МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

Комп’ютерна гра засобами Unreal Engine

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп’ютерні науки»**

Освітня програма «**Прикладне програмування»**

Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи ПП-41

_____Меламед М.П._____
(прізвище та ініціали)

Керівник _  _Криволапов Я.В._____
(прізвище та ініціали)

_____
(науковий ступінь, звання)

Унікальність тексту 97%

Випускна кваліфікаційна робота бакалавра допущена до захисту
рішенням кафедри *прикладних інформаційних систем*
Протокол № від р.
зав. кафедри_____ Плескач В.Л.

Київ – 2023

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

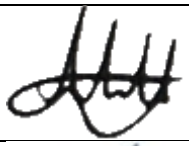
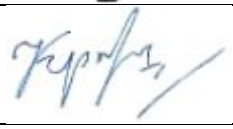
№з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	14.10.2022	Виконано
2.	Видача завдання кваліфікаційної роботи бакалавра	24.10.2022	Виконано
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	31.10.2022	Виконано
4.	Затвердження плану кваліфікаційної роботи бакалавра	01.11.2022	Виконано
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	08.11.2022	Виконано
6.	Підготовка і подання науковому керівнику першого варіанту I розділу роботи	21.12.2022	Виконано
7.	Підготовка і подання науковому керівнику першого варіанту II розділу роботи	31.01.2023	Виконано
8.	Підготовка і подання науковому керівнику першого варіанту III розділу роботи	30.03.2023	Виконано
9.	Подання роботи у першому варіанті	28.04.2023	Виконано
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2023	Виконано
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	**22.05.2023**	Виконано
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	26.05.2023	
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	12.06.2023	
14.	Захист кваліфікаційної роботи бакалавра	26.06.2023	

Здобувач вищої освіти _____ _____
(підпис)

Керівник _____ _____
(підпис)

ВІДОМІСТЬ ДИПЛОМНОЇ РОБОТИ

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Календарний план дипломної роботи	1
Відомість дипломної роботи	1
Анотація	1
Анотація (іноземною мовою – англійською)	1
Зміст	1
Вступ	2
1	8
2	8
3	8
Висновки	1
Перелік використаних джерел	1
Додатки	3

<table>
<tr><td></td><td></td><td></td><td></td><td colspan="3" rowspan="2">ДПП XXXX 00.000.00</td></tr>
<tr><td></td><td>ПІБ</td><td>Підп.</td><td>Дата</td></tr>
<tr><td>Розробн.</td><td>Меламед М.П.</td><td>  </td><td>20.06.2023</td><td rowspan="4">Відомість дипломної роботи</td><td>Лист</td><td>Листів</td></tr>
<tr><td>Керівн.</td><td>Криволапов Я.В.</td><td>  </td><td>20.06.2023</td><td></td><td></td></tr>
<tr><td>Н/контр.</td><td>Макаренко С.А.</td><td></td><td></td><td colspan="2" rowspan="2"></td></tr>
<tr><td>Зав.каф.</td><td>Плескач В.Л.</td><td></td><td></td></tr>
</table>

АНОТАЦІЯ

Дипломна робота: 32 с., 3 табл., 19 рис., 11 джерел.

Ця дипломна робота присвячена проектуванню та розробленню гри за допомогою рушія Unreal Engine 5.

Метою курсової роботи є розробка комп'ютерної гри із застосуванням сучасного ігрового рушія Unreal Engine 5.

Завдання дослідження. Успішне проведення та завершення дослідження передбачає:

- дослідження теоретичних основ побудови відеоігор, зокрема використовуючи рушій Unreal Engine 5;
- дослідження популярних наразі ігрових жанрів та їх поєднань;
- проектування та реалізація комп'ютерної гри.

Об'єктом дослідження кваліфікаційної роботи є процес реалізації комп'ютерних ігор з використанням рушію Unreal Engine 5.

Предметом дослідження кваліфікаційної роботи є сучасні теоретичні, архітектурні та програмні засоби створення відеоігор для платформи Windows, зокрема із використанням сучасних рушіїв, а саме Unreal Engine 5.

Методи дослідження:

- для дослідження ринку на предмет наявності схожих ігор застосовано метод аналізу;
- для формування алгоритму розробки та моделювання проекту використано описовий метод;
- для визначення переваг та недоліків альтернативних реалізацій використано метод порівняння;
- для формування діаграм та моделей використано метод наукового моделювання;

Ключові слова: відео гра, ігрові рушії, ігрові мережеві технології, Unreal Engine 5.

ABSTRACT

Thesis: 32 pages, 3 tables, 19 figures, 11 sources

This thesis is devoted to the design and development of a video game using Unreal Engine 5.

The purpose of the thesis is development of a video game using modern game engine Unreal Engine 5.

To achieve this goal you need to solve these **tasks**:

- To study the theoretical foundations of video games development, in particular using Unreal Engine 5;
- To study popular nowadays game genres and their combinations;
- To design and develop a video game.

Object of study is development of video games using Unreal Engine 5 game engine.

Subjects of study are modern theoretical, architectural and software means of developing video games for Windows platform, in particular using modern game engines, such as Unreal Engine 5.

Research methods:

- To study current video games market for similar games, the analysis method was used;
- To design algorithm of development and project model, the descriptive method was used;
- To determine advantages and disadvantages of alternative implementations, the comparative method was used;
- To form figures and diagrams, the scientific modelling method was used.

Keywords: video game, game engine, game networking, Unreal Engine 5

ЗМІСТ

ВСТУП

Актуальність цієї теми полягає в тому, щоб знайти нові способи поєднання вже відомих ігрових жанрів з метою створення нових, унікальних, більш привабливих для деяких авдиторій ігор.

Метою курсової роботи є розробка комп’ютерної гри із застосуванням сучасного ігрового рушія Unreal Engine 5.

Завдання дослідження. Успішне проведення та завершення дослідження передбачає:

- дослідження теоретичних основ побудови відеоігор, зокрема використовуючи рушій Unreal Engine 5;
- дослідження популярних наразі ігрових жанрів та їх поєднань;
- проектування та реалізація комп’ютерної гри.

Об’єктом дослідження кваліфікаційної роботи є процес реалізації комп’ютерних ігор з використанням рушію Unreal Engine 5.

Предметом дослідження кваліфікаційної роботи є сучасні теоретичні, архітектурні та програмні засоби створення відеоігор для платформи Windows, зокрема із використанням сучасних рушіїв, а саме Unreal Engine 5.

Методи дослідження:

- для дослідження ринку на предмет наявності схожих ігор застосовано метод аналізу;
- для формування алгоритму розробки та моделювання проекту використано описовий метод;
- для визначення переваг та недоліків альтернативних реалізацій використано метод порівняння;
- для формування діаграм та моделей використано метод наукового моделювання;

Практичне значення отриманих результатів полягає в тому, що розроблена комп’ютерна реалізація, що може слугувати дозвіллям для людини, щоб вона мала здатність грати з друзями.

Структура роботи:

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, розподілених на підрозділи та висновку.

РОЗДІЛ 1
ПІДХОДИ ДО РОЗРОБЛЕННЯ КОМП'ЮТЕРНИХ ІГОР НА БАЗІ СУЧАСНИХ ІГРОВИХ РУШІЇВ

1.1. Огляд принципів та підходів до розробки відеоігор

Розробка відеоігор, як окремий напрямок програмування, з'явилася ще в 1947 році, з появою першої в світі інтерактивної програми. Перша комерційна відеогра, Computer Space, вийшла на ринок у 1971 році, і з тих пір відеоігри стали дуже поширеним явищем та популярним способом дозвілля. В 1975 році з'явилася Pong - перша відеогра, доступна вдома.

З самого початку ігри розвивалися дуже швидко, намагаючись завжди надати гравцю найбільш привабливий ігровий процес. При цьому деякі ігри надавали перевагу реалістичності, деякі – глибині механік, деякі – навпаки, простоті освоєння. При цьому, для надання користувачу найкращого досвіду гри та реалізації найбільш пасуючих грі механік, розробники дуже часто потрапляли в ситуації, коли апаратне забезпечення доводилося використовувати на повну, перед цим тижнями оптимізуючи гру. Через це саме в відеоігровій індустрії виникло багато алгоритмів, зорієнтованих не стільки на точність, скільки на швидкість, деякі з яких важко розшифрувати навіть досвідченому розробнику. З часом в індустрії склалося декілька можливих архітектурних підходів до розробки ігор, кожен з яких має власні переваги та недоліки. Розглянемо основні з них:

- об'єктно-орієнтований підхід. Заснований на парадигмі об'єктно-орієнтованого програмування. При такому підході усе в грі – об'єкти, що належать до певних класів. Класи можуть спадкуватися один від одного, що дозволяє перевикористовувати код для створення спеціалізацій ігрових об'єктів. При такому підході дані об'єкту об'єднані з його логікою. Основними недоліками такого підходу є: заплутані ієрархії класів, коли об'єкт має поєднувати в собі властивості декількох інших класів; неефективне використання кешу процесора, адже в об'єктно-орієнтованому програмуванні дуже часто

відбувається перехід за вказівником, через що процесор не може ефективно передбачити, які дані будуть використані наступні;

- компонентний підхід. Цей підхід також заснований на об’єктно-орієнтованому програмуванні, однак при такому підході об’єктами виступають не цілі ігрові сутності, а лише окремі компоненти (наприклад, компонент отримання вводу від гравця, компонент руху, компонент фізики, тощо). Самі ігрові сутності при такому підході складаються з набору компонентів, який і визначає їхню поведінку. При такому підході архітектура гри вільна від складних ієрархій, однак все ще багато використовує вказівники. Існує також модифікація цього методу, при якій сама сутність також містить в собі дані й свою поведінку, специфічну саме для цієї сутності;
- сутність-компонент-система. При цьому підході дані повністю відділенні від поведінки. Архітектура гри ділиться на 2 частини: компоненти та системи. Сутність в пам’яті є не що інше, як простий порядковий номер. Усі компоненти розташовуються в пам’яті поспіль, так, щоб зробити ітерацію за ними як можна більш швидкою. Компоненти містять в собі лише дані. Систему, в свою чергу, отримують лише потрібні їм компоненти необхідних сутностей, і обробляють їх (найчастіше в простому циклі). Наприклад, для виведення спрайту на екран достатньо простої ітерації за компонентами позиції та відображення у всіх тих сутностей, у яких є обидва ці компоненти. Така архітектура дозволяє найефективніше використовувати процесорний кеш (оскільки шанс отримати промах в кеші мінімальний через те, що дані завжди розташовані поруч), а складність архітектури падає через повне розділення даних від поведінки. Недоліком цього підходу є його новизна, а тому –

незвичність для багатьох розробників, а також невелика кількість великих рушіїв, що його підтримують.

В Unreal Engine 5 використовується підхід, що є модифікацією компонентного: актори розбиваються на компоненти, але при цьому також можуть мати специфічні для себе дані та поведінку.

1.2. Стек використовуваних технологій для відеоігор

Для реалізації програмної системи курсової роботи був обраний саме рушій Unreal Engine 5. Щоб обґрунтувати свій вибір, пропоную порівняння трьох, як мені здається, найбільш розвинутих на даний момент безкоштовних рушіїв:

Таблиця 1.1 - Порівняння ігрових рушіїв

Що порівнюється	Unreal Engine 4	Unity3D	Godot
Мова програмування	C++, Blueprint (візуальний)	C#, візуальні скрипти	GDScript, VisualScript (візуальний), C#, GDNative
Ціна	Безкоштовно для проектів, чий прибуток менше $3000 на місяць	Існує декілька планів підписки, є безкоштовний план для некомерційного використання	Повністю безкоштовний

Продовження таблиці 1.1

Підхід до архітектури гри	Мапи складаються з акторів (об’єктів класів, успадкованих від AActor). Актори мають власну логіку та компоненти, які можуть використовуватися для перевикористання логіки у різних акторах. Актори можуть спадкувати інших акторів (Вищезгадана модифікація компонентного підходу)	Усе є GameObject, але GameObject може мати компоненти – скрипти, що визначають логіку роботи. Кожен скрипт є окремим класом, тобто може спадкуватися від інших скриптів (чистий компонентний підходу).	Усе є сцена. Сцена складається з сцен та містить в собі логіку. Кожна сцена також є класом, тож може спадкуватися від інших сцен (чистий ОО підхід).

Продовження таблиці 1.1

Бібліотека для інтерфейсу	Unreal Motion Graphics. Віджети складають окрему ієрархію і взаємодіють з грою через HUD	UI Toolkit – колекція інструментів, що дозволяє створювати інтерфейси у манері, схожій на розробку сайтів. Має свою мову розмітки UXML та мову стилізації USS. uGUI – бібліотека для розробки інтерфейсу за допомогою GameObject. Елементи інтерфейсу є такими ж самими об’єктами, як і все інше.	Елементи інтерфейсу є повноправними сценами, інтерфейс будується за загальними правилами побудови сцени.
Бібліотека з найбільш використовуваними класами для геймплею	Gameplay Framework	Немає	Немає

Продовження таблиці 1.1

Стек для реалізації мережевої взаємодії	Власна бібліотека, заснована на двобічному RPC та реплікації акторів	На даний момент немає. Існує застарівший UNet, Netcode, що зараз в розробці, та деякі сторонні рішення	Власна бібліотека, заснована на двобічному RPC та реплікації акторів.

В цій таблиці відсутні багато позицій для порівняння, таких як можливості анімації, звуку, тощо. Також у таблиці відсутнє порівняння платформ, для яких можливе використання рушію, так як нас зараз цікавить лише платформа ОС Windows.

1.3. Дослідження ринку ігор та популярних жанрів

Для розробки успішної відеогри, передусім треба дослідити, які ігри зараз популярні, які ігри були добре сприйняті критиками, у які ігри досі грають, незалежно від їх віку, тощо. Оскільки перед нами стоїть задача розробити багатокористувацьку гру, то і шукати ми будемо серед багатокористувацьких ігор.

Проведемо дослідження статистики цифрової платформи Steam за допомогою сайту SteamDB:

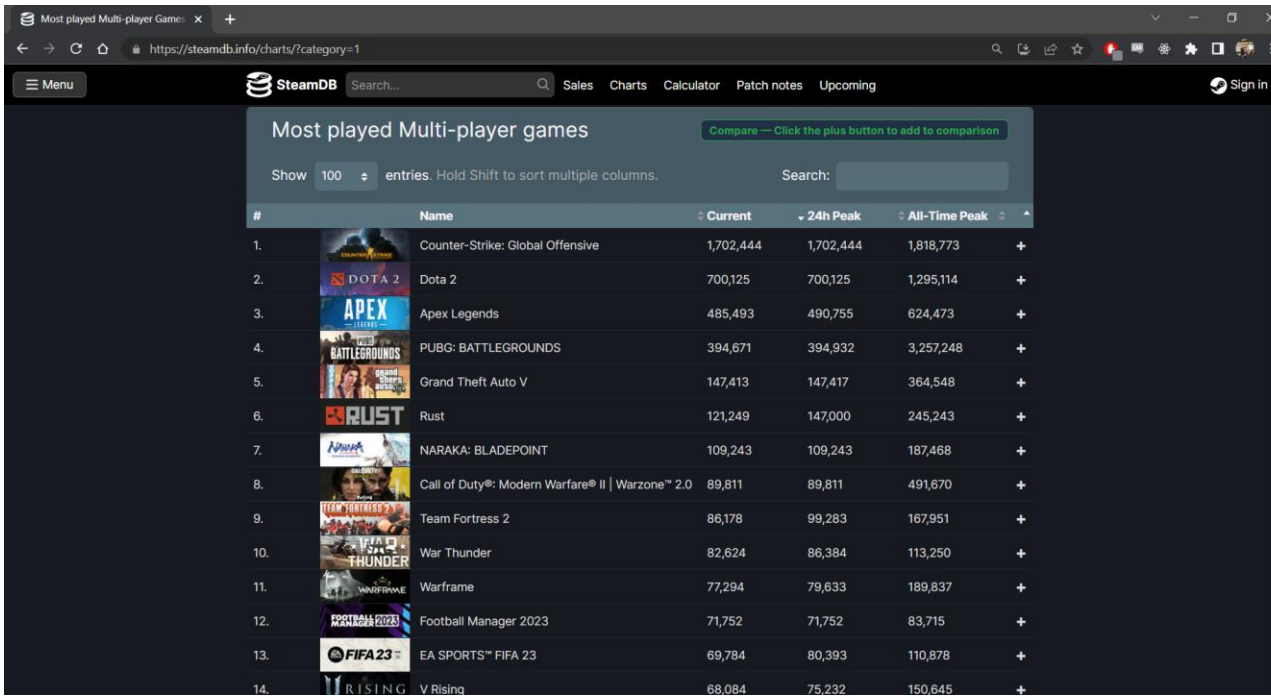


Рисунок 1.1. - Найпопулярніші багатокористувацькі ігри за статистикою SteamDB

Дослідимо механіки перших 5 ігор:

- Counter-Strike: Global Offensive – командний шутер від першої особи. Гравці поділяються на 2 команди по 5 осіб: команда терористів та команда контр-терористів. Задача кожної команди змінюється в залежності від режиму гри, найпопулярніших яких є 2: рятування заручників та закладка бомби. Окрім задач режиму гри, які є асиметричними, спорядження гравців є повністю симетричним, тобто кожен гравець теоретично під час матчу має доступ до будь-якого доступного спорядження.

- Dota 2 – багатокористувацька онлайнова бойова арена, в якій основною задачею кожної команди є руйнування бази супротивника. В кожній команді грає по 5 гравців. На початку матча кожен гравець обирає собі персонажа, кожен з яких має свої унікальні здібності, і під час матчу гравці не можуть змінювати свого персонажу. Таким чином,

важливим чинником перемоги для команди є не лише командна гра під час гри, а й підбір персонажів, що добре поєднуються один з одним.

- Apex Legends – командна гра жанру «королівської битви». Задачею кожної з 20 команд є залишитися останньою командою на полі битви, що невпинно зменшується. При цьому кожен з гравців починає гру лише зі здібностями свого обраного персонажу і без зброї, у той час як зброю усі гравці повинні шукати на полі бою у розпалі матчу. При цьому, на відміну від інших популярних шутерів, гравець, що був «вбитий», не повертається у гру, а вибуває з матчу.
- PUBG: BATTLEGROUNDS - командна гра жанру «королівської битви». Як і в Apex Legends, задачею кожного гравця є залишитися останнім на полі бою, але на відміну від Apex Legends, відсутній поділ на команди, а також механіки персонажів: усі гравці стартують в рівних умовах та не мають унікальних здібностей.
- Grand Theft Auto V – однокористувацька гра з можливістю кооперативної гри до 30 осіб. Оскільки гра є однокористувацькою, розглядання її механік виходить за межі цієї роботи.

З цих 5 ігор ми можемо побачити, що при багатокористувацькій грі гравці в основному зацікавлені в командних шутерах, найчастіше з механіками персонажів (так звані «геройські шутери»). Розглянемо ще декілька популярних ігор того ж жанру:

- Overwatch – багатокористувацький геройський шутер, в якому команди по 6 осіб мають виконувати завдання на карті, в залежності від режиму гри. Основних режимів гри 4: контроль об’єкту, супровід, захват точок та змішаний супровід з захватом точок.
- Valorant - багатокористувацький геройський шутер, що являє собою поєднання механік Counter-Strike Global Offensive з механікою

персонажі. З двох команд обирається та, що атакує, та та, що захищає. Задачею атакуючих є закладення бомби, а задачею захисників – не дати атакуючим закласти бомбу.

Звернувши увагу на все це, було вирішено реалізовувати командний геройський шутер від третьої особи, основною задачею команд в якому є забрати флаг з бази супротивника і перенести на свою базу (режим «захват флагу»), поєднаний з механіками багатокористувацьких онлайн бойових арен: гравці не тільки мають 1 здібність, але й можуть будувати башти та викликати «міньйонів», що керуються ігровим штучним інтелектом.

РОЗДІЛ 2
АНАЛІЗ ПРОГРАМНО-АРХІТЕКТУРНИХ РІШЕНЬ ДЛЯ РЕАЛІЗАЦІЇ КОМП'ЮТЕРНИХ ІГОР

2.1. Інформаційне забезпечення проектованої системи

Розробка програмного продукту вимагає використання певного стеку технологій. В даному випадку у якості стеку використовується рушій Unreal Engine 5 і його компоненти, зокрема:

- Blueprints Visual Scripting: мова візуального програмування, призначена для спрощення роботи над грою програмістів та дизайнерів. Зручна для використання там, де необхідна можливість швидко змінювати значення для швидких дизайн-ітерацій (прикинули значення, поставили, перевірили в грі, і так аж доки не знайдеться збалансоване значення). На рисунку 2.1. можна побачити приклад використання системи Blueprint для програмування HUD-інтерфейсу гравця;

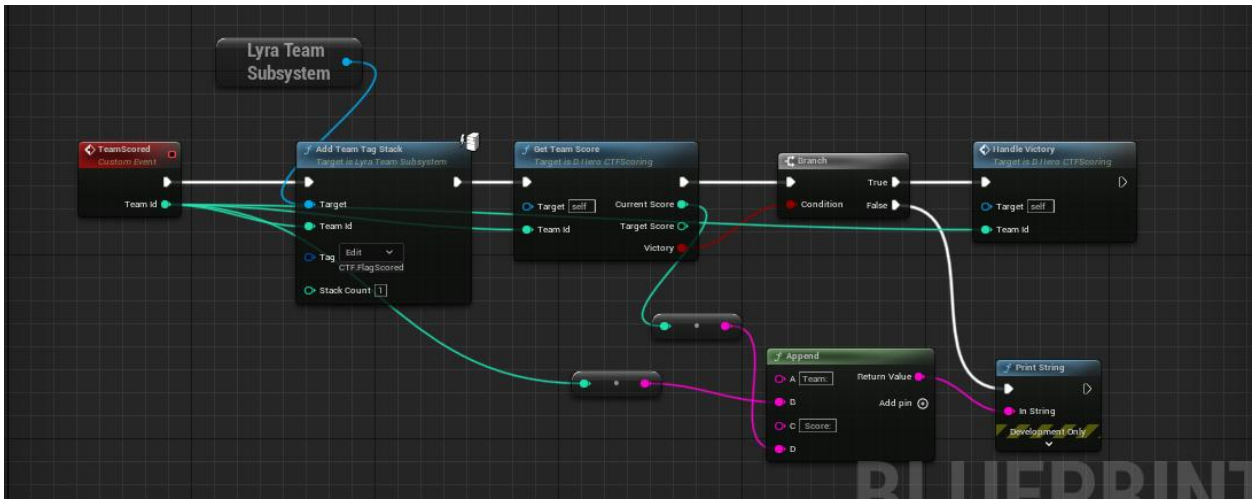


Рисунок 2.1. - Blueprint-скрипт для зарахування флагу команди

- Unreal Motion Graphics: система дизайну, анімації та розробки користувацького інтерфейсу для гри. Увесь інтерфейс складається з ієрархії віджетів, кожен з яких має своє візуальне представлення (що, можливо, також складається з віджетів), а також має свій програмний інтерфейс (атрибути віджету та події). Атрибутам можна надавати

значення як у редакторі, так і зв’язувати їх програмно. Для скриптингу інтерфейсу (реакцій на події та зв’язування даних) використовуються Blueprint-скрипти;

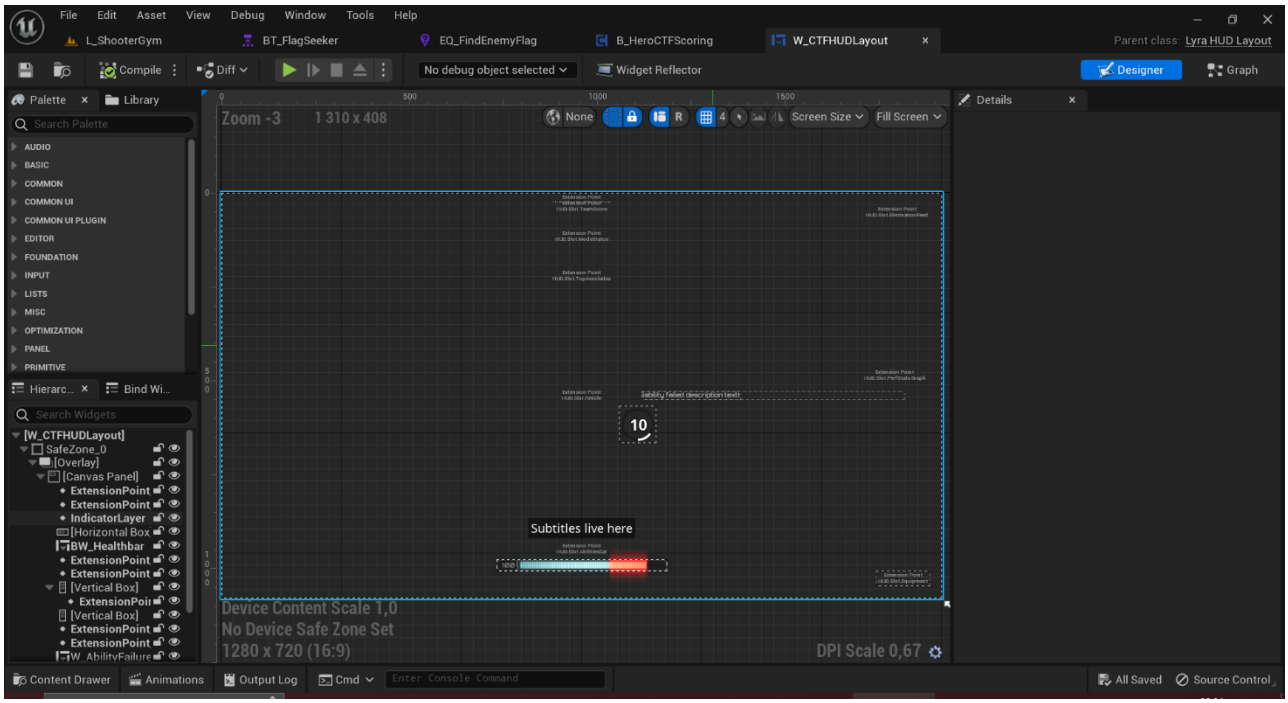


Рисунок 2.2. - Приклад редактора UMG UI (відкритий головний інтерфейс гри)

- Unreal Engine Gameplay Framework: набір класів C++, що містить в собі базові класи, необхідні для створення геймплею, такі як: GameMode (режим гри), GameState (стан гри), Character (персонаж), Pawn (пішка, актор, яким може керувати гравець або ШІ), Camera, HUD (користувацький інтерфейс) тощо. Взаємозв’язок між класами фреймворку показаний на рис. 2.3.

У якості мови програмування Unreal Engine 5 використовує високорівневу мову C++ з єдиною відмінністю: після проходу препроцесора та перед компіляцією файли з кодом оброблюються утилітою Unreal Header Tool, яка налаштовує внутрішню систему рефлексії для використання C++ сумісно з Blueprint. Також замість ручного управління пам’яттю надається перевага використанню власного прибиральника сміття Unreal’а, а для взаємодії з пам’яттю зовнішніх бібліотек – розумні вказівники з лічильником посилань.

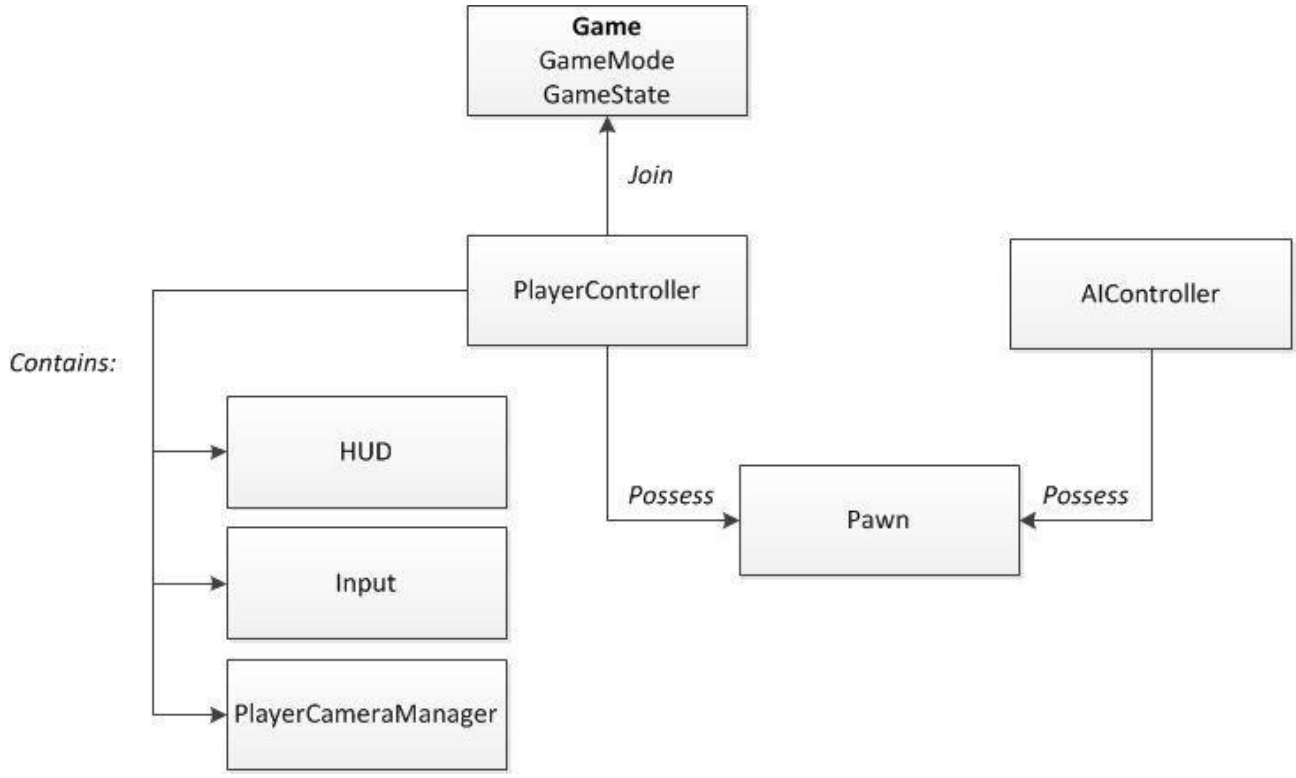


Рисунок 2.3. - Взаємозв’язок класів Gameplay Framework.

2.2. Архітектура мережевої взаємодії в Unreal Engine 5

Реалізація нетворкінгу (мережевої взаємодії) в Unreal Engine 5 засновано на реплікації, двобічному RPC та використанні онлайн-підсистеми (Online subsystem). Unreal Engine за замовчуванням підтримує лише модель авторитарного серверу: уся інформація з серверу на клієнти, у разі невідповідності інформації завжди обирається інформація серверу. Тим не менш, клієнт може мати нереплiковані об’єкти (наприклад, візуальні ефекти, які не треба обробляти на сервері), а також симулювати репліковані об’єкти (для зменшення затримки від вводу гравця до реакції гри).

Підтримується дві моделі серверу:

- виділений сервер (dedicated server) – сервер, що запущений в окремому від клієнта процесі, без підтримки вводу чи виводу. Для гри на такому сервері гравці мають використовувати спеціальні клієнти. Головними перевагами такого серверу є вища продуктивність та можливість

використання окремої інфраструктури. Недоліком є необхідність обслуговування сервера. Зазвичай використовується в змагальних іграх;

- вбудований сервер (listen server) – сервер, об’єднаний з клієнтом одного з гравців. Таким чином, клієнт з сервером об’єднані в один процес. Головною перевагою є відсутність витрат на обслуговування сервера і відсутність необхідності в окремій інфраструктурі взагалі, однак при цьому продуктивність сервера може бути значно нижчою (оскільки той самий комп’ютер має також обробляти ввод та вивід), а швидкість з’єднання дуже залежить від швидкості мережі приймаючого гравця. Зазвичай використовується в кооперативних іграх, або в змаганнях по локальній мережі.

Обмін даними між сервером та клієнтом реалізований за рахунок механізмів реплікації, володіння, та віддаленого виклику процедур. Оскільки Unreal Engine 5 завжди використовує модель авторитарного сервера, то реплікація інформації завжди відбувається в одному напрямі: з сервера на клієнт. Реплікуватися можуть лише об’єкти класів, що успадковані від класу AActor, або успадковані від класу UObject і належать об’єкту типу AActor (тобто рекурсивний виклик методу GetOwner() в кінці кінців поверне об’єкт типу AActor). При цьому не кожен об’єкт типу AActor повинен реплікуватися: за реплікацію відповідає властивість об’єкту bReplicates типу bool (за замовченням false). Таким чином, акторів, які не впливають на плин гри (наприклад, візуальні ефекти), можна не створювати на сервері, а створювати лише на клієнтах.

Окремим механізмом обміну даних між сервером та клієнтом є механізм віддаленого виклику даних (RPC, remote procedure call). Цей механізм дозволяє серверу викликати процедури на клієнті, або клієнтам викликати процедури на сервері. Цей механізм найчастіше використовується для надсилання подій з сервера на клієнт та з клієнта на сервер. Прикладом такого виклику може бути

реакція сервера на ввод користувача: клієнт, отримавши ввод від користувача, відправляє RPC на сервер, сервер обробляє запит та реагує на ввод (дозволяє або забороняє взаємодію користувача). Іншим прикладом може бути створення на клієнтах акторів, що відповідають за візуальні ефекти: оскільки виділений сервер не візуалізує гру, щоб не витрачати ресурси сервера, достатньо надсилати клієнтам запит на створення візуальних ефектів, замість створення та реплікації їх з сервера.

Unreal Engine 5 підтримує наступні типи віддаленого виклику процедур:

- з володіючого клієнту на сервер;
- з сервера на володіючий клієнт;
- з сервера на усі клієнти.

Ще одним важливим механізмом при обміні даних є володіння об’єктом. Кожен об’єкт належить або серверу, або одному з клієнтів. Володіння об’єктом визначається значенням, яке повертає рекурсивний виклик GetOwner(). Якщо повернуте значення має тип APlayerController – цим об’єктом володіє клієнт, до якого відноситься відповідний контролер. В іншому разі об’єктом володіє сервер. Цей механізм використовується для додаткового контролю реплікацією (нема сенсу реплікувати кількість речей в інвентарі усім гравцям, а не тільки гравцю, чий це інвентар), а також для контролю віддаленого виклику процедур (нечесні гравці, маючи доступ до пам’яті комп’ютера, можуть спробувати викликати процедуру від імені іншого гравця).

Не менш важливим компонентом мережевої взаємодії в грі, що використовує рушій Unreal Engine 5 є онлайн-підсистема. Онлайн-підсистема – це набір компонентів, що дозволяють абстрагувати код взаємодії з онлайн-платформами від конкретної онлайн-платформи. Таким чином, розробник має можливість писати один код, який буде використовувати рідну онлайн-систему на кожній платформі (Steam або EOS на ОС Windows, Xbox Live на Xbox Series

X, тощо), не дублюючі його для кожної платформи. Онлайн-підсистема має наступний набір компонентів:

- досягнення;
- зовнішній інтерфейс користувача;
- друзі;
- авторизація та автентифікація;
- доска лідерів;
- присутність;
- покупки в грі;
- ігрові сесії;

2.3. Архітектура штучного інтелекту в Unreal Engine 5

Для реалізації штучного інтелекту в Unreal Engine використовується клас AAIController, який може мати в собі всю логіку поведінки контрольованого актора. Тим не менш, для спрощення реалізації логіки Unreal Engine також надає можливість використання дерев поведінки, доски (blackboard), запити до середовища, та компоненту сприйняття ШІ.

Дерево поведінки – це спосіб представлення поведінки актора, що може бути представлена у вигляді направленого дерева. Основними елементами дерева є композитні вузли, задачі (листя), декоратори та сервіси.

- Композитні вузли – вузли, що мають дочірні вузли. В Unreal Engine за замовчуванням є наступні композитні вузли: послідовність (виконує по черзі усі дочірні вузли до першої невдалої), обирач (намагається виконати по черзі усі дочірні вузли до першої вдалої), паралельна (виконує дочірні вузли паралельно)

- Задачі – вузли, що не мають дочірніх вузлів і представляють атомарну дію. Прикладами є переміщення, використання предмету, очікування, тощо.
- Декоратори – елемент дерева поведінки, що додається до композитного вузла або задачі та модифікує виконання вузла. Наприклад: повторити дію n разів, завжди вважати дію вдалою, тощо.
- Сервіси – елемент дерева поведінки, що додається до композитного вузла або задачі та виконується із заданою частотою завжди, поки виконується гілка. Такими є: встановлення фокуса ШІ, запити до середовища, оновлення даних, тощо.

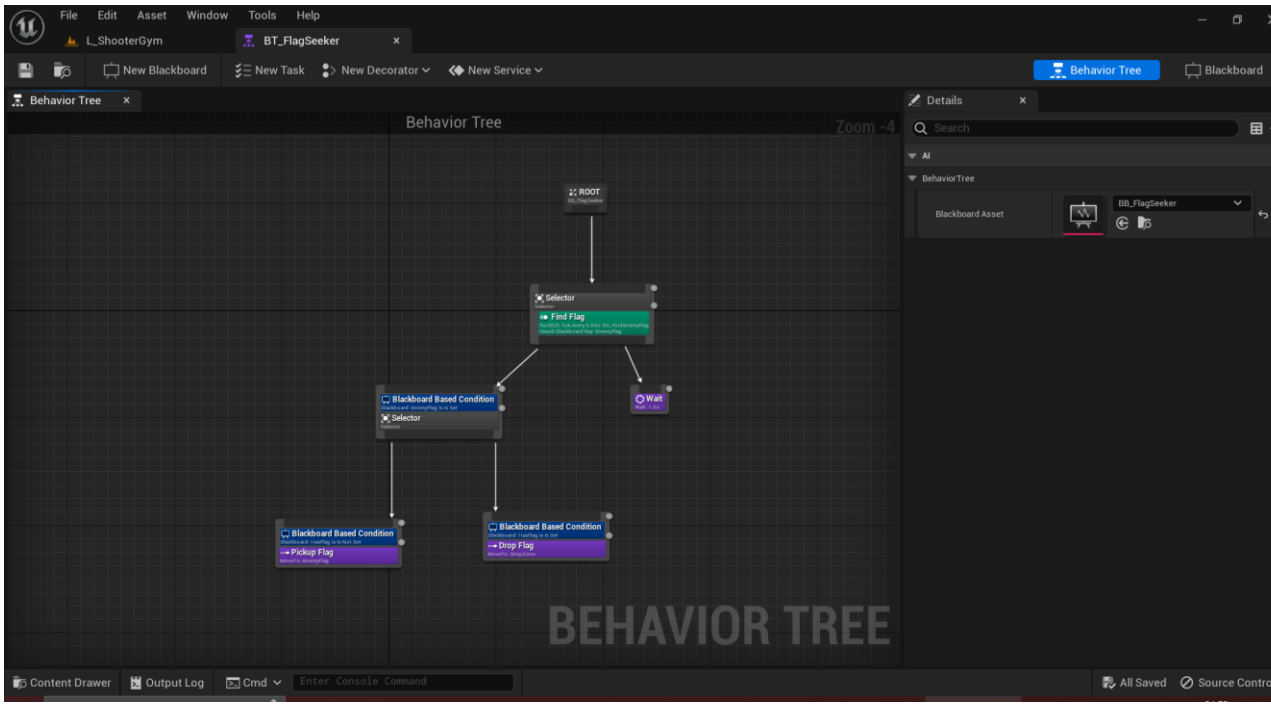


Рисунок 2.4. - Приклад дерева поведінки для підбора флагу

Доска (blackboard) – компонент, який можна назвати «пам’яттю ШІ». Дозволяє зберігати в собі дані, що можуть знадобитись під час виконання дерева поведінки.

Запити до середовища (environment query system) – система, що дозволяє контролерам ШІ отримувати інформацію про навколишнє середовище.

Прикладами можуть бути: «знайти найближче укриття», «обрати випадкову точку на колі радіуса r навколо гравця», тощо.

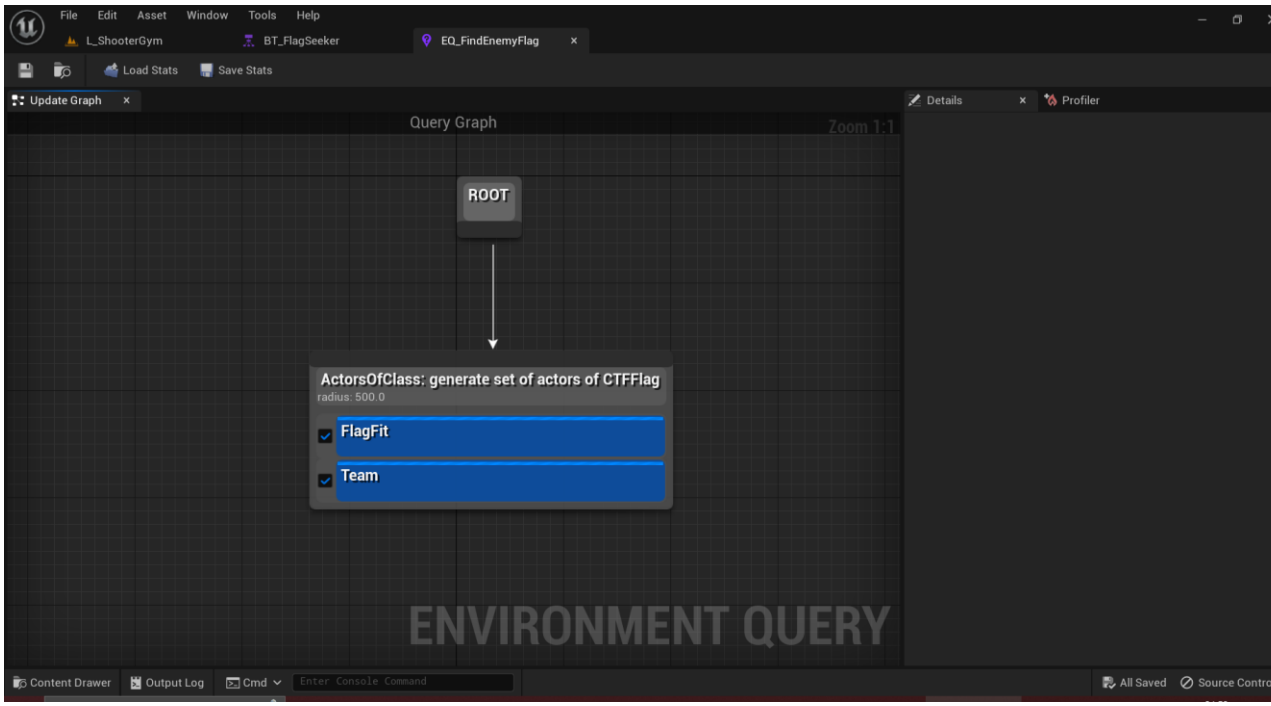


Рисунок 2.5. - Приклад запиту до середовища для пошуку флагу супротивника

Компонент сприйняття ШІ – компонент, що дозволяє контролерам ШІ реагувати на навколишнє середовище імітуючи людське чуття. Прикладами таких «чуттів» є: «зір», «слух», реакція на отримані пошкодження, передбачення дій гравця, тощо.

РОЗДІЛ 3
ІМПЛЕМЕНТАЦІЯ ГРИ DALDOS З ВИКОРИСТАННЯМ РУШІЯ UNREAL ENGINE 5

3.1. Постановка задачі. Технічне завдання

Проаналізувавши основні теоретичні концепції побудови відеоігр та досконально дослідивши популярні зараз жанри відеоігор, виокремивши основні пункти, яких слід дотримуватися, можемо сформулювати основне завдання цієї кваліфікаційної роботи бакалавра:

Спроектувати прототип командного геройського шутеру від третьої особи, основною задачею команд в якому є забрати флаг з бази супротивника і перенести на свою базу (режим «захват флагу»), поєднаний з механіками багатокористувацьких онлайн бойових арен: гравці не тільки мають 1 здібність, але й можуть будувати башти та викликати «міньйонів», що керуються ігровим штучним інтелектом.

Для виконання поставленої мети необхідно послідовно вирішити наступні задачі:

- Розробити основний компонент, що відповідає за логіку гри
- Спроектувати та розробити компонент, що відповідає за взаємодію гравця з грою
- Спроектувати та реалізувати користувацький інтерфейс гри
- Спроектувати та розробити компонент, що відповідає за багатокористувацьку гру онлайн.

3.2. Програмна структура

3.2.1. Загальна архітектура

Для написання прототипу використовується рушій Unreal Engine 5. Як і будь-який фреймворк, він передбачає заздалегідь визначену архітектуру

проекта. Використання цієї архітектури дозволяє домогтися найбільшої ефективності роботи та швидкодії рушія, а також найефективніше використовувати API рушія.

Основними компонентами будь-якого проекту на Unreal Engine є:

Таблиця 3.1 - Основні компоненти гри на Unreal Engine

Назва	Опис
Режим гри	Відповідає за «правила гри»: початок та кінець матчу, логіку взаємодії гравців між собою, тощо
Стан гри	Включає в себе інформацію, пов’язану з ходом гри загальну для всіх гравців
Контролер гравця	«Представник» гравця в грі, відповідає за отримання вводу від гравця та його обробку (або делегування обробки необхідним акторам)
HUD	Відповідає за інтерфейс (Heads-Up Display) гравця
Стан гравця	Включає в себе інформацію, пов’язану з ходом гри конкретного гравця, але не пов’язану з конкретним актором, яким володіє гравець (наприклад, рахунок гравця)

При цьому проект може містити декілька мап, а для кожної мапи окремо встановлюються реалізації кожного з цих компонентів. Мапу можна визначити як стан гри, який має свій набір акторів і режим гри.

3.2.2. Структура плагінів при використанні Lyra

Починаючи з версії рушія 5.0, розробники рушія Epic Games рекомендують використовувати у якості основи гри проєкт Lyra. Головною особливістю Lyra є висока якість заздалегідь реалізованих механік, що часто використовуються (інвентар, розподіл на команди, тощо), а також її висока модульність, що дозволяє розширити або доповнити будь-яку з написаних систем та з легкістю дописувати свої за потреби.

При цьому дещо змінюється архітектура гри та плин ініціалізації. Для досягнення високої модульності Lyra пропонує використовувати спільний клас-спадкоємець AGameMode для усіх режимів гри, на відміну від окремого класу для кожного режиму. При цьому клас режиму гри відповідає лише за функціональність, спільний для усіх режимів, такий як підключення гравців онлайн, створення контролерів гравців, додавання гравців на карту, а також завантаження «досвідів» (experiences). В новій архітектурі досвід – це набір додаткових компонентів, які треба додати до необхідних акторів, і які і реалізують необхідний для режиму гри функціонал. Завдяки такій архітектурі підвищується модульність, спрощується перевикористання коду, а також з’являється можливість динамічно завантажувати та вивантажувати «досвіди», змінюючи правила гри на ходу.

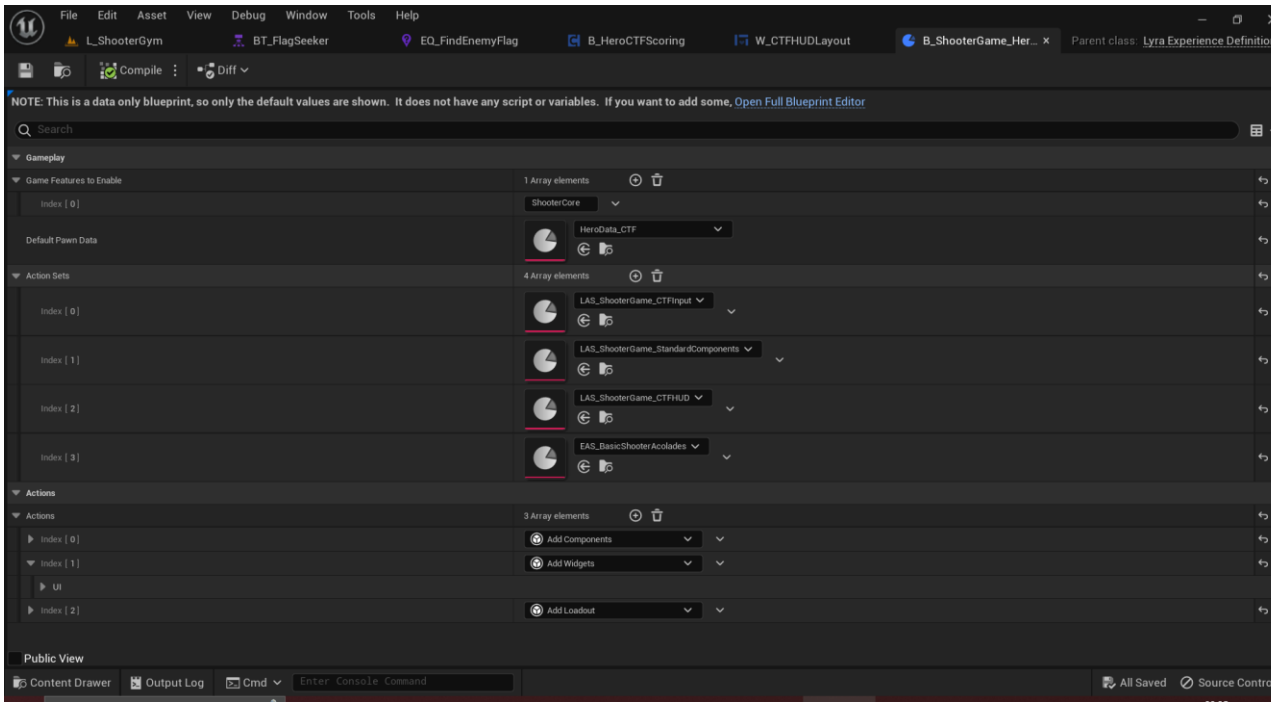


Рисунок 3.1. - Приклад визначення «досвіду» в Unreal Engine 5 Lyra

При використанні такої архітектури з’являється ще одна необхідна зміна в структурі проєкту: можливість виокремити кожен режим гри в плагін, що значно спрощує створення та розповсюдження доповнень для гри.

3.2.3. Структура проекту

Окрім безпосередньо коду, кожна гра також містить «асети» - файли зображень, шрифтів, музики, тощо. На відміну від кода, Unreal Engine не накладає обмежень на структуру директорій асетів, окрім того, що код повинен зберігатися в директорії Source, а асети – в директорії Content.

Оскільки Project Gryphon використовує Lyra, асети, що пов’язані безпосередньо з ігроладом були винесені в плагін ShooterCore, в директорію HeroCTF.

Таблиця 3.3 - Структура директорій в проекті гри Project Gryphon

Директорія	Опис
Abilities	Blueprint-класи, пов’язані зі здібностями гравців
Buildings	Blueprint-класи, пов’язані з будівлями, які можуть будувати гравці
Filters	Blueprint-класи, пов’язані з фільтрами акторів для вибіркового застосування ефектів будівель
Flag	Blueprint-класи, пов’язані з флагом
Input	Асети, пов’язані з вводом користувача
Maps	Карти
Materials	Шейдери
Minions	Blueprint-класи, пов’язані з врагами, що керовані штучним інтелектом

Phases	Blueprint-класи, пов’язані з фазами гри
PlayerComponents	Blueprint-класи, пов’язані з додатковими компонентами актора гравця
UI	Blueprint-класи, пов’язані з інтерфейсом користувача

3.3. Інструкція користувача

Розроблена програма відповідає усім основним вимогам до симуляції такого рода гри: наявний функціонал для гри вшостьох онлайн, головне меню, меню паузи, основний геймплей.

Інтерфейс грального процесу був спроектований за принципами мінімалізму таким чином, щоб у гравця на екрані завжди було мінімум елементів окрім грального поля. При цьому, стиль інтерфейсу був зроблений в стилі наукової фантастики, а сама гра візуально повинна нагадувати боротьбу роботів, а не живих людей.

Після запуску гри користувач потрапляє в головне меню, де він може почати гру або вийти з неї.

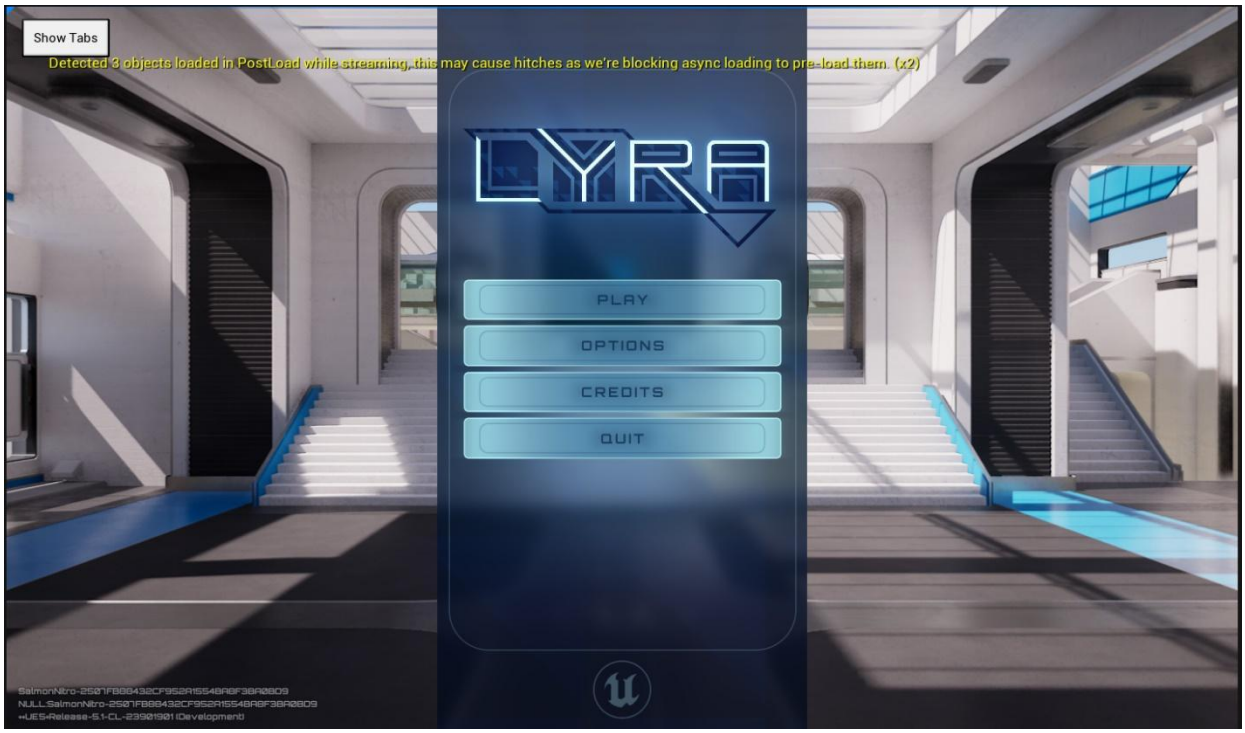


Рисунок 3.2. - Головне меню гри

Після натискання на «Play» гравець потрапляє до екрану головного геймплею.

Рисунок 3.3. - Головний геймплей гри

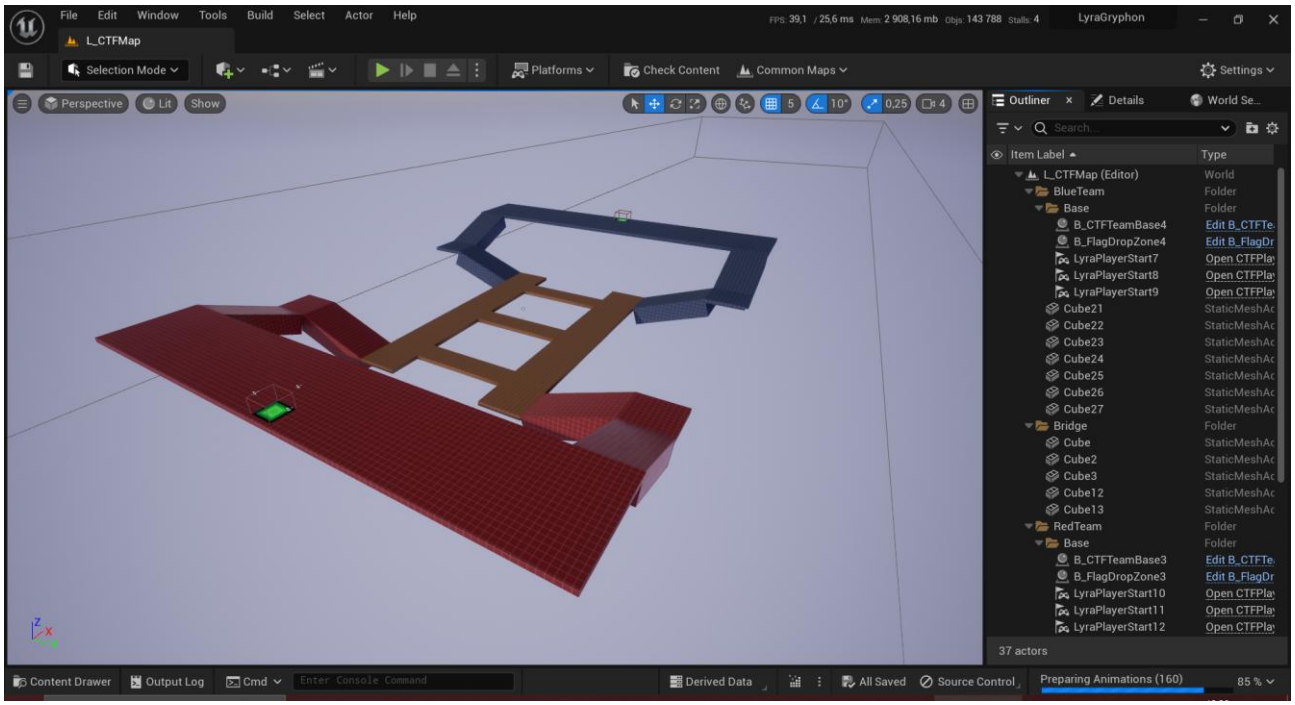


Рисунок 3.4. - Зовнішній вигляд ігрової мапи

Внизу гравець може побачити шкалу здоров’я та свої здібності. При використанні здібності її кнопка стає неактивною та починає заповнюватися шкала перезарядки здібності.

Зверху відображається поточний час матчу, поточний рахунок та кількість балів, яку необхідно набрати для перемоги в матчі.

По центру розташований приціл, а праворуч внизу можна побачити кількість патронів, що залишились в обоймі до перезарядки.

Під час гри гравець може використовувати свої здібності та будувати на карті башти, що надають позитивний ефект союзникам, або негативний ефект ворогам.

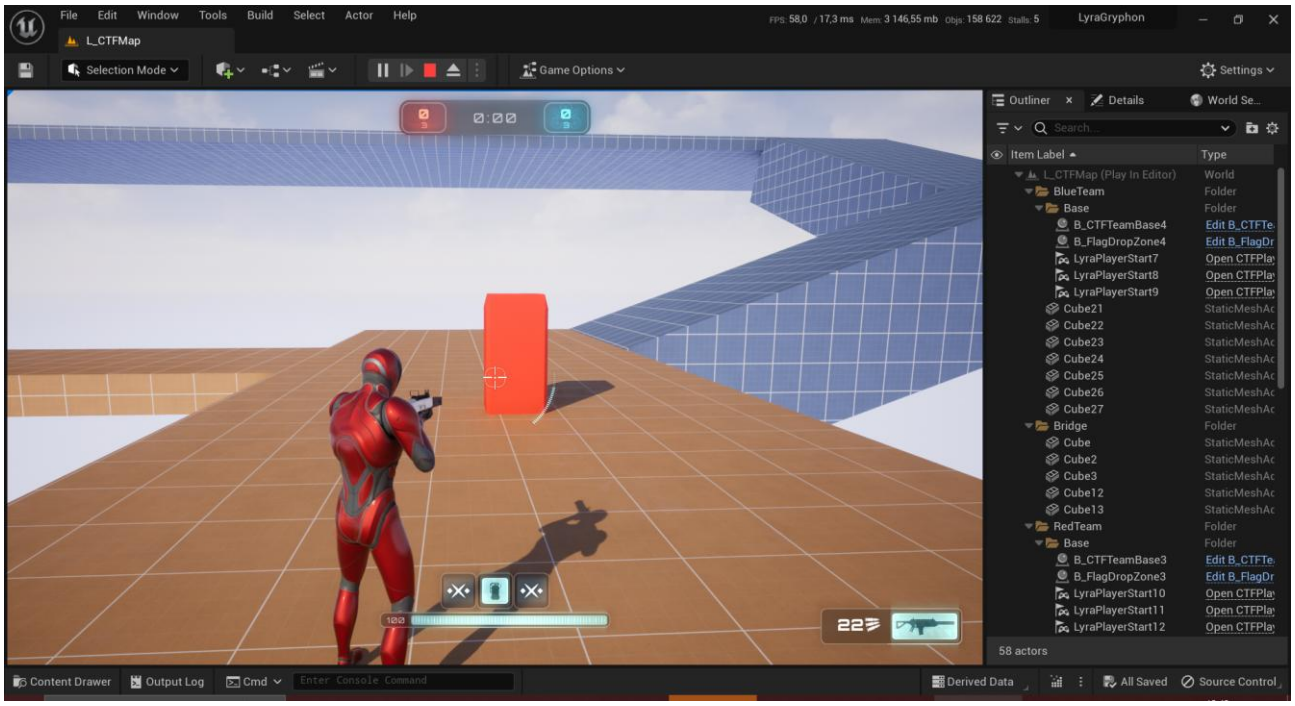


Рисунок 3.5. - Побудована башта, що наносить пошкодження ворогам

Головною задачею гравця є захопити флаг супротивника та доставити його на свою базу.

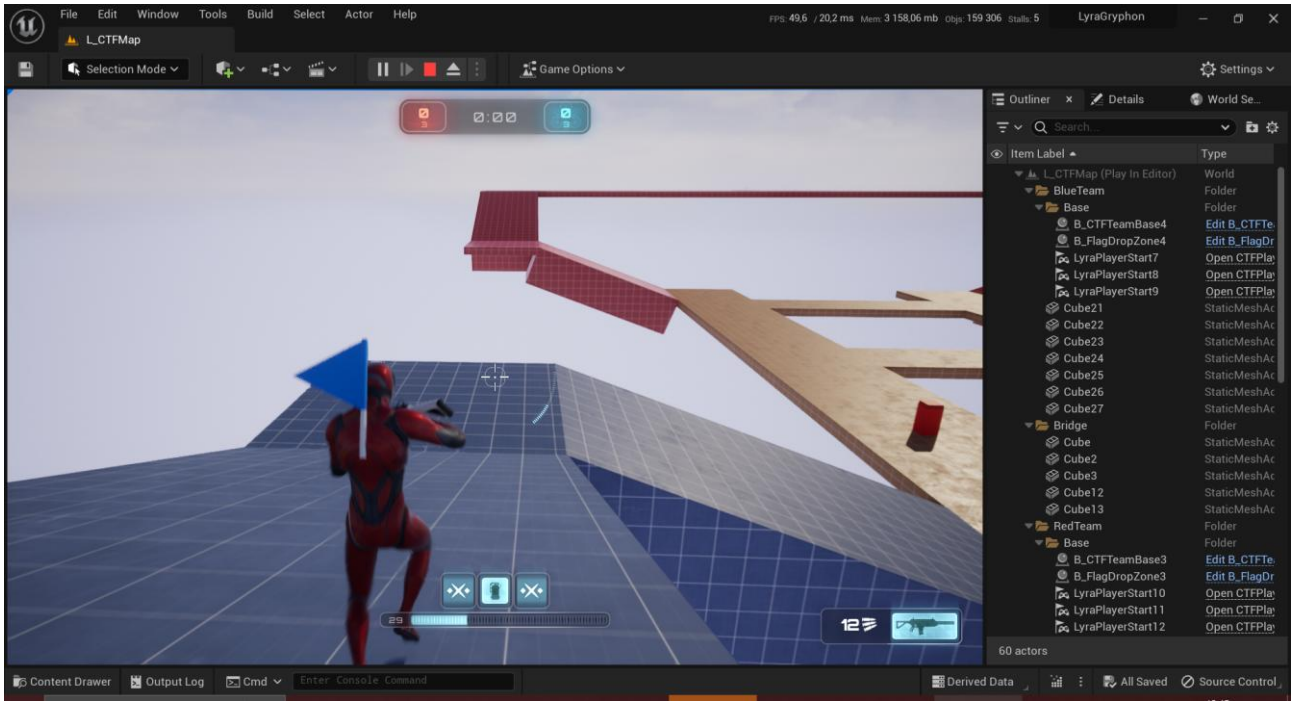


Рисунок 3.6. - Гравець, що захопив флаг, несе його на свою базу

Рисунок 3.7. - Екран перемоги

У грі також наявне меню паузи, через яке гравець може продовжити гру після паузи, вийти в головне меню або в меню налаштувань.

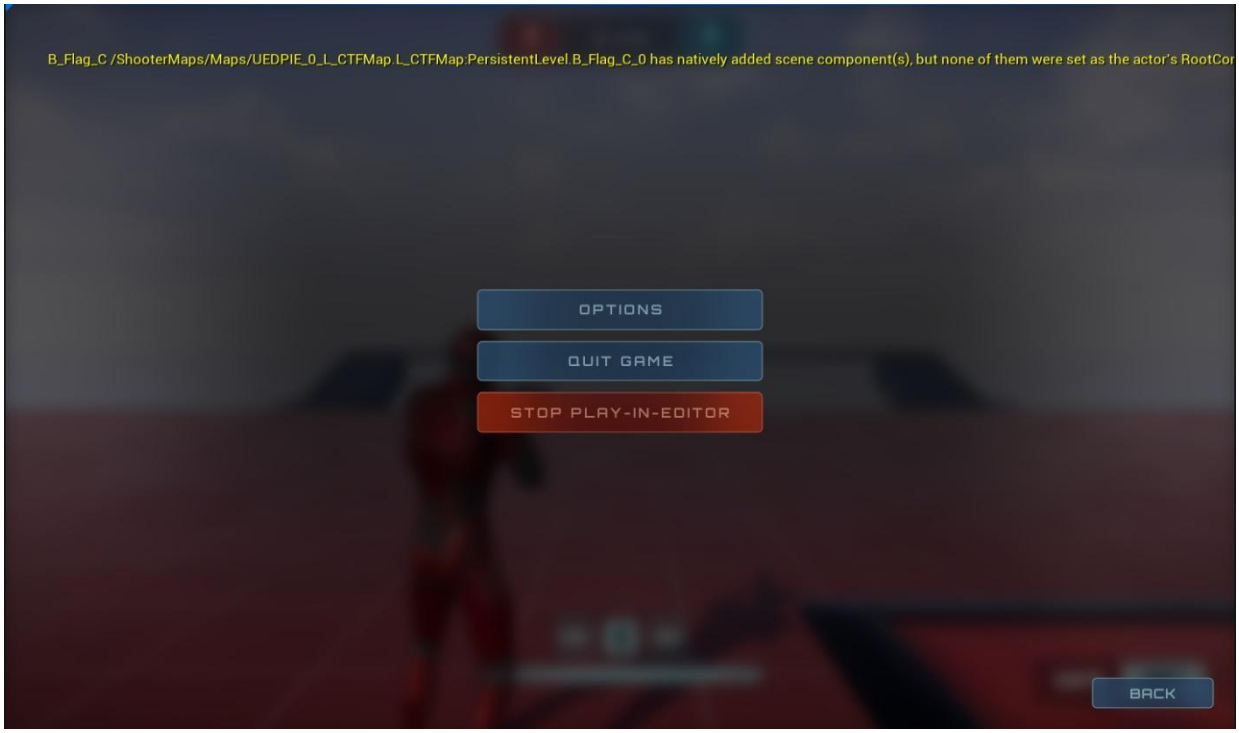


Рисунок 3.8. - Меню паузи

В грі також наявне меню налаштувань, в якому гравці можуть обрати потрібні їм налаштування графіки, мови та вводу.

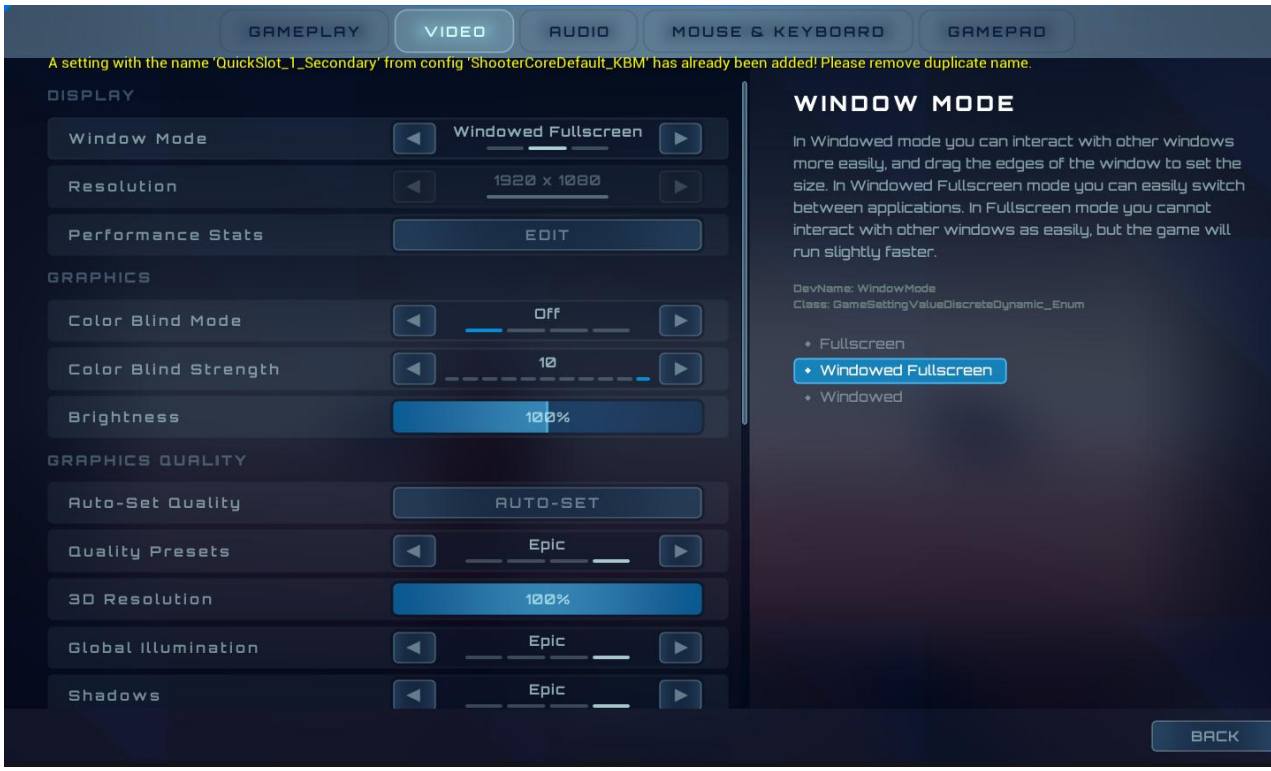


Рисунок 3.9. - Налаштування графіки

При подальшому розвитку програмного забезпечення планується також додавання можливості багатокористувацької гри через мережу.

ВИСНОВОК

Отже, у результаті виконання курсової роботи були виконані та завершені наступні поставлені задачі:

По-перше, були вивчені теоретичні засади сучасної розробки відеоігор, основні концепції їх побудови. Був досліджений ринок відеоігр у платформах Steam, Good Old Games та Epic Games Store на наявність реалізацій аналогічних ігор.

По-друге, на основі обраних для розробки програмно-технологічних рішень було розроблено структуру ігри, включаючи архітектуру взаємодії та взаємозв’язку об’єктів та акторів.

Було проведено дослідження популярних нині жанрів та конкретних ігор, а також спроектовано власну гру на основі результату цих досліджень.

На основі досліджених жанрів та ігор, розробленої архітектури та структури гри було зроблено рішення використовувати для реалізації ігровий рушій Unreal Engine 5 та мови програмування C++ і Blueprint (візуальна). Використовуючи цей стек технологій було розроблено працюючу багатокористувацьку гру Project Gryphon.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Unreal Engine 5 Official Website: веб-сайт. URL: http://unrealengine.com/home (дата звернення: 20.05.2023).
2. Unity3D Home Page: веб-сайт. URL: https://unity.com/ (дата звернення: 20.05.2023).
3. Godot Engine Home Page: веб-сайт. URL: https://godotengine.org/ (дата звернення: 20.05.2023).
4. Entity Systems are the future of MMOG development: веб-сайт. URL: http://t-machine.org/index.php/2007/09/03/entity-systems-are-the-future-of-mmog-development-part-1/ (дата звернення: 20.05.2023).
5. Most played Multiplayer games in Steam: веб-сайт. URL: https://steamdb.info/charts/?category=1 (дата звернення: 20.05.2023).
6. Overwatch Official Website: веб-сайт. URL: https://overwatch.blizzard.com/ (дата звернення: 20.05.2023).
7. Valorant Official Website: веб-сайт. URL: https://playvalorant.com (дата звернення: 20.05.2023).
8. Unreal Engine 5 Network Compendium: веб-сайт. URL: https://cedric-neukirchen.net/docs/category/multiplayer-network-compendium/ (дата звернення: 20.05.2023).
9. Unreal Engine 5 Lyra Sample Game: веб-сайт. URL: https://docs.unrealengine.com/5.1/en-US/lyra-sample-game-in-unreal-engine/ (дата звернення: 20.05.2023).
10. Persistent Data Compendium: веб-сайт. URL: https://wizardcell.com/unreal/persistent-data/ (дата звернення: 20.05.2023).
11. Gameplay Ability System Documentation: веб-сайт. URL: https://github.com/tranek/GASDocumentation (дата звернення: 20.05.2023).

ДОДАТКИ

ДОДАТОК А

```cpp
void ACTFFlagDropZone::OnBeginOverlap(AActor* OverlappedActor, AActor* OtherActor)
{
    if (!HasAuthority())
    {
        return;
    }

    if (!OverlappedActor || !OtherActor)
    {
        return;
    }

    if (UCTFFlagBearerComponent* FlagBearerComponent = OtherActor->FindComponentByClass<UCTFFlagBearerComponent>())
    {
        if (ACTFFlag* Flag = FlagBearerComponent->GetCarriedFlag())
        {
            const ULyraTeamSubsystem* TeamSubsystem = GetWorld()->GetSubsystem<ULyraTeamSubsystem>();
            if (!TeamSubsystem)
            {
                return;
            }

            const bool bFlagBearerIsAlly = TeamSubsystem->CompareTeams( A: OverlappedActor, B: OtherActor) ==
                ELyraTeamComparison::OnSameTeam;
            const bool bFlagIsEnemy = TeamSubsystem->CompareTeams( A: OverlappedActor, B: Flag) ==
                ELyraTeamComparison::DifferentTeams;
            if (bFlagBearerIsAlly && bFlagIsEnemy)
            {
                // OnFlagInDropZone.Broadcast(this, FlagBearerComponent, Flag);
                FCTFFlagScoredMessage Message;
                Message.Verb = TAG_Lyra_FlagOnBase_Message;
                Message.Instigator = OtherActor;

                bool bIsInTeam;
                TeamSubsystem->FindTeamFromActor(OtherActor, &bIsInTeam, &Message.ScorerTeamID);
                TeamSubsystem->FindTeamFromActor(Flag, &bIsInTeam, &Message.FlagTeamID);

                UGameplayMessageSubsystem& MessageSystem = UGameplayMessageSubsystem::Get(GetWorld());
                MessageSystem.BroadcastMessage(Message.Verb, Message);

                FlagBearerComponent->DetachFlag();
                Flag->SetIsWaitingRespawn(true);
                AttachEnemyFlag(Flag);
            }
        }
```

Рисунок А1. - Реалізація повернення флагу на базу

```cpp
void UCTFFlagBearerComponent::OnOverlapWithFlag(ACTFFlag* Flag)
{
    if (CurrentFlag || Flag->IsCarried() || Flag->IsWaitingRespawn())
    {
        return;
    }

    const ULyraTeamSubsystem* LyraTeamSubsystem = GetWorld()->GetSubsystem<ULyraTeamSubsystem>();
    if (!LyraTeamSubsystem)
    {
        return;
    }

    if (LyraTeamSubsystem->CompareTeams(A: GetOwner(), B: Flag) == ELyraTeamComparison::DifferentTeams)
    {
        CurrentFlag = Flag;
        Flag->SetCarrier(this);
        const FAttachmentTransformRules FlagAttachmentRules(InLocationRule: EAttachmentRule::SnapToTarget,
                                                            InRotationRule: EAttachmentRule::SnapToTarget,
                                                            InScaleRule: EAttachmentRule::KeepWorld, bInWeldSimulatedBodies: true);
        if (UMeshComponent* BearerMesh = GetOwner()->FindComponentByClass<UMeshComponent>())
        {
            Flag->AttachToComponent(BearerMesh, FlagAttachmentRules,
                                    FlagAttachSocketName);
        }
        else
        {
            Flag->AttachToActor(GetOwner(), FlagAttachmentRules, FlagAttachSocketName);
        }
        OnFlagChanged.Broadcast(bHasFlag: true);
    }
}
```

Рисунок А2. - Реалізація захоплення флагу

ДОДАТОК Б

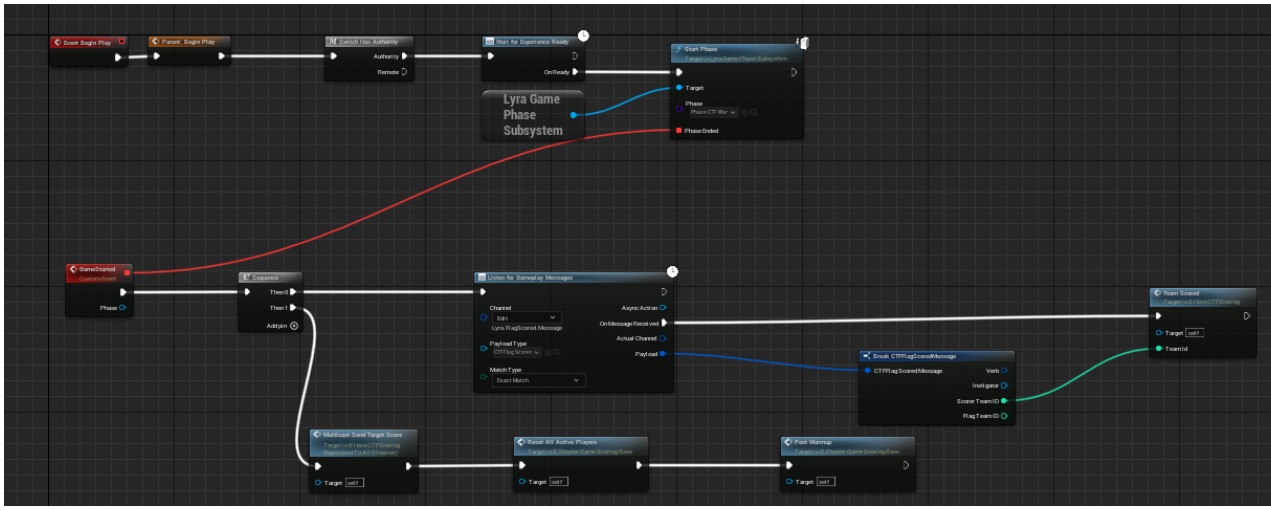


Рисунок Б1. - Реалізація скорингу (підрахунку балів) на мові Blueprint

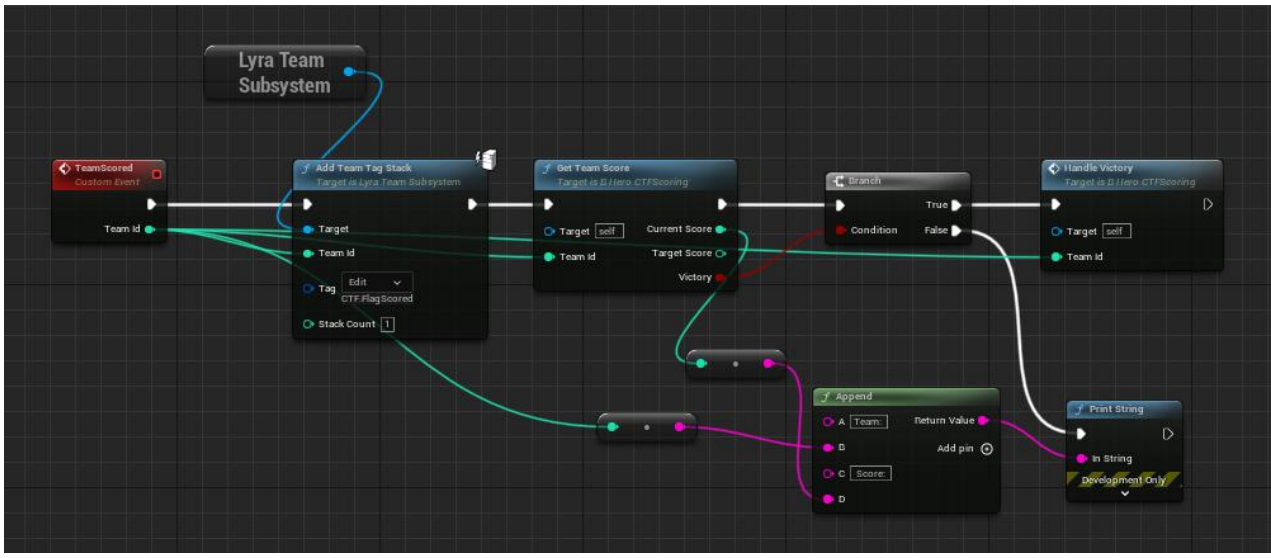


Рисунок Б2. - Реалізація скорингу (підрахунку балів) на мові Blueprint