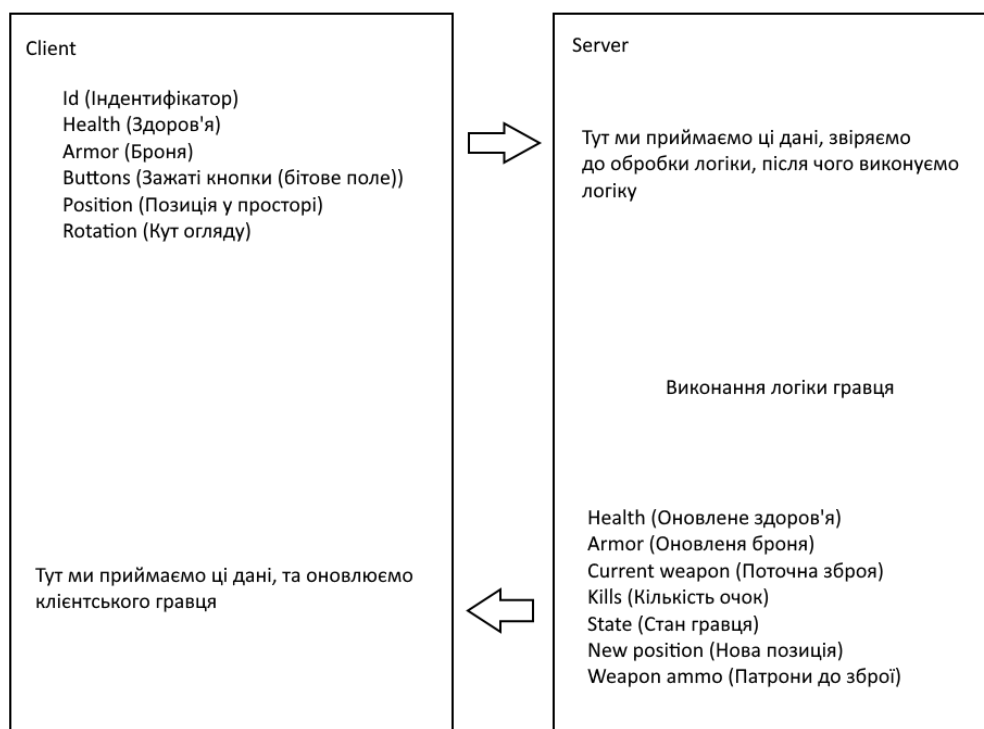


Рисунок 2.3.1 – Схема виконання комунікації гравця

Тут ми зчитуємо ідентифікатор гравця та шукаємо серед клієнтів. Якщо такого гравця не існує, ми пропускаємо обробку функції. В іншому випадку ми прий-

маємо повідомлення і застосовуємо гравцеві з таким індексом. Клас CPlayer представляє ігрову сутність, об'єкт втілює гравця. Схема комунікації для гравців на клієнтській стороні та серверній зображена на рисунку 2.3.1.

На сервері є копія клієнтського гравця, яка в свою чергу виступає у ролі ігрового об'єкта ти виконує логіку, таку як рух чи стрільбу. На стороні клієнта ми просто виступаємо у ролі камери, яку сервер оновлює, включно з параметрами гравця.

На стороні серверу ми маємо:

```
void CPlayer::getFromClient(Network::CNetMessage* msg)
{
    int health, armor;
    Vector3D vPos;

    msg->readBlock((unsigned char*)&health, sizeof(int));
    msg->readBlock((unsigned char*)&armor, sizeof(int));

    if (health != m_iHealth || armor != m_iArmor)
    {
        health = m_iHealth;
        armor = m_iArmor;
    }

    m_iHealth = health;
    m_iArmor = armor;

    msg->read32(&pl_uiPlacedButtons);
    msg->readVector(&vPos);
    msg->readVector(&m_anglRotaion);

    pl_camera->m_front = m_anglRotaion;

    if (vPos == m_vPosition) m_vPosition = vPos;
}
```

Тим часом як на стороні клієнта:

```
void CPlayerClient::sendActionToServer(Network::CNetMessage* msg)
{
    msg->write16(cl_sId);

    msg->writeBlock((byte*)&cl_Health, sizeof(int));
    msg->writeBlock((byte*)&cl_Armor, sizeof(int));

    msg->write32(cl_Buttons);

    msg->writeVector(cl_vPosition);
    msg->writeVector(cl_aRotView);
}
```

Далі ми виконуємо певну ігрову логіку, оновлюємо позиції об'єктів, видалямо сутності, які вже не є дійсними, та розпочинаємо збір даних для надсилання клієнту, де ми реалізували у двох функціях – «writeEntitiesToData» та

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				52
Змн.	Арк.	№ докум.	Підпис	Дата		

«writeToClients». У першій функції ми записуємо ключові поля для надсилення ігрових об'єктів, у наступній ми записуємо параметри гравця та відправляємо клієнтам.

Варто зазначити, що ігрові об'єкти, сутності, ми передаємо їхні хеш-дані ресурсів, позицію, колір та кут огляду, де загалом виходить 22 байта на одну сутність. Процес збору для гравця наведено нижче:

```
for (auto& iter : gameSession.gs_listClientsPlayers)
{
    msgTmp.InitMessage(NET_MAX_BUFFER);

    iter.second->makeForClient(&msgTmp);

    msgTmp.writeBlock(msg->getData(), msg->getLength());
    msgTmp.setCMD(NET_CMD_ACTION);

    if (sv_bGetMSGSize)
    {
        Printf("NETWORK_MSG_SIZE = %d\n", msgTmp.getLength());
        sv_bGetMSGSize = false;
    }

    Network::sendToClient(&msgTmp, iter.second->pl_sID);
}
```

Тут варто звернути увагу на метод «makeForClient» класу гравця. У цьому класі ми записуємо дані для клієнтських гравців, мінімізуючи розмір даних, нижче наведено реалізація класу:

```
void CPlayer::makeForClient(Network::CNetMessage* msg)
{
    msg->writeBlock((byte*) &m_iHealth, sizeof(short));
    msg->writeBlock((byte*) &m_iArmor, sizeof(short));

    byte weapon = pl_ulCurrentWeapon & 0xFF;

    msg->write8(weapon);
    msg->write8(pl_uiNumKills & 0xFF);
    msg->write8(pl_state);

    if (pl_state == PL_STATE_FINISHED)
    {
        msg->writeVector(m_vPosition);
    }
    else
    {
        m_vPosition = getCurrentPosition();
        m_vPosition.y += vCollisionStretch.y * 0.5f;
        msg->writeVector(m_vPosition);
    }

    msg->writeBlock((byte*)&pl_weapons->wp_ammo, sizeof(AmmoWeapon_t));
}
```

Після збору та запису даних для кожного підключеного гравця, ми викликаємо нашу знайому функцію `sendToClient`, формуємо заголовок протокола та надсилаємо дані на клієнтську машину. Після чого йдуть функції, які не стосуються мережевої частини гри. В результаті у нас виходить такий цикл серверу, який оновлюється кожну ітерацію програми:

```
void serverRun(double msc)
{
    Network::CNetMSG msg;
    Network::CNetMSGHolder msgHolder;

    // get message from connected clients
    Network::listenClients( &msgHolder );

    // analyse message from connected clients
    serverAnalyseMSG( &msgHolder );

    // update entity logic
    updateGameLogic(msc, &msg);

    // write data from game simulation to clients
    writeToClients( &msgHolder, &msg );

    CPlayer* pl = gameSession.gs_pePlayer;

    // get view matrix from player camera
    if (Game->gaType == GAME_FINISHED_MATCH)
    {
        pViewPort->m_viewMatrix = pl->pl_camera->getMatrixView();
        pViewPort->m_ViewDir     = pl->pl_camera->m_position;
    }
    else
    {
        pl->getPlayerView(pViewPort);
    }

    // render world room, skybox, brushes
    renderView(&gameSession.gs_world, pViewPort);

    // render entities
    renderEntities(entityManager.em_allServerEntities, pViewPort, pl);

    // if we alive, render weapon
    if (pl->pl_state == PL_STATE_ACTIVE) {
        // render model weapon
        pl->pl_weapons->renderWeapon();
    }

    // draw hud to player
    GameHUD::draw(pViewPort);

    // draw console, if needed
    DrawConsole();
}
```

Ви могли помітити ще один клас, «CNetMSGHolder». Його задача збирати дані, які надійшли з клієнтів, та виконати аналіз на основі команд, шляхом перебору масиву даних.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				54
Змн.	Арк.	№ докум.	Підпис	Дата		

Для того, щоб сервер успішно вимкнувся, він надсилає команду кожному клієнту, повідомляючи, що сеанс гри завершився. Нижче наведено реалізація вимкнення серверу:

```
void serverShutdown()
{
    Network::CNetMSG msg;

    const char* strEndText = "Server shutdown";

    msg.InitMessage(strlen(strEndText) + 1);
    msg.writeBlock((byte*)strEndText, strlen(strEndText));
    msg.setCMD(NET_CMD_STOP_SERVER);

    pConsoleDataBase->deleteConsoleFunction("ghostObjectsInfo()");
    pConsoleDataBase->deleteConsoleVariable("sv_bGetMSGSize");

    for (auto& iter : gameSession.gs_listClientsPlayers)
    {
        Network::sendToClient(&msg, iter.second->pl_sID);
    }

    Network::closeServer();

    /* очищення ресурсів */
}
```

Загалом на сервері ми маємо локального гравця та копії клієнтській гравців. Інакше кажучи, якщо ми виступаємо у ролі серверу, ми виступаємо у ролі адміністратора ігрової сесії, а це означає, що клієнти не матимуть доступ до ігрової сесії, наприклад, використовуючи взломані значення – сервер їх виправить.

Клієнтська частина

Робота клієнтської частини схожа з роботою серверу, але тут на рівні ігрової сесії чи програми, клієнт немає особливої логіки – він виконує роль «камери». Ми розглянемо основну реалізацію даного ігрового клієнта, використовуючи мережеві функції для роботи з нашим протоколом.

Для приєднання до серверу ми вказуємо адресу, після чого викликається функція для підключення до серверу:

```
bool JoinGame(void)
{
    CFileName FileName = "Data\\Levels\\Multiplayer\\DM1.wo";

    try
    {
        clientSetUp(FileName);
        processConnectToServer();
    }
    catch (char* e)
    {
    }
```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        Printf("Cannot join game:\n%s\n", e);
        Network::closeClient();
        return false;
    }

    openConsole = false;
    return true;
}

```

Спочатку ми налаштуємо клієнта, потім підключаємось до серверу. У разі помилки ми отримаємо відповідне повідомлення, та функція поверне недійсне значення щодо невдалого підключення. В іншому разі клієнт буде вважатися підключеним та готовим до гри.

Створення клієнта аналогічне зі сервером, лише викликається функція «startServer». Нижче наведено фрагмент коду реалізації:

```

void clientSetUp(const CFileName& strFileName)
{
    Network::startClient();
    gameSession.LoadWorld(strFileName);
    renderPrepareWorld(&gameSession.gs_world);

    ...

    CFileName m_strTexture = "Data\\Models\\Players\\Test\\Box.te";

    FileSetGlobalPatch(m_strTexture);

    pDataBase->loadModel("Data\\Models\\Players\\Test\\Box.mo");
    pDataBase->loadTexture(m_strTexture);

    createChecksumSession(&net_uiCRC);
}

```

Тепер переходимо до найцікавішого – процес підключення клієнта до серверу. Ідея алгоритму наступна: ми надсилаємо серверу збір CRC, цілісність даних рівня, та запит на підключення. Сервер на своїй стороні звіряє дані. Якщо такі дані не сходяться – сервер відправляє клієнту про помилку та відключає від сесії. Якщо все успішно, сервер надсилає дані ігрової сесії для клієнту, після отримання цих даних клієнт надсилає запит на приєднання гравця до ігрової сесії. З цього моменту клієнт вважається підключеним до серверу та ігрової сесії. Нижче наведено реалізацію алгоритму:

```

void proccessConnectToServer(void)
{
    char string[128];
    size_t sizestring;

    /* first we need connect to server */
    Network::CNetMessage msg;
    msg.InitMessage(64);
}

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				56
Змн.	Арк.	№ докум.	Підпис	Дата		

```

msg.writeBlock(getPublicKey(), 32 * sizeof(byte));
msg.write32(net_uiCRC);
msg.setCMD(NET_CMD_CONNECT);
Network::sendToServer( &msg );
msg.Clear();

/* second we get from server info and add player */

Network::getFromServer(&msg);

if (msg.getData() == nullptr)
    Throw("Cannot get answer from server\n");

if (msg.msg_netCMD == NET_CMD_BAD)
{
    strcpy(string, (const char*)msg.getData());
    Throw("%s", string);
}

gameSession.gs_clPlayer = new CPlayerClient;

msg.read16(&gameSession.gs_clPlayer->cl_sId);
msg.read32(&sizestring);

msg.readBlock((byte*)string, sizestring);

Printf("%s\n", string);

msg.Clear();

msg.InitMessage(1024);

msg.write16(gameSession.gs_clPlayer->cl_sId);
msg.write8(plr_colColor);
msg.write32(strlen(plr_strPlayerName));
msg.writeBlock((byte*)plr_strPlayerName, strlen(plr_strPlayerName));
msg.setCMD(NET_CMD_ADD_PLAYER);

Network::sendToServer( &msg );

/* done */

GameHUD::setup();
}

```

У разі вдалого підключення, ми переходимо безпосередньо у цикл клієнта. За своєю структурою він дещо схожий з сервером, але є певні відмінності, наприклад, при аналізі повідомлень, отриманих від серверу. Розглянемо процес аналізу і отримання відповіді від серверу:

```

void clientGetMsg(Network::CNetMessage* msg)
{
    Network::getFromServer(msg);

    // kill client if server has stopped match
    if (msg->msg_netCMD == NET_CMD_STOP_SERVER) clientDisconnect(msg);

    // if we change table on server side, just read structure and get action from
server
    if (msg->msg_netCMD == NET_CMD_EVENT_CHANGE_SCORE)
    {

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

```

* 8);
    msg->readBlock((byte*)&gameSession.gs_playerScore[0], sizeof(playerScore_t)
    Network::getFromServer(msg);
}

if (msg->msg_netCMD != NET_CMD_ACTION) return;

clientClear();

// first we get info for player
CPlayerClient* cl = gameSession.getClientPlayer();

cl->applyActions(msg);

unsigned int iModelCRC, iTextureCRC;

// second we get info for entities
while ( 1 )
{
    if (msg->msg_uiCurrentPosition >= msg->msg_uiNumData) break;

    msg->read32(&iModelCRC);
    msg->read32(&iTextureCRC);

    Vector3D vPos = VECTOR3_NONE;

    msg->readVector(&vPos);

    byte yaw;
    byte index;

    msg->read8(&yaw);
    msg->read8(&index);

    CModelObject* model = pDataBase->loadModelCRC(iModelCRC);
    CTextureObject* texture = pDataBase->loadTextureCRC(iTextureCRC);

    assert(model != NULL);
    assert(texture != NULL);

    CModel* mo = new CModel;

    mo->SetModel(model);
    mo->SetTexture(texture);
    mo->SetUpModel();

    mo->modelMatrix = glm::mat4(1.f);
    mo->modelMatrix = glm::translate(mo->modelMatrix, vPos);
    mo->modelMatrix = glm::rotate(mo->modelMatrix, AngleToRad(float(yaw)) *
float((SDL_GetTicks64() * 0.001)), VECTOR3(0, 1, 0));

    EntityInfo* info = new EntityInfo;
    info->color = playerColorTables[index];
    info->model = mo;

    entities.push_back(info);
}
}

```

Аналіз повідомлень дещо невеликий, у порівнянні з сервером, але свою роль він виконує. Також можна помітити, що присутня нова команда –

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				58
Змн.	Арк.	№ докум.	Підпис	Дата		

NET_CMD_EVENT_CHANGE_SCORE, яка надсилається коли оновлюється таблиця результатів. Як було сказано раніше, нові команди завжди повинні додаватися з кінця та не перезаписувати зарезервовані команди для протоколу. Із цікавого можна помітити оброблення команди вимкнення серверу. Реалізація цієї команди виконує очищення ресурсів, закриття сокету та виконання функції завершення програми. Нижче наведена її реалізація:

```
void clientDisconnect(Network::CNetMessage* msg)
{
    GameHUD::clear();

    renderClearWorld(&gameSession.gs_world);

    if (gameSession.gs_clPlayer)
        delete gameSession.gs_clPlayer;

    gameSession.gs_clPlayer = NULL;

    Network::closeClient();

    pConsoleDataBase->deleteConsoleFunction("getPlayerPosition()");

    clientClear();

    char strText[64];

    msg->readBlock((byte*)strText, msg->getLength());
    strText[msg->getLength()] = 0;

    Printf("Stopped client, reason: %s\n", strText);

    pConsoleDataBase->executeFunc("Quit()");
}
```

Повернімось тепер до частини отримання повідомлень. Клієнтський гравець отримує дані від серверу, по аналогії тому, як сервер отримує дані від клієнта. Нижче можна побачити реалізацію:

```
void CPlayerClient::applyActions(Network::CNetMessage* msg)
{
    msg->readBlock((byte*)&cl_Health, sizeof(short));
    msg->readBlock((byte*)&cl_Armor, sizeof(short));

    byte w;

    msg->read8(&w);
    msg->read8((byte*)&cl_kills);
    msg->read8(&cl_state);

    hud_bShowScore = cl_state;

    cl_CurrentWeapon = w;

    msg->readVector(&cl_vPosition);

    msg->readBlock((byte*)&cl_Weapons->wp_amm0, sizeof(AmmoWeapon_t));
```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				59
Змн.	Арк.	№ докум.	Підпис	Дата		

```
}
```

Ці дані вважаються вже основними для клієнта, навіть якщо він змінить певні поля, у підсумку сервер відкорегує.

Після погодження даних для клієнта йде крок отримання даних для візуалізації ігрових сутностей. В цілому це те саме, що і на серверній частині, тільки на сервері ми записуємо, на клієнті ми зчитуємо ці дані.

Після чого йде крок надсилання даних у сервер. Перед цим ми записуємо те, які клавіші затиснуті у цей момент. У гравця, клієнтська сторона та серверна, є бітове поле, розмірністю на 32 біт. Кожен біт визначає яка клавіша була натиснута в певний момент виклику. Після чого ми відправляємо основні поля серверу, де той, в свою чергу, виконує логіку гравця. Наступним кроком ми надсилаємо серверу, викликаючи функцію рівня протоколу – «sendToServer». Реалізацію вищеписаних моментів наведено нижче:

```
void clientSentToServer(Network::CNetMessage* msg)
{
    msg->InitMessage(512);
    msg->setCMD(NET_CMD_ACTION);

    CPlayerClient* plcl = gameSession.getClientPlayer();

    plcl->doActions();
    plcl->setView(pViewPort);
    plcl->sendActionToServer(msg);

    Network::sendToServer(msg);
}
```

Алгоритм завершення сеансу клієнта в якомусь розумінні схожий з тим, як завершується сервер: ми очищуємо ресурси та надсилаємо серверу про бажання від'єднатися. Сервер отримує відповідь від клієнта та видаляє на рівні нашого протоколу та на рівні ігрової сесії. Нижче наведено лістинг коду вимкнення клієнта:

```
void clientShutDown()
{
    // if player null, this means we are not in game here
    if (gameSession.gs_clPlayer == nullptr) return;

    GameHUD::clear();

    renderClearWorld(&gameSession.gs_world);

    Network::CNetMessage msg;
    msg.InitMessage(20);

    msg.write16(gameSession.gs_clPlayer->cl_sId);
    msg.setCMD(NET_CMD_DISCONNECT_FROM_SERVER);
}
```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

```

Network::sendToServer(&msg);

if (gameSession.gs_clPlayer)
    delete gameSession.gs_clPlayer;

gameSession.gs_clPlayer = NULL;

Network::closeClient();

pConsoleDataBase->deleteConsoleFunction("getPlayerPosition()");

clientClear();
}

```

Загалом ми реалізували те, як гра відбувається на обох частинах – на стороні серверу та на стороні клієнта, за допомогою написаних функцій для роботи з нашим мережевим протоколом. Оскільки грою сингплеєру (гра наодинці), вважався сервер, де просто виконувалась логіка сутностей, ми впровадили багатокористувацький режим, добавивши клієнта. Скажімо, ми могли б також реалізувати це за допомогою протоколу TCP, однак для відеогри такого жанру, як шутер, де важлива швидкість, протокол не підійшов би, так як працює зі встановленим з'єднанням. Ми могли б використати чистий протокол UDP, але знову ж, виникла би проблема визначення даних, таких як команда, а також механізм дублювання PDU та засоби шифрування та стиснення.

2.4 Захист даних

Наш додаток та протокол готові до роботи. Ми реалізували на основі теоретичних знань протокол поверх UDP, розробили мережеву частину комунікації, побудовану на Windows Socket API, а також впровадили у відеогру, розробивши клас «CNetMessage» для комунікації між рівнем ігрової сесії та рівнем протоколу. Однак дані, які ми надсилаємо, можуть в будь-який момент використати чи прочитати. Зазвичай дані, які ми надсилаємо містять просто значення гравця, хеш поля моделей чи текстур і їхнє використання зі зловмисною метою менш імовірне. Однак ситуація змінюється при надсиланні текстових повідомлень, де є загроза перехоплення сторонньою особою. Саме тому у мережевому протоколі передбачене також шифрування даних.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				61
Змн.	Арк.	№ докум.	Підпис	Дата		

Як було обговорено раніше, ми зможемо шифрувати повністю дані, або лише текст. Ми зможемо зашифрувати конфіденційні дані повністю, але втратимо продуктивність програми та мережі, натомість ми зможемо зашифрувати лише текст.

Для такої задачі ми використаємо криптографічну бібліотеку `libsodium` [14], яка реалізує шифрування даних, цифровий підпис та обмін ключами. Алгоритм шифрування використовується «ChaCha20-Poly1305», який відноситься до класу автентифікованого шифрування з пов'язаними даними (AEAD), який поєднує потоковий шифр ChaCha20 з кодом автентифікації повідомлень Poly1305. Його основна перевага – це продуктивність та безпека. Для шифрування алгоритм використовує 256-бітний ключ та 96-бітне поле унікальності (nonce) з розширенням шифротексту розмірністю 128 біт. Також ця бібліотека вирішує ще одну проблему у криптографії – обмін ключами між двома сторонами сеансів. Обмін ключами здійснюється за алгоритмом Діффі-Геллмана з використанням еліптичної кривої (Рис. 2.4.0).

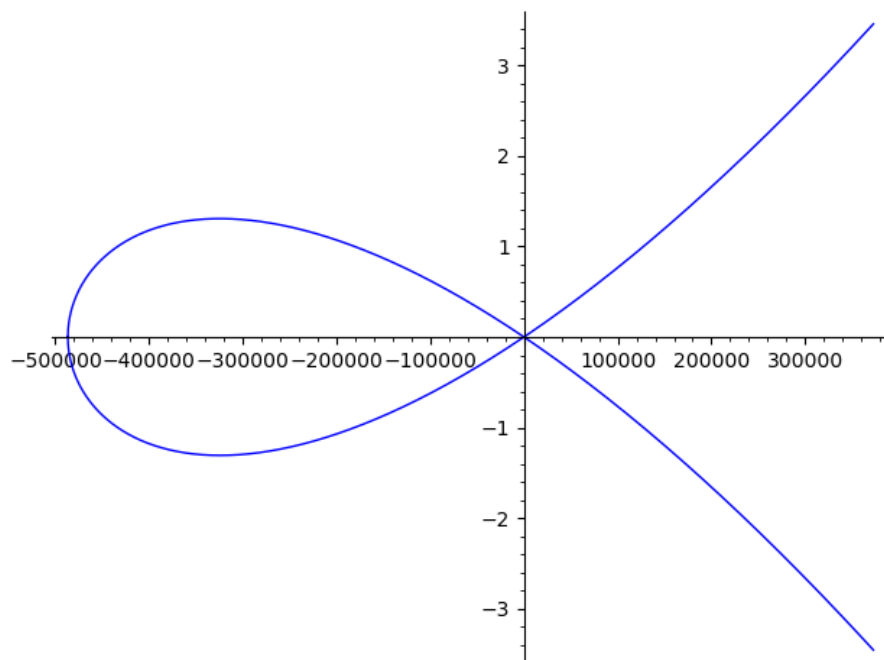


Рисунок 2.4.0 – Еліптична крива Curve25519

Після встановлення бібліотеки, та підключення до мережевої частини нашого ігрового рушія можна розпочинати роботу. На етапі запуску програми ми викликаємо функцію «`initializeProtection`», її реалізація наведено нижче:

```
void initializeProtection()
{
    int result = sodium_init();

    if (result < 0)
```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				62
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    {
        Error("Cannot initialize cryptographic library! sodium_init() returned %d",
result);
    }
    sodium_memzero(cl_pk, crypto_kx_PUBLICKEYBYTES * sizeof(byte));
    sodium_memzero(cl_sk, crypto_kx_SECRETKEYBYTES * sizeof(byte));
    sodium_memzero(cl_rx, crypto_kx_SESSIONKEYBYTES * sizeof(byte));
    sodium_memzero(cl_tx, crypto_kx_SESSIONKEYBYTES * sizeof(byte));

    sodium_memzero(sv_pk, crypto_kx_PUBLICKEYBYTES * sizeof(byte));
    sodium_memzero(sv_sk, crypto_kx_SECRETKEYBYTES * sizeof(byte));
    sodium_memzero(sv_tx, crypto_kx_SESSIONKEYBYTES * sizeof(byte));

    sodium_memzero(sv_keys, sizeof(struct serverKeysForClient));
}

```

Варто зазначити що функції, створені для роботи з цією бібліотекою не можуть бути імпортовані з DLL, що означає що ми не зможемо використати їх у Game.DLL та Power.EXE. Можна звернути увагу, що значення, які ми заповнюємо нулями – це ключі.

При створенні сервера або клієнта, у функції «startServer» та «startClient» відповідно, додаємо виклик функції для створення ключів – публічного та закритого. Псевдокод для клієнта та серверу наведено нижче:

```

void generateServerKeys()
{
    if (crypto_kx_keypair(sv_pk, sv_sk) < 0)
    {
        Throw("Cannot generate crypt-keys\n");
    }
}

void generateClientKeys()
{
    if (crypto_kx_keypair(cl_pk, cl_sk) < 0)
    {
        Throw("Cannot generate crypt-keys\n");
    }
}

```

Для шифрування даних, а також для виконання обміну ключами ми маємо чотири основних ключі:

- публічний ключ;
- секретний ключ;
- пара спільних секретних ключів.

Для обчислення пари секретних ключів, на основі публічного використовується алгоритм HKDF (ключовий дериваційний алгоритм).

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				63
Змн.	Арк.	№ докум.	Підпис	Дата		

У попередньому пункті на етапі під'єднання клієнта до сервера ви могли побачити такий рядок:

```
msg.writeBlock(getPublicKey(), 32 * sizeof(byte));
```

Це означає, що клієнт надсилає серверу свій публічний ключ який сервер використовує для обрахунку спільних секретних ключів, і у разі успіху поверне клієнту свій спільний ключ [14]. Основна схема реалізації наведена на рисунку 2.4.1.

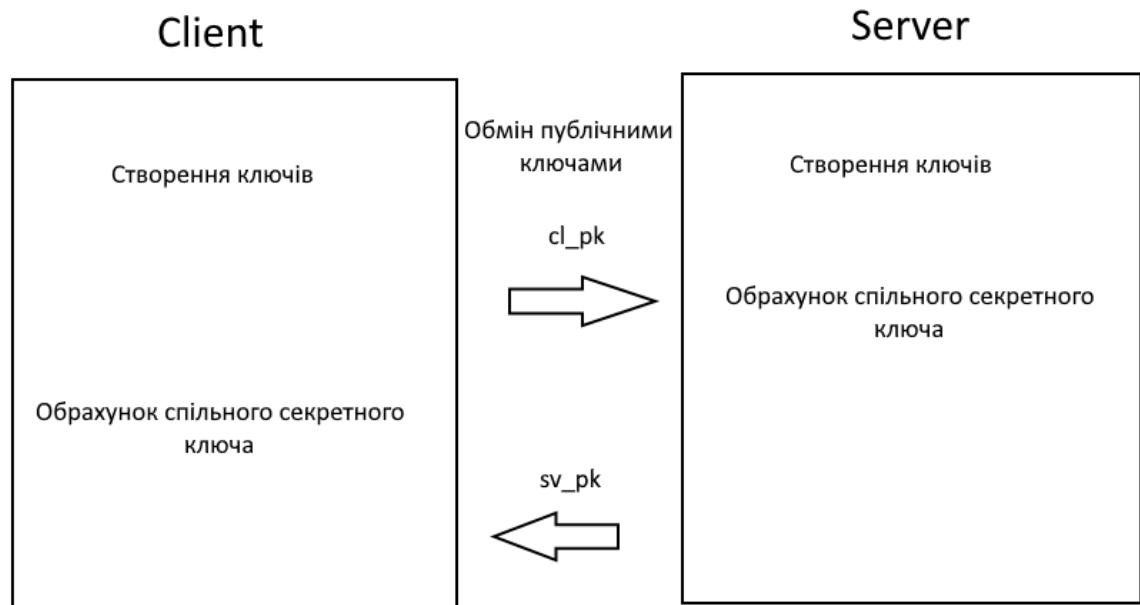


Рисунок 2.4.1 – Схема обміну ключами алгоритмом Діффі-Геллмана

Нижче наведена реалізація для обох сторін, клієнта та сервера:

```
bool exchangeOnClientSideKey(Network::CNetMessage* msg)
{
    msg->readBlock(sv_pk, crypto_kx_PUBLICKEYBYTES);

    if (crypto_kx_client_session_keys(cl_rx, cl_tx, cl_pk, cl_sk, sv_pk) != 0)
    {
        Printf("Cannot get public server key!\n");
        return false;
    }

    return true;
}

bool exchangeOnServerSideKey(Network::CNetMessage* msg, unsigned int iClient)
{
    msg->readBlock(cl_pk, crypto_kx_PUBLICKEYBYTES);

    if (crypto_kx_server_session_keys(sv_keys[iClient].sv_rx, sv_tx, sv_pk, sv_sk,
    cl_pk) != 0)
    {
        Printf("Cannot get public client[%d] key!\n", iClient);
        return false;
    }

    return true;
}
```

sv_keys – це структура, яка містить обраховані спільні ключі для розшифрування даних. Після того, як ми обчислили ключі, ми маємо отримати наступний результат який наведено у таблиці 2.4.0.

Таблиця 2.4.0 – Порівняння спільних ключів

Клієнт	Сервер
Client_TX = Server_RX	Server_RX = Client_TX
Client_RX = Server_TX	Server_TX = Client_RX

Тепер за потреби ми виконуємо шифрування та розшифрування даних. Нижче наведено реалізацію шифрування:

```
bool encryptMessage(CNetMSG* msg)
{
    unsigned int msgLen = msg->getLength();

    Printf("Len = %d\n", msgLen);

    byte nonce[crypto_secretbox_NONCEBYTES];
    randombytes_buf(nonce, sizeof(nonce));

    byte* buff = (byte*)calloc(crypto_secretbox_MACBYTES + msgLen, sizeof(byte));
    assert(buff != NULL);

    if (isServer())
    {
        if (crypto_secretbox_easy(buff, msg->getData(), msgLen, nonce, sv_tx) != 0)
        {
            free(buff);
            Printf("Error to encrypt msg\n");
            return false;
        }
    }
    else
    {
        if (crypto_secretbox_easy(buff, msg->getData(), msgLen, nonce, cl_tx) != 0)
        {
            free(buff);
            Printf("Error to encrypt msg\n");
            return false;
        }
    }

    byte lastCMD = msg->msg_netCMD;

    msg->Clear();

    msg->InitMessage(crypto_secretbox_NONCEBYTES + msgLen +
crypto_secretbox_MACBYTES);
    msg->writeBlock(nonce, crypto_secretbox_NONCEBYTES);
    msg->writeBlock(buff, crypto_secretbox_MACBYTES + msgLen);
    msg->setCMD(lastCMD);
    free(buff);
    // msg->dumpHexToFile("Enc.txt");
    return true;
}
```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				65
Змн.	Арк.	№ докум.	Підпис	Дата		

```
}
```

Реалізація розшифрування:

```
bool decryptMessage(CNetMSG* msg, int iClient)
{
    //    msg->dumpHexToFile("Dec.txt");
    byte nonce[crypto_secretbox_NONCEBYTES];

    msg->readBlock(nonce, crypto_secretbox_NONCEBYTES);

    unsigned int full = msg->getLength();
    unsigned int cipherLen = full - crypto_secretbox_NONCEBYTES;
    unsigned int plainLen = cipherLen - crypto_secretbox_MACBYTES;

    byte* bufftext = (byte*)calloc(plainLen, sizeof(byte));
    byte* buffcipher = (byte*)calloc(cipherLen, sizeof(byte)); // cipher = mac +
    ciphertext

    assert(buffcipher != NULL);
    assert(bufftext != NULL);

    msg->readBlock(buffcipher, cipherLen);

    if (isServer())
    {
        if (crypto_secretbox_open_easy(bufftext, buffcipher,
                                       cipherLen, nonce, sv_keys[iClient].sv_rx) != 0)
        {
            Printf("Cannot decrypt message from client\n");
            free(bufftext);
            free(buffcipher);
            return false;
        }
    }
    else
    {
        if (crypto_secretbox_open_easy(bufftext, buffcipher,
                                       cipherLen, nonce, cl_rx) != 0)
        {
            free(buffcipher);
            free(bufftext);
            Printf("Cannot decrypt message from server");
            return false;
        }
    }

    byte lastCMD = msg->msg_netCMD;

    msg->Clear();
    msg->setCMD(lastCMD);
    msg->InitMessage(plainLen);
    msg->writeBlock(bufftext, plainLen);

    free(bufftext);
    free(buffcipher);

    return true;
}
```

Наші основні функції для роботи з шифруванням даних, а також механізмом обміну ключами готові. Тепер потрібно впровадити у мережевий код рушія та гри.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

При формуванні протоколу ми встановлюємо параметр для шифрування та змінюємо прапорець бітового поля нашого протоколу. На стороні серверу та клієнта це виглядає наступним чином:

```
if (packet->ubNetCMD == NET_CMD_SEND_TEXT && net_bCryptText && !net_bCryptAllData)
{
    if (net_ubCryptType == 1)
    {
        packet->uwFlags |= NET_UDP_USE_CRYPT_OWN;
        encryptMessage(msg);
    }
    else if (net_ubCryptType == 2)
        packet->uwFlags |= NET_UDP_USE_CRYPT_AES256;
    else
        packet->uwFlags |= 0;
}

if (net_bCryptAllData && !net_bCryptText)
{
    if (net_ubCryptType == 1)
    {
        packet->uwFlags |= NET_UDP_USE_CRYPT_OWN;
        encryptMessage(msg);
    }
    else if (net_ubCryptType == 2)
        packet->uwFlags |= NET_UDP_USE_CRYPT_AES256;
    else
        packet->uwFlags |= 0;
}

packet->uwNumData = msg->getLength();
```

Після того, як ми визначили певні бітові поля, ми зашифруємо дані, та надсилаємо відправнику. При отриманні цих даних ми звіряємо команду та бітові поля, якщо дані потрібно розшифрувати. Нижче наведено псевдокод цієї операції:

```
if (packet->uwFlags & NET_UDP_USE_CRYPT_OWN)
{
    bResult = decryptMessage(pMsg, clId);

    if (!bResult)
    {
        Printf("Invalid text message\n");
    }
    else
    {
        memset(buffer, 0, sizeof(buffer));
        pMsg->readBlock((byte*)buffer, pMsg[j]->getLength());
        Printf("%s\n", buffer);
    }
}
else
{
    memset(buffer, 0, sizeof(buffer));
    pMsg->readBlock((byte*)buffer, packet->uwNumData);
    Printf("%s\n", buffer);
}
```

Тепер ми реалізуємо функцію надсилання текстового повідомлення між сервером та клієнтом. Лістинг коду наведено нижче:

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				67
Змн.	Арк.	№ докум.	Підпис	Дата		

```

void sendText(const char* strText)
{
    assert(strText != nullptr);

    char buff[1024];

    memset(buff, 0, sizeof(buff));

    strcpy(buff, strText);

    CNetMessage msg;

    msg.InitMessage(strlen(buff) + 1);
    msg.writeBlock((byte*)buff, strlen(buff));
    msg.setCMD(NET_CMD_SEND_TEXT);

    if (isServer())
    {
        FOREACH_ARRAY(NET_MAX_CLIENTS)
        {
            if (netDataGlobals.clients[i].lastSeqNumber != 0)
            {
                sendToClient(&msg, i);
            }
        }
    }
    else
    {
        sendToServer(&msg);
    }
}

```

В результаті чого ми отримуємо шифрування даних з певними опціями, наприклад, ми зможемо увімкнути шифрування для всіх даних, для тексту або взагалі не використовувати шифрування даних. Останній варіант покращує продуктивність програми, але не забезпечує конфіденційність даних, в той час як перший навпаки.

Висновки до другого розділу

У даному розділі було розроблено та розглянуто особливості розробки мережових додатків, а також реалізація власного мережевого протоколу на основі транспортного протоколу UDP, з урахуванням усіх теоретичних аспектів розглянутих у попередньому розділі.

Ми створили протокол з підтримкою впорядкування пакетів, шифрування даних, а також мінімізацією розміру, для уникнення фрагментації. Наш мережевий протокол на основі команди формує PDU для надсилання даних та подальшої роботи з ним. Для полегшення роботи над протоколом було створено спеціальний клас повідомлень для збору даних.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				68
Змн.	Арк.	№ докум.	Підпис	Дата		

Після реалізації задуманих ідей ми створили невеличку бібліотеку для полегшення роботи над сокетом Windows, які потрібні для комунікації між хостами. У ній ми реалізували архітектуру клієнт-сервер з використанням нашого протоколу.

Наступним нашим кроком було створено демонстраційну гру з LAN-з'єднанням та впровадження роботи клієнт-серверу у цей додаток, з використанням нашої бібліотеки та протоколу. Наша гра розподіляється на три модуля, серед яких дві динамічні зв'язувальні бібліотеки та програма виконання. Це необхідно було для створення модульності проекту, а також орієнтації при розробці таких додатків. Загалом було використано наступні бібліотеки: Bullet Engine, SDL2, OpenGL32, lib-sodium, glm math та системні бібліотеки ОС Windows.

Варто зазначити, що ми впровадили також шифрування для протокола, де ми можемо забезпечити конфіденційність даних. Ми зможемо також відключити шифрування даних, як результат – підвищиться продуктивність, хоч і знизиться конфіденційність.

Якщо порівняти роботу протоколу з роботою інших протоколів, у нього є схожі методи з протоколом HTTPS, де також підтримується шифрування та надсилаються запити, подібно до наших команд. Наш протокол також має схожості з TCP, де підтримується ASK та опцій, подібних до бітового поля нашого протоколу.

В цілому ми виконали завдання, реалізувавши надійний та захищений протокол для нашого додатка як відеогра. Його використання можливе не тільки для ігор, а й для програм, де швидкість та продуктивність є важливим аспектом їхньої архітектури.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3 ТЕСТУВАННЯ РОБОТИ ПРОТОКОЛУ

3.1 Консоль додатка

У цьому розділі будуть подані результати нашої розробки, а також тестування протоколу та відеогри у різних ситуаціях. Перш за все, ознайомимось з консоллю гри, яка водночас є і меню гри (Рис. 3.1.0).

```
-----
In the Engine version 1.0
Initialization OpenGL 4.0...
-----
OpenGL 4.0 created succesfull
Intel Intel(R) UHD Graphics 620 4.6.0 - Build 31.0.101.1999
-----
Working In the Engine directory patch: C:\Games\In-the-Power\
-----
Compile shaders...
All shaders compiled succesful
-----
Loading game library: Game.dll
Game library: "Game.dll" loaded succesfull
Player name = Username
***** Dummy console type menu *****
to create new game, print in console: startGame()
to join local host, print in console: joinGame()
to search LAN game, print in console: searchLAN()
to help for console commands, print: helpConsole()
:

```

Рисунок 3.1.0 – Консоль додатка

У консолі ми можемо викликати функції, які знаходяться у будь-якому адресному просторі, чи то ігровий рушій, чи віконний додаток. Також тут є можливість змінити параметри гри та мережі.

Для того щоб, виконати функцію, у консолі потрібно прописати назву цієї функції, звертаючи увагу на дужки та регістри. Приблизний виклик функції наведено на рисунку 3.1.1.

:PrintAllSymbols(X

Рисунок 3.1.1 – Виклик функції

Якщо такої функції у нас немає, відповідно консоль проінформує нас про те, що дана функція у базі даних не записана, відповідно ми не зможемо її виконати. Для прикладу ми ввели неіснуючу функцію у консоль (Рис.3.1.2).

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

```
-----
Symbol "reportLoadedObkects()" not found!
:|
```

Рисунок 3.1.2 – Повідомлення про те, що даної функції не існує

У консолі присутні і параметри, такі як змінні. Якщо ми ввели зміну, ми отримуємо її значення на даний момент, але якщо ми задали змінній значення, вона автоматично приймає для будь-якого параметру, наприклад відключення шифрування для текстових повідомлень (рис. 3.1.3).

```
< net_bCryptText = 1
:net_bCryptText = 0
```

Рисунок 3.1.3 – Приклад встановлення параметру

Як у випадку з функціями, при неправильному введенні консоль нас проінформує те, що даного символу не існує. На даний момент доступні всього 11 функцій для відеогри (рис.3.1.4). Для того, щоб переглянути всі доступні символи, натисніть клавішу «tab», вона автоматично надрукує «PrintAllSymbols()», для виведення всієї бази даних оболонки у консоль.

```
Functions:
PrintAllSymbols()
createError()
createWarning()
Quit()
reportLoadedObkects()
getClientPosition()
printLoadedObkects()
getEntityInfo()
startGame()
joinGame()
ghostObkectsInfo()
```

Рисунок. 3.1.4 – Доступні функції

Також у консолі наявна реалізація команди. Воно дещо схоже на функцію, але головна відмінність в тому, що команда може використати функції з параметрами та ті, які повертають значення. Команди завжди розпочинаються з символу «/». На даний момент реалізована лише одна команда, її синтаксис виглядає наступним чином:

/connect x.x.x.x

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

Дана команда виконує підключення клієнта до серверу за заданою IP-адресою. Є також функція «joinGame()», яка використовується на локальному хостові.

Стрілками угору чи стрілками вниз ми зможемо побачити останні команди, функції чи змінні, які ми вводили. Максимальна кількість змінних, збережених при введенні, становить 128. Якщо це число перевищити, останні записані символи перезаписуються.

Після завершення програми, весь текст та вся інформація яка була виведенна у консоль зберігається у файл журналювання. Це зручно, коли у буфер консолі записані помилки чи винятки.

3.2 Комп'ютерна гра

У попередньому пункті було розглянуто консоль гри, через яку можна змінювати налаштування мережі, ігрового сеансу або спостерігати за об'єктами. У цьому пункті ми сконцентруємось на самій відеоігрі.

Мінімальні на даний момент системні вимоги:

- ОС Windows 10;
- Відеокарта, сумісна з OpenGL 4.0+;
- Оперативна пам'ять 512 мб;
- Мережевий адаптер;
- Місце на жорсткому диску 128 мб.

Гра не вимоглива до апаратного забезпечення, оскільки вона призначена для тестування мережевої оболонки та роботи протоколу.

Жанр нашої гри – це шутер від першої особи. Суть гри доволі проста: набрати першому найбільшу кількість очків, убиваючи при цьому ворожих гравців. Гравець який помер, відроджується у випадковій точці відродження, розташованій у спеціальних місцях ігрової мапи. Варто звернути увагу, що ці очки можна також втратити, якщо гравець упаде за межами ігрової мапи або помре від власної зброї. В такому разі гравцю забирається одне очко.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		72

Основне управління грою на даний момент – це клавіатура та комп’ютерна миша. У таблиці 3.2.0 наведені клавіші та їхню роль у грі.

Таблиця 3.2.0 – Управління грою

Клавіша	Призначення
W	Рух прямо
A	Рух ліворуч
S	Рух задом
D	Рух праворуч
SPACE	Стрибок
ESCAPE	Вихід з гри
TAB	Перегляд результатів гравців
~ чи `	Консоль
3	Повноекранний режим (1080р)
9	Скріншот гри
Ліва кнопка миші	Стрільба
Покрутка колеса миші угору	Вибір наступної зброї
Покрутка колеса миші униз	Вибір попередньої зброї

Для того, щоб ми змогли змінити ім’я гравця, необхідно відкрити файл «PConfig.cfg» та написати бажане ім’я та код кольору (Рис. 3.2.0). Основні коди кольорів наведено у таблиці 3.2.1.

Таблиця 3.2.1 – Коди кольору гравця

Код	0	1	2	3	4	5	6	7
-----	---	---	---	---	---	---	---	---

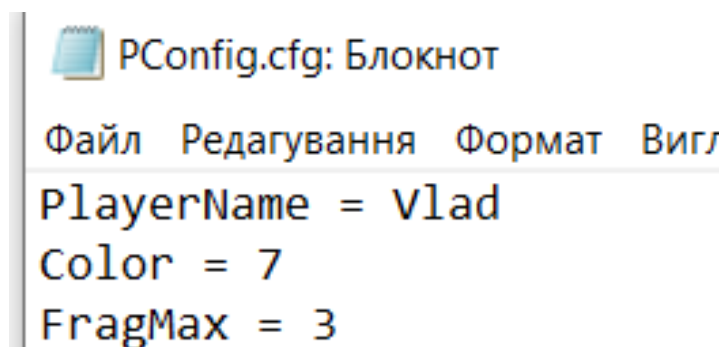


Рисунок 3.2.0 – Конфігураційний файл для гри

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				73
Змн.	Арк.	№ докум.	Підпис	Дата		



Рисунок 3.2.1 – Інтерфейс гри

Нарешті, розглянемо інтерфейс гри (Рис. 3.2.1). Для простоти, вся інформація подається у текстовому вигляді. Нижче наведено опис елементу, за позначенням на рисунку 3.2.1:

1. Таблиця результатів;
2. Очки гравця та загальна кількість, яку необхідно першому набрати;
3. Відсоток броні гравця;
4. Відсоток здоров'я гравця;
5. Патрони для зброї, може бути відсутнім;
6. Приціл;
7. Зброя гри.

3.3 Тестування роботи протоколу

Для того щоб створити сервер, ми у консолі вводимо функцію запуску гри. Після чого ми попадаємо у ігровий світ, виступаючи у ролі серверу (рис. 3.2.0)



Рисунок 3.3.0 – введення команди для консолі

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

Для підключення до серверу у нас є два способи – вказати явну IP-адресу серверу, або запустити на локальному хості. У консолі на машині клієнтській, введіть адресу серверу (Рис. 3.3.1).



Рисунок 3.3.1 – Підключення до серверу локальній мережі

У разі успішності, ми підключаємось до гри серверу як клієнтський гравець, результат наведено на рисунку 3.3.2.

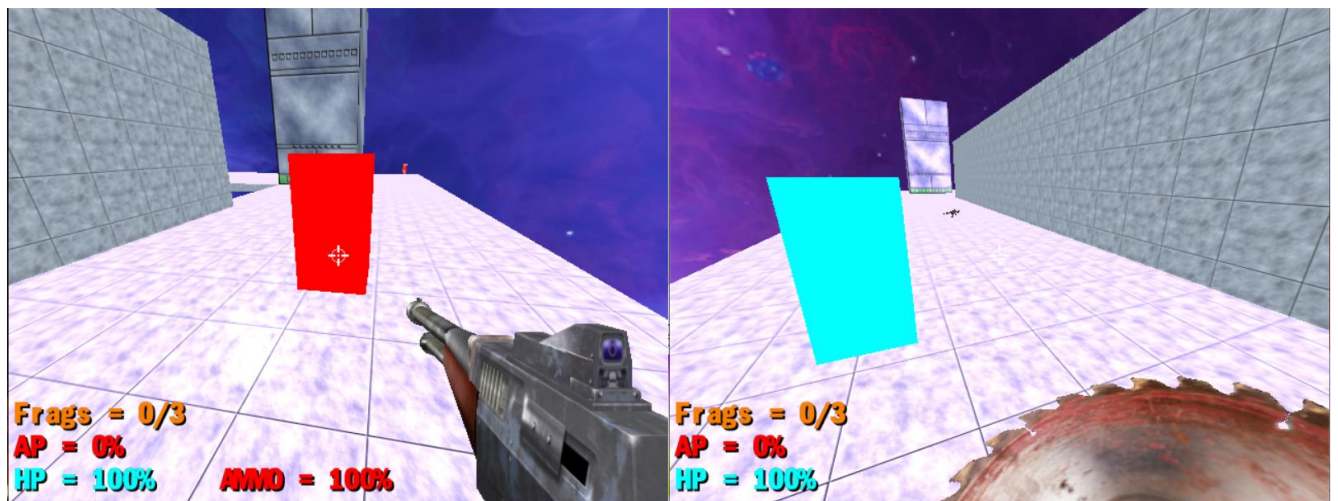


Рисунок 3.3.2 – Зображення хостів праворуч сервер, ліворуч клієнт

Таким чином, ми зможемо грати у локальній мережі гру. Варто зазначити, що брандмауер Windows може блокувати UDP порт, який у нас за замовчуванням для додатка 5910. Для цього потрібно у брандмауері встановити правило, яке дозволяє певному UDP порту надходити у мережу.

У попередньому розділі ми реалізували шифрування текстових повідомлень, або і всіх повідомлень у мережі. Ми могли б використати Wireshark для перехоплення трафіку у мережі, однак для загального порівняння ми зможемо записати необроблені дані, які надійшли з мережі. Щоб уникнути багаторазового запису у файл, опція вимикається одразу після запису. Для цього у консолі ми увімкнемо таку опцію та порівняємо результат (рис 3.2.3).

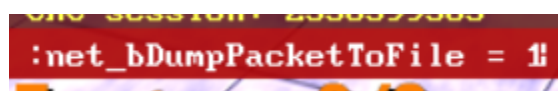


Рисунок 3.3.3 – Увімкнення запису даних у файл

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				75
Змн.	Арк.	№ докум.	Підпис	Дата		

Тепер нехай клієнт надішле текстове повідомлення. Ми реалізували просту тестову функцію для надсилання звичайного тексту (Рис. 3.3.4). Після цього сервер повинен отримати відповідне повідомлення (рис. 3.3.5).

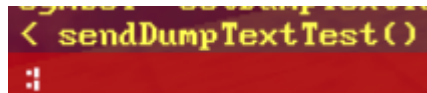


Рисунок – 3.3.4 Надсилання тексту серверу



Рисунок – 3.3.5 Отримання тексту від клієнта

Окрім повідомлення, у нас повинен з'явився бінарний файл, у якому містяться усі дані, отримані з функції recvfrom, включно з заголовком нашого протоколу (Рис. 3.3.6).

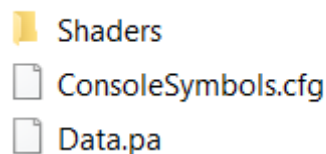


Рисунок 3.3.6 – файл пакету даних (Data.pa)

Для аналізу даних ми цей файл зможемо відкрити у hex-редакторі, для визначення байтів (Рис. 3.3.7).

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Декодований текст
00000000	05	22	00	4E	00	00	00	00	00	B3	12	00	00	31	A6	01	0".N.....i...l .
00000010	00	00	00	00	00	8D	47	2B	95	70	9C	B4	4C	7C	BB	33	KG+•pъrL »3
00000020	2E	4C	97	BD	B6	51	83	43	FF	10	50	E6	16	83	68	85	.L-SŹQŹСя.Рж.Źh...
00000030	62	A4	31	93	A2	C6	00	33	00	BF	B4	5D	4B	28	27	C6	ь=1"ŹЖ.3.Źr]K('Ж
00000040	D9	CE	75	3A	D2	32	33	33	D8	7E	0C	E0	0A	89	2B	D2	ЩOu:T233Ш~.a.Ź+T
00000050	B9	D2	16	2C	A3	9A	75	4F	1C	39	D9	3A	E9	82	37	3C	ŹT.,JъuO.9Щ:Ź,7<
00000060	54	02	87	00	00	00	00	00	00	00	00	00	00	00	00	00	T.Ź+.....
00000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Рис. 3.3.7 Отримані сирі дані з клієнта

На рисунку вище ми бачимо, що 1 байт це номер команди протоколу, де 05 це команда **NET_CMD_SEND_TEXT**. Текст, який ми надіслали, має вигляд послідовності довільних символів. Це свідчить про те, що текстові повідомлення між клієнтом та сервером шифруються, а отже, наша реалізація спрацювала.

Тепер вимкнемо шифрування тексту. Для цього ми скористаємось опцією для відключення шифрування текстових повідомлень на клієнті (Рис. 3.3.8).



Рисунок 3.3.8 Вимкнення опції шифрування текстових повідомлень

На сервері, включаємо знову опцію запису даних у файл, тепер після того, як ми отримали дані з клієнта, ми знову зможемо побачити бінарний файл. Відкривши його у hex-редакторі, ми бачимо відкритий текст (рис. 3.2.9).

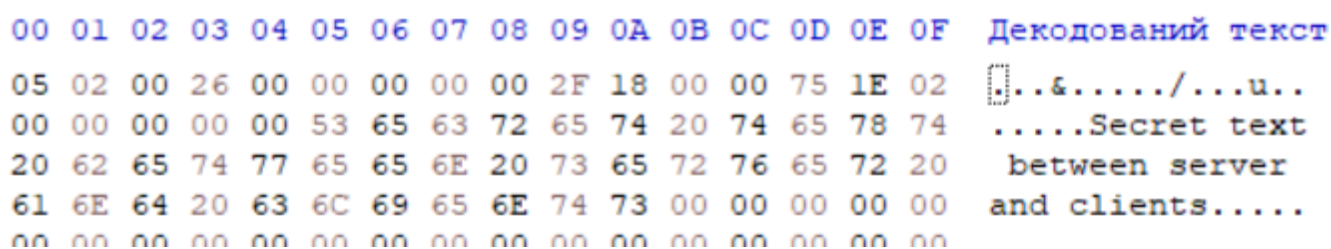


Рисунок 3.3.9 – Відкритий текст

При вимкненні опції шифрування даних, текст подається у відкритому вигляді і будь-хто з умінням перехоплення пакетів може прочитати відкритий текст. Тому задля конфіденційності опція шифрування текстових повідомлень завжди повинна бути увімкнена.

Висновки з третього розділу

Третій розділ присвячений тестування нашого додатку, з використанням нашого протоколу та мережевої бібліотеки. Тестування проводилось у локальній мережі на двох робочих станціях, з операційною системою Windows 10 та відеоадаптером з підтримкою OpenGL 4.0.

Для налагодження та тестування у додаток було впроваджено віртуальну консоль, щоб ми змогли встановити певні параметри, для діагностування та налагодження мережі, ігрового процесу та додатка. За допомогою консолі ми зможемо як запустити гру, під'єднатися до серверу та вийти з гри. Також ми зможемо увімкнути певні опції, такі як шифрування даних, а у перспективі – стиснення (на даний момент стиснення не реалізовано).

Результати тестування додатку показали, що реалізація захищеного комунікаційного протоколу для мережевої відеогри вдалась, про що свідчить можливість підключення гравців до серверу, а також зашифровані текстові повідомлення.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

Метою кваліфікаційної роботи було створення захищеного протоколу для комп'ютерної відеогри. Розробка даного продукту передбачала реалізацію мережевого протоколу, з підтримкою шифрування даних, упорядкування пакетів, надійності, розробка мережевої архітектури, та створення власне тестової онлайн відеогри, з використанням LAN-мережі.

За основу ми взяли протокол UDP, оскільки створення протоколу з нуля для такого додатку, як відеогра створить певні проблеми з передачею даних у маршрутизаторі, через невизначеність протоколу. Поверх UDP ми реалізували власний протокол, з підтримкою шифрування даних, впорядкування пакетів за порядковим номером, а також з підтримкою надійності.

Основна ідея нашого протоколу це команди – за допомогою них ми спілкуємось у мережі через як клієнт та сервер..

У додатку реалізовано зв'язок між програмними рівнями, рівнем протоколу, призначеним для надсилання даних за допомогою бібліотеки Windows Socket API, та рівнем програми, де безпосередньо діє логіка клієнта та сервера. Для підтримки зв'язку ми реалізували клас, який буде записувати дані та команду, де на основі команди генерується заголовок протоколу та здійснюється підготовка даних до комунікації.

Усю цю реалізацію ми впровадили у розроблену відеогру, яка поділена на дві динамічно зв'язувальні бібліотеки та програму виконання. Для полегшення розробки, гра використовує вільні бібліотеки, такі як SDL2, libsodium, Bullet Engine, OpenGL32, а також системні бібліотеки WinApi32 та Winsock2.

Результати тестів показали, що ми виконали своє завдання – створити захищений протокол для онлайн гри. При різних параметрах ми тестували те, чи дійсно дані захищені, що свідчить про успішну роботу механізму обміну ключами.

Загалом створення додатків такого типу та реалізація мережевого протоколу досить – складна задача, яка вимагає високого рівня сформованості навичок у галузях комп'ютерної інженерії та кібербезпеки. Розроблений протокол та ігровий додаток демонструють базові принципи реалізації таких проєктів на практиці. Їх

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

можна вдосконалювати, розширюючи наявний функціонал. Водночас, уже на даному етапі, даний проєкт може бути використано для ознайомлення з базовими концепціями створення ігор та їхньої мережної частини, з урахуванням проблем, які постають перед розробниками.

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Andrew Tanenbaum, David Wetherall, «Computer Networks 5th edition», New Jersey, 2012 – 960 с.
2. Б.Ю. Жураковський, І.О. Зенів, «РОЗРОБКА ТА РЕАЛІЗАЦІЯ МЕРЕЖНИХ ПРОТОКОЛІВ НАВЧАЛЬНИЙ ПОСІБНИК», Київ: КПІ ім. Ігоря Сікорського, 2020 – 462 с.
3. Mikael Degermark and Stephen Pink, «Issues in the Design of a New Network Protocol».
4. Douglas E. Comer, «Internetworking With TCP/IP Vol I: Principles, Protocols, and Architecture Sixth Edition», 2013 – 733 с.
5. RFC 793 — Transmission Control Protocol, 1981.
6. RFC 768 — User Data Protocol, 1980.
7. [Електроний ресурс] – режим доступу до ресурсу:
<https://www.cisco.com/c/en/us/support/docs/ip/generic-routing-encapsulation-gre/25885-pmtud-ipfrag.html>
8. [Електроний ресурс] – режим доступу до ресурсу:
<https://notes.shichao.io/tcpv1/ch10/>
9. [Електроний ресурс] – режим доступу до ресурсу:
<https://medium.com/@karthikayanmailsamy/5-layer-network-model-made-simplified-e813da0913ba>
10. [Електроний ресурс] – режим доступу до ресурсу:
<https://learn.microsoft.com/en-us/windows/win32/winsock/tcp-ip-raw-sockets-2>
11. Joshua Glazer, Sanjay Madhav «Multiplayer Game Programming», 2015 – 385 с.
12. [Електроний ресурс] – режим доступу до ресурсу:
<https://learn.microsoft.com/en-us/troubleshoot/windows-client/setup-upgrade-and-drivers/dynamic-link-library>
13. [Електроний ресурс] – режим доступу до ресурсу:
<https://gameprogrammingpatterns.com/game-loop.html>

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				81
Змн.	Арк.	№ докум.	Підпис	Дата		

14.[Електроний ресурс] – режим доступу до ресурсу: <https://doc.libsodium.org/>

15.[Електроний ресурс] – режим доступу до ресурсу:
<https://learn.microsoft.com/en-us/windows/win32/winsock/windows-sockets-start-page-2>

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

ДОДАТКИ

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				83
Змн.	Арк.	№ докум.	Підпис	Дата		

Реалізація методів класу CNetMessage

Лістинг коду:

```

namespace Network
{
    CNetMessage::CNetMessage(void)
    {
        msg_data = NULL;
        msg_uiCurrentPosition = 0;
        msg_uiLastPosition = msg_uiCurrentPosition;
        msg_uiMaxData = 0;
        msg_uiNumData = 0;
        msg_netCMD = NET_CMD_NONE;
    }

    CNetMessage::CNetMessage(size_t numData)
    {
        msg_data = NULL;
        msg_uiCurrentPosition = 0;
        msg_uiLastPosition = msg_uiCurrentPosition;
        msg_uiMaxData = 0;
        msg_uiNumData = 0;

        msg_netCMD = NET_CMD_NONE;

        InitMessage(numData);
    }

    CNetMessage::~CNetMessage(void)
    {
        Clear();
    }

    void CNetMessage::Clear()
    {
        if (msg_data != NULL)
            free(msg_data);
        msg_data = NULL;
        msg_uiCurrentPosition = 0;
        msg_uiLastPosition = 0;
        msg_uiNumData = 0;
        msg_uiMaxData = 0;
        msg_netCMD = 0;
    }

    CNetMessage& CNetMessage::operator=(CNetMessage& other)
    {
        Clear();

        this->msg_uiCurrentPosition = other.msg_uiCurrentPosition;
        this->msg_uiLastPosition = other.msg_uiLastPosition;
        this->msg_uiNumData = other.msg_uiNumData;
        this->msg_uiMaxData = other.msg_uiMaxData;
        this->msg_netCMD = other.msg_netCMD;

        if (other.getData() == NULL)
        {
            this->msg_data = NULL;
            return *this;
        }
    }
}

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				84
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        memcpy(this->msg_data, other.getData(), other.msg_uiNumData);

        return *this;
}

void CNetMessage::copyData(byte* data, size_t numData)
{
    if (numData > NET_MAX_BUFFER)
        Throw("Network message is too large!\n");

    Clear();

    //msg_data = (byte*)malloc(numData);

    //if (msg_data == NULL)
    //    Error("Cannot allocate message!\n");

    msg_uiMaxData = numData;
    msg_uiNumData = numData;

    msg_data = (byte*)copyMemory(data, numData);
}

void CNetMessage::setData(byte* data, size_t numData)
{
    Clear();

    msg_data = data;
    msg_uiNumData = numData;
    msg_uiMaxData = numData;
}

void CNetMessage::InitMessage(size_t numData)
{
    if (numData > NET_MAX_BUFFER)
        Throw("Network message is too large!\n");

    Clear();

    msg_data = (byte*)malloc(numData);

    if (msg_data == NULL)
        Error("Cannot allocate message!\n");

    memset(msg_data, 0, sizeof(numData));

    msg_uiMaxData = numData;
}

void CNetMessage::dumpHexToFile(const std::string& strFile)
{
    CFileName filename = "System\\" + strFile;

    FileSetGlobalPatch(filename);

    FILE* F = fopen((const char*)filename, "w");

    assert(F != NULL);

    FOREACH_ARRAY(getLength())
    {
        fprintf(F, "0x%X ", msg_data[i]);
    }
}

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				85
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        fclose(F);
    }

    /***** Writing methods *****/

    void CNetMessage::write8(const byte& b)
    {
        if (msg_data == NULL)
        {
            Printf("Cannot write data, because data = NULL\n");
            return;
        }

        if (msg_uiNumData + sizeof(byte) >= NET_MAX_BUFFER) {
            Throw("Network message is to large!\n");
        }

        msg_data[msg_uiNumData] = b;
        msg_uiNumData += sizeof(unsigned char);
    }

    void CNetMessage::write16(const unsigned short& s)
    {
        write8(s >> 8);
        write8(s & 0xFF);
    }

    void CNetMessage::write32(const unsigned int& i)
    {
        write8((i & 0xFF000000) >> 24);
        write8((i & 0x00FF0000) >> 16);
        write8((i & 0x0000FF00) >> 8);
        write8((i & 0x000000FF) >> 0);
    }

    void CNetMessage::write64(unsigned long long& ll)
    {
        write8((ll >> 56) & 255ui8);
        write8((ll >> 48) & 255ui8);
        write8((ll >> 40) & 255ui8);
        write8((ll >> 32) & 255ui8);
        write8((ll >> 24) & 255ui8);
        write8((ll >> 16) & 255ui8);
        write8((ll >> 8) & 255ui8);
        write8((ll >> 0) & 255ui8);
    }

    void CNetMessage::writeFloat(const float& f)
    {
        //writeBlock((byte*)&f, sizeof(float));
        write32(*(unsigned int*)&f);
    }

    void CNetMessage::writeDouble(const double& d)
    {
        write64(*(unsigned long long*)&d);
    }

    void CNetMessage::writeString(std::string& str)
    {
        write32(str.size());
        writeBlock((byte*)str.c_str(), str.size());
    }

    void CNetMessage::writeBlock(byte* data, size_t num)
    {

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				86
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        FOREACH_ARRAY(num)
        {
            write8(data[i]);
        }

        /*
        if (msg_uiNumData + num >= NET_MAX_BUFFER) {
            Printf("Cannot write block data! Buffer overflow!\n");
            return;
        }

        memcpy(msg_data + msg_uiNumData + num, data, num);

        msg_uiNumData += num;*/
    }

    void CNetMessage::writeVector(Vector3D& v)
    {
        writeFloat(v.x);
        writeFloat(v.y);
        writeFloat(v.z);
    }

    /***** Reading methods *****/

    void CNetMessage::read8(byte* b)
    {
        if (msg_uiCurrentPosition + sizeof(unsigned char) > NET_MAX_BUFFER ||
            msg_uiCurrentPosition > msg_uiNumData)
        {
            Printf("Cannot read message max = %d, your = %d, read = %d, msg = %d\n",
                NET_MAX_BUFFER, msg_uiNumData, msg_uiCurrentPosition, msg_netCMD);
            return;
        }

        *b = msg_data[msg_uiCurrentPosition];

        msg_uiCurrentPosition += sizeof(unsigned char);
    }

    void CNetMessage::read16(unsigned short* s)
    {
        unsigned char a, b;

        read8(&a);
        read8(&b);

        *s = (a << 8) | (b << 0);
    }

    void CNetMessage::read32(unsigned int* i)
    {
        unsigned char a, b, c, d;
        read8(&a);
        read8(&b);
        read8(&c);
        read8(&d);

        // yes its weird, buuut its just same method which uses below
        *i = RGBA_toHEX(a, b, c, d);

        //*i = (a << 24) | (b << 16) | (c << 8) | (d << 0);
    }

    void CNetMessage::read64(unsigned long long* ll)
    {
        unsigned char a, b, c, d, e, f, g, h;

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				87
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        read8(&a);
        read8(&b);
        read8(&c);
        read8(&d);
        read8(&e);
        read8(&f);
        read8(&g);
        read8(&h);

        *ll = (unsigned long long(a) << 56ui64) |
               (unsigned long long(b) << 48ui64) |
               (unsigned long long(c) << 40ui64) |
               (unsigned long long(d) << 32ui64) |
               (unsigned long long(e) << 24ui64) |
               (unsigned long long(f) << 16ui64) |
               (unsigned long long(g) << 8ui64) |
               (unsigned long long(h) << 0ui64);
    }
    void CNetMessage::readFloat(float* f)
    {
        unsigned int i;
        read32(&i);
        *f = *(float*)&i;
    }
    void CNetMessage::readDouble(double* d)
    {
        unsigned long long ll;
        read64(&ll);
        *d = *(unsigned long long*)&ll;
    }
    void CNetMessage::readString(std::string& str)
    {
        unsigned int length;
        read32(&length);

        char* data = (char*)malloc(length);
        readBlock((byte*)data, length);
        data[length] = 0;

        str = data;

        free(data);
    }
    void CNetMessage::readBlock(byte* data, size_t num)
    {
        FOREACH_ARRAY(num)
        {
            read8(&data[i]);
        }
    }

    void CNetMessage::readVector(Vector3D* v)
    {
        readFloat(&v->x);
        readFloat(&v->y);
        readFloat(&v->z);
    }

    /***** I/O methods *****/

    void CNetMessage::rewindMSG(void)
    {
        msg_uiLastPosition = msg_uiCurrentPosition;
        msg_uiCurrentPosition = 0;
    }
    void CNetMessage::setPos(const unsigned int& newPos)

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				88
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    {
        msgh_uiCurrentPosition = newPos;
    }

byte* CNetMessage::getData(void)
{
    return msg_data;
}

CNetMessageHolder::CNetMessageHolder()
{
    bCleaned = 0;
    msgh_uiNum = 0;
}

CNetMessageHolder::~CNetMessageHolder()
{
    //if (!bCleaned) Clear();
    Clear();
}

void CNetMessageHolder::Clear()
{
    FOREACH_ARRAY(msgh_uiNum)
    {
        delete msgh_aIncommingMessages[i];
    }

    msgh_aIncommingMessages.clear();
    msgh_uiNum = 0;
    //bCleaned = true;
}

void CNetMessageHolder::setNew(int i)
{
}

CNetMessage* CNetMessageHolder::getMSG(int index)
{
    return msgh_aIncommingMessages[index];
}

void CNetMessageHolder::setMSG(int index)
{
}

};

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		89

Функція створення серверу

```

void startServer()
{
    if (net_iNumClients > NET_MAX_CLIENTS || net_iNumClients <= 0)
    {
        Throw("Trying to set clients %d, but max clients are %d",
net_iNumClients, NET_MAX_CLIENTS);
    }

    net_uiSequenceTick = 0;

    memset(&netDataGlobals, 0, sizeof(struct NetData));

    Printf("Start initialize server...\n");

    pubSendingData = (byte*)malloc(NET_MAX_DATASEND);

    if (pubSendingData == NULL)
    {
        Throw("Cannot create buffer data for sending msg");
    }

    memset(pubSendingData, 0, NET_MAX_DATASEND);

    // step 1 set up winsock
    WSADATA wsaData;

    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
    {
        Throw("Cannot open winsock");
    }

    Printf("Winsock opened\n");

    // step 2 get local address from pc
#ifdef NET_ENABLE_LAN
    ULONG bufferSize = 0;
    DWORD result = GetAdaptersAddresses(AF_INET, GAA_FLAG_INCLUDE_PREFIX,
nullptr, nullptr, &bufferSize);
    if (result != ERROR_BUFFER_OVERFLOW)
    {
        WSACleanup();
        Throw("GetAdaptersAddresses failed to get buffer size");
    }

    char* buffer = (char*)malloc(bufferSize);

    PIP_ADAPTER_ADDRESSES pAdapterAddresses =
reinterpret_cast<PIP_ADAPTER_ADDRESSES>(buffer);

    result = GetAdaptersAddresses(AF_INET, GAA_FLAG_INCLUDE_PREFIX, nullptr,
pAdapterAddresses, &bufferSize);
    if (result != ERROR_SUCCESS)
    {
        free(buffer);
        WSACleanup();
        Throw("GetAdaptersAddresses failed with error: %d", result);
    }
#endif
}

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        PrintF("Addresses: ");

        for (PIP_ADAPTER_ADDRESSES pAdapter = pAdapterAddresses; pAdapter !=
nullptr; pAdapter = pAdapter->Next)
        {
            if (pAdapter->OperStatus == IfOperStatusUp && pAdapter->
FirstUnicastAddress != nullptr)
            {
                for (PIP_ADAPTER_UNICAST_ADDRESS pUnicast = pAdapter->
FirstUnicastAddress; pUnicast != nullptr; pUnicast = pUnicast->Next)
                {
                    if (pUnicast->Address.lpSockaddr->sa_family == AF_INET)
                    {
                        sockaddr_in* pIPv4 =
reinterpret_cast<sockaddr_in*>(pUnicast->Address.lpSockaddr);
                        char ipAddress[INET_ADDRSTRLEN];
                        if (inet_ntop(AF_INET, &(pIPv4->sin_addr),
ipAddress, INET_ADDRSTRLEN) != nullptr)
                        {
                            PrintF("%s ", ipAddress);
                        }
                    }
                }
            }
        }

        PrintF("\n");

        free(buffer);

#endif

//SOCKET serverSocket = socket(AF_INET, SOCK_DGRAM, 0);
netDataGlobals.nd_socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (netDataGlobals.nd_socket == INVALID_SOCKET)
{
    //strAddresses.clear();

    WSACleanup();
    Throw("Socket creation failed: %s\n", lastErrorWSA());
}

u_long notBlock = 1;

if (ioctlsocket(netDataGlobals.nd_socket, FIONBIO, &notBlock) ==
SOCKET_ERROR)
{
    closesocket(netDataGlobals.nd_socket);
    WSACleanup();
    Throw("Failed to setup nonblocking socket: %s\n", lastErrorWSA());
}

// step 4 ssetup server address and client

netDataGlobals.serverAddr.sin_family = AF_INET;
#if 0//NET_ENABLE_LAN
inet_pton(AF_INET, "127.0.0.1", &netDataGlobals.serverAddr.sin_addr);
#else
netDataGlobals.serverAddr.sin_addr.s_addr = INADDR_ANY;
#endif
netDataGlobals.serverAddr.sin_port = htons(NET_PORT);

// step 5 bind socket

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				91
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if (bind(netDataGlobals.nd_socket, (sockaddr*)&netDataGlobals.serverAddr,
sizeof(netDataGlobals.serverAddr)) == SOCKET_ERROR)
        {
            closesocket(netDataGlobals.nd_socket);
            WSACleanup();
            Throw("Failed bind UDP socket %s\n", lastErrorWSA());
        }

        Printf("Server was started succesfull\n");

        Printf("-----\n");

        netDataGlobals.bOpenAsServer = true;
        netDataGlobals.bOpened = true;
        //netMainServer.

        memset(&netMainServer, 0, sizeof(struct NetServer));

        // in default max clients always 2, where first is server (local client)
        netMainServer.ns_numMaxClients = net_iNumClients;

        FOREACH_ARRAY(NET_MAX_CLIENTS)
        {
            netDataGlobals.clients[i].clientsSize =
sizeof(netDataGlobals.clients[i].clientsAddr);
            // netDataGlobals.clientsSize[i] = sizeof(netDataGlobals.clientsAddr[i]);
        }

        generateServerKeys();
    }

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				92
Змн.	Арк.	№ докум.	Підпис	Дата		

Заголовочний файл з мережевими функціями та протоколом

```

#pragma once

#pragma comment(lib, "IPHLPAPI.lib")
#pragma comment(lib, "ws2_32.lib")

/*
-----> Networks defines <-----
*/

/*

Enables lan network. This means we can connect from LAN/WLAN to server computer.

For debugging network game, just disable it

*/
#define NET_ENABLE_LAN 0

// version of protocol
#define NET_PROTOCOL_VERSION 0x96
// port
#define NET_PORT 5910
// max possible clients
#define NET_MAX_CLIENTS 8

#define NET_MAX_BUFFER 1441

#define NET_MAX_PACKET_SIZE \
((sizeof(byte)) + (sizeof(unsigned short)) + (sizeof(unsigned short)) + \
(sizeof(unsigned int)) + (sizeof(unsigned int)) + \
(sizeof(unsigned long long)))

#define NET_MAX_DATASEND (NET_MAX_BUFFER) + (NET_MAX_PACKET_SIZE)

/*
-----> Network command (NetCMD) <-----
*/

extern bool net_bCryptText;
extern bool net_bCryptAllData;
extern bool net_bCheckCRCData;
extern byte net_ubCryptType; // 0 - none, 1 - aes256, 2 - own algorithm
extern bool net_bDumpPacketToFile;

extern unsigned int net_uiSequenceTick;
extern int net_iNumClients;

enum eNetCMD
{
    // layer of network communication

    NET_CMD_BAD,
    NET_CMD_NONE,
    NET_CMD_CONNECT,
    NET_CMD_DISCONNECT_FROM_SERVER,
    NET_CMD_STOP_SERVER,
    NET_CMD_SEND_TEXT,

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				93
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    // layer of game/application

    NET_CMD_ACTION,
    NET_CMD_ADD_PLAYER,
    NET_CMD_REMOVE_PLAYER,
    NET_CMD_EVENT_CHANGE_SCORE
};

/* get last error WSA data */
extern const char* lastErrorWSA(void);

/*
-----> Network protocol flags <-----
*/

/*
Note:

flags in packet (uiFlags) generates from network message (ubNetCMD). Other flags,
generate from engine
network system, like which compression we use or encryption algorithm

*/

#define NET_UDP_RELIABLE ( 1u << 0 ) // reliable packets must have ACK
#define NET_UDP_UNRELIABLE ( 1u << 1 ) // unreliable packets cannot have ACK,
so engine dont care if we gets packet
#define NET_UDP_USE_ZLIB_COMPRESSION ( 1u << 2 ) // use ZLIB compression library
#define NET_UDP_USE_ZSTD_COMPRESSION ( 1u << 3 ) // use ZSTD compression library
#define NET_UDP_USE_CRYPT_AES256 ( 1u << 4 ) // crypt messages with aes256
#define NET_UDP_USE_CRYPT_OWN ( 1u << 5 ) // you can add own algorithm, but if we
have on both sides: server and client
#define NET_UDP_USE_CRC ( 1u << 6 )
#define NET_UDP_ACK ( 1u << 7 )
#define NET_UDP_NONE ( 0 )

#define NET_UDP_FULL \
((NET_UDP_RELIABLE) | (NET_UDP_USE_ZSTD_COMPRESSION) | \
(NET_UDP_USE_CRYPT_AES256) | (NET_UDP_USE_CRC))
/*
////////////////////////////////////

@brief namespace for network classes, functions
and structures

////////////////////////////////////
*/

namespace Network
{

#pragma pack(push, 1)
/*
@brief Main network protocol uses for Engine sending packets.

ITNP (In The Network Protocol)
*/
struct ITNP
{
    /* network cmd message */
    byte ubNetCMD;
    /* packet flags */
    unsigned short uwFlags;
    /* num of data send */

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				94
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        unsigned short uwNumData;
        /* checksum data */
        unsigned int uiCRC;
        /* number of sequence */
        unsigned int uiSequenceID;
        /* time when we send packet */
        unsigned long long ullTimeSend;
};
#pragma pack(pop)

typedef struct _ITNP ITNP_t;

struct NetClient
{
    sockaddr_in clientsAddr;    // address from client
    int clientsSize;           // size of client

    unsigned int lastSeqNumber; // last sequNumber of client

    unsigned long long time;    // time
};

struct NetData
{
    SOCKET nd_socket;           // main socket
    sockaddr_in serverAddr;     // server address

    struct NetClient clients[NET_MAX_CLIENTS]; // client on server, active only
on server side

    unsigned int uiLastTick;    // last sequence id

    bool bOpenAsServer;         // check if we server
    bool bOpened;               // check if we open socket
};

struct NetServer
{
    struct NetClient ns_Clients[NET_MAX_CLIENTS];
    int ns_numClients;          // current players count
    int ns_numMaxClients;      // max player count in server (ns_numMaxClients <=
NET_MAX_CLIENTS)
    long long ns_time;
};

/*
Network message. This can be like file i/o, but only for memory. This Sending
message data to server/client
communication from high level (<GameName>.dll) to low level (Engine.dll)

TODO: this supports only unsigneds integers, make support with negative values
*/
class ENGINE_API CNetMSG
{
public:

    unsigned int msg_uiNumData;    // num data of network message
    unsigned int msg_uiMaxData;    // max data of network message
    unsigned int msg_uiCurrentPosition; // current position of network message
    unsigned int msg_uiLastPosition; // last positions (uses for rewind)

    byte* msg_data;                // main data in bytes
    byte msg_netCMD;               // network command

public:

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				95
Змн.	Арк.	№ докум.	Підпис	Дата		

```

CNetMessage(void);
CNetMessage(size_t numData);
~CNetMessage(void);

inline void Clear();

inline void InitMessage(size_t numData);

inline void setData(byte* data, size_t numData);

inline void copyData(byte* data, size_t numData);

inline void setCMD(byte cmd) { msg_netCMD = cmd; }

/***** Writing methods *****/

inline void write8(const byte &b);
inline void write16(const unsigned short &s);
inline void write32(const unsigned int &i);
inline void write64(const unsigned long long &ll);

inline void writeFloat(const float &f);
inline void writeDouble(const double &d);
inline void writeString(std::string& str);
inline void writeBlock(byte* data, size_t num);
inline void writeVector(Vector3D& v);

/***** Reading methods *****/

inline void read8(byte* b);
inline void read16(unsigned short* s);
inline void read32(unsigned int* i);
inline void read64(unsigned long long* ll);
inline void readFloat(float* f);
inline void readDouble(double* d);
inline void readString(std::string& str);
inline void readBlock(byte* data, size_t num);
inline void readVector(Vector3D* v);

/***** I/O methods *****/

inline void rewindMSG(void);
inline void setPos(const unsigned int& newPos);
inline byte* getData(void);

void dumpHexToFile(const std::string &strFile);

inline unsigned int getLength() { return msg_uiNumData; }

CNetMessage& operator=(CNetMessage& other);
};
/*
Array holder of network message
*/
class CNetMessageHolder
{
public:
    int msgh_uiNum;
    // CNetMessage msg[NET_MAX_CLIENTS]; // FIXME: in GServer.cpp crash because data
    // can be NULL!!!
    std::vector<CNetMessage*> msgh_aIncommingMessages;

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				96
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        bool bCleaned;
public:
    ENGINE_API CNetMessageHolder();
    ENGINE_API ~CNetMessageHolder();

    ENGINE_API void Clear();

    ENGINE_API void setNew(int i);

    ENGINE_API CNetMessage* getMSG(int index);
    ENGINE_API void setMSG(int index);
};
// encrypt message
void Encrypt(byte* data, size_t size, byte* key, size_t keySize);
// decrypt message
void Decrypt(byte* data, size_t size, byte* key, size_t keySize);

bool encryptMessage(CNetMessage* msg);
bool decryptMessage(CNetMessage* msg, int iClient = 0);

// send Text to all clients
ENGINE_API void sendText(const char* strText);

/*
-----> Server interface <-----
*/

// initialiaze server
ENGINE_API void startServer(void); // see Server.cpp
// close server
ENGINE_API void closeServer(void); // see Server.cpp
// send data to clients
ENGINE_API void sendToClients(CNetMessage* msg);
ENGINE_API void sendToClients(CNetMessageHolder* msgHolder);
// send to client
ENGINE_API void sendToClient(CNetMessage* msg, int iClient);

// get data from clients
ENGINE_API void getPackets(CNetMessageHolder* holder);

ENGINE_API void listenClients(CNetMessageHolder* holder);

// check if we server
ENGINE_API inline bool isServer(void);

/*
-----> Client interface <-----
*/

ENGINE_API void startClient(void); // see Client.cpp
ENGINE_API void closeClient(void); // see Client.cpp

ENGINE_API void sendToServer(CNetMessage* msg);

ENGINE_API void getFromServer(CNetMessage* msg);

/*
-----> Connect/dissconnect <-----
*/

};

// initialiaze libsodium library

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				97
Змн.	Арк.	№ докум.	Підпис	Дата		

```

extern void initializeProtection();
// generate server keys
extern void generateServerKeys();
// generate client keys
extern void generateClientKeys();
// destroy server and clientkeys
extern void destroyKeys();
// exchange key on client side -> gets public server key
extern bool exchangeOnClientSideKey(Network::CNetMSG* msg);
// exchange key on server side -> gets public client key
extern bool exchangeOnServerSideKey(Network::CNetMSG* msg, unsigned int iClient);
// gets public key
extern ENGINE_API byte* getPublicKey(void);

/* check sum */
extern ENGINE_API unsigned int net_uiCRC;

extern Network::NetData netDataGlobals;

/* server structure, active only on server computer */
extern Network::NetServer netMainServer;
/* client structure, active only on client computer */
extern Network::NetClient netMainClient;
/* main buffer data sended to client/server */
extern byte* pubSendingData;

extern ENGINE_API char *strAddressString;

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		98

Комунікація між клієнтом та сервером

```

/*
-----> client - server communication in low level <-----
*/

struct Network::NetServer netMainServer;

#define ENABLE_OLD_COMMUNICATION_NET 0

unsigned int net_uiSequenceTick;
byte net_ubCryptType = 1;
unsigned int net_uiCRC = 0;
bool net_bCryptText = true;
bool net_bCryptAllData = false;
bool net_bCheckCRCData = false;
bool net_bDumpPacketToFile = false;

byte* pubSendingData = NULL;

namespace Network
{
    /*
    //////////////////////////////////////
    Server communication
    //////////////////////////////////////
    */

    void sendToClient(CNetMessage* msg, int iClient)
    {
        assert(msg != NULL);
        assert(msg->getData() != NULL);
        // if (msg == NULL) return;
        // if (msg->getData() == NULL) return;

        ITNP_t* packet = NULL;

        memset(pubSendingData, 0, NET_MAX_DATASEND);
        packet = (ITNP_t*)pubSendingData;
        packet->uiSequenceID = net_uiSequenceTick;
        packet->ubNetCMD = msg->msg_netCMD;
        packet->ullTimeSend = SDL_GetTicks64();
        packet->uiCRC = 0;

        if (packet->ubNetCMD == NET_CMD_SEND_TEXT && net_bCryptText &&
            !net_bCryptAllData)
        {
            if (net_ubCryptType == 1)
            {
                packet->uwFlags |= NET_UDP_USE_CRYPT_OWN;
                encryptMessage(msg);
            }
            else if (net_ubCryptType == 2)
                packet->uwFlags |= NET_UDP_USE_CRYPT_AES256;
            else
                packet->uwFlags |= 0;
        }

        if (net_bCryptAllData && !net_bCryptText)

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				99
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{
    if (net_ubCryptType == 1)
    {
        packet->uwFlags |= NET_UDP_USE_CRYPT_OWN;
        encryptMessage(msg);
    }
    else if (net_ubCryptType == 2)
        packet->uwFlags |= NET_UDP_USE_CRYPT_AES256;
    else
        packet->uwFlags |= 0;
}

packet->uwNumData = msg->getLength();

memcpy(pubSendingData + sizeof(ITNP_t), msg->getData(), msg->getLength());

//Printf("Sending to client[%d]...\n", iClient);

if (sendto(netDataGlobals.nd_socket, (char*)pubSendingData,
NET_MAX_DATASEND, 0, (sockaddr*)&netDataGlobals.clients[iClient].clientsAddr,
netDataGlobals.clients[iClient].clientsSize) == SOCKET_ERROR)
{
    Printf("^2Warning: cannot send to client[%i]: %s\n", iClient,
lastErrorWSA());
}

}

void listenClients(CNetMSGHolder* holder)
{
    if (net_uiSequenceTick == 0xFFFFFFFF) net_uiSequenceTick = 0;

    net_uiSequenceTick++;

    sockaddr_in addrWhere; // address where we getting data
    int iLenAddr = sizeof(sockaddr_in); // length of address
    int iBytes; // num of bytes, without size UDP
    CNetMSG msg; // msg for stuff things
    CNetMSG* pMsg = NULL;

    char buffer[1024]; // buffer, usually for text
    int j = -1;

    bool bResult = false;
    unsigned int c = 0;
    CFileStream fileDummy;

    while ( j <= 7 )
    {
        j++;

        memset(pubSendingData, 0, NET_MAX_DATASEND);

        iBytes = recvfrom(netDataGlobals.nd_socket, (char*)pubSendingData,
NET_MAX_DATASEND, 0, (sockaddr*)&addrWhere, &iLenAddr);

        if (iBytes <= 0) break;

        // because max clients 8
        unsigned short clId = 666;

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				100
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        /*          // find incoming address
        FOREACH_ARRAY(netMainServer.ns_numMaxClients)
        {
            if (memcmp(&netDataGlobals.clientsAddr[i], &addrWhere,
sizeof(sockaddr_in)) == 0)
            {
                clId = i;
                break;
            }
        }
        */

        FOREACH_ARRAY(netMainServer.ns_numMaxClients)
        {
            if (memcmp(&netDataGlobals.clients[i].clientsAddr, &addrWhere,
sizeof(sockaddr_in)) == 0)
            {
                clId = i;
                break;
            }
        }

        /*          // get incoming address
        if (clId == 666)
        {
            FOREACH_ARRAY(netMainServer.ns_numMaxClients)
            {
                if (netDataGlobals.clientsAddr[i].sin_port == 0)
                {
                    netDataGlobals.clientsAddr[i] = addrWhere;
                    clId = i;
                    break;
                }
            }
        }
        */

        if (clId == 666)
        {
            FOREACH_ARRAY(netMainServer.ns_numMaxClients)
            {
                if (netDataGlobals.clients[i].clientsAddr.sin_port == 0)
                {
                    netDataGlobals.clients[i].clientsAddr = addrWhere;
                    clId = i;
                    break;
                }
            }
        }

        // get info from packet
        if (clId != 666)
        {

            ITNP_t* packet = (ITNP_t*)pubSendingData;

            // if we have double packet, or packet has incorrect order,
just skip this
            if (packet->uiSequenceID <=
netDataGlobals.clients[clId].lastSeqNumber)
            {
                continue;
            }
            else
            {
                netDataGlobals.clients[clId].lastSeqNumber = packet-
>uiSequenceID;
            }
        }
    }
}

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				101
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }

    switch (packet->ubNetCMD)
    {
    case NET_CMD_CONNECT:

        holder->msg_h_uiNum++;

        pMsg = new CNetMessage;
        pMsg->copyData(pubSendingData + sizeof(ITNP_t), packet->
        uwNumData);

        //holder->msg[j].copyData(pubSendingData +
        sizeof(ITNP_t), packet->uwNumData);

        bResult = exchangeOnServerSideKey(pMsg, clId);

        pMsg->read32(&c);

        holder->msg_h_aIncommingMessages.push_back(pMsg);

        //holder->msg[j].read32(&c);

        if (c != net_uiCRC)
        {
            msg.InitMessage(64);
            memset(msg.msg_data, 0, 64 * sizeof(unsigned
            char));

            strcpy(buffer, "Request from server: bad CRC");
            msg.write32(strlen(buffer));
            msg.writeBlock((byte*)buffer, strlen(buffer));
            msg.setCMD(NET_CMD_BAD);

            sendToClient(&msg, clId);

            memset(&netDataGlobals.clients[clId].clientsAddr,
            0, sizeof(sockaddr_in));

            netDataGlobals.clients[clId].lastSeqNumber = 0;
        }
        if (!bResult)
        {
            msg.InitMessage(512);
            memset(msg.msg_data, 0, 512 * sizeof(unsigned
            char));

            strcpy(buffer, "Request from server: Bad key
            exchange");

            msg.write32(strlen(buffer));
            msg.writeBlock((byte*)buffer, strlen(buffer));
            msg.setCMD(NET_CMD_BAD);

            sendToClient(&msg, clId);

            memset(&netDataGlobals.clients[clId].clientsAddr,
            0, sizeof(sockaddr_in));

            netDataGlobals.clients[clId].lastSeqNumber = 0;
        }
        else
        {
            strcpy(buffer, "Request from server: client
            connected");

            msg.InitMessage(128);

            msg.writeBlock(getPublicKey(), 32 * sizeof(byte));
            msg.write16(clId);
            msg.write32(strlen(buffer));
            msg.writeBlock((byte*)buffer, strlen(buffer));

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				102
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        msg.setCMD(NET_CMD_CONNECT);

        sendToClient(&msg, clId);

    }

    break;
case NET_CMD_DISCONNECT_FROM_SERVER:

    memset(&netDataGlobals.clients[clId].clientsAddr, 0,
sizeof(sockaddr_in));

    netDataGlobals.clients[clId].lastSeqNumber = 0;

    pMsg = new CNetMessage;

    holder->msg_h_uiNum++;

    pMsg->copyData(pubSendingData + sizeof(ITNP_t), packet-
>uwNumData);

    pMsg->setCMD(NET_CMD_DISCONNECT_FROM_SERVER);
    holder->msg_h_aIncommingMessages.push_back(pMsg);

    break;
case NET_CMD_SEND_TEXT:

    if (net_bDumpPacketToFile)
    {
        fileDummy.open("System\\Data.pa", 0, 0);
        fileDummy.WriteToFile(pubSendingData,

        fileDummy.closeFile();
        net_bDumpPacketToFile = false;
    }

    holder->msg_h_uiNum++;

    pMsg = new CNetMessage;
    pMsg->copyData(pubSendingData + sizeof(ITNP_t), packet-
>uwNumData);

    pMsg->setCMD(NET_CMD_SEND_TEXT);
    if (packet->uwFlags & NET_UDP_USE_CRYPT_OWN)
    {
        bResult = decryptMessage(pMsg, clId);

        if (!bResult)
        {
            Printf("Invalid text message\n");
        }
        else
        {
            memset(buffer, 0, sizeof(buffer));
            pMsg->readBlock((byte*)buffer, pMsg-
>getLength());

            Printf("%s\n", buffer);
        }
    }
    else
    {
        memset(buffer, 0, sizeof(buffer));
        pMsg->readBlock((byte*)buffer, packet->uwNumData);
        Printf("%s\n", buffer);
    }

    holder->msg_h_aIncommingMessages.push_back(pMsg);
    break;

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				103
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        case NET_CMD_BAD:

            Printf("NET_CMD_BAD\n");

            break;
        default:

            pMsg = new CNetMessage;
            holder->msg_h_uiNum++;
            pMsg->copyData(pubSendingData + sizeof(ITNP_t), packet->uwNumData);

            pMsg->setCMD(packet->ubNetCMD);

            holder->msg_h_aIncommingMessages.push_back(pMsg);
            break;
        }
    }
}

/*
////////////////////////////////////

Client communication

////////////////////////////////////
*/

void sendToServer(CNetMessage* msg)
{
    //if (msg == NULL) return;
    assert(msg != NULL);

    if (net_uiSequenceTick == 0xFFFFFFFF) net_uiSequenceTick = 0;

    net_uiSequenceTick++;

    unsigned int length = msg->getLength() + sizeof(ITNP_t);

    if (length > NET_MAX_DATASEND)
    {
        Printf("Cannot send data, because size bigger than maximum possible
data\n");
        return;
    }

    memset(pubSendingData, 0, NET_MAX_DATASEND);

    ITNP_t* packet = (ITNP_t*)pubSendingData;

    packet->uiSequenceID = net_uiSequenceTick;
    packet->ubNetCMD      = msg->msg_netCMD;
    packet->ullTimeSend   = SDL_GetTicks64();
    packet->uiCRC          = 0;
    packet->uwFlags       |= NET_UDP_UNRELIABLE;

    // text always encrypted, but you can change this option
    if (packet->ubNetCMD == NET_CMD_SEND_TEXT && net_bCryptText &&
!net_bCryptAllData)
    {
        if (net_ubCryptType == 1)
        {
            packet->uwFlags |= NET_UDP_USE_CRYPT_OWN;

            encryptMessage(msg);
        }
    }
}

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				104
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
    else if (net_ubCryptType == 2)
        packet->uwFlags |= NET_UDP_USE_CRYPT_AES256;
    else
        packet->uwFlags |= 0;
}
// you can encrypt all data
if (net_bCryptAllData && !net_bCryptText)
{
    if (net_ubCryptType == 1)
    {
        packet->uwFlags |= NET_UDP_USE_CRYPT_OWN;
        encryptMessage(msg);
    }
    else if (net_ubCryptType == 2)
        packet->uwFlags |= NET_UDP_USE_CRYPT_AES256;
    else
        packet->uwFlags |= 0;
}
// this sends if we have packet unreliable
if (packet->ubNetCMD == NET_CMD_CONNECT)
{
    packet->uwFlags &= ~NET_UDP_UNRELIABLE;
    packet->uwFlags |= NET_UDP_RELIABLE;
}

packet->uwNumData = msg->getLength();

memcpy(pubSendingData + sizeof(ITNP_t), msg->getData(), packet->uwNumData);

if (sendto(netDataGlobals.nd_socket, (const char*)pubSendingData,
NET_MAX_DATASEND, 0, (sockaddr*)&netDataGlobals.serverAddr,
sizeof(netDataGlobals.serverAddr)) == SOCKET_ERROR)
{
    Printf("Cannot send data to server: %s\n", lastErrorWSA());
}

packet = NULL;
}

void getFromServer(CNetMSG* msg)
{
    //if (msg == NULL) return;

    ITNP_t* packet = NULL;

    sockaddr_in serverAddrNew;

    int size = sizeof(serverAddrNew);

    char buffer[1024];

    int bytes;

    double start = (double(SDL_GetTicks64()) * 0.001) + 10.0;

    while (1)
    {
        memset(pubSendingData, 0, NET_MAX_DATASEND);

        bytes = recvfrom(netDataGlobals.nd_socket, (char*)pubSendingData,
NET_MAX_DATASEND, 0, (sockaddr*)&serverAddrNew, &size);

        if (bytes > 0) break;
    }
}

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				105
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if (start < (double(SDL_GetTicks64()) * 0.001))
        {
            Warning("Server: request timeout\n");
            pConsoleDataBase->executeFunc("Quit()");
        }
    }
    packet = (ITNP_t*)pubSendingData;

    // if we have doubled packets, just ignore it, command of stop server must
be ignored
    if (packet->uiSequenceID <= netDataGlobals.uiLastTick && packet->ubNetCMD !=
NET_CMD_STOP_SERVER)
    {
        msg->InitMessage(15);
        memset(msg->msg_data, 0, sizeof(byte) * 15);
        msg->setCMD(NET_CMD_BAD);
        return;
    }
    else
    {
        netDataGlobals.uiLastTick = packet->uiSequenceID;
    }

    msg->copyData(pubSendingData + sizeof(ITNP_t), packet->uwNumData);
    msg->msg_netCMD = packet->ubNetCMD;

    if (msg->msg_netCMD == NET_CMD_CONNECT)
    {
        bool b = exchangeOnClientSideKey(msg);
        if (!b) msg->msg_netCMD = NET_CMD_BAD;
    }

    if (msg->msg_netCMD == NET_CMD_SEND_TEXT)
    {
        if (packet->uwFlags & NET_UDP_USE_CRYPT_OWN)
        {
            bool bResult = decryptMessage(msg);

            if (!bResult)
            {
                Printf("Invalid text message\n");
            }
            else
            {
                memset(buffer, 0, sizeof(buffer));
                msg->readBlock((byte*)buffer, packet->uwNumData);
                Printf("%s\n", buffer);
            }
        }
        else
        {
            memset(buffer, 0, sizeof(buffer));
            msg->readBlock((byte*)buffer, packet->uwNumData);
            Printf("%s\n", buffer);
        }
    }
}

};

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		106

3бip CRC32 для рівня

```

void createChecksumSession(unsigned int* crc)
{
    CFileStream fstream;

    if (!fstream.open(gameSession.gs_strFileName, 0, 1))
    {
        Throw("Cannot open world file \"%s\" for crc32",
gameSession.gs_strFileName);
    }

    CRC32::start(crc);

    fstream.calculateFileCRC(crc);

    for (unsigned int i = 0; i < gameSession.getWorld()->gw_numBrushes; i++)
    {
        CBrush* br = gameSession.getWorld()->gw_brushes[i];

        for (unsigned int j = 0; j < br->m_uPolygonsCount; j++)
        {
            CFileStream file;

            FileDeleteAbsolutePatch(br->m_brushPolygons[j]->tex01FileName);

            bool bResult = file.open(br->m_brushPolygons[j]->tex01FileName, 0, 1);

            if (!bResult) {
                Throw("Cannot open brush texture file \"%s\" for crc32", br-
>m_brushPolygons[j]->tex01FileName);
            }

            file.calculateFileCRC(crc);

            file.closeFile();

        }
    }

    for (unsigned int i = 0; i < gameSession.getWorld()->gw_numRooms; i++)
    {
        CRoomBase* room = &gameSession.getWorld()->gw_rooms[i];

        for (unsigned int j = 0; j < room->ro_polygons.size(); j++)
        {
            CFileStream file;

            FileDeleteAbsolutePatch(room->ro_polygons[j].tex01FileName);

            bool bResult = file.open(room->ro_polygons[j].tex01FileName, 0, 1);

            if (!bResult) {
                Throw("Cannot open room texture file \"%s\" for crc32", room-
>ro_polygons[j].tex01FileName);
            }

            file.calculateFileCRC(crc);

            file.closeFile();

        }
    }
}

```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				107
Змн.	Арк.	№ докум.	Підпис	Дата		

```
CRC32::end(crc);  
PrintF("CRC session: %u\n", *crc);  
fstream.closeFile();  
}
```

		Корнійчук В.В.			ККІКБ.КР.Б – 123 – 06 – ПЗ	Арк.
		Головня О.С.				108
Змн.	Арк.	№ докум.	Підпис	Дата		