

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Національний університет «Запорізька політехніка»

Інформаційна безпека та електронні комунікації  
(повне найменування факультету)

Кафедра інформаційних технологій електронних засобів  
(повне найменування кафедри)

## Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему: ДОСЛІДЖЕННЯ РОБОТИ ШТУЧНОГО ІНТЕЛЕКТУ ЯК  
ДОПОМІЖНОГО ІНСТРУМЕНТУ В РОЗРОБЦІ НАВЧАЛЬНИХ ІГОР, НА  
ПРИКЛАДІ МЕРЕЖЕВОГО ПРОЄКТУ  
STUDY OF THE WORK OF ARTIFICIAL INTELLIGENCE AS AN AUXILIARY  
TOOL IN THE DEVELOPMENT OF EDUCATIONAL GAMES, USING THE  
EXAMPLE OF A NETWORK PROJECT

Виконав: студент(ка) 2 курсу, групи БК-512м  
Спеціальності 172 Телекомунікації та  
радіотехніка

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інтелектуальні технології мікросистемної  
радіоелектронної техніки

ПАВЛЮЧЕНКО Р.В.

(ПРИЗВИЩЕ та ініціали)

Керівник ФАРАФОНОВ О.Ю.

(ПРИЗВИЩЕ та ініціали)

Рецензент КОРОЛЬКОВ Р.Ю.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Національний університет «Запорізька політехніка»

Факультет Інформаційна безпека та електронні комунікації  
Кафедра Інформаційні технології електронних засобів  
Ступінь вищої освіти магістр  
Спеціальність 172 «Телекомунікації та радіотехніка»  
(код і найменування)  
Освітня програма (спеціалізація) Інтелектуальні технології мікросистемної радіоелектронної техніки  
(назва освітньої програми (спеціалізації))

**ЗАТВЕРДЖУЮ**

В.о. завідувач кафедри ІТЕЗ, к.т.н, доц.

О.Ю. МАЛИЙ

“22” листопада 2023 року

**ЗАВДАННЯ**

**НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)**

ПАВЛЮЧЕНКО Ростислав Васильович

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження роботи штучного інтелекту як допоміжного інструменту в розробці навчальних ігор, на прикладі мережевого проєкту

Study of the work of artificial intelligence as an auxiliary tool in the development of educational games, using the example of a network project

керівник проєкту (роботи) к.т.н., ФАРАФОНОВ Олексій Юрійович,  
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові.)

затверджені наказом закладу вищої освіти від “21” листопада 2023 року № 448

2. Строк подання студентом проєкту (роботи) 27 грудня 2023 року

3. Вихідні дані до проєкту (роботи) \_\_\_\_\_

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Вступ, 1 Загальна частина, 2 Розробка мережевого проєкту, 3 Дослідження роботи штучного інтелекту.

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

## 6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
Розділи 1-3	ФАРАФОНОВ О.Ю., доцент	04.09.2023	08.12.2023
Нормоконтроль	ПОСПЕЄВА І. Є., ст.викладач	08.12.2023	19.12.2023

7. Дата видачі завдання «04» вересня 2023 року.

## КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Розробка мережевого проєкту	2 тиждень	Розділ 2
4	Робота зі штучним інтелектом	3-4 тижні	Розділ 3
5	Аналіз роботи штучного інтелекту.	5 тиждень	Розділ 3
6	Оформлення пояснювальної записки та відповідної документації	6-7 тиждень	Додатки
7	Нормоконтроль та рецензування.	8 тиждень	
8	Захист дипломної роботи.	9 тиждень	

Студент(ка)

Ростислав ПАВЛЮЧЕНКО  
( підпис ) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

Олексій ФАРАФОНОВ  
( підпис ) (Ім'я ПРИЗВИЩЕ)

## РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:  
с.101, табл. 0, рис. 17, дод. 2, джерел 12.

ШТУЧНИЙ ІНТЕЛЕКТ, ГЕНЕРАТИВНИЙ ШТУЧНИЙ ІНТЕЛЕКТ,  
UNREAL ENGINE, ONLINE SUBSYSTEM, BLUEPRINT, СЕРВЕР, HOST,  
CLIENT, VISUAL STUDIO, CHATGPT.

Об'єкт дослідження: Штучний інтелект як допоміжний інструмент в розробці навчальних ігор.

Предмет дослідження: Робота штучного інтелекту як інструменту, що надає підтримку у написанні коду для розробки мережевого проекту навчальної гри.

Мета дослідження - вивчення та оцінка ефективності використання штучного інтелекту як допоміжного інструменту в процесі розробки ігор з використанням мережевого проекту, зокрема, аналіз його внеску в полегшення написання коду.

Для реалізації мережевого проекту використовувалась мова програмування C++, програма Unreal Engine 5.3, середовище розробки Visual Studio 22, сервіс Epic Online Services.

В результаті був розроблений демонстраційний мережевий проект на основі якого проводились дослідження роботи штучного інтелекту.

## ABSTRACT

Explanatory note to the diploma qualifying work of the magister: 101 pages, 0 tables, 17 figures, 2 appendixes, 12 sources.

ARTIFICIAL INTELLIGENCE, GENERATIVE ARTIFICIAL INTELLIGENCE, UNREAL ENGINE, ONLINE SUBSYSTEM, BLUEPRINT, SERVER, HOST, CLIENT, VISUAL STUDIO, CHATGPT.

Research object: Artificial intelligence as an auxiliary tool in the development of educational games.

Research subject: The work of artificial intelligence as a tool that provides support in writing code for the development of a network project of an educational game.

The purpose of the study is to study and evaluate the effectiveness of using artificial intelligence as an auxiliary tool in the process of developing games using a network project, in particular, analyzing its contribution to facilitating the writing of code.

To implement the network project were used the C++ programming language, the Unreal Engine 5.3 program, the development environment Visual Studio 22, and the Epic Online Services service.

As a result, a demonstration network project was developed on the basis of which studies of the work of artificial intelligence were conducted.

## ЗМІСТ

Перелік скорочень та умовних позначень.....	6
Вступ.....	7
1 Загальна частина.....	8
1.1 Огляд тенденцій до розвитку та використання штучного інтелект.....	8
1.2 Використання штучного інтелекту у ігровій індустрії.....	11
1.3 Огляд Unreal Engine та його можливостей для розробки ігор.....	13
2 Розробка мережевого проекту.....	16
2.1 Визначення основної концепції гри, постановка цілі. Ігрові механіки....	16
2.2 Створення ігрового світу. Інструменти для моделювання.....	17
2.3 Мережева архітектура Unreal Engine.....	27
2.4 Базові класи рушію.....	33
2.5 Реалізація механік.....	45
2.6 UI та реалізація підключення гравців один до одного.....	53
2.7 Аналіз імплементації створення сесії OSS EOS .....	59
3 Дослідження роботи штучного інтелекту.....	64
3.1 Опис проблем та сценаріїв де можна використати штучний інтелект при розробці.....	64
3.2 Використання штучного інтелекту для написання коду реалізації підключення гравців один до одного.....	66
3.3 Аналіз відповідей штучного інтелекту на проблеми які виникли при розробці проекту.....	74
Висновки.....	87
Перелік джерел посилання.....	88
Додаток А Код класу ACyberFCharacter.....	89
Додаток Б Код класу ACyberFPlayerController.....	96

## ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ШІ - Штучний інтелект

ГШІ - Генеративний штучний інтелект

UE – Unreal Engine

BP – Blueprint

ПК – Персональний комп'ютер

OSS – Online Subsystem

EOS – Epic Online Services

## ВСТУП

Сучасний етап розвитку інформаційних технологій відзначений стрімким розширенням застосування штучного інтелекту в різноманітних галузях. Однією з перспективних областей використання штучного інтелекту є розробка ігор. Дана робота спрямована на дослідження можливостей штучного інтелекту у якості допоміжного інструменту в розробці навчальних ігор, на прикладі мережевого проекту.

Зростання інтересу до використання інноваційних технологій у сфері навчання та розвитку комп'ютерних ігор створює необхідність у вивченні можливостей ефективного використання штучного інтелекту в цьому контексті. На тлі цього, актуальність даного дослідження визначається потребою в розкритті потенціалу штучного інтелекту в якості інструменту для спрощення та оптимізації процесу створення навчальних ігор, зокрема з використанням мережевих проектів.

У цьому контексті, важливим аспектом є вивчення впливу штучного інтелекту на розробку коду навчальних ігор та визначення можливостей покращення їхньої функціональності. При цьому, розглядаються не лише аспекти поліпшення якості гри, але і можливості спрощення процесу розробки без прив'язки до особливостей навчання.

Даний дослідження має на меті розкриття потенціалу використання штучного інтелекту в якості допоміжного інструменту в розробці навчальних ігор з використанням мережевого проекту, що сприятиме подальшому розвитку сучасних методів навчання та збагаченню функціональних можливостей комп'ютерних ігор у навчальному середовищі.

## 1 ЗАГАЛЬНА ЧАСТИНА

### 1.1 Огляд тенденцій до розвитку та використання штучного інтелекту

#### 1.1.1 Визначення штучного інтелекту та генеративного штучного інтелекту

Штучний інтелект (ШІ) — це галузь інформатики, яка займається створенням програм та систем, здатних виконувати завдання, що зазвичай потребують людського інтелекту. Штучний інтелект орієнтований на розробку алгоритмів та моделей, які надають комп'ютерам здатність вчитися, приймати рішення, розуміти природну мову, взаємодіяти з оточенням та вирішувати завдання, що раніше вважалися характерними тільки для людей.

Генеративний штучний інтелект (ГШІ) — це підгалузь штучного інтелекту, яка фокусується на розвитку систем, спроможних генерувати новий контент або інформацію. Такі системи здатні створювати тексти, зображення, аудіо та інші дані, які можуть надавати враження творчості або творчої продуктивності. Прикладами генеративного штучного інтелекту є глибокі генеративні моделі, такі як генеративні прийняття, в яких модель може генерувати нові дані, користуючись вивченими закономірностями з навчального набору.

#### 1.1.2 Нові вражаючі генеративні моделі штучного інтелекту

Наприкінці 2022 року, світ дізнався про Chat GPT. Його можливості вразили багатьох та підштовхнули світові компанії до використання, власної розробки, генеративного штучного інтелекту. Нижче наведено приклади генеративних моделей ШІ.

**CLIP (Contrastive Language–Image Pre-training).** Розроблений OpenAI, CLIP дозволяє моделі аналізувати текст та зображення разом. Ця модель може розуміти контекст іншим чином, ніж попередні моделі, і вона здатна відрізнити зображення за текстовими описами та навпаки.

**DALL-E.** Інша розробка від OpenAI, DALL-E є генеративною моделлю, яка може створювати унікальні зображення на основі текстового опису. Наприклад: ви можете запросити йому створити зображення "черепашка, зроблена з карамелі".

**GPT-3 (Generative Pre-trained Transformer 3).** Від OpenAI, GPT-3 - це масштабна мовна модель, яка вражає своєю здатністю розуміти та генерувати людсько-подібний текст. Вона може виконувати завдання відповіді на питання, перекладу, створення коду та багато інших.

**StyleGAN 2.** Це покращення оригінальної моделі StyleGAN і використовується для генерації реалістичних зображень обличь.

**MuZero.** Розроблений DeepMind, MuZero вражає своєю здатністю вчитися і генерувати стратегії в середовищі, навіть без заздалегідь заданого правила гри.

Звісно світ не стоїть на місці та розвивається. З'являються нові генеративні моделі, оновлюються існуючі. Компанії інвестують у розробку штучного інтелекту, анонсують своє бачення ГШІ.

### 1.1.3 Де використовують штучний інтелект

Штучний інтелект використовується в багатьох сферах життя, від бізнесу та медицини до розваг та безпеки. Нижче наводиться приклади того, як ШІ використовується сьогодні.

У бізнесі ШІ використовується для автоматизації завдань, прийняття рішень та прогнозування тенденцій. Наприклад: а) ШІ використовується для обслуговування клієнтів; б) аналізу даних та ведення переговорів.

У медицині ШІ використовується для діагностики захворювань, розробки нових ліків та персоналізації лікування. Наприклад: а) ШІ використовується для аналізу медичних зображень; б) розробки нових вакцин та створення штучних органів.

У сфері розваг ШІ використовується для створення реалістичніших ігрових персонажів, генерування контенту та особистого досвіду споживачів.

Наприклад: а) ШІ використовується для створення штучного інтелекту в іграх; б) написання творчого контенту та персоналізації рекомендацій.

У сфері безпеки ШІ використовується для виявлення злочинів, попередження тероризму та захисту критичної інфраструктури. Наприклад: а) ШІ використовується для аналізу відео; б) розпізнавання обличчя та захисту від кібератак.

Ось деякі інші приклади того, як ШІ використовується сьогодні:

У фінансах ШІ використовується для оцінки ризику, управління портфелями та надання фінансових консультацій.

У виробництві. ШІ використовується для автоматизації завдань, оптимізації процесів та підвищення ефективності.

У транспортній сфері ШІ використовується для розробки самокерованих автомобілів, управління повітряним рухом та підвищення безпеки дорожнього руху.

У енергетиці. ШІ використовується для оптимізації використання енергії, прогнозування попиту та розвитку нових джерел енергії.

ШІ є швидко розвиваючою областю, і його використання продовжує розширюватися. Очікується, що ШІ буде мати все більший вплив на наше життя в майбутньому.

#### **1.1.4 Як Google використовує штучний інтелект**

Google використовує ШІ в багатьох своїх продуктах і послугах, щоб зробити їх більш корисними та інформативними для користувачів. Нижче наведено приклади того, як Google використовує ШІ.

У пошуку, Google використовує ШІ для розуміння запитів користувачів і надання найбільш релевантних результатів пошуку. Наприклад: а) ШІ використовується для аналізу мови запиту; б) визначення сенсу запиту та відбору найбільш релевантних веб-сторінок.

У перекладі, Google використовує ШІ для перекладу тексту з однієї мови на іншу. Наприклад: а) ШІ використовується для розуміння граматики та лексики двох мов; б) створення точного перекладу та адаптації перекладу до контексту;

У картах, Google використовує ШІ для надання користувачам актуальної інформації про навколишній світ. Наприклад: а) ШІ використовується для визначення поточних умов на дорозі; б) відстеження громадського транспорту та надання рекомендацій щодо маршрутів.

У Google Assistant, Google використовує ШІ для розуміння запитів користувачів і надання допомоги в різних завданнях. Наприклад: а) ШІ використовується для розпізнавання мови; б) відповіді на запитання та виконання команд.

У Google Фото, Google використовує ШІ для розпізнавання об'єктів, осіб та місць на фотографіях. Наприклад: а) ШІ використовується для створення альбомів; б) автоматичного сортування фотографій та додавання підписів до фотографій.

Google також використовує ШІ для розробки нових продуктів і послуг. Наприклад: Google працює над створенням самокерованих автомобілів, які використовують ШІ для навігації та уникнення перешкод.

Google вважає, що ШІ має потенціал змінити світ на краще. Компанія прагне використовувати ШІ для створення продуктів і послуг, які роблять життя людей більш зручним, продуктивним і безпечним.

## **1.2 Використання штучного інтелекту у ігровій індустрії**

Штучний інтелект широко використовується в ігровій індустрії, надаючи новаторські можливості для поліпшення геймплею та створення більш реалістичного віртуального світу. Інтелектуальні противники в іграх стають все більш складними завдяки алгоритмам поведінки, які навчаються від гравця та адаптуються до його стратегій. Графіка та анімації стають більш реалістичними завдяки використанню глибоких генеративних моделей. Голосові та мовні

інтерфейси взаємодії з гравцями також стають стандартом, а системи розпізнавання мови дозволяють відтворювати реалістичні голоси персонажів.

Штучний інтелект використовується для динамічної адаптації сценаріїв, реагуючи на вибори гравців та модифікуючи гру відповідно. Автоматизовані системи тестування використовують штучний інтелект для виявлення помилок та аналізу поведінки гравців. Генеративні алгоритми використовуються для створення унікального вмісту, такого як рівні, персонажі та предмети. Інструменти на основі штучного інтелекту сприяють автоматизації дизайну гри, дозволяючи розробникам швидше створювати геймплей та рівні. Системи рекомендацій персоналізують геймплей, враховуючи стиль гри та інтереси гравця.

Ці приклади відображають, як штучний інтелект трансформує ігрову індустрію, надаючи гравцям більш складні та персоналізовані геймплей-враження.

В межах ігрової індустрії штучний інтелект використовується для створення унікального мистецтва, генерації звуків та голосів, не обов'язково використовуючи генеративні моделі. Наприклад: а) алгоритми штучного інтелекту можуть бути використані для автоматизованого створення арт-об'єктів у грі; б) аудіо-ефектів та голосових ліній персонажів. Вони можуть адаптуватися до контексту гри та взаємодіяти з гравцем, надаючи більш імерсивний інтерфейс та сприяючи більш надійному відтворенню віртуального світу.

Наприклад: системи звукового дизайну, використовуючи алгоритми штучного інтелекту, можуть адаптувати акустичні ефекти до конкретних обставин у грі, змінюючи звучання в залежності від подій або дій гравця. У сфері мистецтва, штучний інтелект може допомагати розробникам генерувати текстури, фони та інші візуальні елементи, що створюють відчуття унікальності та естетичності гри.

Щодо голосового супроводу, системи розпізнавання мови та синтезу мови на основі штучного інтелекту можуть дозволити персонажам реагувати на

гравців, адаптувати свої висловлювання до ситуацій та навіть навчатися від взаємодії з гравцем.

Ці приклади віддзеркалюють варіативні застосування штучного інтелекту для творчих аспектів в ігровій індустрії, надаючи гравцям більш глибокий іммерсивний досвід.

### **1.3 Огляд Unreal Engine та його можливостей для розробки ігор**

#### **1.3.1 Unreal Engine, що це, та для чого**

Unreal Engine (UE) - це ігровий рушій (game engine), який розробляється і підтримується компанією Epic Games. Рушій створений для розробки та створення ігор, а також для реалізації віртуальної реальності, архітектурного візуалізації, тренування симуляцій та інших візуалізаційних проектів [1]. Нижче наведено основні характеристики та можливості Unreal Engine.

**Графіка та візуалізація.** Unreal Engine вражає своєю здатністю генерувати високоякісну графіку, включаючи реалістичні ефекти світла, тіні, води та інші візуальні аспекти. Він підтримує різні функції рендерингу, включаючи рейтрейсінг (ray tracing).

**Множинна платформена підтримка.** Unreal Engine дозволяє розробляти ігри та додатки для різних платформ, таких як ПК, консолі, мобільні пристрої та віртуальна реальність.

**Мова програмування.** Unreal Engine використовує мову програмування C++, що дозволяє розробникам створювати складні та потужні ігрові системи. Також має візуальну мову програмування Blueprint.

**Вбудовані інструменти розробки.** Unreal Engine має широкий набір вбудованих інструментів розробки, таких як редактор матеріалів, система фізики, система анімації, система штучного інтелекту, та інші.

**Спільнота та ресурси.** Unreal Engine користується великою та активною спільнотою розробників. Крім того, є багато документації, онлайн-ресурсів та навчальних матеріалів для підтримки новачків.

**Реальний час.** Unreal Engine відомий своєю здатністю працювати в реальному часі, що дозволяє розробникам швидко переглядати та вдосконалювати геймплей та візуальні ефекти.

Unreal Engine став широко використовуваним інструментом у галузі розробки відеоігор та візуалізаційних проектів, завдяки своїй потужності та гнучкості.

### 1.3.2 Переваги Unreal Engine

Особисто для себе я виділяю такі переваги:

- UE є безкоштовним для початківців, що дає змогу вивчати програму та робити проекти цілком легально та у повному обсязі;
- UE має відкритий початковий код, що дає можливість глибокого вивчення рушія та його подальшого модифікування;
- UE має багато вбудованих інструментів, які йдуть одразу з програмою;
- багато ігор рівня AAA (з великим бюджетом на розробку).

Слід зазначити, що Unreal Engine це важка програма, яка потребує багато місця на жорсткому диску та обчислювальних можливостей персонального комп'ютера (ПК).

### 1.3.3 Чому саме Unreal Engine

Я маю достатній досвід роботи з Unreal Engine, включаючи комерційний. В мене є розуміння архітектури та досвід роботи с мережевими проектами, та досвід створення мережових проектів. Опираючись на пункти 1.3.1 та 1.3.2, та маючи власний досвід роботи з програмою - я зможу оцінити роботу штучного інтелекту при розробці.

Так як UE велика програма з величезною кількістю коду, який є відкритим та у повному вільному доступі, використання штучного інтелекту як

допоміжного інструменту може дати необхідну інформацію швидше ніж її шукати у документації, або на інших інформаційних ресурсах.

## **2 РОЗРОБКА МЕРЕЖЕВОГО ПРОЕКТУ**

### **2.1 Визначення основної концепції гри, постановка цілі. Ігрові механіки**

#### **2.1.1 Концепція та цілі**

Дивлячись різноманітні відеоролики на ігрову тематику мені попадались відео з грою, де ігрові персонажі, якими керують люди, грають у невеликий футбол. Правила прості, треба використовуючи ігрові механіки, заштовхати м'яч у ворота противника та усіма можливими засобами перешкоджати попаданню м'яча у свої ворота. Для свого мережевого проекту я обрав саме таку ідею для реалізації, але дещо у спрощеному вигляді.

Ціль даного розділу познайомитись у загальних рисах з процесом створення мережевого проекту у рушії Unreal Engine, виявлення проблемних місць де треба шукати додаткову інформацію у документації, на форумах. Подалі, у 3 розділі буде проаналізовано роботу генеративного штучного інтелекту, як допоміжного інструменту у пошуку інформації, відповідей на проблемні питання.

#### **2.1.2 Ігрові механіки**

Обрана концепція гри передбачає наявність декількох ключових ігрових механік, які роблять її унікальною, та захоплюючою для гравців. Основна ідея полягає в тому, щоб взаємодіяти з ігровим світом, використовуючи різноманітні техніки, для досягнення цілі, а сама: а) забити м'яч у ворота суперника; б) водночас заважати противнику забити м'яч.

Ходьба та прискорення відкривають перед гравцем можливість вільно пересуватися по полі та швидко реагувати на гострі геймплейні ситуації. Проте, це пов'язано з витратою витривалості, що вимагає від гравця ретельно розраховувати свої дії.

Важливим аспектом є також механіка відновлення витривалості, яка вимагає від гравця стратегічного управління своїми ресурсами та обережного використання прискорення для досягнення оптимальних результатів.

Найбільше емоцій та тактичних рішень вносить механіка удару по м'ячу. Гравець має можливість вдарити по м'ячу, що надає грі стратегічний компонент та робить кожен раунд непередбачуваним.

Загалом обрані ігрові механіки створюють цікавий та динамічний геймплей, сприяючи взаємодії гравців та створюючи унікальний характер проекту.

Отже гра буде уявляти собі невелике змагання між двома гравцями які грають по інтернету. Ціль гри – забити більше м'ячів у ворота противника та захищати свої. Гравці можуть штовхати та бити по м'ячу. Щоб наздогнати м'яч чи супротивника, гравець може на короткий час прискоритись. Прискорення не можна використовувати постійно, повинен пройти певний час для його відновлення. Наявність втомлює робить гру унікальною та відрізняє її від схожих ігор.

## **2.2 Створення ігрового світу. Інструменти для моделювання**

### **2.2.1 Підготовка до роботи**

Для реалізації проекту буде використовуватись Unreal Engine 5.3 [1]. Також потрібні необхідні версії бібліотек та середовище розробки. Детальніше можна ознайомитись за документацією [2]. Ось необхідний перелік:

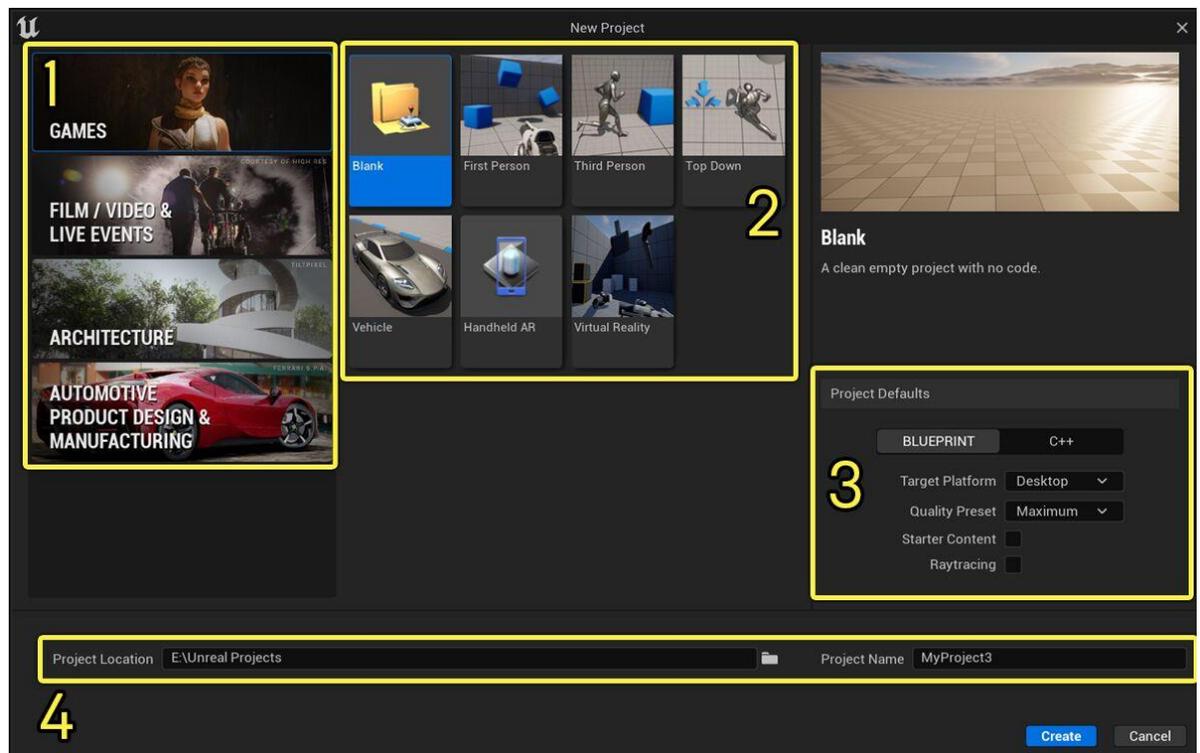
- Visual Studio 2022 v17.4 або новіше;
- Windows SDK 10.0.18362 або новіше;
- LLVM clang 14.0.1;
- .NET 4.6.2 Targeting Pack;
- .NET 6.0.

## 2.2.2 Створення проекту

Коли ви запускаєте Unreal Engine, браузер Unreal Project Browser відкривається автоматично. Тут ви можете:

- створить новий проект;
- відкрийте існуючий проект;
- керуйте існуючими проектами.

На рисунку 2.1 показано кроки для створення нового проекту в Unreal Engine.



- 1- категорія розробки;
- 2- шаблони проектів;
- 3- налаштування проекту;
- 4- шлях для розміщення проекту.

**Рисунок 2.1 - Створення нового проекту в Unreal Engine з вікна браузера проектів**

Для проекту я обрав категорію розробки – «Games», шаблон проекту - «Third Person», у налаштуваннях обрав C++ та вказав зручний шлях для розміщення проекту [3].

Слід зазначити, що якщо створити проект на Blueprint (BP) (див. рис. 2.1), то це буде цілком проект з використанням мови графічного програмування Blueprint. Не зважаючи на це, у подальшому часі, можна створити C++ файли, та використовувати їх. Але якщо створити C++ проект, то не можливо буде повернутися до виключно Blueprint типу проекту.

Blueprint та C++ код добре поєднуються та взаємодіють один з одним. Blueprint класи створюються як спадкоємці C++ класів. Вони можуть викликати функції які створені у C++, бачити та змінювати змінні, перевизначити методи з C++ коду. Звісно у Blueprint можна створювати свої змінні, функції. Але простого зворотного виклику з C++ коду, методу який створений тільки у BP немає. Тому якщо потрібно викликати метод у C++ коді, логіка якого буде у Blueprint-і, з початку треба об'явити її у C++ коді, та зробити певні позначення. Детальніше буде розписано у підрозділі 2.4.

### 2.2.3 Інтерфейс Unreal Engine

Після створення проекту відкривається основна частина програми Unreal Engine. Детальніше можна ознайомитись із інтерфейсом у документації [4]. Нижче наведено опис основних елементів інтерфейсу.

**Рядок меню.** Використовуйте ці меню для доступу до команд і функцій редактора.

**Головна панель інструментів.** Містить ярлики для деяких найпоширеніших інструментів і редакторів у Unreal Engine, а також ярлики для входу в режим відтворення (запуск гри в Unreal Editor) і для розгортання вашого проекту на інших платформах.

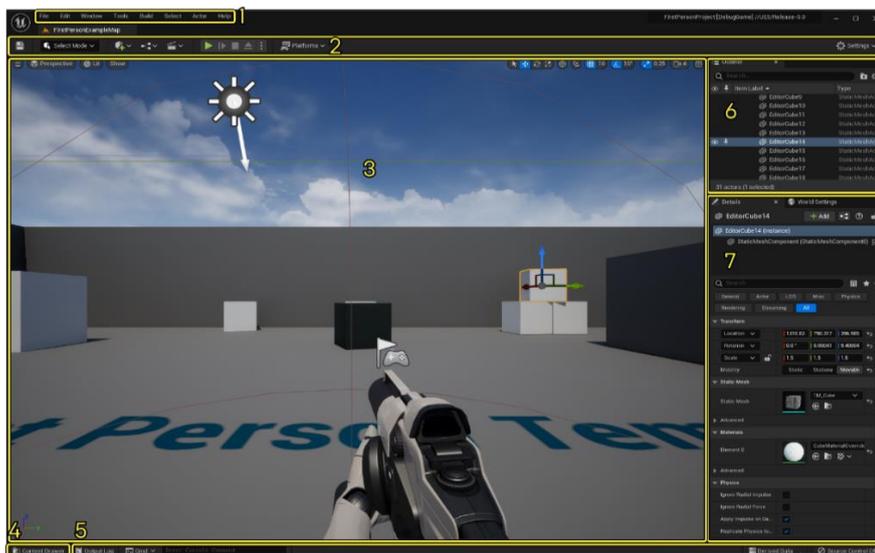
**Видове вікно рівня.** Відображає вміст вашого рівня, камери, актори, статичні сітки тощо.

**Кнопка ящика вмісту.** Відкриває панель вмісту, з якої можна отримати доступ до всіх ресурсів у вашому проекті.

нижня панель інструментів. Містить ярлики для командної консолі, журналу виводу та функцій похідних даних. Також відображає статус керування джерелом.

**Контурник.** Відображає ієрархічне дерево всього вмісту вашого рівня;

Панель деталей. З'являється, коли ви вибираєте актора. Відображає різні властивості цього актора, наприклад його: а) трансформацію (положення на рівні); б) статичну сітку; в) параметри матеріалу та фізики. На цій панелі відображаються різні налаштування залежно від того, що ви вибрали у вікні перегляду рівня.



- 1- рядок меню;
- 2- головна панель інструментів;
- 3- видове вікно рівня;
- 4- кнопка ящика вмісту;
- 5- нижня панель інструментів;
- 6- контурник;
- 7- панель деталей.

**Рисунок 2.2 - Інтерфейс Unreal Editor за замовчуванням у Unreal Engine 5**

Головна панель інструментів містить ярлики деяких інструментів і команд, які найчастіше використовуються в Unreal Editor. Нижче наведено опис на які області поділяється ця панель.

**Кнопка збереження.** Натисніть цю кнопку, щоб зберегти поточний відкритий рівень.

**Вибір режиму.** Містить ярлики для швидкого перемикання між різними режимами для редагування вмісту на вашому рівні:

- виберіть Редагування;
- пейзажне редагування;
- редагування листа;
- розпис сіткою;
- редагування переломів;
- редагування пензля.

**Ярлики вмісту.** Містить ярлики для додавання та відкриття поширених типів вмісту в редакторі рівнів, а саме:

- створити;
- креслення;
- кінематограф.

**Елементи керування режимом відтворення.** Містить кнопки швидкого доступу («Відтворити», «Пропустити», «Зупинити» та «Вилучити») для запуску гри в редакторі.

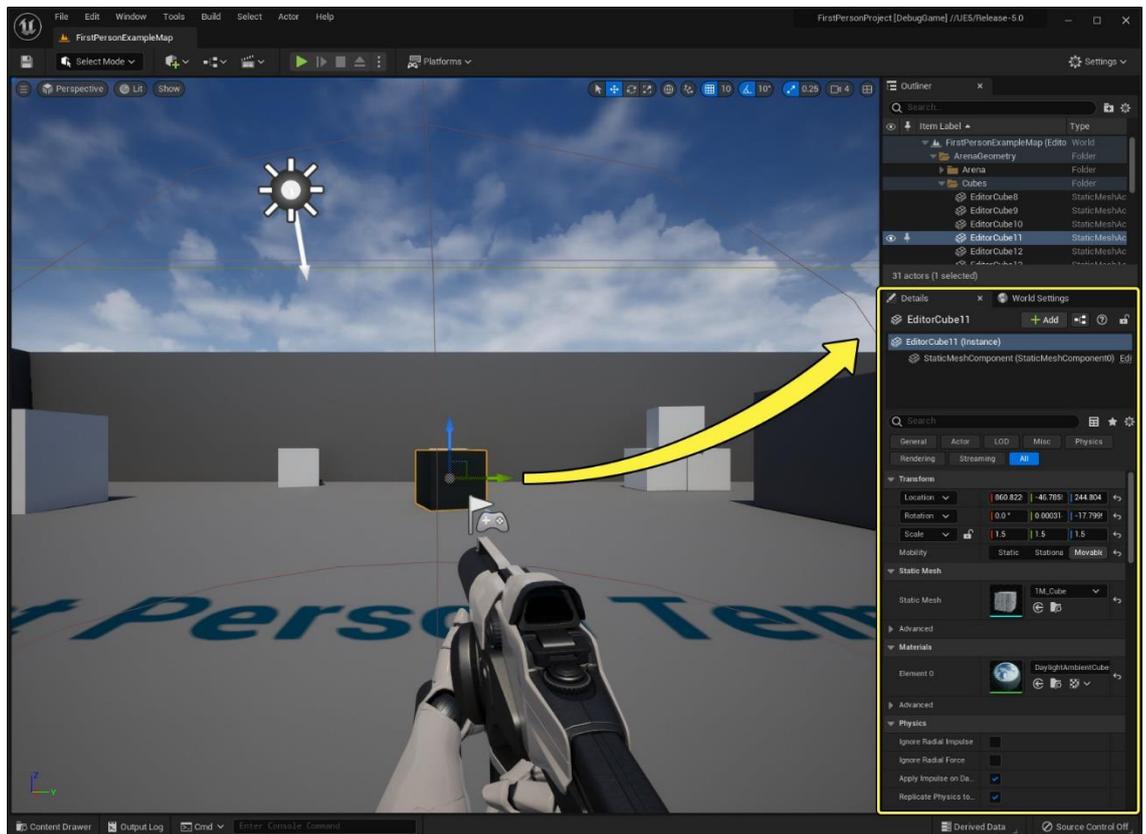
**Меню платформ.** Містить серію параметрів, які можна використовувати для налаштування, підготовки та розгортання вашого проекту на різних платформах, таких як робочий стіл, мобільний пристрій або консолі.

**Налаштування.** Містить різні параметри для редактора Unreal Editor, вікна перегляду редактора рівнів і поведінки гри.



- 1- кнопка збереження;
- 2- вибір режиму;
- 3- ярлики вмісту;
- 4- елементи керування режимом відтворення;
- 5- меню платформ;
- 6- налаштування.

**Рисунок 2.3 – Головна панель інструментів**



**Рисунок 2.4 – Панель деталей**

Панель деталей яка зображена на рисунку 2.4 може відрізнятися за змістом в залежності від типу обраного об'єкту. Параметрів може бути більше або менше.

## 2.2.4 Інструменти для моделювання

Моделювання світу є невід'ємною та важливою частиною розробки відеоігор, оскільки воно визначає основні параметри та характеристики вигаданого віртуального простору, в якому відбудеться весь геймплей. Введення моделювання світу в геймдев проект дозволяє досягти кількох ключових мет цієї стадії розробки.

По-перше: моделювання дозволяє розробникам створювати віртуальні світи, які є концептуально та естетично цілісними. Це означає, що гравець може заглибитись у ігровий світ, який вражає своєю атмосферою та деталями. Моделювання дозволяє створити реалістичні або фантастичні ландшафти, архітектуру та об'єкти, що формують неповторний візуальний досвід.

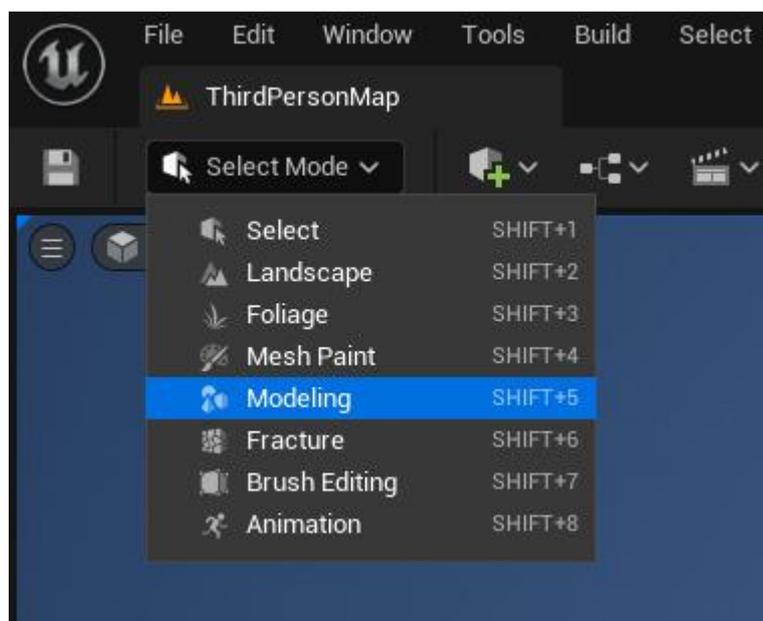
По-друге: моделювання дозволяє розробникам ефективно керувати ресурсами гри. Правильно створені та оптимізовані моделі сприяють оптимальному використанню пам'яті та процесорної потужності, забезпечуючи плавну та ефективну гру.

По-третє: моделювання є ключовим етапом у визначенні геймплейних параметрів та правил взаємодії гравців із світом. Від добре створених моделей залежить реалістичність та динаміка геймплею, а також можливість реалізації різноманітних інтерактивних елементів.

Щоб досягти цих цілей, використання відповідних інструментів для моделювання стає важливою складовою розробки відеоігор. Unreal Engine має власні інструменти для моделювання [5].

По за замовчуванням вони можуть бути вимкнені, тому їх потрібно увімкнути, для цього користуючись рядком меню, потрібно відкрити вікно з додатками (plugins), та знайти потрібний додаток. Ось шлях: Edit > Plugins > Modeling Tools Editor Mode.

Далі треба перезапустити рушій (запропонує). Після чого можна вибрати відповідний режим у панелі інструментів, як показано на рисунку 2.5.



**Рисунок 2.5 – Вибір режиму з панелі інструментів**

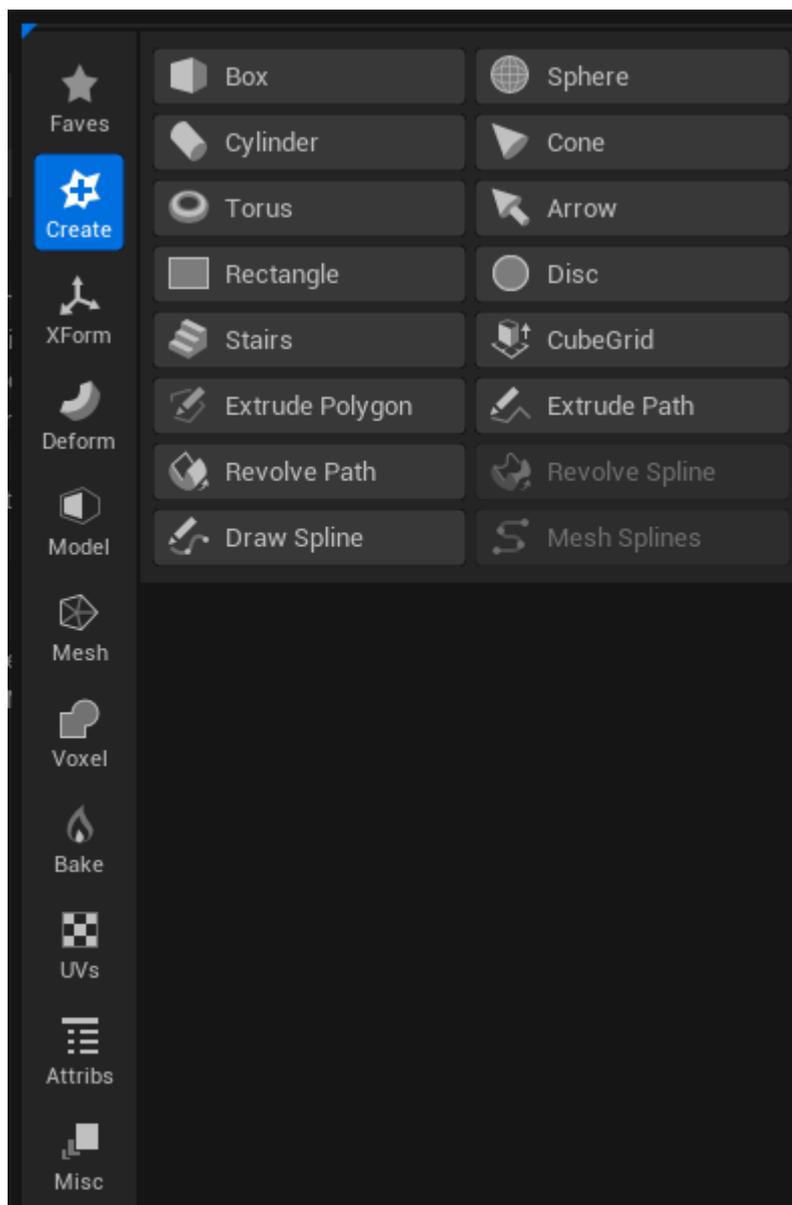
Modeling Mode в Unreal Engine 5 — це режим, який дозволяє користувачам створювати та редагувати 3D-моделі. Він пропонує широкий спектр інструментів та функцій, які роблять його потужним інструментом для створення 3D-вмісту. Нижче наведено перелік основних функції Modeling Mode.

**Створення базових форм.** Ви можете використовувати інструменти, такі як сплайни, полігони та маніпулятори, щоб створювати базові форми для ваших моделей.

**Редагування деталей.** Ви можете використовувати інструменти, такі як полігональна модифікація, скульптура та текстурювання, щоб редагувати деталі ваших моделей.

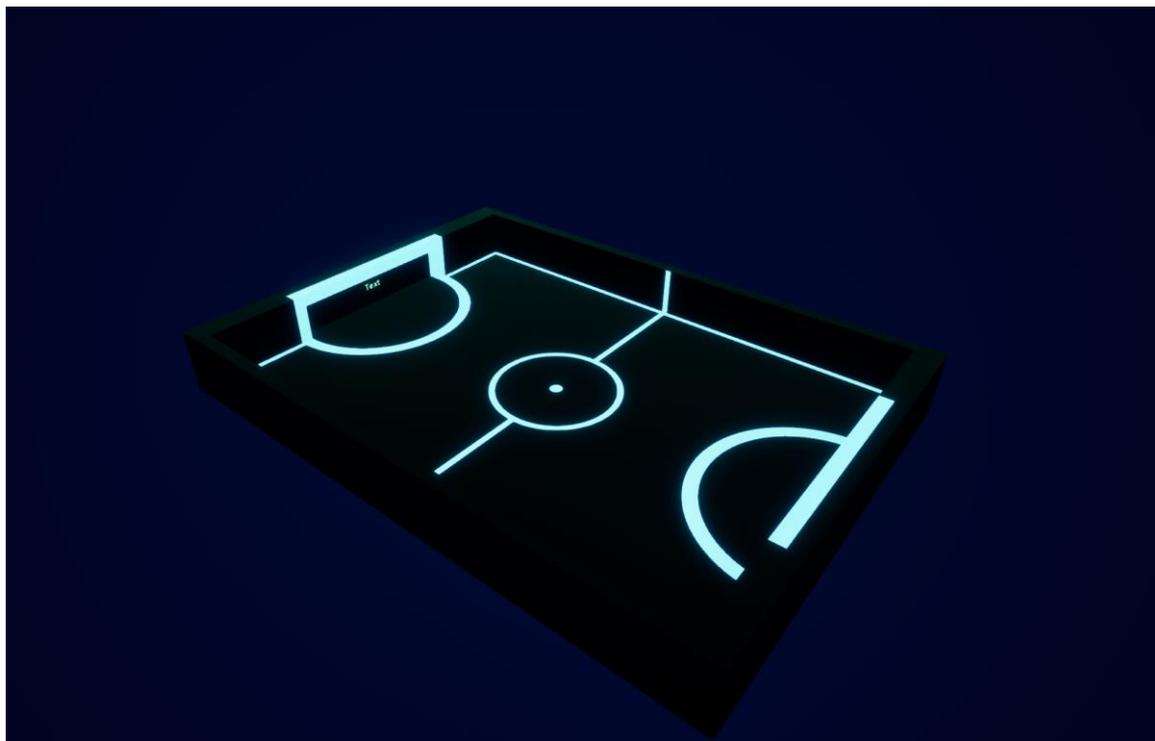
**Імпорт та експорт.** Ви можете імпортувати та експортувати 3D-моделі з інших програм, таких як Maya або Blender.

Modeling Mode — це потужний інструмент, який можна використовувати для створення різноманітних 3D-моделей. Він ідеально підходить для створення персонажів, декорацій та інших об'єктів для ігор, фільмів та інших медіа.



**Рисунок 2.6 – Панель інструментів Modeling Mode**

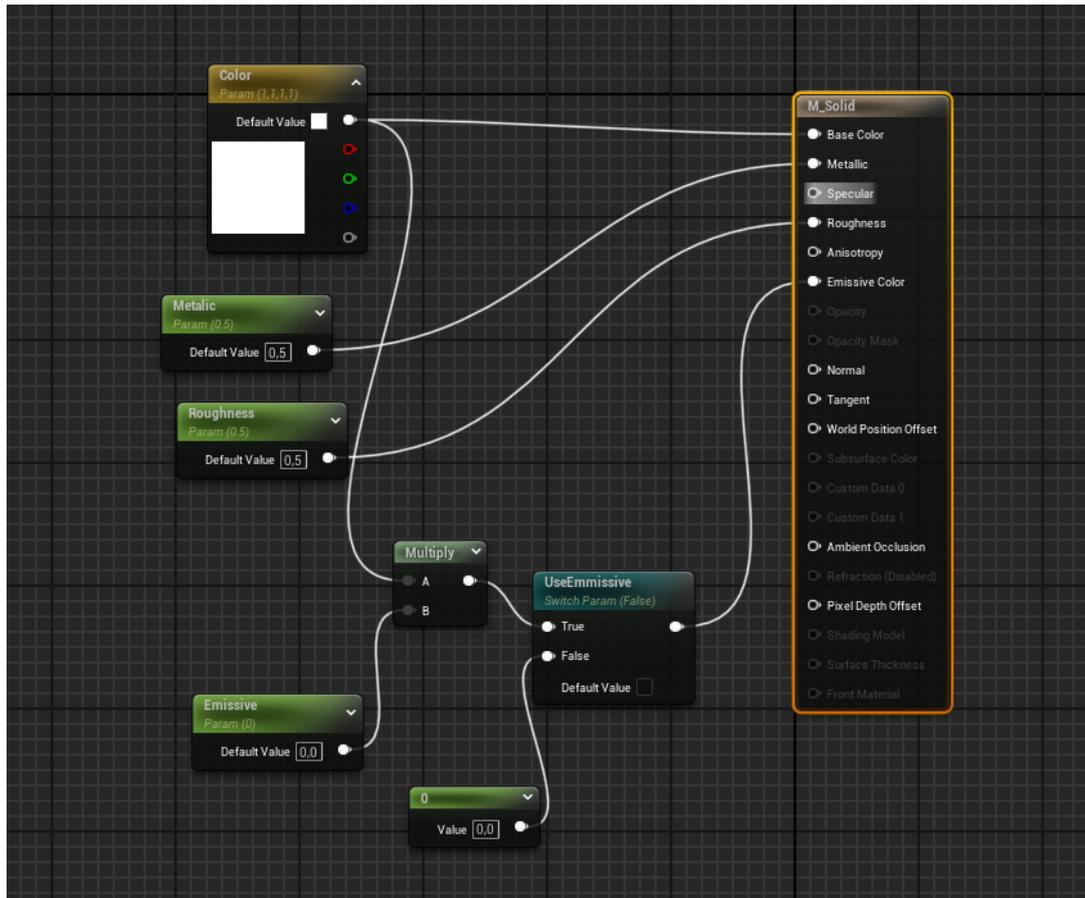
Modeling Mode має власну панель інструментів та режим роботи, детальніше можна ознайомитись у документації [5]. Цей режим наразі у статусі «Beta», тобто можливі помилки при роботі. Для реалізації моєї задумки я користувався примітивами з вкладки «Create», та «CubeGrid».



**Рисунок 2.7 – Ігрова арена**

На рисунку 2.7 продемонстрована ігрова арена яку я зробив за допомогою режиму моделювання (Modeling Mode). Звісно можна було обійтися без цього режиму та використати готові примітиви, але так в нас є більше контролю при розробці.

До моделей вже назначені необхідні матеріали, що відповідають за її зовнішній вигляд (колір, коефіцієнт металевості тощо). Цей розділ у Unreal Engine я не розглядатиму, детальніше можна дізнатись у документації [6].



**Рисунок 2.8 – Логіка базового матеріалу**

Для арени я створив один базовий матеріал, з якого зробив «копії» де відрізняються параметри, що роблять «копію» унікальною за виглядом, але такою ж самою за логікою. Логіка базового матеріалу зображена на рисунку 2.8.

### 2.3 Мережева архітектура Unreal Engine

Перш ніж перейти до реалізації механік, треба познайомитись із мережевою архітектурою у Unreal Engine. Це дуже важливо, бо ми повинні розуміти як працює передача даних між гравцями, як виконується логіка яку викликали з іншого ПК, так як взагалі викликати віддалене спрацювання коду. Ці знання полегшують нам роботу та зекономлять час на розробку коду, переробку, оскільки знаючи про архітектуру, ми можемо спроектувати, продумати, логіку реалізації механік.

### 2.3.1 основні терміни та поняття

Я надам спрощене пояснення термінів якого в цілому буде достатньо для розуміння. Для деяких термінів виділені спеціальні пункти де більш детально розкривається термін.

Хост (Host) – Гравець, який виступає у ролі сервера.

Клієнт – Гравець, який підключається до сервера, хосту.

Online Subsystem (OSS) – це інтерфейс із відповідною реалізацією, який забезпечує загальний спосіб доступу до функцій онлайн-сервісів, таких як:  
а) Steam; б) Xbox Live; в) Facebook тощо [7].

Реплікація (Replication) – Механізм передачі даних та синхронізації між гравцями, між клієнтами та сервером.

### 2.3.2 Online Subsystem

Для кращого розуміння що таке Online Subsystem (OSS), треба почати з далеку. Для роботи інтернету є основні відомі протоколи IP, TCP, UDP. Це самий нижчий рівень. Однак для програміста перший рівень абстракції який доступний для реалізації інтернет з'єднання це Сокет (Socket).

Сокети (англ. socket - заглиблення, гніздо, роз'єм) — назва програмного інтерфейсу для забезпечення обміну даними між процесами. Процеси при такому обміні можуть виконуватися як на одній ЕОМ, так і на різних ЕОМ, пов'язаних між собою мережею. Сокет - абстрактний об'єкт, що представляє кінцеву точку з'єднання.

Сокет це програмний інтерфейс реалізація якого знаходиться на боці операційної системи. Тобто для кожної платформи (UE дозволяє робити ігри на різні платформи) буде свій інтерфейс специфічний для кожної із платформ.

За вибір та роботу по передачі даних у Unreal Engine відповідає клас UNetDriver. Він також може бути специфічним для платформи. Грубо кажучи

цей клас, це обгортка рушія над викликом системних функцій з бібліотеки для певної операційної системи.

Також є сторонні сервіси та їх бібліотеки які використовують інтернет з'єднання та дозволяють створювати, знаходити ігрові сесії та приєднуватись до них. І ось інтерфейс Online Subsystem гарантує що, незалежно від платформи (реалізації під платформу), єдині зміни, які розробники повинні внести, це коригування конфігурації для кожної підтримуваної служби.

Тож коли ми викликаємо метод із створенням сесії з коду гри, через інтерфейс OSS, ми викликаємо певний метод певної реалізації. У той реалізації можуть бути специфічні виклики для платформи, та виклики які в кінцевому шляху звертаються до класу UNetDriver та відправляють данні через інтернет.

OSS – це великий інтерфейс, який включає в себе інші інтерфейси для реалізації певних можливостей [7]. Наприклад інтерфейс для створення сесій, що дозволяє:

- створювати сесії;
- знаходити сесії;
- приєднуватись до сесії;
- відправляти запрошення на приєднання, тощо.

Для наших потреб знадобиться OSS Null – для початкової реалізації інтернет з'єднання між гравцями, OSS EOS (Epic Online Services) – для можливості підключення через глобальну мережу, інтерфейси: а) IOnlineSessionInterface, IOnlineIdentityInterface.

Підводячи підсумки Online Subsystem дає змогу розробнику використовувати функції мережевого з'єднання не замислюючись над тим яка реалізація у середині.

### **2.3.3 Реплікація**

Реплікація у рушії Unreal Engine є важливою та потужною функцією, яка дозволяє синхронізувати дані між клієнтом та сервером у мережевих іграх. Цей

процес грає критичну роль у забезпеченні коректності та консистентності гри для всіх учасників.

На практиці, реплікація дозволяє відтворювати деякі об'єкти та події в мережевому середовищі так, щоб кожен гравець отримувал актуальну інформацію про події, що відбуваються в грі. Це може стосуватися руху ігрових об'єктів, змін в стані персонажів, або інших ігрових аспектів.

Реплікація в Unreal Engine використовує концепцію "реплікованих змінних" та "реплікованих функцій". Репліковані змінні дозволяють автоматично синхронізувати значення між клієнтом та сервером, забезпечуючи єдність стану об'єктів у грі. Репліковані функції, з свого боку, дозволяють викликати функції на віддаленому клієнті чи сервері, забезпечуючи спільні ігрові події.

Цей механізм дозволяє створювати мережеві ігри зі збалансованим та стабільним геймплеєм, де кожен учасник отримує адекватну інформацію про те, що відбувається в ігровому світі. Реплікація в Unreal Engine є ключовим елементом для створення ігор, які надають враження від гри в реальному часі в мережевому середовищі.

#### **2.3.4 Реплікація змінних**

Реплікація змінних - це процес передачі інформації про змінні з одного клієнта до іншого в багатокористувацькій грі. В Unreal Engine реплікація змінних реалізована за допомогою системи, яка називається Replicated Variable.

Система Replicated Variable працює за наступним принципом:

- сервер створює змінну і присвоює їй унікальний ідентифікатор;
- сервер зберігає інформацію про змінну у своїй базі даних;
- сервер періодично відправляє клієнтам інформацію про змінені змінні;
- клієнти отримують інформацію про змінені змінні і відображають її на своєму екрані.

Інформація про змінні, яка реплікується, називається реплікованими змінними. Репліковані змінні можуть бути будь-якими, включаючи такі типи даних, як цілі числа, дійсні числа, строки, булеві значення і т.д.

Для того, щоб змінна була реплікована, вона повинна бути позначена як реплікована. Це можна зробити, присвоюючи змінній властивість `Replicated`.

Реплікація змінних в Unreal Engine може бути конфігурована за допомогою різних параметрів. Наприклад: можна вказати, як часто буде відправлятися інформація про змінені змінні.

### 2.3.5 Типи реплікації змінних

У Unreal Engine існує два основних типи реплікації змінних:

Реплікація змінних з фіксованим інтервалом - це найбільш поширений тип реплікації змінних. У цьому випадку інформація про змінені змінні відправляється клієнтам заздалегідь через визначений інтервал часу.

Реплікація змінних за запитом - це більш ефективний тип реплікації змінних, який використовується для реплікації змінних, які змінюються рідко. У цьому випадку інформація про змінені змінні відправляється клієнтам тільки за запитом.

### 2.3.6 Приклади використання реплікації змінних

Реплікацію змінних можна використовувати для реалізації різних функцій в багатокористувацьких іграх. Наприклад: реплікацію змінних можна використовувати для наступних цілей:

- синхронізація стану об'єктів;
- передача інформації про події;
- реалізація кооперативних ігор.

Реплікацію змінних можна використовувати для того, щоб синхронізувати здоров'я персонажів в багатокористувацькій грі. Реплікацію змінних також

можна використовувати для того, щоб повідомляти клієнтам про зміні стану об'єкта, наприклад: а) об'єкт впав; б) був знищений.

### 2.3.7 Remote Procedure Calls

Remote Procedure Call (RPC) - це технологія в програмуванні, яка дозволяє викликати функції чи методи на віддаленому сервері, ніби вони викликаються локально. У контексті розробки відеоігор та мережевих додатків, RPC використовується для передачі команд та даних між клієнтом та сервером в мережевому середовищі.

RPC є потужним інструментом для організації взаємодії між компонентами програми, розташованими на різних вузлах мережі. Він дозволяє реалізувати віддалене викликання функцій чи методів так, щоб вони виконувалися на віддаленому сервері, а результати поверталися на локальний вузол.

У розробці відеоігор, коли маємо справу з мережевим геймплеєм, RPC використовується для синхронізації дій гравців, обміну інформацією про стан об'єктів у грі та викликання подій на віддалених пристроях.

Наприклад: якщо гравець натискає на кнопку стрільби у відео ігровому шутері, виклик RPC може передати інформацію про цю дію на сервер, який потім розсилатиме команди в інші пристрої для відтворення відповідної дії. Це дозволяє гравцям взаємодіяти один з одним у реальному часі, навіть якщо вони знаходяться на великій відстані один від одного у мережі.

### 2.3.8 Види RPC викликів у Unreal Engine

В Unreal Engine існують кілька видів RPC подій, які дозволяють вам взаємодіяти між клієнтами та сервером в мережевому середовищі гри. Ось декілька основних видів RPC подій в Unreal Engine:

- server RPC;
- client RPC;

- multicast RPC.

**Server RPC (Server-to-Client RPC)** - Цей тип RPC викликається на сервері і виконується на всіх підключених клієнтах. Використовується для відсилання інформації від сервера до всіх клієнтів. Наприклад: погодження змін у стані гравця на всіх клієнтах після виконання певної дії.

**Client RPC (Client-to-Server RPC)** - Цей тип RPC викликається на клієнті і виконується на сервері. Використовується для відсилання інформації від клієнта до сервера. Наприклад: вказівка серверу про зміни у стані гравця.

**Multicast RPC (Server & Clients)** - Цей тип RPC викликається на сервері, але виконується на всіх підключених клієнтах і самому сервері. Використовується для відсилання інформації всім гравцям у грі. Наприклад: відтворення вибуху або іншого події, яка повинна бути синхронізована на всіх гравців.

Ці види RPC подій в Unreal Engine дозволяють розробникам ефективно взаємодіяти між клієнтами та сервером для синхронізації гри в мережевому середовищі.

## 2.4 Базові класи рушію

Unreal Engine - це потужний інструмент для розробки відеоігор, який надає розробникам широкий спектр базових класів для побудови своїх ігор. В цьому підрозділі розглянемо ключові класи UE, що утворюють фундамент для розробки ігрового вмісту.

Unreal Engine має свій стандарт коду та правила найменування класів, структур [8]. Тому у назві класу міститься спеціальний префікс, такий як «U» чи «A». Ось витяг зі стандарту [8]:

- класи шаблонів мають префікс T;
- класи, які успадковуються від, UObject мають префікс U;
- класи, які успадковуються від, AActor мають префікс A.

Основні базові класи:

- UObject – базовий клас для об'єктів у рушію;
- AActor – базовий початковий клас який можна розміщувати на сцені (у світі);
- UGameInstance – клас який існує продовж усього циклу життя гри;
- AGameMode – клас із базовими налаштуваннями гри, відповідає за створення об'єктів типу: а) APlayerController; б) APlayerState; в) APawn;
- APlayerState – клас для збереження стану гравця. Також зберігає ідентифікатор гравця у мережі;
- APlayerController – клас який відповідає за опрацювання вхідних даних, наприклад: з клавіатури;
- APawn – базовий клас яким може керувати гравець;
- ACharacter – спадкоємець класу APawn, який підходить для створення персонажів якими можна керувати.

Далі буде надано більш детальний опис класів. Об'єкти з наданих базових класів з префіксом «А», створюються автоматично при старті гри, та існують на рівні. Навіть якщо ми не створювали ці класи, вони все одно існують у архітектурі рушію і необхідні для правильного функціонування гри.

Отримати доступ до APlayerController з APawn в Unreal Engine можна за допомогою функції GetController():

```
APlayerController* MyPlayerController = MyPawn->GetController();
```

З APlayerController можна отримати доступ до APlayerState та APawn таким чином:

```
APlayerState* MyPlayerState = MyPlayerController->PlayerState;
```

```
APawn* MyPlayerPawn = MyPlayerController->GetPawn();
```

Щоб отримати доступ до AGameMode треба викликати GetWorld() для отримання світу і потім викликати GetAuthGameMode():

```
UWorld* MyWorld = GetWorld();
```

```
if (MyWorld)
```

```

{
    AGameModeBase* MyGameMode = MyWorld->GetAuthGameMode();
}

```

### 2.4.1 UObject

UObject є основним класом у системі об'єктів Unreal Engine та є ключовою складовою для розробки ігор та віртуальних середовищ. Основна роль UObject - це надання базового функціоналу для усіх ігрових об'єктів та компонентів. Нижче наведено кілька ключових аспектів, що стосуються UObject.

**Спадкоємність.** Всі класи у Unreal Engine, які пов'язані з грою, спадкоємці класу UObject. Це включає в себе такі класи, як AActor, UActorComponent, UTexture, та багато інших.

**Життєвий цикл.** Об'єкти, що належать до класу UObject, мають життєвий цикл, котрий керується системою збору сміття Unreal Engine. Коли об'єкт більше не використовується або потребує звільнення пам'яті, система автоматично його видаляє.

**Метаклас та метаоб'єкти.** Unreal Engine використовує систему метакласів та метаоб'єктів для забезпечення динамічного змінювання властивостей об'єктів під час виконання гри.

**Рефлексія.** Об'єкти класу UObject підтримують рефлексію, що дозволяє розробникам здійснювати динамічний доступ до властивостей та методів об'єктів.

**Ідентифікація та серіалізація.** Об'єкти UObject можуть бути ідентифіковані унікальними іменами та піддаються серіалізації, що дозволяє зберігати та відновлювати їх стан.

**Динамічне створення об'єктів.** Розробники можуть динамічно створювати об'єкти класу UObject під час виконання гри, що важливо для сценаріїв, де створення об'єктів залежить від деяких умов чи подій.

**Інтеграція з графічним інтерфейсом.** Клас `UObject` також забезпечує базовий функціонал для інтеграції з графічним інтерфейсом `Unreal Editor`, де розробники можуть створювати, редагувати та налаштовувати об'єкти.

Використання класу `UObject` є невід'ємною частиною розробки в `Unreal Engine`, і розуміння його основних принципів дозволяє розробникам зручно та ефективно працювати з системою об'єктів та компонентів в рамках гри.

## 2.4.2 `AActor`

`AActor` є одним з ключових класів у системі `Unreal Engine` та представляє об'єкт, який може бути поміщений на сцену гри. Цей клас відіграє критичну роль у визначенні об'єктів у грі, включаючи персонажі, предмети, ефекти, та інші об'єкти взаємодії. Нижче наведено більше деталей щодо класу `AActor`.

**Компоненти та прикріплення.** `AActor` може містити різноманітні компоненти, такі як `USceneComponent`, `UStaticMeshComponent`, `UCameraComponent`, і багато інших. Ці компоненти визначають візуальний вигляд, фізичні властивості та інші аспекти об'єкта.

**Трансформація.** Кожен `AActor` має трансформацію, що визначає його положення, орієнтацію та масштаб на сцені. Це дозволяє розміщувати об'єкти в просторі та керувати їхньою орієнтацією.

**Життєвий цикл.** `AActor` пройде через різні етапи свого життєвого циклу, такі як створення, ініціалізація, усунення, та інші. Це важливо для правильної роботи гри та взаємодії з іншими елементами системи.

**Керування рухом.** Об'єкти класу `AActor` можуть взаємодіяти з системою фізики та мати можливість керувати рухом. Рух може бути здійснений як за допомогою фізики, так і шляхом простого переміщення об'єкта з одного місця на інше.

**Інтеграція з ігровим світом.** `AActor` може взаємодіяти з ігровим світом та отримувати події, такі як зіткнення, дотики, або інші події, що можуть впливати на його стан та поведінку.

**Реплікація у мережі.** `AActor` підтримує мережеву реплікацію, що дозволяє синхронізувати його стан між сервером та клієнтами у мережевих іграх.

**Події та функції.** Розробники можуть визначати власні події та функції для обробки різних сценаріїв взаємодії та поведінки об'єкта.

Інспектор та редагування в Unreal Editor - Клас `AActor` підтримує зручний інспектор та можливості редагування в редакторі Unreal Editor, що полегшує роботу з об'єктами під час розробки.

`AActor` є основою для багатьох ігрових об'єктів в Unreal Engine, і його правильне використання дозволяє розробникам створювати різноманітні та ефективні ігри.

### 2.4.3 `UGameInstance`

`UGameInstance` є одним із важливих класів у системі Unreal Engine та відіграє ключову роль у зберіганні та обміні даними між різними частинами гри, а також управлінні життєвим циклом гри. Нижче наведені деякі ключові аспекти цього класу.

**Глобальні дані гри.** `UGameInstance` дозволяє зберігати глобальні дані гри, які повинні залишатися сталими протягом всього життєвого циклу гри. Це може бути корисно для зберігання стану гравця, рівня складності, або будь-яких інших глобальних параметрів.

**Життєвий цикл.** `UGameInstance` існує протягом всього життєвого циклу гри, від моменту запуску до його завершення. Це дозволяє зберігати дані, які мають існувати поза межами конкретного рівня чи екземпляра гравця.

**Менеджмент рівнів та `Uniform Resource Identifier`.** `UGameInstance` може взаємодіяти з системою рівнів та вікнами гри, дозволяючи змінювати рівні, завантажувати нові ресурси та взаємодіяти з різними частинами гри.

**Робота з мережею.** У мережевих іграх `UGameInstance` може використовуватися для управління підключеннями гравців, обміну даними між клієнтами та сервером, та синхронізації глобальних подій.

**Відсутність графічного представлення.** Основна особливість UGameInstance полягає в тому, що він не пов'язаний з конкретним представленням графіки. Він працює в тлі та не відображає жодних візуальних компонентів.

**Серіалізація та зберігання даних.** UGameInstance може використовуватися для зберігання та обміну даними між різними рівнями гри. Це зручно для збереження стану гравця, налаштувань гри та інших параметрів.

**Події та функції.** Розробники можуть визначати власні події та функції у класі UGameInstance, які можуть бути викликані з будь-якої частини гри.

**Збереження стійкості інформації.** Завдяки тривалому циклу життя, UGameInstance можна використовувати для забезпечення стійкості даних, які не повинні скидатися між рівнями або сесіями гри.

UGameInstance є важливим елементом для реалізації загальних механізмів та логіки гри, яка не пов'язана безпосередньо з візуальною частиною. Це дає розробникам широкий функціонал для роботи з глобальними даними та управління різними аспектами гри.

#### 2.4.4 AGameMode

AGameMode - це клас в Unreal Engine, який визначає правила та параметри гри для конкретного рівня або групи рівнів гри. Кожен рівень у грі має свій власний об'єкт AGameMode, який визначає, як гра повинна вести себе на цьому рівні. Нижче наведені ключові аспекти класу AGameMode.

**Правила та параметри гри.** AGameMode визначає основні правила та параметри гри. Це може включати в себе такі речі, як кількість життів гравця, час гри, кількість очок, потрібних для перемоги, тощо.

**Життєвий цикл гри.** Клас AGameMode включає різні фази життєвого циклу гри, такі як підготовка до початку гри, сама гра, завершення гри та інші. Це дозволяє розробникам визначати поведінку гри на кожному етапі.

**Система очків та перемоги.** Клас може визначати, як гравці заробляють очки та умови для перемоги. Наприклад: визначення того, що гравець перемагає, коли досягає певної кількості очок чи виконує певні завдання.

**Створення гравців та об'єктів.** `AGameMode` визначає, як гравці та інші об'єкти з'являються на сцені гри. Це включає в себе місця створення, області безпеки та інші параметри.

**Логіка гри та події.** Клас може містити логіку для обробки різних подій гри, таких як смерть гравця, зіткнення, перемога та інші. Це дозволяє розробникам налаштовувати гру під свої потреби.

**Мережева робота.** Клас `AGameMode` також включає функціонал для роботи в мережевому середовищі, дозволяючи синхронізувати стан гри між сервером та клієнтами.

**Контроль неактивних гравців.** Визначає, як гра взаємодіє з неактивними гравцями. Це може включати в себе визначення того, чи можуть неактивні гравці впливати на гру.

**Налаштування та параметри.** `AGameMode` дозволяє розробникам встановлювати різні параметри та налаштування, які впливають на гру, і дозволяє створювати різноманітні сценарії та варіанти гри.

Клас `AGameMode` є важливим компонентом для керування логікою гри та визначенням умов її виконання. Розробники використовують цей клас для створення унікальних та захоплюючих геймплеїв для своїх ігор.

### 2.4.5 `APlayerController`

`APlayerController` є класом в Unreal Engine, який відповідає за керування гравцем та його взаємодію з грою. Кожен гравець в грі пов'язаний із своїм власним екземпляром `APlayerController`, який відповідає за обробку введення, управління персонажем та іншими аспектами взаємодії гравця з ігровим світом. Нижче наведені деякі ключові аспекти цього класу.

**Введення та управління.** APlayerController обробляє введення від гравця, таке як натискання клавіш, рух миші, джойстика чи інших пристроїв введення. Він перетворює це введення на відповідні команди та управління героєм або персонажем.

**Керування камерою.** Клас відповідає за управління камерою, яка пов'язана з гравцем. Це може включати в себе рух камери, прицілювання та інші аспекти огляду гравцем ігрового світу.

**Мережева взаємодія.** APlayerController взаємодіє з мережевим середовищем, керуючи локальним гравцем та синхронізуючі його стан з іншими гравцями у мережі.

**Орієнтація та рух.** Клас визначає орієнтацію та рух гравця в ігровому просторі. Він може включати такі аспекти, як повороти, переміщення та керування швидкістю руху.

**Інтеграція з ігровим світом.** APlayerController взаємодіє з ігровим світом, оброблюючи події, такі як зіткнення, збори ресурсів та взаємодія з іншими об'єктами на сцені.

**UI та Інтерфейс.** Клас може включати логіку для відображення та оновлення інтерфейсу користувача (UI), такого як віджети, панелі інформації та інші елементи, які пов'язані з інтерфейсом гравця.

**Логіка гри та події.** APlayerController може містити логіку для обробки різних подій гри, таких як завершення завдань, отримання нагород, тощо.

**Перехоплення подій вводу.** Розробники можуть перехоплювати різні події вводу, такі як натискання клавіш, миші чи інших пристроїв, для створення власної логіки гравця.

**Інтеграція із системою анімацій.** Клас може взаємодіяти з системою анімацій для відтворення рухів та анімацій гравця.

APlayerController виступає важливою роллю у взаємодії гравця та гри, визначаючи способи керування та обробки вводу. Цей клас дозволяє розробникам налаштовувати відчуття та поведінку гравця в грі.

## 2.4.6 A Pawn

A Pawn - це базовий клас у системі Unreal Engine, який представляє ігровий об'єкт, яким може керувати гравець або штучний інтелект. Клас A Pawn є частиною ієрархії класів Unreal Engine, яка включає в себе такі об'єкти, як персонажі, транспортні засоби, та інші контрольовані ігрові об'єкти. Нижче наведені деякі ключові аспекти цього класу.

**Управління рухом.** A Pawn дозволяє визначати та керувати рухом об'єкта в ігровому світі. Це може включати в себе переміщення вперед/назад, повороти, а також інші аспекти руху.

**Введення та керування.** Клас може обробляти введення гравця або іншого джерела та виконувати відповідні дії в залежності від цього введення.

**Фізична взаємодія.** A Pawn може взаємодіяти з системою фізики, що дозволяє об'єкту реагувати на силу та зіткнення в ігровому світі.

**Анімація та моделі.** Об'єкти класу A Pawn можуть мати асоційовані 3D-моделі та анімації для відображення їхнього вигляду та поведінки в грі.

**Колізії та зіткнення.** Клас взаємодіє з системою колізій та зіткнень, що дозволяє об'єкту реагувати на перешкоди в ігровому світі.

**Керування камерою.** Деякі об'єкти A Pawn можуть мати можливість контролювати камеру, яка слідкує за ними або дозволяє гравцеві взаємодіяти з навколишнім світом.

**Мережева робота.** A Pawn підтримує мережеву взаємодію, що дозволяє керувати об'єктом на стороні клієнта та сервера в мережевих іграх.

**Логіка гравця та AI (штучний інтелект).** Клас може бути керований або гравцем, або штучним інтелектом, в залежності від контексту гри.

**Інтерфейс та відображення.** A Pawn може мати власний інтерфейс користувача та графічне відображення для взаємодії з гравцем.

**Життєвий Цикл.** Об'єкти класу A Pawn проходять через різні етапи життєвого циклу, включаючи створення, ініціалізацію та знищення.

Клас `APawn` служить основою для створення різноманітних ігрових об'єктів, які можуть бути керовані гравцем або штучним інтелектом, і відіграє ключову роль у взаємодії гравця з ігровим світом.

### 2.4.7 ACharacter

`ACharacter` є підкласом класу `APawn` в Unreal Engine і представляє об'єкт гри, який може бути керований гравцем або штучним інтелектом та має додаткову функціональність для роботи з персональними характеристиками та анімацією персонажів. Нижче наведені деякі ключові аспекти цього класу.

**Керування рухом та камерою.** `ACharacter` додає функціональність для керування рухом та камерою персонажа. Включає в себе обробку введення гравця та взаємодію з системою камери.

**Анімації та скелет.** Клас підтримує анімації персонажа та можливість роботи зі скелетною анімацією. Це дозволяє відтворювати різні рухи та стани персонажа.

**Керування життям та здоров'ям.** `ACharacter` може мати вбудовану систему керування життям та здоров'ям персонажа. Включає в себе можливість визначення поточного стану здоров'я та лікування.

**Інтеграція з об'єктами.** Персонаж може взаємодіяти з об'єктами та середовищем гри, виконуючи різноманітні дії, такі як збирання предметів чи взаємодія з іншими персонажами.

**Керування Виносками (Ragdoll).** Деякі версії Unreal Engine надають можливість активувати режим виносок (ragdoll), коли персонаж потерпає від сильних ударів або пошкоджень.

**Мережева взаємодія.** Клас підтримує роботу в мережевому середовищі, дозволяючи керувати персонажем на стороні клієнта та сервера в мережевих іграх.

**Керування станами (States).** Має можливість визначення різних станів персонажа, таких як рух, спокій, атака та інші. Це важливо для правильної обробки анімацій та взаємодії гравця з персонажем.

**Кастомізація поведінки.** Розробники можуть налаштовувати поведінку персонажа за допомогою належного коду та параметрів.

**Здібності та навички.** ACharacter дозволяє визначати різні навички та здібності персонажа, такі як стрибки, біг, атаки та інші.

Клас ACharacter використовується для створення рухомих та інтерактивних персонажів в грі, які можуть взаємодіяти з гравцем та оточуючим світом.

#### 2.4.8 APlayerState

APlayerState є класом в Unreal Engine, який представляє стан конкретного гравця в грі. Кожен гравець, який підключений до гри, має свій власний об'єкт APlayerState, який зберігає інформацію про цього гравця. Нижче наведені деякі ключові аспекти цього класу.

**Ідентифікація гравця.** APlayerState містить унікальний ідентифікатор гравця, який використовується для відслідковування та ідентифікації конкретного гравця в грі.

**Інформація про гравця.** Клас містить різноманітну інформацію про гравця, таку як ім'я, статус (онлайн/офлайн), очки та інші дані, які можуть бути важливими для відображення на екрані чи управління гравцями.

**Очки та ранги.** APlayerState може відстежувати кількість набраних гравцем очків та його ранг в грі.

**Статус готовності.** Деякі ігри використовують APlayerState для відстеження готовності гравця до початку гри.

**Мережева взаємодія.** APlayerState служить як точка обміну інформацією про гравців між клієнтом та сервером в мережевому режимі гри.

**Спеціальні параметри та властивості.** Розробники можуть додавати свої власні параметри та властивості до `APlayerState` відповідно до конкретних потреб гри.

**Логіка обробки подій.** Клас містить логіку для обробки подій, таких як вибування гравця, отримання/втрата очок, зміни статусу тощо.

**Синхронізація стану гравців.** `APlayerState` гарантує синхронізацію стану гравців між клієнтом та сервером у мережевій грі.

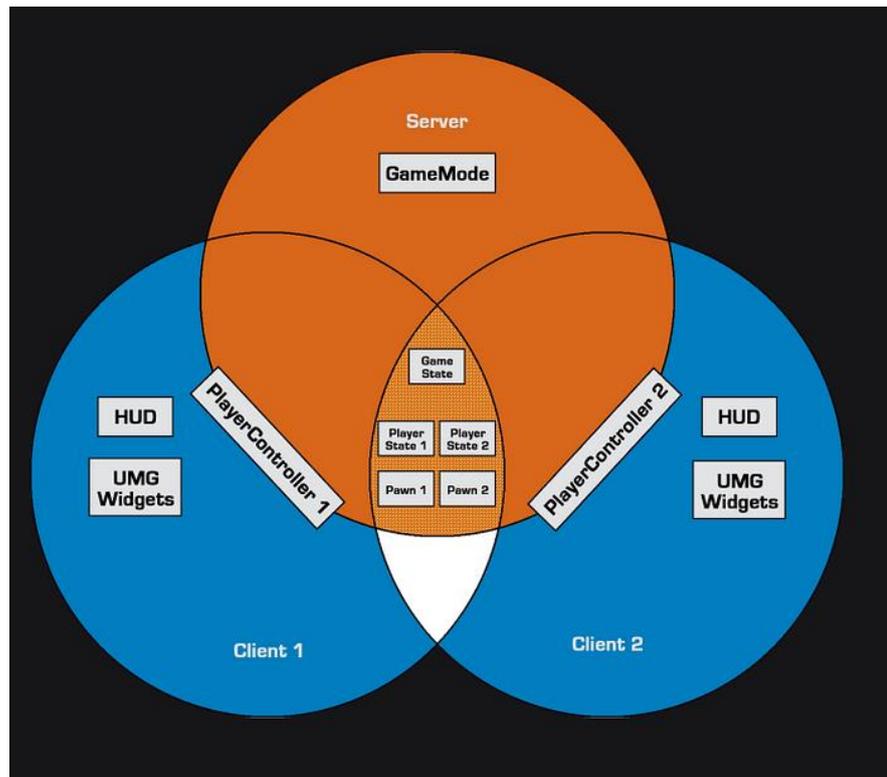
**Зберігання налаштувань гравця.** Може містити налаштування, які пов'язані з конкретним гравцем та впливають на його геймплей або візуальний вигляд.

**Мультиплеєрні функції.** `APlayerState` надає функції для роботи з мережовим середовищем, такі як відправка повідомлень чи синхронізація даних.

`APlayerState` грає важливу роль у відстеженні та керуванні інформацією про кожного гравця в грі, забезпечуючи єдність та синхронізацію в мережових та одиночних іграх.

#### 2.4.9 Класи у мережевій архітектурі

Оскільки ми робимо мережовий проект, слід розуміти де саме у мережі будуть розміщуватись ці класи. Тобто між якими мережевими об'єктами (сервером, клієнтом) вони будуть синхронізуватись. Це розуміння важливе для побудови ігрової архітектури, логіки. На рисунку 2.9 зображено та які класи та у якому мережевому просторі вони знаходяться.



**Рисунок 2.9 – Мережеве розміщення класів**

Виключно на сервері знаходиться клас `AGameMode`, та класи які створюються виключно на сервері (за написаною логікою розробника гри) та із вимкненою реплікацією.

Виключно на клієнті знаходяться класи які відповідають за інтерфейс (UI).

На сервері та відповідному клієнті знаходиться клас `APlayerController`. Основна частина та створення цього класу знаходиться на стороні клієнта, але на сервері знаходиться його мережева копія, що дозволяє серверу виконувати певні керування.

На сервері та між клієнтами знаходяться класи: а) `AGameState`; б) `APlayerState`; в) `APawn`.

## 2.5 Реалізація механік

Необхідні механіки:

- механіка переміщення;

- механіка прискорення;
- механіка витривалості та її відновлення;
- механіка удару по м'ячу.

Необхідні класи:

- клас гравця;
- клас м'ячу;
- клас воріт;
- клас GameMode;
- клас PlayerController.

Також буде можливість відновити камеру у початкову позицію та зміни кольору гравців. Звісно це все повинно працювати у мережі, синхронізуватись. Нагадаю що, для реалізації проекту був використаний шаблонний проект.

Рушій має реалізований певний функціонал загального призначення. Але для кожного проекту потрібна своя певна логіка, функціонал, а для цього треба створювати свої класи та методи. Наприклад методи: а) створення сесії; б) пошуку сесії. Вже є готові методи, ноди, але код у цих методах виконується асинхронно і розробнику потрібно власноруч писати потрібну йому логіку після завершення асинхронного коду. У моєму випадку я зроблю логіку яка після вдалого створення сесії, завантажує на певну мапу, а після знаходження, відбувається приєднання та завантаження на карту. Також для моїх потреб є певні налаштування сесії, тому оскільки це все є специфічним функціоналом для проекту, для цієї логіки потрібно створювати власні методи, які у собі будуть викликати метод надані рушієм. Докладніше про реалізацію методів створення та пошуку сесії буде пізніше.

Також у вибраному шаблоні реалізовано переміщення, але не має функції бігу. Для функції бігу мені ще потрібно враховувати механіку витривалості, такого функціоналу рушій не має. Навіть якщо він був, там була одна реалізація, яка може відрізнитися від моїх задумок.

Отже рушій має базові методи загального призначення, використовуючи їх можна створити свою власну унікальну логіку. Unreal Engine дозволяє створювати різноманітні проекти різних видів. Для кожного виду, певного проекту, потрібен свій функціонал який неможливо передбачити наперед.

### 2.5.1 Вже реалізований функціонал

У вибраному шаблоні вже реалізована механіка переміщення, стрибку. Зроблена обробка натискання кнопок. За це відповідає компонент `UCharacterMovementComponent`.

`UCharacterMovementComponent` є компонентом для руху персонажа в Unreal Engine. Цей компонент призначений для управління фізикою та анімацією руху персонажа. Нижче наведені деякі ключові аспекти `UCharacterMovementComponent`.

**Рух по землі.** Компонент визначає рух персонажа по землі, включаючи ходьбу та біг. Він враховує різні аспекти, такі як нахил, прискорення та обмеження швидкості.

**Сповільнення та зупинка.** Відповідає за логіку гальмування та зупинки персонажа. Це може бути важливим для реалістичної поведінки руху.

**Стрибки та гравітація.** `UCharacterMovementComponent` дозволяє налаштувати параметри стрибків та взаємодії персонажа з гравітацією.

**Лазання та сходження по схилах.** Компонент може взаємодіяти з різними поверхнями, включаючи сходження та лазання по схилах.

**Рух по воді та занурення.** `UCharacterMovementComponent` включає підтримку руху по воді та можливості занурювання персонажа.

**Анімація руху.** Дозволяє синхронізувати параметри руху з системою анімації персонажа для плавного відтворення рухових анімацій.

**Мережевий рух.** Компонент оптимізований для роботи в мережевому режимі гри, забезпечуючи синхронізацію руху між клієнтом та сервером.

**Перевірка колізій та зіткнень.** Включає в себе логіку для виявлення колізій та зіткнень персонажа з оточуючим середовищем.

**Налаштування руху.** `UCharacterMovementComponent` має велику кількість параметрів, які можна налаштовувати, щоб долучити рух персонажа до конкретних вимог гри.

Цей компонент є ключовим для визначення поведінки руху персонажа в грі, і його налаштування може значно впливати на геймплей та відчуття гри.

### 2.5.1 Створення нових Input

За обробку натискання клавіш та вхідних даних для керування ігровим персонажем відповідає система Enhanced Input [9].

Enhanced Input — це новий API в Unreal Engine 5, який покращує введення та взаємодію з гравцями. Він забезпечує більш плавну та інтуїтивну взаємодію з іграми, а також дає розробникам більше контролю над відчуттям ігор. Нижче наведено функції які включає в себе Enhanced Input.

**Підтримка різних типів введення.** Enhanced Input підтримує широкий спектр типів введення, включаючи контролери, клавіатури, миші та сенсорні екрани. Це дозволяє розробникам створювати ігри, які можна комфортно грати на будь-якому пристрої.

**Інтуїтивне управління.** Enhanced Input має простий та інтуїтивний інтерфейс, який полегшує розробникам настройку та керування вводом.

**Підтримка відстеження погляду.** Enhanced Input підтримує відстеження погляду, що дозволяє гравцям керувати грою, дивлячись на екран.

Enhanced Input є потужним інструментом, який може допомогти розробникам створювати більш захоплюючі та інтуїтивні ігри.

Нижче наведені деякі конкретні приклади того, як Enhanced Input можна використовувати в іграх.

У шутерах від першої особи Enhanced Input можна використовувати для покращення точності стрільби та управління рухом.

У пригодницьких іграх Enhanced Input можна використовувати для покращення взаємодії гравців з навколишнім середовищем.

У спортивних іграх Enhanced Input можна використовувати для покращення відчуття управління персонажем.

Enhanced Input все ще знаходиться в розробці, але він вже доступний для використання в Unreal Engine 5.

Отже мені треба створити 3 об'єкти типу «InputAction», для обробки удару, для прискорення та для відкривання UI (реалізуємо у пункті 2.6). Я назвав їх: а) IA\_Push; б) IA\_Sprint; в) IA\_UI. Далі я додав їх у файл з контекстом інпутів, який створений у шаблоні. Він має назву «IMC\_Default». В ньому я за прикладом додав свої InputAction`s, та назначив їм клавіші.

Далі у кодї, у класї «ACyberFCharacter» за аналогією створюю змінні для того щоб можна було зв'язати створені об'єкти з певною логікою обробки. Ось як це виглядає:

```
UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input,
meta = (AllowPrivateAccess = "true"))
```

```
UInputAction* SprintAction;
```

```
UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input,
meta = (AllowPrivateAccess = "true"))
```

```
UInputAction* PushAction;
```

```
UPROPERTY(EditAnywhere, BlueprintReadOnly, meta =
(AllowPrivateAccess = "true"))
```

```
UCapsuleComponent* PushCapsule;
```

Нам потрібно використовувати макрос UPROPERTY [10], для того щоб система відобразила цю змінну у блупринті, та ми змогли вказати посилання на раніше створений об'єкт.

Також слід зробити методи для обробки натискання клавіш.

```
void Push();
```

```

        UFUNCTION(BlueprintCallable,                               BlueprintNativeEvent,
meta=(DisplayName = "Push"))
        void K2_Push();
        void StartSprint();
        void StopSprint();
        UFUNCTION(BlueprintCallable, Server, Reliable)
        void StartSprint_S();
        UFUNCTION(BlueprintCallable, Server, Reliable)
        void StopSprint_S();

```

Метод `K2_Push` буде перевизначений у блупренті, та мати назву «Push». Метод «`StartSprint_S`» та «`StopSprint_S`», можуть викликатись у блупренті, та будуть реплікуватись на сервер. Тобто клієнт буде викликати виконання логіки на сервері. За ці налаштування відповідає макрос `UFUNCTION` [11].

У методі «`SetupPlayerInputComponent`» ми робимо прив'язку відповідного `InputAction` до певного методу. Ось як це виглядає:

```

        EnhancedInputComponent->BindAction(PushAction, ETriggerEvent::Started,
this, &ACyberFCharacter::Push);
        EnhancedInputComponent->BindAction(SprintAction, ETriggerEvent::Started,
this, &ACyberFCharacter::StartSprint);
        EnhancedInputComponent->BindAction(SprintAction,
ETriggerEvent::Completed, this, &ACyberFCharacter::StopSprint);

```

### 2.5.2 Реалізація бігу, витривалості, удару по м'ячу

Біг реалізований просто. Це звичайне перемикання булевої змінної та зміна значення змінної максимальної швидкості. Код реалізації для бігу:

```

void ACyberFCharacter::StartSprint()
{
    StartSprint_S();
}

```

```

void ACyberFCharacter::StopSprint()
{
    StopSprint_S();
}

void ACyberFCharacter::StartSprint_S_Implementation()
{
    const float MinTimeToActivate = 1.f;
    isSprintMode = (SprintTime - UsedSprintTime) > MinTimeToActivate;

    if (!isSprintMode) { return; }

    GetCharacterMovement()->MaxWalkSpeed = SprintMaxSpeed;
}

void ACyberFCharacter::StopSprint_S_Implementation()
{
    if (!isSprintMode) { return; }

    isSprintMode = false;
    GetCharacterMovement()->MaxWalkSpeed = DefaultWalkMaxSpeed;
}

```

Витривалість теж має просту реалізацію. Безперервно можна бігати 3 секунди. Відновлюється витривалість з 0 до 100% за 2 секунди. Ці параметри можна налаштувати окремо у блупринті. Витрата чи відновлення витривалості виконуються виключно на стороні серверу та у кожному кадрі.

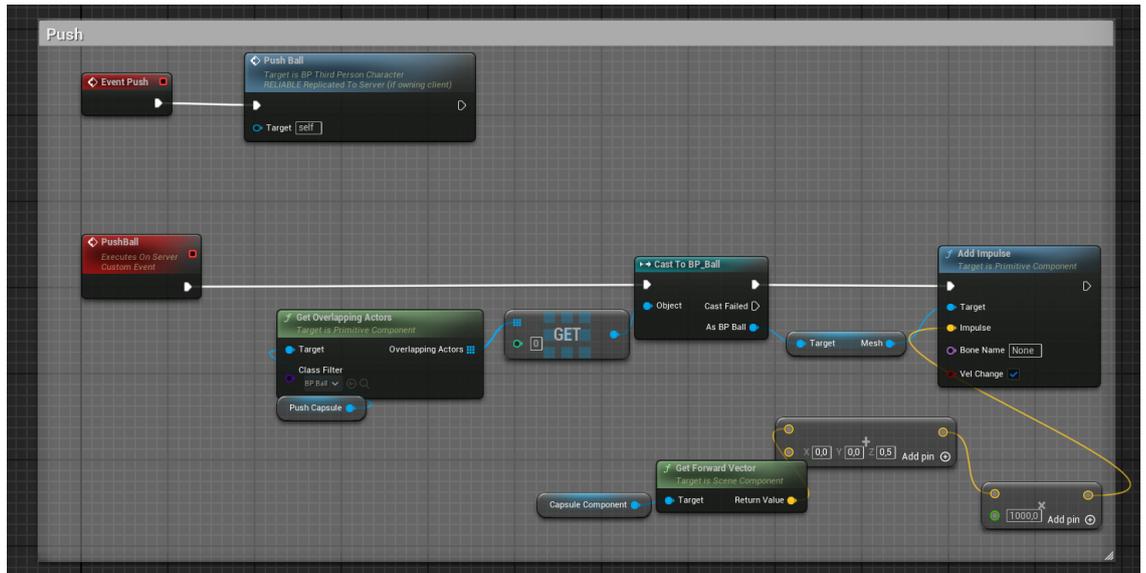
Якщо кнопка для бігу натиснута, то витривалість зменшується на різницю часу між кадрами. Для 60 кадрів у секунду це значення буде дорівнювати 0,016 мс. Коли кнопка бігу не натиснута, то відбувається відновлення витривалості.

```

void ACyberFCharacter::Tick(float DeltaSeconds)
{
    Super::Tick(DeltaSeconds);
    if (GetLocalRole() == ROLE_Authority)
    {
        if (isSprintMode)
        {
            UsedSprintTime += DeltaSeconds;
            if (SprintTime - UsedSprintTime <= 0)
            {
                StopSprint();
            }
        }
        else if (UsedSprintTime != 0)
        {
            const float TargetFPS = 60;
            const float CurrnetFPS = 1 / DeltaSeconds;
            const float SprintResumePart =
(TargetSprintResumeValuePerSecond * TargetFPS) / CurrnetFPS;

            UsedSprintTime -= SprintResumePart * SprintTime;
            UsedSprintTime = FMath::Clamp(UsedSprintTime, 0,
UsedSprintTime);
        }
    }
}

```



**Рисунок 2.10 – Реалізація логіки удару по м'ячу**

Реалізація логіки удару по м'ячу зроблена тільки у блупрінті та зображена на рисунку 2.10. Клієнт після обробки натискання відповідної кнопки, відправляє виклик логіки удару на сервер. На сервері йде перевірка, чи знаходиться м'яч поблизу гравця і якщо знаходиться, то додаємо м'ячу імпульс у напрямку погляду моделі гравця. Саме моделі, а не напрямку зору камери.

Повний код з класу `ACyberFCharacter` наданий у додатку А

### 2.5.3 Функціональне призначення інших класів

Проект також має інші класи з певною логікою, яка необхідна для повноцінної роботи гри. Нижче наведений опис деяких з цих класів.

Клас «`BP_Ball`» - це наш м'яч. У ньому знаходиться відповідна модель, налаштована колізія, увімкнена реплікація та реплікація переміщення, є логіка переміщення на початкову позицію.

Клас «`BP_Goal`» - це ворота. Вони містять спеціальний тригер, який реагує на потрапляння в нього м'яча та компонент з відображенням тексту. Цей компонент відображає кількість забитих голів у ці ворота. Обробка попадання

м'яча у ворота виконується на сервері. Змінна яка відповідає за рахунок є реплікованою, тому що її треба синхронізувати між сервером та клієнтами.

Клас «BP\_GameMode» - відповідає за налаштування стандартних класів та в моєму випадку це основна серверна логіка. Тут є логіка початку гри, переміщення на початкові позиції, поява м'ячу та гравців.

Клас «BP\_PlayerController» - тут знаходиться логіка створення UI та опрацювання натискання на елементи UI. У цей клас я розмістив логіку: а) створення сесії; б) пошуку сесії; в) приєднання до сесії.

## 2.6 UI та реалізація підключення гравців один до одного

### 2.6.1 Що таке UI

Інтерфейс користувача (UI) — це сукупність всіх елементів, що дозволяють користувачеві взаємодіяти з програмою або грою. У контексті розробки ігор UI відповідає за те, щоб гравець міг легко розуміти, що відбувається в грі, і приймати рішення. Нижче наведено опис основних елементів як включає UI.

**Текст.** Текст використовується для передачі інформації гравцю. Він може використовуватися для пояснення правил гри, інформування про стан гравця або просто для створення атмосфери.

**Ікони.** Ікони використовуються для представлення дій або функцій. Вони можуть бути простими або складними, але повинні бути легко зрозумілими для гравця.

**Графіка.** Графіка використовується для створення привабливого та інтуїтивно зрозумілого інтерфейсу. Вона може включати в себе фони, персонажів та інші елементи.

Нижче наведено опис принципів UI-дизайну.

**Відповідність цілям.** UI повинен відповідати цілям і завданням гри. Він повинен бути простим у використанні та зрозумілим для гравців різного рівня досвіду.

**Відповідність стилю.** UI повинен відповідати стилю гри. Він повинен гармонійно вписуватися в загальну атмосферу гри.

**Ефективність.** UI повинен бути ефективним у передачі інформації гравцю. Він повинен бути легким для розуміння і використання.

Розробка UI — це складний процес, що вимагає вдумливого планування та дизайну. Нижче наведені фактори які інженери-розробники повинні враховувати.

**Цілі та завдання гри.** Для чого призначена гра? Які функції повинен виконувати UI?

**Цільова аудиторія.** Для кого призначена гра? Які у них навички та досвід?

**Технології.** Які технології будуть використовуватися для розробки гри? Які обмеження накладають ці технології?

Успішний UI-дизайн може зробити гру більш захоплюючою та інтуїтивно зрозумілою для гравців. Він може допомогти гравцям легко зрозуміти, що відбувається в грі, і приймати рішення, які сприяють успіху.

## 2.6.2 Створення власного UI

Створення UI у Unreal Engine відбувається завдяки Unreal Motion Graphics [12].

Unreal Motion Graphics (UMG) — це вбудована система UI-дизайну, яка дозволяє створювати UI-елементи за допомогою простого у використанні інтерфейсу.

UMG — це потужний інструмент, який дозволяє створювати складні UI-інтерфейси з використанням широкого спектру елементів. Нижче наведено опис основні елементи, які включає UMG.

**Widget Blueprints** — це основа для всіх UI-елементів у UMG. Вони містять всі властивості та поведінку елемента.

**UI Widgets** — це готові до використання елементи, які можна вставляти в Widget Blueprint. Вони включають текстові поля, кнопки, графічні елементи та інші елементи.

**Layout** — це система розміщення UI-елементів на екрані.

Нижче наведено опис основних елементів UI, які можна використовувати в Unreal Engine.

**Текст.** Використовується для передачі інформації гравцю. Він може використовуватися для пояснення правил гри, інформування про стан гравця або просто для створення атмосфери.

**Ікони.** Використовуються для представлення дій або функцій. Вони можуть бути простими або складними, але повинні бути легко зрозумілими для гравця.

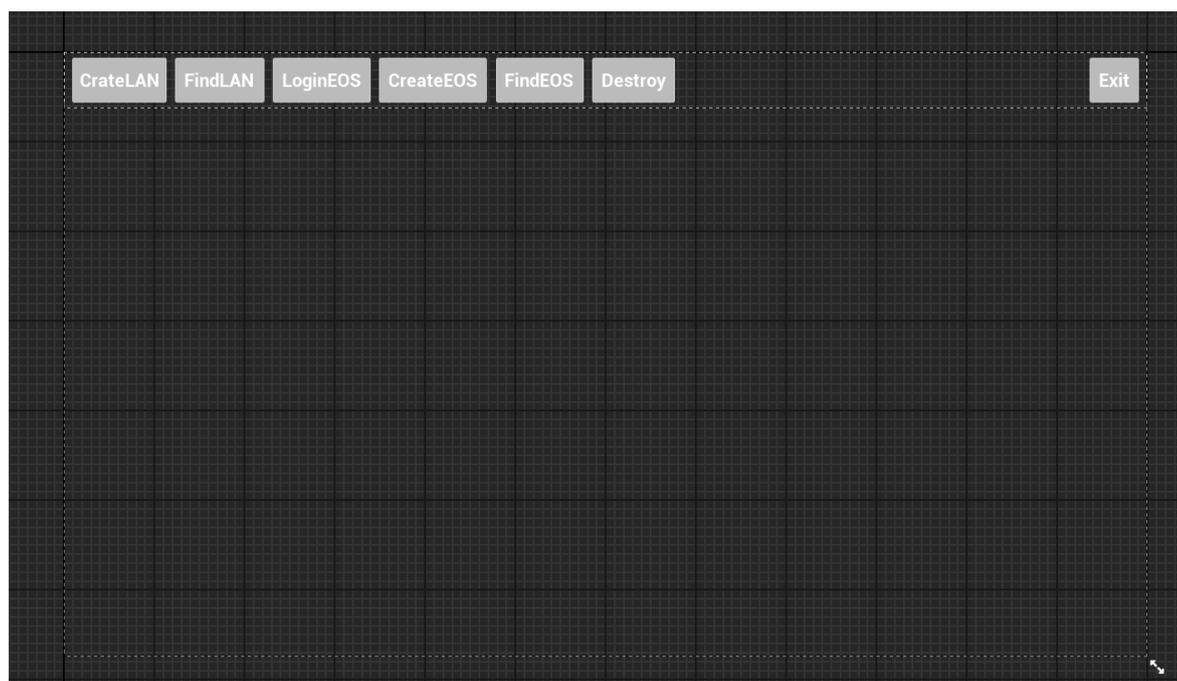
**Графіка.** Використовується для створення привабливого та інтуїтивно зрозумілого інтерфейсу. Вона може включати в себе фони, персонажів та інші елементи.

Крім основних елементів, існує ряд додаткових елементів UI, які можна використовувати в Unreal Engine. Ці елементи включають:

- віджет-контейнери, які використовуються для об'єднання декількох UI-елементів в один;
- слайдери, які використовуються для вибору значення з певного діапазону;
- перемикачі, які використовуються для вибору між двома значеннями;
- радіокнопки, які використовуються для вибору одного з декількох значень;
- текстові поля, які використовуються для введення тексту;
- ім'я поля, які використовуються для введення імені;
- паролне поле, які використовуються для введення пароля;
- кнопки, які використовуються для виконання певної дії;
- списки, які використовуються для відображення набору елементів;

- таблиці, які використовуються для відображення набору даних у вигляді таблиці;
- стрічки прокрутки, які використовуються для відображення елементів, які не вміщуються на екрані.

Для реалізації UI у проекті нам знадобляться такі елементи: а) Layout; б) Контейнер; в) Кнопка; г) Текстове поле.



**Рисунок 2.11 – UI у проекті**

Уся основна обробка натискання кнопок відбувається у класі «BP\_PlayerController». Я завчасно створив кнопки для реалізації підключення до глобальної мережі оскільки на цьому функціоналі буде досліджуватись робота штучного інтелекту як помічника у розробці.

### **2.6.3 Підключення по локальній мережі**

Локальна мережа (ЛМ) - це комп'ютерна мережа, яка об'єднує комп'ютери та інші пристрої в межах обмеженої географічної області, такої як один будинок, одне приміщення, чи невеликий кампус. Ці мережі можуть бути створені з метою

обміну ресурсами, таких як файли, принтери, інтернет-з'єднання, а також для розгортання спільних сервісів. Нижче наведено опис основних характеристик локальних мереж.

**Розташування.** Локальні мережі охоплюють невелику територію, зазвичай в межах одного будинку, офісу або кампусу. Вони не перетинають межі об'єкта (наприклад: не виходять за межі офісу чи компанії).

**Топологія.** Топологія локальної мережі описує, як фізично або логічно підключені комп'ютери та інші пристрої в цій мережі. Загальні топології включають зірку, кільце, лінійну, дерево, та інші.

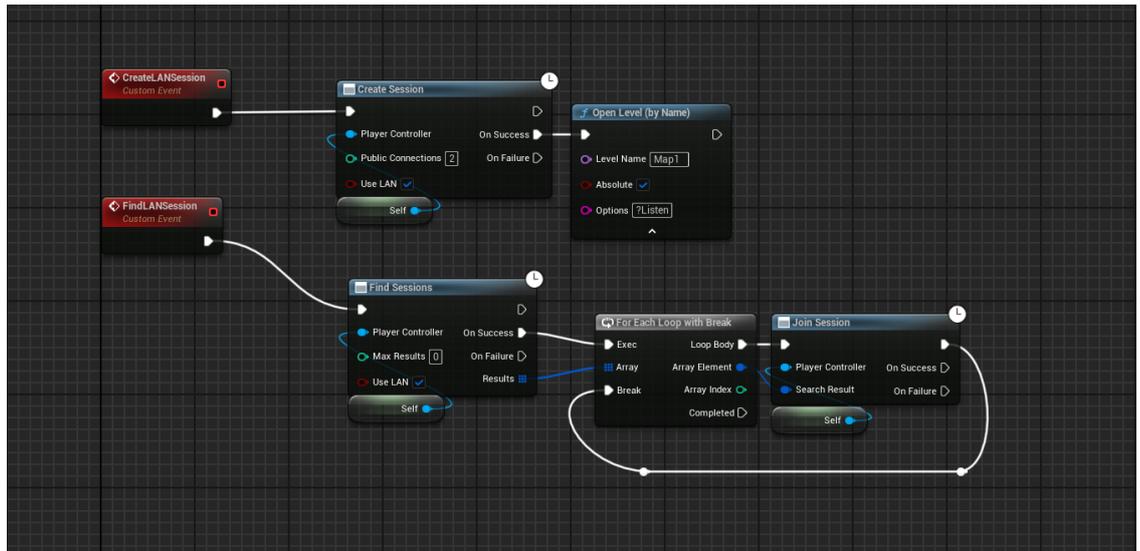
**Швидкість та пропускна здатність.** Локальні мережі можуть мати високу швидкість передачі даних та велику пропускну здатність, оскільки вони призначені для обмеженого числа пристроїв та користувачів.

**Спільний доступ до ресурсів.** Головною метою локальної мережі є забезпечення спільного доступу до ресурсів, таких як файли, друк, програми, а також обмін інформацією між користувачами та пристроями.

**Протоколи та стандарти.** Локальні мережі використовують різні протоколи та стандарти для забезпечення надійного обміну даними. Наприклад: Ethernet є одним із найпоширеніших стандартів для локальних мереж.

**Масштабованість.** Локальні мережі можуть бути масштабованими, але зазвичай їх розміри обмежені географічною областю. Для великих територій використовуються інші типи мереж, такі як розгалужена мережа (WAN).

Локальні мережі можуть використовуватися в домашньому оточенні, в офісах, навчальних закладах та інших установах для забезпечення зручної комунікації та обміну ресурсами.



**Рисунок 2.13 – Реалізація створення та підключення до сесії.**

За замовчуванням у Unreal Engine використовується Online Subsystem Null, яка якраз і створює ігрові сесії у локальній мережі. Звісно якщо мати статичну IP адресу, або виділений сервер, то можна під'єднатись напряму. Реалізація підключення виконана у блупренті та зображена на рисунку 2.13. Для створення, пошуку та приєднання використовую «ноди» які існують у рушію.

## 2.7 Аналіз імплементації створення сесії OSS EOS

OSS EOS дає можливість грати через локальну мережу. Наразі ми використовуємо OSS NULL що дозволяє грати через локальну мережу, або якщо маємо виділений сервер. Реалізацію підключення через OSS EOS буде зроблено у наступному розділі, а зараз проаналізуємо імплементацію створення сесії OSS EOS.

Відповідальний метод за це: `FOnlineSessionEOS::CreateSession`, він приймає ідентифікатор користувача, ім'я сесії, та налаштування сесії.

```
FNamedOnlineSession* Session = GetNamedSession(SessionName);
if (Session == nullptr){
if (IsRunningDedicatedServer() // EOSSubsystem->UserManager-
>GetLoginStatus(HostingPlayerNum) >= ELoginStatus::UsingLocalProfile)
```

```
{...}}
```

Першим кроком йде перевірка чи вже створена сесія, перевіряється за ім'ям, якщо ні, то далі перевіряється чи авторизувався гравець до системи EOS, а бо чи це виділений сервер.

Далі йде створення екземпляру класу сесії, яка зберігає ім'я та налаштування. Оновлюються данні з переданого аргументу функції.

```
Session = AddNamedSession(SessionName, NewSessionSettings);
check(Session);

Session->SessionState = EOnlineSessionState::Creating;

Session->OwningUserId          =      EOSSubsystem->UserManager-
>GetUniquePlayerId(HostingPlayerNum);

Session->OwningUserName        =      EOSSubsystem->UserManager-
>GetPlayerNickname(HostingPlayerNum);

Session->NumOpenPrivateConnections          =
NewSessionSettings.NumPrivateConnections;

Session->NumOpenPublicConnections          =
NewSessionSettings.NumPublicConnections;

Session->HostingPlayerNum = HostingPlayerNum;

Session->SessionSettings.BuildUniqueId = GetBuildUniqueId();
```

До сесії додається ім'я користувача, його ідентифікатор, який буде ідентифікатором сесії та версія зібраної гри. Зберігати версію гри потрібно, щоб уникнути конфліктів із різною кодовою базою, або базою контенту, бо хтось може грати на новішій версії, де є суттєві зміни по коду і це може призвести до неприємних помилок.

Наступним кроком йде перевірка чи це буде у локальній мережі чи ні, та створювати сесію чи лобі. В залежності від перевірки, викликаються певні методи для створення. Результат перевірок залежить від налаштувань сесії, які передаються аргументом до виклику методу. У моєму проекті не буде використовуватись локальна мережа при використанні OSS EOS також не буде використовуватись створення лобі.

```

if (!NewSessionSettings.bIsLANMatch)
{
    if (Session->SessionSettings.bUseLobbiesIfAvailable)
    {
        Result = CreateLobbySession(HostingPlayerNum, Session);
    } else {
        Result = CreateEOSSession(HostingPlayerNum, Session);}
    } else {
        Result = CreateLANSession(HostingPlayerNum, Session);}
}

```

У FOnlineSessionEOS::CreateEOSSession йде адаптування та перенесення значень налаштувань з рушію до бібліотеки EOS та йде створення сесії шляхом виклику функції з бібліотеки «EOS\_Sessions\_CreateSessionModification».

```

EOS_Bool bPresenceEnabled = EOS_FALSE;
bPresenceEnabled = (Session->SessionSettings.bUsesPresence //
Session->SessionSettings.bAllowJoinViaPresence //
Session->SessionSettings.bAllowJoinViaPresenceFriendsOnly //
Session->SessionSettings.bAllowInvites) ? EOS_TRUE : EOS_FALSE;
FSessionCreateOptions Options(TCHAR_TO_UTF8(*Session-
>SessionName.ToString()));
Options.MaxPlayers = Session->SessionSettings.NumPrivateConnections +
Session->SessionSettings.NumPublicConnections;
Options.LocalUserId = EOSSubsystem->UserManager-
>GetLocalProductUserId(HostingPlayerNum);
Options.bPresenceEnabled = bPresenceEnabled;
const auto BucketIdUtf8 = StringCast<UTF8CHAR>(*GetBucketId(Session-
>SessionSettings));
Options.BucketId = (const char*)BucketIdUtf8.Get();

```

```

EOS_EResult                                     ResultCode                                     =
EOS_Sessions_CreateSessionModification(EOSSubsystem->SessionsHandle,
&Options, &SessionModHandle);

```

Далі, після створення, йде встановлення адреси серверу (хосту) у системі сервісів EOS.

```

FName NetDriverName = GetDefault<UNetDriverEOS>()->NetDriverName;
FInternetAddrEOS      TempAddr(LexToString(Options.LocalUserId),
NetDriverName.ToString(), GetTypeHash(NetDriverName.ToString()));
HostAddr = TempAddr.ToString(true);
char HostAddrAnsi[EOS_OSS_STRING_BUFFER_LENGTH];
FCStringAnsi::Strncpy(HostAddrAnsi,      TCHAR_TO_UTF8(*HostAddr),
EOS_OSS_STRING_BUFFER_LENGTH);

```

```

EOS_SessionModification_SetHostAddressOptions HostOptions = { };
HostOptions.ApiVersion = 1;
UE_EOS_CHECK_API_MISMATCH(EOS_SESSIONMODIFICATION_SETH
OSTADDRESS_API_LATEST, 1);
HostOptions.HostAddress = HostAddrAnsi;
EOS_EResult                                     HostResult                                     =
EOS_SessionModification_SetHostAddress(SessionModHandle, &HostOptions);

```

Останнім кроком йде оновлення даних про сесію, на цьому етапі це буде встановлення відсутніх даних, такі як максимальна кількість гравців. Ці данні оновлюються на серверах сервісів EOS.

```

SetPermissionLevel(SessionModHandle, Session);
SetMaxPlayers(SessionModHandle, Session);
SetInvitesAllowed(SessionModHandle, Session);
SetJoinInProgress(SessionModHandle, Session);
SetAttributes(SessionModHandle, Session);
EOS_Sessions_UpdateSessionOptions CreateOptions = { };
CreateOptions.ApiVersion = 1;

```

```
UE_EOS_CHECK_API_MISMATCH(EOS_SESSIONS_UPDATESESSION_A  
PI_LATEST, 1);
```

```
CreateOptions.SessionModificationHandle = SessionModHandle;  
EOS_Sessions_UpdateSession(EOSSubsystem->SessionsHandle,  
&CreateOptions, Callback, Callback->GetCallbackPtr());  
EOS_SessionModification_Release(SessionModHandle);
```

Отже метод реалізація методу створення сесії не просто створює сесію, а й робить певні налаштування та адаптування даних. Данні про налаштування сесії відправляються на сервери EOS де будуть використовуватись для пошуку сесій, та надсилатися гравцеві (клієнту), на запит про пошук сесії.

## 3 ДОСЛІДЖЕННЯ РОБОТИ ШТУЧНОГО ІНТЕЛЕКТУ

### 3.1 Опис проблем та сценаріїв де можна використати штучний інтелект при розробці

#### 3.1.1 Основна ціль дослідження

Основною цілю використання штучного інтелекту є скорочення часу на пошук інформації, відповіді на запитання. Unreal Engine є багатомодульною програмою з великою кількістю класів та можливостями. Тому інформації яку треба знати для вдалої роботи із рушієм потрібно багато. Але не уся інформація потрібна завжди, або потрібно вивчити якийсь новий інструмент, тому ШІ буде корисним у вивченні чогось нового або у відновленні старих знань.

Я маю на меті дослідити як ГШІ впорається із написанням коду та написанням інструкції з інтеграції Online Subsystem EOS.

Online Subsystem EOS — це плагін для Unreal Engine, який дозволяє розробникам ігор додавати онлайн-функціональності в свої проекти. Плагін підтримує широкий спектр функцій, нижче наведено приклад деяких функцій.

**Реєстрація та вхід гравців.** Плагін дозволяє гравцям реєструватися та входити в гру за допомогою облікових записів: а) Epic Games; б) Xbox Live; в) PlayStation Network або будь-якої іншої підтримуваної платформи.

**Спілкування гравців.** Плагін дозволяє гравцям спілкуватися один з одним за допомогою текстового чату, голосового чату та інших засобів.

**Гра між платформами.** Плагін підтримує гру між платформами, що дозволяє гравцям з різних платформ грати разом.

**Сервера.** Плагін дозволяє розробникам створювати та керувати власними серверами для своїх ігор.

Online Subsystem EOS є потужним інструментом, який може допомогти розробникам ігор додати онлайн-функціональності в свої проекти без необхідності писати власний код. Нижче наведені деякі з переваг використання Online Subsystem EOS.

**Легкість використання.** Плагін простий у використанні та не вимагає від розробників ігор знань про мережеве програмування.

**Широкий спектр функцій.** Плагін підтримує широкий спектр функцій, що дозволяє розробникам ігор створювати повнофункціональні онлайн-ігри.

**Підтримка між платформної гри.** Плагін підтримує гру між платформами, що дозволяє гравцям з різних платформ грати разом.

Online Subsystem EOS невідмінно від Online Subsystem Null дозволяє створювати та шукати сесії через глобальну мережу. Що дає змогу гравцям грати у гру коли вони фізично знаходяться у різних місцях, різні інтернет провайдери.

Для повноцінної роботи Online Subsystem EOS треба зробити певні налаштування, та використовувати C++ код на відміно від того як зроблено приєднання по локальній мережі через блупринт логіку.

### 3.1.2 Де ще та як можна використовувати ШІ

З точки зору програмування генеративний штучний інтелект можна використовувати для:

- інструкцій із увімкнення реплікацій, створення реплікаційних подій (у тому випадку коли розробник з цім не працював ніколи);
- пошуку відповіді на питання, чому якийсь об'єкт не синхронізується;
- для пошуку відповідей на проблеми при синхронізації логіки;
- складання певних формул, або роз'яснення формул.

Це приклади які стосуються саме проекту який розроблений у 2й частині.

Варіантів для використання значно більше, наприклад:

- перероблення коду з однієї мови програмування на іншу;
- генерування коду, написання алгоритму;
- оптимізація коду.

У інших сферах генеративний штучний інтелект можна використовувати для:

- створення зображень з концепцією гри (Art);

- створення текстур;
- створення діалогів;
- написання сценарію;
- створення звуків, озвучення;
- переклад на інші мови.

## **3.2 Використання штучного інтелекту для написання коду реалізації підключення гравців один до одного**

### **3.2.1 Що треба зробити**

Перш ніж почати дослідження, треба розуміти те що будемо вважати вдалим кінцевим результатом. Якщо задавати питання ГШІ та вирішувати проблему у сфері з якою ще не працював, то це буде вирішення проблеми методом проб то помилок. Оскільки я маю досвід роботи з Online Subsystem EOS, то можу визначити кінцевий результат.

Тож, щоб у грі запрацювала Online Subsystem EOS потрібно:

- зробити певні налаштування проекту;
- підписатися на зворотній виклик у разі виконання методів з OSS;
- зробити метод для авторизації у систему EOS;
- зробити методи для створення сесії, пошуку, приєднання;
- зробити обробку зворотних викликів, на створення, пошук та приєднання.

### **3.2.2 Генеративний штучний інтелект**

Для генерування коду будуть використовуватись ChatGPT та Бард (Bard). Переважно ChatGPT.

ChatGPT - це модель обробки природної мови, яку розробив OpenAI на основі архітектури GPT (Generative Pre-trained Transformer). GPT є серією моделей, призначених для вирішення завдань з обробки природної мови.

"ChatGPT" зазвичай вказує на конкретну версію моделі, спроектовану для взаємодії з користувачами через чат або інші текстові вводи.

Ця модель навчається на величезних обсягах текстових даних і може генерувати текст, відповідаючи на запитання, формулювати відгуки та виконувати різні завдання, пов'язані з мовленнєвою обробкою. Вона може використовуватися для створення чат-ботів, асистентів і для вирішення різних завдань в сфері обробки природної мови.

Бард — це велика мовна модель, також відома як розмовний штучний інтелект або чат-бот, навчений бути інформативним і всеосяжним. Бард навчений на величезній кількості текстових даних і може спілкуватися та генерувати текст, подібний до людського, у відповідь на широкий спектр підказок і запитань. Наприклад: Бард може надавати резюме фактичних тем або створювати історії. Бард все ще перебуває в розробці, але навчився виконувати багато видів завдань.

Бард є потужним інструментом, який можна використовувати для різних цілей. Наприклад: його можна використовувати для:

- надання загального огляду фактичних тем;
- творення історій;
- переклад мов;
- написання різних видів творчого контенту;
- відповіді на ваші запитання в інформативному та всебічному порядку.

Бард постійно вдосконалюється, і я в нетерпінні дізнатися, що він зможе робити в майбутньому.

### **3.2.3 Генерування коду**

У цьому підрозділі буде надано код який згенерував ГШІ. Огляд та аналіз відповідей ГШІ та згенерованого коду буде надано у підрозділі 3. Слід розуміти що генеративний штучний інтелект – це допоміжний інструмент, а не вирішення

проблем. Тому і ставитись до нього потрібно відповідно та розуміти що відповіді можуть буди не точними, мати помилки.

Увесь код який наданий у цьому підрозділі є згенерований ГШІ. Повний код із моїми корективами можна знайти у додатку Б. Цей код, це реалізація функцій які викликаються з `BP_PlayerController`, які у свою чергу спрацьовують при натисканні відповідних кнопок у `UI` який був створений у розділі 2, підпункт 2.6.2.

Очікуваний результат на запит про підписання на зворотній виклик: згенерований код, який через інтерфейс `OSS` робить передплату на зворотній виклик подій, для створення сесії, пошуку та приєднання.

Мій запит: «відповідай тільки українською мовою. Потрібно згенерувати код для UE5 для передплати на делегати в `Online subsystem`: створення сесії, пошук сесії, приєднання сесії».

Згенерований код:

*«// Підписка на делегат створення сесії*

*OnlineSessionInterface-*

*>OnCreateSessionCompleteDelegates.AddUObject(this,  
&YourClass::OnCreateSessionComplete);*

*// Підписка на делегат пошуку сесій*

*OnlineSessionInterface-*

*>OnFindSessionsCompleteDelegates.AddUObject(this,  
&YourClass::OnFindSessionsComplete);*

*// Підписка на делегат приєднання до сесії*

*OnlineSessionInterface->OnJoinSessionCompleteDelegates.AddUObject(this,  
&YourClass::OnJoinSessionComplete);»*

Також він зробив базову генерацію обробки подій. Там немає ніякої логіки бо я не вказував контекст и чого я хочу. До обробки ще повернемось згодом.

Очікуваний результат на запит про створення сесії: згенерований код, який через інтерфейс `OSS` створює сесію. Також реалізація функції зворотного виклику, яка у разі успішного створення переміщує гравця на певну мапу.

Мій запит: «Unreal engine Online Subsystem. Потрібен метод створення сесії EOSSession, при успішному створенні сесії потрібно перейти на карту "Map1"».

Згенерований код:

```
// Функція для створення сесії
void YourGameSessionClass::CreateSession()
{
    IOnlineSessionPtr SessionInterface = GetOnlineSessionInterface();
    if (SessionInterface.IsValid())
    {
        // Використовуйте ваші параметри для створення сесії (ім'я сесії,
        кількість гравців і т.д.)
        FOnlineSessionSettings SessionSettings;
        SessionSettings.bIsLANMatch = false;
        SessionSettings.NumPublicConnections = 4;
        SessionSettings.bShouldAdvertise = true;
        // Додайте інші налаштування сесії за потреби
        // Створення сесії
        SessionInterface->CreateSession(0, NAME_GameSession,
        SessionSettings);
    }
}

// Обробник делегата після створення сесії
void YourGameSessionClass::OnCreateSessionComplete(FName
SessionName, bool bWasSuccessful)
{
    if (bWasSuccessful)
    {
        // Сесія успішно створена, тепер ви можете переходити на іншу
        карту
```

```

// Наприклад, використовуйте AGameModeBase для зміни карти
AGameModeBase* GameMode = GetWorld()->GetAuthGameMode();
if (GameMode)
{
    FString MapName = "Map1"; // Замініть на фактичне ім'я вашої
карти
    GameMode->TravelToMap(MapName);
}
}
}

```

Очікуваний результат на запит про пошук сесії: згенерований код, який через інтерфейс OSS шукає сесію. Також реалізація функції зворотного виклику, яка у разі успішного пошуку, викликає через OSS функцію приєднання до сесії.

Мій запит: «Метод пошуку сесії. При успішному знаходженні сесії потрібно приєднатися до неї і перейти на карту».

Згенерований код:

```

// Функція для пошуку сесій
void YourGameSessionClass::FindSessions()
{
    IOnlineSessionPtr SessionInterface = GetOnlineSessionInterface();
    if (SessionInterface.IsValid())
    {
        // Використовуйте ваші параметри для пошуку сесій (наприклад,
максимальна кількість сесій, інші фільтри і т. д.)
        SessionSearch = MakeShareable(new FOnlineSessionSearch());
        // Ініціалізація параметрів пошуку (якщо потрібно)
        // SessionSearch->bIsLanQuery = false;
        // SessionSearch->MaxSearchResults = 10;
        // Пошук сесій
        SessionInterface->FindSessions(0, SessionSearch.ToSharedRef());
    }
}

```

```

    }
}
Згенерований код для підключення до знайденої сесії:
// Обробник делегата після завершення пошуку сесій
void YourGameSessionClass::OnFindSessionsComplete(bool bWasSuccessful)
{
    if (bWasSuccessful)
    {
        // Пошук сесій завершено успішно, тепер ви можете приєднатися до
        знайдених сесій
        TArray<FOnlineSessionSearchResult> SearchResults = SessionSearch-
>SearchResults;
        if (SearchResults.Num() > 0)
        {
            // Обираємо першу знайдену сесію, але ви можете вибрати сесію
            відповідно до вашої логіки
            FOnlineSessionSearchResult& SearchResult = SearchResults[0];
            IOnlineSessionPtr SessionInterface = GetOnlineSessionInterface();
            if (SessionInterface.IsValid())
            {
                // Приєднання до сесії
                SessionInterface->JoinSession(0, NAME_GameSession,
                SearchResult);
            }
        }
    }
}

```

Очікуваний результат на запит про обробку події успішного приєднання до сесії: згенерований код, який у разі успішного приєднання, завантажує гравця (клієнта) до сервера.

Мій запит: «Код, коли приєднання до сесії успішне, потрібно перейти на карту».

Згенерований код:

```
// Обробник делегата після приєднання до сесії
void YourGameSessionClass::OnJoinSessionComplete(FName SessionName,
EOnJoinSessionCompleteResult::Type Result)
{
    if (Result == EOnJoinSessionCompleteResult::Success)
    {
        // Приєднання до сесії успішне, тепер ви можете переходити на іншу
карту
        // Наприклад, використайте AGameModeBase для зміни карти
AGameModeBase* GameMode = GetWorld()->GetAuthGameMode();
        if (GameMode)
        {
            FString MapName = "Map1"; // Замініть на фактичне ім'я вашої
карти
            GameMode->ServerTravel(MapName);
        }
    }
}
```

Отже в цілому генеративний штучний інтелект (ChatGPT), впорався із задачею по генерації коду. Він згенерував загальний код, робив в ньому пояснювальні коментарі. Так, цей код можна перенести до проекту, але його трішки потрібно буде адаптувати, адже це згенеровано як приклад, а не під конкретний проект.

```

void ACyberFPlayerController::EOSCreateSession()
{
    FOnlineSessionSettings SessionSettings;
    SessionSettings.NumPublicConnections = 2;
    SessionSettings.NumPrivateConnections = 0;
    SessionSettings.bShouldAdvertise = true;
    SessionSettings.bIsLANMatch = false;
    SessionSettings.bAllowInvites = true;
    SessionSettings.bUsesPresence = true;
    SessionSettings.bAllowJoinViaPresence = true;
    SessionSettings.bAllowJoinViaPresenceFriendsOnly = false;
    SessionSettings.Set(SEARCH_KEYWORDS, FString("Session"), EOnlineDataAdvertisementType::ViaOnlineService);
    SessionSettings.Set(SETTING_MAPNAME, FString("Map1"), EOnlineDataAdvertisementType::ViaOnlineService);
    if (OSS != nullptr)
    {
        if (IOnlineSessionPtr Session = OSS->GetSessionInterface())
        {
            Session->CreateSession(0, CF_SESSION_NAME, SessionSettings);
        }
    }
}

void ACyberFPlayerController::EOSFindSession()
{
    if (OSS != nullptr)
    {
        if (IOnlineSessionPtr Session = OSS->GetSessionInterface())
        {
            if (!SessionSearch.IsValid())
            {
                SessionSearch = MakeShareable(new FOnlineSessionSearch);
            }
            SessionSearch->QuerySettings.Set(SEARCH_PRESENCE, true, EOnlineComparisonOp::Type::Equals);
            SessionSearch->MaxSearchResults = 10;
            SessionSearch->bIsLanQuery = false;

            Session->FindSessions(0, SessionSearch.ToSharedRef());
        }
    }
}

```

**Рисунок 3.1 - Код для створення та пошуку сесії**

На рисунку 3.1 надано код який є у проекті для створення та пошуку сесії. Він базується на відповідях ГШІ, але перероблений за моїми вподобаннями. Я розширив параметри для створення та для пошуку сесії. Повний код надано у додатку Б.

### **3.3 Аналіз відповідей штучного інтелекту на проблеми які виникли при розробці проекту**

#### **3.3.1 Аналіз відповідей на запитання про підключення OSS EOS**

Перш ніж почати генерування коду, треба було налаштувати проект та плагін Online Subsystem EOS. ChatGPT повноцінно із запитом не справився, бо в

його базі немає нових даних. У своїх запитах я позначав що це для Unreal Engine 5. UE 5 з'явився не дуже давно, але все ж таки раніше ніж останнє оновлення бази даних ГШІ. Тим паче що рушій та плагін оновлюються та можуть буди специфічні сценарії, то дуже важко сформулювати повноцінну інструкцію. Базову загальну інструкцію ГШІ надав, також у ній були посилання на ресурси. В цілому для початкового розуміння цього може бути достатньо, але все одно треба знаходити відповідну документацію та робити відповідно неї.

Отже мій запит був таким: «Unreal engine online subsystem eos. Як підключити». Відповідь ГШІ зображено на рисунку 3.2. Я забув надати певні налаштування з мови, тому мова відрізняється.

Функціонал ГШІ залежить від бази даних на якій він навчався, черпає данні для відповідей. Завдяки інтернету, доступ до даних став простіший, але і кількість інформації значно зростає. ChatGPT не має якогось конкретного цілеспрямування, він охоплює різні області та сфери діяльності людини. Тому не дивно що у світі де багато різноманітної, специфічної інформації, яка постійно зростає, не вистачає нових даних у базі даних ГШІ.

### ChatGPT

На момент окончания моего последнего обновления в январе 2022 года не существует информации о том, что Unreal Engine поддерживает подсистему EOS (Epic Online Services) "из коробки". Однако вы можете использовать SDK EOS для добавления поддержки в свой проект. Процедура может измениться, так что удостоверьтесь, что вы используете актуальную документацию, предоставленную Epic Games и EOS.

Вот общий процесс подключения SDK EOS к проекту Unreal Engine:

1. **Регистрация в Epic Games и получение ключа разработчика:**
  - Зарегистрируйтесь на [сайте Epic Games](#).
  - Зайдите в [панель разработчика Epic Games](#), создайте новое приложение и получите ключ разработчика.
2. **Скачивание и установка SDK EOS:**
  - Посетите [страницу загрузки Epic Online Services SDK](#), чтобы скачать последнюю версию SDK EOS.
  - Распакуйте архив и следуйте инструкциям по установке, предоставленным в документации.
3. **Настройка проекта Unreal Engine:**
  - Откройте свой проект Unreal Engine.
  - Включите поддержку онлайн-сервисов в проекте, если ещё этого не сделали. Обычно это можно сделать в настройках проекта.
4. **Интеграция SDK в проект:**
  - Добавьте файлы SDK в ваш проект.
  - Обновите файлы конфигурации и настройте параметры, такие как ключ разработчика, в соответствии с документацией EOS.
5. **Использование функций EOS в коде:**
  - Используйте функции и классы из SDK EOS в своем коде Unreal Engine для работы с онлайн-сервисами.
6. **Тестирование:**
  - Проведите тестирование, чтобы убедиться, что подключение работает правильно и взаимодействие с онлайн-сервисами происходит без проблем.

Обратитесь к документации EOS и Unreal Engine для более подробных инструкций и обновленной информации о подключении подсистемы EOS к Unreal Engine.

## Рисунок 3.2 – Відповідь ГШІ про налаштування OSS EOS

Проте, не зважаючи на відсутність певної інформації, ГШІ намагався надати загальний опис кроків які необхідно виконати. Щоб дати правильну відповідь, людині потрібно знайти відповідну документацію, далі проаналізувати її та виділити основні кроки, які необхідно реалізувати. З розвитком ГШІ, думаю це усе стане можливим та більш точним, що буде полегшувати пошук людині та економити час.

### 3.3.2 Аналіз відповідей на запитання про підписання на зворотній виклик функцій

Як правило, при створенні об'єкту класу, де буде використання Online Subsystem, робляться підписування на зворотній виклик функцій. У русію це називаються делегати та підписання на делегати.

Простими словами делегат – це клас (об'єкт класу) який містить в собі показники на функції, та може викликати ці функції одразу. Він має певний функціонал, такий як: а) додати функцію до делегату (підписатися); б) видалити з делегату (відписатися); в) викликати підписані функції.

Делегатів у Online Subsystem багато, та знаходяться у різних інтерфейсах, тому буває важко так згадати назву потрібного делегату та те як до нього отримати доступ.

Генерацію коду ГШІ зробив правильно, звісно це загальний варіант і про це він писав у своїй відповіді. У згенерованому коді він використовує змінну «OnlineSessionInterface», я спитав як її отримати, і він згенерував потрібний код, де є звертання до модулю та взяття потрібного показника на інтерфейс.

Також він згенерував налаштування та наголосив щоб у файлі налаштувань проекту був заданий потрібний модуль для OSS. То б то замість OSS Null було вказано OSS EOS. Про ці налаштування я не запитував, але ГШІ про них написав, що є важливим кроком для повноцінної роботи OSS EOS. Тобто якщо про це розробник не знав, або забув, ГШІ про це нагадав сам виходячи із контексту задачі.

Особисто мене вразило те, що ГШІ надав приклад у файлі налаштувань, параметрів для налаштування сесії, яку будемо створювати. Вразило це тим, бо про це я не знав, та зазвичай ці параметри задаються при у коді перед створенням сесії.

Ще є одна особливість із роботою Online Subsystem. Так як OSS це плагін який включає в себе модуль, то для роботи в коді із OSS та вдалої збірки проекту, треба підключити модулі до проекту. ГШІ одразу про це на сказав, тому довелось

про це запитувати на пряму. Я знаю, що треба підключати модулі, знаю які та де саме, але якщо людина в перше із цим затикається, то їй буде не зрозуміло, чого його код не компілюється. Коли я запитав про модулі, тобто використав свої знання об їх існуванні, ГШІ написав про необхідні модулі, та те де і як їх потрібно підключити до проекту. Вони підключаються у коді у певному файлі, тому він згенерував код.

Модуль – це код виділений у одну предметну область. Його можна підключити ти вилучити із проекту. Також модуль компілюється разом із проектом.

Плагін – це модуль але із декількома відмінностями. Він компілюється окремо від проекту. Вмикається та вимикається, через браузер плагінів. Може містити файли контенту (не коду). Плагін більше походить на невеличкий самостійний проект, який інтегрується у інші проекти.

### **3.3.3 Аналіз відповідей на запитання про створення сесії, та опрацювання вдалого створення**

Я просив згенерувати код для створення сесії та код опрацювання вдалого створення. Коли сесія успішно створена, нам треба перейти на певну мапу і відкрити прослуховування на приєднання [7].

Код згенерований правильно, є налаштування сесії та виклик створення. Він також згенерував код для переходу на певну мапу, яку я вказав у своєму запиті. Тільки він використовує трішки інші методи для переходу на мапу.

```
FString MapName = "Map1";
```

```
GameMode->TravelToMap(MapName);
```

Відповідно до документації [7] я замінив код на:

```
GetWorld()->ServerTravel(FString("Map1?listen"));
```

Важливою відмінністю є атрибут «?listen», що дозволяє грі, прослуховувати вхідні підключення інших гравців. Також я відредагував

налаштування сесії відповідно до моїх вимог. Оскільки я не конкретизував параметри, ГШІ згенерував загальний код.

### 3.3.4 Аналіз відповідей на запитання про пошук сесії, та приєднання до сесії

За моєю задумкою, коли система знаходить сесію відбувається підключення до сесії. Уразі вдалого підключення, гравця (клієнта) завантажує на сервер, певну мапу.

Код для пошуку сесії згенеровано вірно. Слід зазначити, що там використовується змінна яку потрібно створити у .h файлі. ГШІ у даній відповіді про це не сказав. Також він надав приклад налаштування для пошуку сесій. Цей код закоментований та наданий як приклад. Але це добре, бо одразу є розуміння про такі можливості та навіть приклад є.

Обробка знаходження сесії теж вдала. У моєму випадку функціоналу тут мало, тому розповісти нічого. Якщо розширювати логіку, то замість миттєвого підключення, можна виводити інформацію об знайдених сесіях, створити браузер сесії.

А ось обробка вдалого підключення та завантаження до серверу не є вдалою. Нижче згенерований код переміщення на мапу до сервера.

```
FString MapName = "Map1";
GameMode->ServerTravel(MapName);
```

Відповідно до документації [7] для клієнтів потрібно використовувати код який наведено нижче.

```
FString JoinedAddress;
Session->GetResolvedConnectString(SessionName, JoinedAddress);
if (!JoinedAddress.IsEmpty())
{
    if (UWorld* World = GetWorld())
    {
```

```

    ClientTravel(JoinedAddress, TRAVEL_Absolute);
}
}

```

Важливою відмінністю є метод «ClientTravel», який переміщає саме клієнта та за певною адресою. Метод «ServerTravel» викликається тільки з серверу та переміщає усіх гравців на певну карту.

Повний код із реалізацією підключення та використанням Online Subsystem EOS надано у додатку Б. У кінцевій реалізації були проведені певні зміни у згенерованому коді, щоб відповідати розробленому проекту.

```

void ACyberFPlayerController::OnCreateSessionComplete(FName SessionName, bool bWasSuccessful)
{
    if(bWasSuccessful)
    {
        GetWorld()->ServerTravel(FString("Map1?listen"));
    }
}

void ACyberFPlayerController::OnJoinSessionComplete(FName SessionName, EOnJoinSessionCompleteResult::Type Result)
{
    if (Result == EOnJoinSessionCompleteResult::Success)
    {
        if (OSS != nullptr)
        {
            if (IOnlineSessionPtr Session = OSS->GetSessionInterface())
            {
                FString JoinedAddress;
                Session->GetResolvedConnectString(SessionName, JoinedAddress);
                if (!JoinedAddress.IsEmpty())
                {
                    if (UWorld* World = GetWorld())
                    {
                        ClientTravel(JoinedAddress, TRAVEL_Absolute);
                    }
                }
            }
        }
    }
}

```

**Рисунок 3.3 – виправлений код згенерований ГШІ**

На рисунку 3.3 зображено код, який використовується у проекті. Він написаний з урахуванням проведеного аналізу згенерованого коду ГШІ. Аналіз коду проводився відповідно до пункту 3.3.3 та пункту 3.3.4.

### 3.3.5 Аналіз пошуку рішення з проблемою переміщення гравців

Під час розробки ігрового процесу я зіткнувся із проблемою, переміщення гравця у потрібну точку (телепортація). Окрім того, що мені треба було змінити координати гравця, мені ще треба було змінити його поворот. Виклик телепортації виконується із сторони сервера, та повинен реплікуватись до клієнтів. Проблема полягає в тому, що на сервері все гаразд, другий гравець перемістився та повернувся так як треба, а на стороні другого гравця, він перемістився, але не розгорнувся та дивиться у той бік як і до переміщення. Із цією проблемою я вирішив звернутися до ChatGPT.

Очікуваний результат – пояснення проблеми, надання розуміння, що могло піти не так. За можливістю, надати код для прикладу із вирішенням проблеми. Результат вважається вдалим, коли з'явиться розуміння із вирішенням проблеми.

Мій запит:

«Відповідай лише українською мовою. unreal engine, multiplayer game, grpc event. Чому при виклику телепортації гравця на сервері з урахуванням повороту. На сервері гравці телепортуються нормально, з поворотом, а на клієнті поворот той, який був до телепортації»

Відповідь на мій запит:

«Проблема, ймовірно, пов'язана з тим, як ви синхронізуєте дані про повороти між сервером і клієнтом в мережевій грі на Unreal Engine. Для того щоб зробити правильну телепортацію з урахуванням повороту на обох сторонах, вам слід коректно передавати і синхронізувати дані про поворот між клієнтом і сервером.

Переконайтеся, що ви правильно використовуєте RPC (Remote Procedure Call) події для виклику телепортації на обох сторонах. Коли ви викликаєте подію на сервері, передавайте інформацію про те, наскільки був повернутий гравець. Потім сервер повинен передати цю інформацію клієнту під час телепортації.

Вам також може знадобитися використовувати функції, такі як `SetActorLocationAndRotation`, щоб встановити як місце розташування, так і поворот під час телепортації.

Важливо також перевірити, чи ви правильно синхронізуєте дані між клієнтом і сервером при використанні RPC подій, і чи не виникає конфліктів у передачі даних.»

Чесно кажучи, так зразу і не зрозуміло у чому проблема, тому я не думаючи, роблю інший запит: «Згенеруй правильний код для телепортації. З Сервера беремо `FTransform`».

На що, ChatGPT відповів:

«Звісно, ось приклад коду на мові C++ для телепортації з використанням `FTransform` на стороні сервера та синхронізації повороту між клієнтом і сервером за допомогою RPC подій в Unreal Engine:

У вашому .h файлі гравця (`PlayerController` або `Character`):

```
#pragma once
#include "GameFramework/PlayerController.h"
#include "MyPlayerController.generated.h"
UCLASS()
class YOURGAME_API AMyPlayerController : public APlayerController
{
    GENERATED_BODY()
public:
    // RPC подія для телепортації
    UFUNCTION(Server, Reliable, WithValidation)
    void ServerTeleportWithRotation(FTransform NewTransform);
    // Локальна функція для виклику телепортації на клієнті
    void TeleportWithRotation(FTransform NewTransform);
};
```

У вашому .cpp файлі гравця (`PlayerController` або `Character`):

```
#include "MyPlayerController.h"
```

```

#include "Net/UnrealNetwork.h"

void
AMyPlayerController::ServerTeleportWithRotation_Implementation(FTransform
NewTransform)
{
    // Встановлення нового розташування та повороту на сервері
    GetCharacter()->SetActorTransform(NewTransform);
    // Виклик локальної функції на клієнті для встановлення того ж самого
розташування та повороту
    TeleportWithRotation(NewTransform);
}

bool AMyPlayerController::ServerTeleportWithRotation_Validate(FTransform
NewTransform)
{ return true; }

void AMyPlayerController::TeleportWithRotation(FTransform NewTransform)
{
    // Локально встановлюємо нове розташування та поворот на клієнті
    GetCharacter()->SetActorTransform(NewTransform);
}

// Деяка інша логіка гравця...

Приклад виклику телепортації на сервері:
if (HasAuthority())
{
    // Отримання поточного розташування та повороту гравця
    FTransform NewTransform = GetCharacter()->GetActorTransform();
    // Встановлення нового розташування та повороту (наприклад,
телепортація на нові координати зі збереженням повороту)
    NewTransform.SetLocation(FVector(NewX, NewY, NewZ));
    NewTransform.SetRotation(FQuat(NewRotation));
    // Виклик RPC події на сервері для телепортації з урахуванням повороту

```

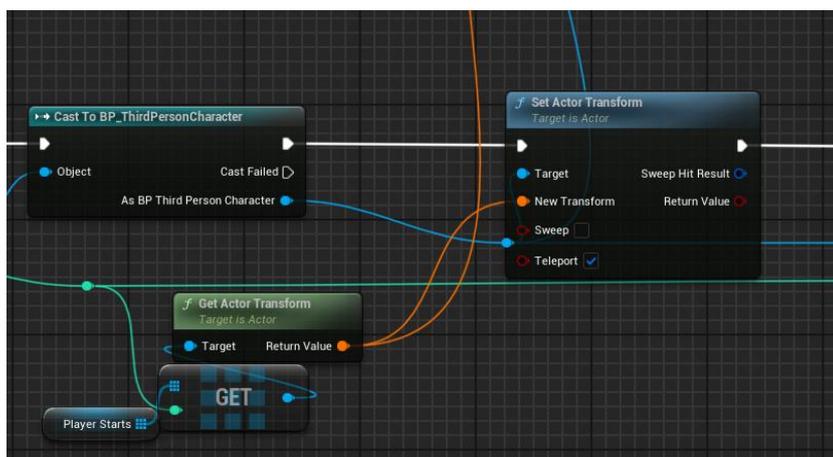
```

ServerTeleportWithRotation(NewTransform);
}

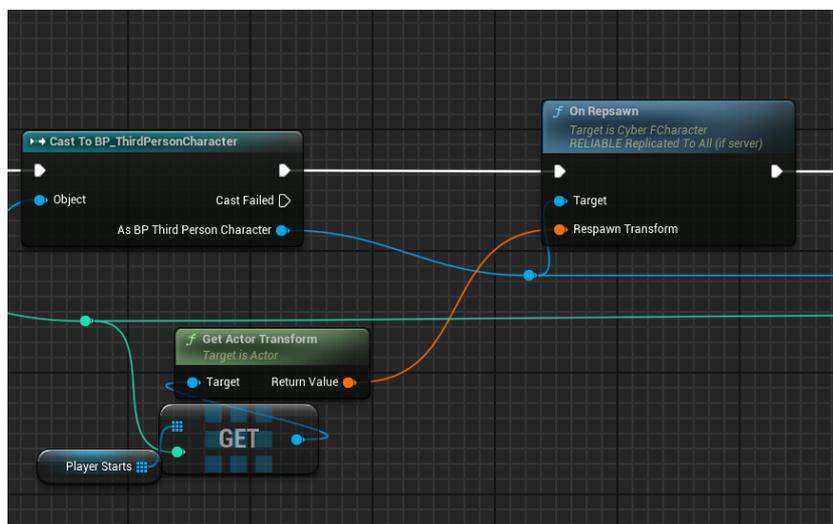
```

Цей код дозволяє вам викликати телепортацію на сервері, яка буде коректно синхронізована з клієнтом, враховуючи поворот.».

Для мене цієї інформації було достатньо. Я зрозумів що я не зовсім правильно зробив логіку переміщення. Я це робив тільки на сервері (рис. 3.4 а), а потрібно було створити RPC функцію, яку потрібно викликати із сервера, а виконуватись вона буде на клієнтах (рис. 3.4 б).



а)



б)

**Рисунок 3.4 – Приклад коду з описаною проблемою. а) код із помилкою; б) виправлений код**

C++ код блоку який вирішив проблему та зображений на рисунку 3.4 б – об’ява методу:

```
UFUNCTION(BlueprintCallable, NetMulticast, Reliable)
```

```
void OnRespawn(FTransform RespawnTransform);
```

реалізація:

```
void ACyberFCharacter::OnRespawn_Implementation(FTransform RespawnTransform)
{
    SetActorTransform(RespawnTransform);
}
```

ChatGPT наштовхнув мене на вирішення мого питання, але сам він не вирішив це питання. У згенерованому коді він створив RPC метод, який викликається на сервері. У своєму рішенні я зробив метод, який викликається з сервера на усіх клієнтах. Тобто, його рішення, телепортує тільки на сервері.

Клас `BP_ThirdPersonCharacter` має налаштування із синхронізацією переміщення. Тому коли на сервері персонаж змінив свою позицію, його нові координати синхронізуються між усіма гравцями. А ось поворот при телепортуванні – ні. Бо розрахунки з повороту виконуються окремо для кожного гравця на їх стороні і вони можуть відрізнитися один від одного. Данні які відповідають за розрахунок поворота – синхронізуюся, тому коли гравець почне рухатись, у інших гравців він почне дивитися у ту сторону куди йде.

Моїх знань вистачило щоб усвідомити виникнення цієї проблеми глядячи на відповіді ChatGPT. У більш складних запитаннях або коли не має достатнього досвіду треба зробити більше ітерацій із запитами та перевіркою. Все ще залежить від того як складено запит. Якщо робити більш конкретний запит, то відповіді будуть з урахуванням цього. Ось приклад питання по цій проблемі з урахуванням того, що ми вже знаємо рішення.

Мій запит «Перероби під `multicast` виклик. У його реалізації буде застосовуватися трансформ, який ми отримуємо при візі цього методу. Метод візитуватиметься з сервера для кожного гравця».

Відповідь ГШІ: «Звісно, ось оновлений код, де використовується Multicast для телепортації всіх гравців на сервері:

У вашому .h файлі гравця (PlayerController або Character):

```
#pragma once
#include "GameFramework/PlayerController.h"
#include "MyPlayerController.generated.h"
UCLASS()
class YOURGAME_API AMyPlayerController : public APlayerController
{
    GENERATED_BODY()
public:
    // Multicast подія для телепортації всіх гравців
    UFUNCTION(NetMulticast, Reliable)
    void MulticastTeleportAllPlayers(FTransform TargetTransform);
};
```

У вашому .cpp файлі гравця (PlayerController або Character):

```
#include "MyPlayerController.h"
#include "Net/UnrealNetwork.h"
void
AMyPlayerController::MulticastTeleportAllPlayers_Implementation(FTransform
TargetTransform)
{
    // Викликається на всіх клієнтах для телепортації всіх гравців до
вказаної точки та з вказаним поворотом
    GetCharacter()->SetActorTransform(TargetTransform);
}
// Деяка інша логіка гравця...
```

Приклад виклику телепортації на сервері:

```
if (HasAuthority())
{
```

```

// Вказуємо трансформ для телепортації всіх гравців
FTransform TargetTransform = FTransform(FRotator(NewRotation),
FVector(NewX, NewY, NewZ));
// Виклик Multicast події для телепортації всіх гравців
MulticastTeleportAllPlayers(TargetTransform);
}

```

Цей код використовує Multicast подію для телепортації всіх гравців до вказаної точки та повороту. Будьте впевнені, що ви налаштували всі необхідні параметри мережі і забезпечили, що ці методи визиваються на сервері.».

Як ми бачимо із відповіді, код який згенерував ГШІ не такий самий як і мій, написаний власноруч. Відрізняються тільки ім'я методу та клас де розміщений цей метод. А в цілому, логіка така сама.

Тож відповідь ГШІ залежить від правильно поставленого запитання, конкретизації та складності проблеми. Більш складну проблему треба розділяти на менші. Слід зазначити, що ChatGPT має обмежену кількість символів на відповідь. Це зроблено з тою цілю, щоб відповіді були стислими, а не великими. Якщо потрібно доповнити відповідь, або щось пояснити, можна про це попросити та ChatGPT зробить наступну відповідь, яка буде доповнювати попередню.

## ВИСНОВКИ

Розвиток штучного інтелекту та генеративного штучного інтелекту є ключовими напрямками в сучасному інформаційному та технологічному просторі. Штучний інтелект охоплює широкий спектр методів та технологій, спрямованих на створення систем, здатних виконувати завдання, які раніше вважалися виключно людськими. Від розпізнавання образів до мовного аналізу та прийняття рішень, ШІ стає все більш необхідною складовою нашого повсякденного життя.

З іншого боку, генеративний штучний інтелект визначається здатністю системи створювати новий контент або інформацію, яка не була введена в її навчальний набір. Це відкриває нові горизонти у творчості, мистецтві та науці. Генеративні моделі, такі як GPT-3, демонструють вражаючі здібності у генерації тексту, зображень та навіть музики.

У ході роботи було створено гру яка поєднує у себе різноманітні ігрові механіки та дозволяє грати по мережі. Було проведення ознайомлення із програмою для створення ігор - ігровим рушієм Unreal Engine 5. Було проведено огляд інтерфейсу програми та базових класів які грають ключову роль у створенні власної гри.

Завдяки генеративному штучному інтелекту був згенерований код, який використовує певний модуль рушію та реалізує можливість грати гравцям через глобальну мережу. Робота із ChatGPT та генерацією коду, дала змогу не тільки реалізувати можливість підключення гравців один до одного, а й дала можливість проаналізувати роботу ГШІ.

Був зроблений аналіз згенерованого коду, та надані коментарі по ньому. Підводячи підсумок, ГШІ можна використовувати у якості асистента при розробці навчальних ігор, але слід розуміти що це лише допоміжний інструмент та не завжди він дасть потрібну відповідь на поставлені запитання.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Unreal Engine main site [Електронний ресурс]. – Режим доступу: <https://www.unrealengine.com/en-US>
2. Unreal Engine: Platform SDK Upgrades [Електронний ресурс]. – Режим доступу: <https://docs.unrealengine.com/5.3/en-US/unreal-engine-5.3-release-notes/#platformsdkupgrades>
3. Unreal Engine: Creating a New Project [Електронний ресурс]. – Режим доступу: <https://docs.unrealengine.com/5.3/en-US/creating-a-new-project-in-unreal-engine/>
4. Unreal Engine: Unreal Editor Interface [Електронний ресурс]. – Режим доступу: <https://docs.unrealengine.com/5.3/en-US/unreal-editor-interface/>
5. Unreal Engine: Modeling and Geometry Scripting [Електронний ресурс]. – Режим доступу: <https://docs.unrealengine.com/5.3/en-US/modeling-and-geometry-scripting-in-unreal-engine/>
6. Unreal Engine: Materials [Електронний ресурс]. – Режим доступу: <https://docs.unrealengine.com/5.3/en-US/unreal-engine-materials/>
7. Unreal Engine: Online Subsystem [Електронний ресурс]. – Режим доступу: <https://docs.unrealengine.com/5.3/en-US/online-subsystem-in-unreal-engine/>
8. Unreal Engine: Coding Standard [Електронний ресурс]. – Режим доступу: <https://docs.unrealengine.com/4.26/en-US/ProductionPipelines/DevelopmentSetup/CodingStandard/>
9. Unreal Engine: Enhanced Input [Електронний ресурс]. – Режим доступу: <https://docs.unrealengine.com/5.0/en-US/enhanced-input-in-unreal-engine/>
10. Unreal Engine: Property Specifiers [Електронний ресурс]. – Режим доступу: <https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/GameplayArchitecture/Properties/Specifiers/>
11. Unreal Engine: Function Specifiers [Електронний ресурс]. – Режим доступу: <https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/GameplayArchitecture/Functions/Specifiers/>
12. Unreal Engine: UMG UI Designer [Електронний ресурс]. – Режим доступу: <https://docs.unrealengine.com/5.0/en-US/umg-ui-designer-for-unreal-engine/>

## ДОДАТОК А

### Код класу ACyberFCharacter

```
.h
// Copyright Epic Games, Inc. All Rights Reserved.
#pragma once
#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "Logging/LogMacros.h"
#include "CyberFCharacter.generated.h"
class USpringArmComponent;
class UCameraComponent;
class UInputMappingContext;
class UInputAction;
struct FInputActionValue;
DECLARE_LOG_CATEGORY_EXTERN(LogTemplateCharacter, Log, All);
UCLASS(config=Game)
class ACyberFCharacter : public ACharacter
{
    GENERATED_BODY()
    /** Camera boom positioning the camera behind the character */
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta = (AllowPrivateAccess = "true"))
    USpringArmComponent* CameraBoom;
    /** Follow camera */
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta = (AllowPrivateAccess = "true"))
    UCameraComponent* FollowCamera;

    /** MappingContext */
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
    UInputMappingContext* DefaultMappingContext;
    /** Jump Input Action */
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
    UInputAction* JumpAction;
    /** Move Input Action */
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
    UInputAction* MoveAction;
    /** Look Input Action */
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
    UInputAction* LookAction;
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
    UInputAction* SprintAction;
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, meta = (AllowPrivateAccess = "true"))
    UInputAction* PushAction;
    UPROPERTY(EditAnywhere, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))
    UCapsuleComponent* PushCapsule;
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Setup, meta = (AllowPrivateAccess = "true"))
```

```

float DefaultWalkMaxSpeed = 600.f;
UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Setup, meta = (AllowPrivateAccess = "true"))
float SprintMaxSpeed = 1200.f;
UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Setup, meta = (AllowPrivateAccess = "true"))
float SprintTime = 3.f;
UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Setup, meta = (AllowPrivateAccess = "true"))
float SprintResumeTime = 2.f;
float TargetSprintResumeValuePerSecond;
double time;

protected:
    float UsedSprintTime = 0;
    bool isSprintMode = false;

public:
    ACyberFCharacter();

protected:
    /** Called for movement input */
    void Move(const FInputActionValue& Value);
    /** Called for looking input */
    void Look(const FInputActionValue& Value);
    void Push();
    UFUNCTION(BlueprintCallable, BlueprintNativeEvent, meta=(DisplayName = "Push"))
    void K2_Push();
    void StartSprint();
    void StopSprint();
    UFUNCTION(BlueprintCallable, Server, Reliable)
    void StartSprint_S();
    UFUNCTION(BlueprintCallable, Server, Reliable)
    void StopSprint_S();

protected:
    // APawn interface
    virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent) override;
    // To add mapping context
    virtual void BeginPlay();
    virtual void Tick(float DeltaSeconds) override;

public:
    /** Returns CameraBoom subobject */
    FORCEINLINE class USpringArmComponent* GetCameraBoom() const { return CameraBoom; }
    /** Returns FollowCamera subobject */
    FORCEINLINE class UCameraComponent* GetFollowCamera() const { return FollowCamera; }
    bool GetIsSprintMode() const;
    UFUNCTION(BlueprintCallable, NetMulticast, Reliable)
    void OnRepsawn(FTransform RespawnTransform);
    UFUNCTION(BlueprintCallable, BlueprintNativeEvent)
    void ResetCamera();
};

```

```

.cpp
// Copyright Epic Games, Inc. All Rights Reserved.
#include "CyberFCharacter.h"
#include "Engine/LocalPlayer.h"
#include "Camera/CameraComponent.h"
#include "Components/CapsuleComponent.h"
#include "GameFramework/CharacterMovementComponent.h"
#include "GameFramework/SpringArmComponent.h"
#include "GameFramework/Controller.h"
#include "EnhancedInputComponent.h"
#include "EnhancedInputSubsystems.h"
#include "InputActionValue.h"
DEFINE_LOG_CATEGORY(LogTemplateCharacter);
/////////////////////////////////////////////////////////////////
// ACyberFCharacter
ACyberFCharacter::ACyberFCharacter()
{
    // Set size for collision capsule
    GetCapsuleComponent()->InitCapsuleSize(42.f, 96.0f);
    // Don't rotate when the controller rotates. Let that just affect the camera.
    bUseControllerRotationPitch = false;
    bUseControllerRotationYaw = false;
    bUseControllerRotationRoll = false;
    // Configure character movement
    GetCharacterMovement()->bOrientRotationToMovement = true; // Character moves in the direction of input...
    GetCharacterMovement()->RotationRate = FRotator(0.0f, 500.0f, 0.0f); // ...at this rotation rate
    // Note: For faster iteration times these variables, and many more, can be tweaked in the Character Blueprint
    // instead of recompiling to adjust them
    GetCharacterMovement()->JumpZVelocity = 700.f;
    GetCharacterMovement()->AirControl = 0.35f;
    GetCharacterMovement()->MaxWalkSpeed = DefaultWalkMaxSpeed;
    GetCharacterMovement()->MinAnalogWalkSpeed = 20.f;
    GetCharacterMovement()->BrakingDecelerationWalking = 2000.f;
    GetCharacterMovement()->BrakingDecelerationFalling = 1500.0f;
    // Create a camera boom (pulls in towards the player if there is a collision)
    CameraBoom = CreateDefaultSubobject<USpringArmComponent>(TEXT("CameraBoom"));
    CameraBoom->SetupAttachment(RootComponent);
    CameraBoom->TargetArmLength = 400.0f; // The camera follows at this distance behind the character
    CameraBoom->bUsePawnControlRotation = true; // Rotate the arm based on the controller
    // Create a follow camera
    FollowCamera = CreateDefaultSubobject<UCameraComponent>(TEXT("FollowCamera"));
    FollowCamera->SetupAttachment(CameraBoom, USpringArmComponent::SocketName); // Attach the camera to
the end of the boom and let the boom adjust to match the controller orientation
    FollowCamera->bUsePawnControlRotation = false; // Camera does not rotate relative to arm
    // Note: The skeletal mesh and anim blueprint references on the Mesh component (inherited from Character)
    // are set in the derived blueprint asset named ThirdPersonCharacter (to avoid direct content references in C++)

```

```

PushCapsule = CreateDefaultSubobject<UCapsuleComponent>(TEXT("PushCapsule"));
PushCapsule->SetupAttachment(RootComponent);
PushCapsule->SetRelativeLocation(FVector(55, 0, -30));
PushCapsule->SetCapsuleHalfHeight(50);
PushCapsule->SetCapsuleRadius(30);
PushCapsule->SetGenerateOverlapEvents(true);
}
void ACyberFCharacter::BeginPlay()
{
    // Call the base class
    Super::BeginPlay();
    //Add Input Mapping Context
    if (APlayerController* PlayerController = Cast<APlayerController>(Controller))
    {
        if (UEnhancedInputLocalPlayerSubsystem* Subsystem =
        ULocalPlayer::GetSubsystem<UEnhancedInputLocalPlayerSubsystem>(PlayerController->GetLocalPlayer()))
        {
            Subsystem->AddMappingContext(DefaultMappingContext, 0);
        }
    }
    const float TargetFPS = 60;
    TargetSprintResumeValuePerSecond = (1 / TargetFPS) / SprintResumeTime;
}
void ACyberFCharacter::Tick(float DeltaSeconds)
{
    Super::Tick(DeltaSeconds);
    if (GetLocalRole() == ROLE_Authority)
    {
        if (isSprintMode)
        {
            UsedSprintTime += DeltaSeconds;
            if (SprintTime - UsedSprintTime <= 0)
            {
                StopSprint();
            }
        }
        else if (UsedSprintTime != 0)
        {
            const float TargetFPS = 60;
            const float CurrnetFPS = 1 / DeltaSeconds;
            const float SprintResumePart = (TargetSprintResumeValuePerSecond * TargetFPS) /
            CurrnetFPS;

            UsedSprintTime -= SprintResumePart * SprintTime;
            UsedSprintTime = FMath::Clamp(UsedSprintTime, 0, UsedSprintTime);
        }
    }
}

```

```

}
bool ACyberFCharacter::GetIsSprintMode() const
{
    return isSprintMode;
}
/////////////////////////////////////////////////////////////////
// Input
void ACyberFCharacter::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
    // Set up action bindings
    if (UEnhancedInputComponent* EnhancedInputComponent =
Cast<UEnhancedInputComponent>(PlayerInputComponent)) {
        // Jumping
        EnhancedInputComponent->BindAction(JumpAction, ETriggerEvent::Started, this,
&ACharacter::Jump);
        EnhancedInputComponent->BindAction(JumpAction, ETriggerEvent::Completed, this,
&ACharacter::StopJumping);
        // Moving
        EnhancedInputComponent->BindAction(MoveAction, ETriggerEvent::Triggered, this,
&ACyberFCharacter::Move);
        // Looking
        EnhancedInputComponent->BindAction(LookAction, ETriggerEvent::Triggered, this,
&ACyberFCharacter::Look);
        EnhancedInputComponent->BindAction(PushAction, ETriggerEvent::Started, this,
&ACyberFCharacter::Push);
        EnhancedInputComponent->BindAction(SprintAction, ETriggerEvent::Started, this,
&ACyberFCharacter::StartSprint);
        EnhancedInputComponent->BindAction(SprintAction, ETriggerEvent::Completed, this,
&ACyberFCharacter::StopSprint);
    }
    else
    {
        UE_LOG(LogTemplateCharacter, Error, TEXT("%s' Failed to find an Enhanced Input component! This
template is built to use the Enhanced Input system. If you intend to use the legacy system, then you will need to update this C++ file."),
*GetNameSafe(this));
    }
}
void ACyberFCharacter::Move(const FInputActionValue& Value)
{
    // input is a Vector2D
    FVector2D MovementVector = Value.Get<FVector2D>();
    if (Controller != nullptr)
    {
        // find out which way is forward
        const FRotator Rotation = Controller->GetControlRotation();
        const FRotator YawRotation(0, Rotation.Yaw, 0);

```

```

        // get forward vector
        const FVector ForwardDirection = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::X);
        // get right vector
        const FVector RightDirection = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::Y);
        // add movement
        AddMovementInput(ForwardDirection, MovementVector.Y);
        AddMovementInput(RightDirection, MovementVector.X);
    }
}
void ACyberFCharacter::Look(const FInputActionValue& Value)
{
    // input is a Vector2D
    FVector2D LookAxisVector = Value.Get<FVector2D>();
    if (Controller != nullptr)
    {
        // add yaw and pitch input to controller
        AddControllerYawInput(LookAxisVector.X);
        AddControllerPitchInput(LookAxisVector.Y);
    }
}
void ACyberFCharacter::Push()
{
    K2_Push();
}
void ACyberFCharacter::K2_Push_Implementation()
{}
void ACyberFCharacter::StartSprint()
{    StartSprint_S(); }
void ACyberFCharacter::StopSprint()
{    StopSprint_S();}
void ACyberFCharacter::StartSprint_S_Implementation()
{
    const float MinTimeToActivate = 1.f;
    isSprintMode = (SprintTime - UsedSprintTime) > MinTimeToActivate;
    if (!isSprintMode) { return; }
    GetCharacterMovement()->MaxWalkSpeed = SprintMaxSpeed;
}
void ACyberFCharacter::StopSprint_S_Implementation()
{
    if (!isSprintMode) { return; }
    isSprintMode = false;
    GetCharacterMovement()->MaxWalkSpeed = DefaultWalkMaxSpeed;
}
void ACyberFCharacter::OnRespawn_Implementation(FTransform RespawnTransform)
{
    SetActorTransform(RespawnTransform);
}

```

```
        ResetCamera();  
    }  
    void ACyberFCharacter::ResetCamera_Implementation()  
    {}
```

## ДОДАТОК Б

### Код класу ACyberFPlayerController

```
.h
// Fill out your copyright notice in the Description page of Project Settings.
#pragma once
#include "CoreMinimal.h"
#include "GameFramework/PlayerController.h"
#include "Interfaces/OnlineSessionInterface.h"
#include "CyberFPlayerController.generated.h"
class IOnlineSubsystem;
UCLASS()
class CYBERF_API ACyberFPlayerController : public APlayerController
{
    GENERATED_BODY()
protected:
    virtual void BeginPlay() override;
    IOnlineSubsystem* OSS;
    UFUNCTION(BlueprintCallable, Category = EOS)
    void EOSLogin();
    UFUNCTION(BlueprintCallable, Category = EOS)
    void EOSCreateSession();
    UFUNCTION(BlueprintCallable, Category = EOS)
    void EOSFindSession();
    UFUNCTION(BlueprintCallable, Category = EOS)
    void EOSDestroySession();
    void OnLoginComplete( int32 LocalUserNum, bool bWasSuccessful, const FUniqueNetId& UserId, const FString& Error);
    void OnCreateSessionComplete(FName SessionName, bool bWasSuccessful);
    void OnFindSessionsComplete(bool bWasSuccessful);
    void OnJoinSessionComplete(FName SessionName, EOnJoinSessionCompleteResult::Type Result);
    static const FName CF_SESSION_NAME;
    TSharedPtr<FOnlineSessionSearch> SessionSearch;
};
.cpp
// Fill out your copyright notice in the Description page of Project Settings.
#include "CyberFPlayerController.h"
#include "OnlineSessionSettings.h"
#include "OnlineSubsystem.h"
#include "GameFramework/PlayerState.h"
#include "Interfaces/OnlineIdentityInterface.h"
#include "Interfaces/OnlineSessionDelegates.h"
#include "Interfaces/OnlineSessionInterface.h"
#include "Kismet/GameplayStatics.h"
#include "Online/OnlineSessionNames.h"
const FName ACyberFPlayerController::CF_SESSION_NAME = FName(TEXT("LBSession"));
void ACyberFPlayerController::BeginPlay()
```

```

{
    Super::BeginPlay();
    const FName OSS_Name = FName("EOS");
    OSS = IOnlineSubsystem::Get(OSS_Name);
    if (OSS != nullptr)
    {
        if (IOnlineIdentityPtr Identity = OSS->GetIdentityInterface())
        {
            Identity->AddOnLoginCompleteDelegate_Handle(0, FOnLoginCompleteDelegate::CreateUObject(this,
&ThisClass::OnLoginComplete));
        }
        if (IOnlineSessionPtr Session = OSS->GetSessionInterface())
        {
            Session-
>AddOnCreateSessionCompleteDelegate_Handle(FOnCreateSessionCompleteDelegate::CreateUObject(this,
&ThisClass::OnCreateSessionComplete));
            Session-
>AddOnFindSessionsCompleteDelegate_Handle(FOnFindSessionsCompleteDelegate::CreateUObject(this,
&ThisClass::OnFindSessionsComplete));
            Session-
>AddOnJoinSessionCompleteDelegate_Handle(FOnJoinSessionCompleteDelegate::CreateUObject(this,
&ThisClass::OnJoinSessionComplete));

        }
    }
}

void ACyberFPlayerController::EOSLogin()
{
    if (OSS != nullptr)
    {
        if (IOnlineIdentityPtr Identity = OSS->GetIdentityInterface())
        {
            FOnlineAccountCredentials Credentials;
            Credentials.Type = "AccountPortal";
            Identity->Login(0, Credentials);
        }
    }
}

void ACyberFPlayerController::EOSCreateSession()
{
    FOnlineSessionSettings SessionSettings;
    SessionSettings.NumPublicConnections = 2;
    SessionSettings.NumPrivateConnections = 0;
    SessionSettings.bShouldAdvertise = true;
    SessionSettings.bIsLANMatch = false;
    SessionSettings.bAllowInvites = true;
}

```

```

SessionSettings.bUsesPresence = true;
SessionSettings.bAllowJoinViaPresence = true;
SessionSettings.bAllowJoinViaPresenceFriendsOnly = false;
SessionSettings.Set(SEARCH_KEYWORDS, FString("Session"), EOnlineDataAdvertisementType::ViaOnlineService);
SessionSettings.Set(SETTING_MAPNAME, FString("Map1"), EOnlineDataAdvertisementType::ViaOnlineService);
if (OSS != nullptr)
{
    if (IOnlineSessionPtr Session = OSS->GetSessionInterface())
    {
        Session->CreateSession(0, CF_SESSION_NAME, SessionSettings);
    }
}
}
void ACyberFPlayerController::EOSFindSession()
{
    if (OSS != nullptr)
    {
        if (IOnlineSessionPtr Session = OSS->GetSessionInterface())
        {
            if (!SessionSearch.IsValid())
            {
                SessionSearch = MakeShareable(new FOnlineSessionSearch);
            }
            SessionSearch->QuerySettings.Set(SEARCH_PRESENCE, true,
EOnlineComparisonOp::Type::Equals);
            SessionSearch->MaxSearchResults = 10;
            SessionSearch->bIsLanQuery = false;
            Session->FindSessions(0, SessionSearch.ToSharedRef());
        }
    }
}
void ACyberFPlayerController::EOSDestroySession()
{
    if (OSS != nullptr)
    {
        if (IOnlineSessionPtr Session = OSS->GetSessionInterface())
        {
            Session->DestroySession(CF_SESSION_NAME);
        }
    }
}
void ACyberFPlayerController::OnLoginComplete(int32 LocalUserNum, bool bWasSuccessful, const FUniqueNetId&
UserId,
const FString& Error)
{
    if(bWasSuccessful)

```

```

    {
        PlayerState->SetUniqueId(UserId);
    }
}

void ACyberFPlayerController::OnCreateSessionComplete(FName SessionName, bool bWasSuccessful)
{
    if(bWasSuccessful)
    {
        GetWorld()->ServerTravel(FString("Map1?listen"));
    }
}

void ACyberFPlayerController::OnFindSessionsComplete(bool bWasSuccessful)
{
    if (!bWasSuccessful)
    {
        return;
    }
    for (int32 index = 0; index < SessionSearch->SearchResults.Num(); index++)
    {
        if (SessionSearch->SearchResults.IsValidIndex(index))
        {
            if (OSS != nullptr)
            {
                if (IOnlineSessionPtr Session = OSS->GetSessionInterface())
                {
                    UE_LOG(LogOnlineSession, Log, TEXT("JoinToSession"));
                    Session->JoinSession(0, CF_SESSION_NAME, SessionSearch-
>SearchResults[index]);
                }
            }
        }
    }
}

void ACyberFPlayerController::OnJoinSessionComplete(FName SessionName, EOnJoinSessionCompleteResult::Type
Result)
{
    if (Result == EOnJoinSessionCompleteResult::Success)
    {
        if (OSS != nullptr)
        {
            if (IOnlineSessionPtr Session = OSS->GetSessionInterface())
            {
                FString JoinedAddress;
                Session->GetResolvedConnectString(SessionName, JoinedAddress);
                if (!JoinedAddress.IsEmpty())
                {

```

```
        if (UWorld* World = GetWorld())
        {
            ClientTravel(JoinedAddress, TRAVEL_Absolute);
        }
    }
}
}
```