Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет інформаційних радіотехнологій та технічного захисту інформації
(повна назва)

Кафедра медіаінженерії та інформаційних радіоелектронних систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти *другий (магістерський)*
(позначення документа)

Створення ігрового додатку на Android в Unreal Engine 4 з використанням 3D моделювання
(тема)

Виконав:
студент 2 курсу, групи *СТМм-22-1*
Нікіта НІКІТІН
(прізвище, ініціали)

Спеціальність *171 Електроніка*
(код і повна назва спеціальності)

Тип програми *освітньо-професійна*
(освітньо-професійна або освітньо-наукова)

Освітня програма *Системи, технології і комп'ютерні засоби мультимедіа*
(повна назва освітньої програми)

Керівник *ст. викл. Вікторія КОЛІСНИК*
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Володимир КАРТАШОВ
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційних радіотехнологій та технічного захисту інформації
Кафедра Медіаінженерії та інформаційних радіоелектронних систем

Рівень вищої освіти другий (магістерський)
Спеціальність 171 Електроніка.
(код і повна назва)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма "Системи, технології і комп'ютерні засоби мультимедіа"

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Студентові *Нікітіну Нікіті Романовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Створення ігрового додатку на Android в Unreal Engine 4 з використанням 3D моделювання*
затверджена наказом по університету від *" 20 " 11 2023 р. № 1371 СТ*
2. Термін подання студентом роботи *08.01.2024 р.*
3. Вихідні дані до проекту (роботи)
1. Розробка моделі 3D об'єкту для ігрового додатку на Android
2. Розробка необхідних текстур для різних 3D об'єктів для ігрового додатку
3. Розробка прототипу ігрового додатку на Android за допомогою двигуна Unreal Engine

4. Перелік питань, що потрібно опрацювати в роботі
ВСТУП
1. Огляд існуючих аналогів
2. Аналіз технічного забезпечення та обґрунтування вибору інструментів
3. Моделювання та текстурінг 3D об'єктів для ігрового додатку на Android
4. Реалізація прототипу ігрового додатку у Unreal Engine
5. Тестування та оптимізація
ВИСНОВКИ
ПЕРЕЛІК ПОСИЛАНЬ
ДОДАТКИ

5. Перелік графічного матеріалу із зазначенням обов'язкових креслеників, схем, плакатів, комп'ютерних ілюстрацій:
1. Гра та її механіки; 2. Огляд платформ та програмного забезпечення для створення ігор; 3. Переваги обраного програмного забезпечення та платформи; 4. Етапи 3D моделювання; 5. Створення проекту та локації прототипу гри; 6. Створення логіки прототипу гри; 7. Основні оптичні прилади та системи навігації; 8. Методи оптимізації застосовані у прототипу; 11. Висновки прототипу гри.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Огляд існуючих аналогів	20.11.23–28.11.23	
2.	Аналіз технічного забезпечення та обґрунтування вибору інструментів	21.11.23–28.11.23	
3.	Моделювання та текстурінг 3D об’єктів для ігрового додатку на Android	23.11.23–02.12.23	
4.	Реалізація прототипу ігрового додатку у Unreal Engine	01.12.23–15.12.23	
5.	Тестування та оптимізація	15.12.23–20.12.23	
6.	Графічна частина роботи	15.12.23–20.12.23	
7.	Перевірка керівником	20.12.23-24.12.23	
8.	Перевірка на академічний плагіат	24.12.23-26.12.23	
9.	Перевірка завідувачем кафедри, рецензування	27.12.23-31.12.23	

Дата видачі завдання _____ *20.11.2023 р.*

Студент _____ *Нікіта НІКІТІН*
(підпис)

Керівник роботи _____ *ст. викл. Вікторія КОЛІСНИК*
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи має: 101 с., 96 рис., 2 додатків, 18 джерел.

UNREAL ENGINE, BLENDER, 3D МОДЕЛЮВАННЯ, BLUEPRINT, ANDROID, ВІЗУАЛЬНИЙ СКРІПТИНГ, ІГРОВИЙ ДВИГУН, ШТУЧНИЙ ІНТЕЛЕКТ

Об'єкт дослідження – ігрові додатки.

Предмет дослідження – методи створення та оптимізації ігрових додатків.

Мета кваліфікаційної роботи – розробка прототипу ігрового додатку використовуючи ігровий двигун Unreal Engine.

Методи дослідження – теоретичний аналіз, алгоритмування, графічне моделювання, статистична обробка даних.

У даній роботі проведено огляд існуючих ігрових жанрів, класифікацію вимог ігор під різні платформи, детальний огляд ігрових рушіїв, аналіз програм під 3D моделювання, реалізацію ігрового контенту гри використовуючи програми для 3D моделювання, етапи створення додатку, розробка логіки додатку використовуючи візуальний скріптинг Blueprint, створення візуального інтерфейсу гри.

ABSTRACT

Explanatory note of the qualification work has: 101 p., 96 draw, 2 applications, 18 sources.

UNREAL ENGINE, BLENDER, 3D MODELING, BLUEPRINT, ANDROID, VISUAL SCRAPING, GAME ENGINE, ARTIFICIAL INTELLIGENCE

The object of research is game applications.

The subject of research is methods of creating and optimizing game applications.

The purpose of the qualification work is to develop a game application prototype using the Unreal Engine game engine.

Research methods – theoretical analysis, algorithmization, graphic modeling, statistical data processing.

This paper provides an overview of existing game genres, classification of game requirements for different platforms, a detailed overview of game engines, analysis of 3D modeling programs, implementation of game content using 3D modeling programs, stages of application creation, development of application logic using Blueprint visual scripting, creation visual interface of the game.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

3D – різновид комп’ютерної графіки;

API – інтерфейс програмування програми;

AR – доповнена реальність;

BLUEPRINTS – система візуального скриптингу;

FPS – частота зміни кадрів;

MESH – полігональна сітка у 3D графіці;

UE – ігровий двигун Unreal Engine;

UV – перетворення або розгортка у 3D графіці;

VR – віртуальна реальність;

ПК – персональний комп’ютер;

ШІ – штучний інтелект.

ЗМІСТ

ВСТУП

В сучасному світі мобільні ігри стали не тільки невід'ємною частиною розваг та розважальної індустрії, але й важливим компонентом розвитку технологічного сегменту ринку. Швидкість зростання потужності мобільних пристроїв та їх доступність роблять їх ідеальною платформою для ігор різних жанрів і рівнів складності. За останні десятиліття ігрова індустрія взяла світ геймінгу, перетворивши його у справжню сферу культури та розваги. Мобільні платформи, зокрема Android, відіграли важливу роль у цьому розвитку, надаючи гравцям зручний доступ до ігор у будь-який час та в будь-якому місці. Однак створення ігрових додатків для Android вимагає від розробників високого рівня експертизи в області програмування, дизайну та 3D моделювання.

Кваліфікаційна робота присвячена дослідженню та практичній реалізації створення ігрового додатку на платформі Android за допомогою відомого інструмента розробки ігор - Unreal Engine 4, з акцентом на використанні 3D моделювання. Unreal Engine 4 є одним із найпотужніших та популярних геймдев-движків, що надає безмежні можливості для розробки вражаючих ігор, які працюють на різних платформах. Використання 3D моделювання дозволяє створювати реалістичні світи та персонажі, що робить гру більш привабливою для гравців.

Метою даної роботи є розгляд процесу створення ігрового додатку для платформи Android в середовищі Unreal Engine 4 з використанням 3D моделювання. Дослідження включає в себе аналіз теоретичних основ та практичні кроки розробки такого додатку, від вибору теми та концепції гри, до її оптимізації для мобільної платформи.

Практична цінність роботи полягає в можливості створення власних ігор на мобільних пристроях за допомогою потужного інструменту розробки та розширення розуміння процесу геймдевелопменту. В аналізі інструментів,

методів та підходів, які дозволяють створювати якісні ігри для мобільних пристроїв на платформі Android, використовуючи Unreal Engine 4 та 3D моделювання. Розглянуті теоретичні аспекти створення ігор для Android, проектування та розробки ігрового додатку з використанням 3D моделювання, програмування та оптимізація гри, тестування та налагодження. Результати цього дослідження допоможуть розкрити потенціал Unreal Engine 4 для створення мобільних ігор та вивчити важливі етапи їх розробки.

Ця робота відображає сучасні тенденції у світі розробки мобільних ігор та відповідає на актуальні питання, пов'язані з розробкою ігор для Android-платформи за допомогою Unreal Engine 4 і 3D моделювання.

1 ОГЛЯД ІСНУЮЧИХ АНАЛОГІВ

1.1 Розвиток ігрової індустрії

Ігрова індустрія є однією з найбільш динамічних та швидкозмінних галузей в світі розважальних технологій. Історія розвитку ігор відзначається значущими змінами, починаючи від перших примітивних аркадних ігор до сучасних високотехнологічних ігрових проектів [1-3].

Початки ігрової індустрії можна відстежити до середини 20-го століття, коли були створені перші примітивні аркадні ігри, такі як "Pong" та "Space Invaders". З тих часів ігри зазнали значних технологічних та культурних змін, розвиваючись від аркадних автоматів до консолей та особистих комп'ютерів.
З 1970-1980 роки вважаються "золотою ерою" консольних ігор, коли були створені такі легендарні ігри, як "Super Mario Bros.", "The Legend of Zelda", та "Pac-Man". Цей період відзначався великими досягненнями в галузі геймдизайну та графіки, і відзначився зростанням популярності ігор.

Починаючи з 1980-1990 роки принесли з собою революцію в ігровій індустрії завдяки особистим комп'ютерам. Поява графічних інтерфейсів та операційних систем, таких як MS-DOS та Windows, сприяла розвитку ігор для ПК. З'явилися такі ігри, як "Doom", "SimCity", та "Warcraft", які визначили нові стандарти геймдизайну.

Поява смартфонів та планшетів на початку 21-го століття мобільні ігри стали дуже популярними. Розробники створюють ігри для різних мобільних платформ, використовуючи технології сенсорного керування та графічні можливості сучасних пристроїв. А розширення можливостей смартфонів дало змогу створювати ігри які не поступаються комп'ютерним, що призвело до росту ігрових платформ та створення нової ери ігор.

Останнім розвитком стала віртуальна реальність (VR) та розширена реальність (AR), які внесли суттєві зміни в ігрову індустрію. VR ігри

дозволили гравцям зануритися в ігровий світ, а AR ігри поєднують віртуальні елементи з реальним оточенням [4-15].

1.2 Огляд жанрів ігор

Ігрова індустрія завжди була відома своєю здатністю до постійних змін і інновацій у створенні нових типів ігор та жанрів. Тому виокремлюють основні типи та жанри ігор, а решта вважається поміссю цих можливостей. Кожен тип ігор відрізняється своєю цільовою спрямованістю, тобто має певну "мету", яку переслідують суб'єкти гри. Ігрова мета - це важливий фактор, який значною мірою визначає і зміст гри, і її структуру, виражену в ігрових правилах, її художню спрямованість. Розвиток цих типів і жанрів є ключовим аспектом історії ігрової індустрії, оскільки він відображає реакцію розробників на змінюються під технологічні можливості та смаки гравців. Основними жанрами ігор є: аркади, рольові (RPG), головоломки, тактичні шутери, мультиплеєрні ігри, симулятори, хоррор, стратегії, action, візуальні романи [4].

Аркади. Ці ігри були одними з перших типів ігор і відзначалися швидким геймплеєм та великим акцентом на реакцію гравця. З розвитком технологій, таких як процесори та графічні можливості, аркади стали більш складними та різноманітними. Серед класичних представників цього жанру варто відзначити "Pac-Man" і "Space Invaders", приклад на рис. 1.1.

Рисунок 1.1 – Приклад аркадних ігор "Pac-Man" і "Space Invaders" [4]

Рольові ігри (RPG). Рольові ігри спеціалізуються на історії та розвитку персонажів. Характерними ознаками рольових ігор було: у головного героя, або героїв, і ворогів є певна кількість параметрів (умінь, характеристик, навичок), які визначають їхню силу та здібності, є опрацьований і великий світ, сюжетна лінія, розгалужені діалоги з різними варіантами відповідей, безліч різних персонажів зі своїми цілями і характерами, велика кількість різних предметів. "Final Fantasy" та "The Elder Scrolls" - це приклади відомих RPG, приклад на рис. 1.2.

Рисунок 1.2 – Приклад рольових ігор "Final Fantasy" та "The Elder Scrolls" [4]

Головоломки. Цей вид ігри вимагають логічного мислення та розв'язання різних завдань. Гравці повинні розгадувати головоломки, розміщені в ігровому світі. Оскільки здебільшого це ігри, що використовують площину, вони або двомірні та використовують "вид зверху" на ігрову площину, або, у разі тривимірної графіки, використовують перспективу з легким нахилом ігрової поверхні. "Tetris" та "Portal" - це добре відомі представники цього жанру, приклад на рис. 1.3.

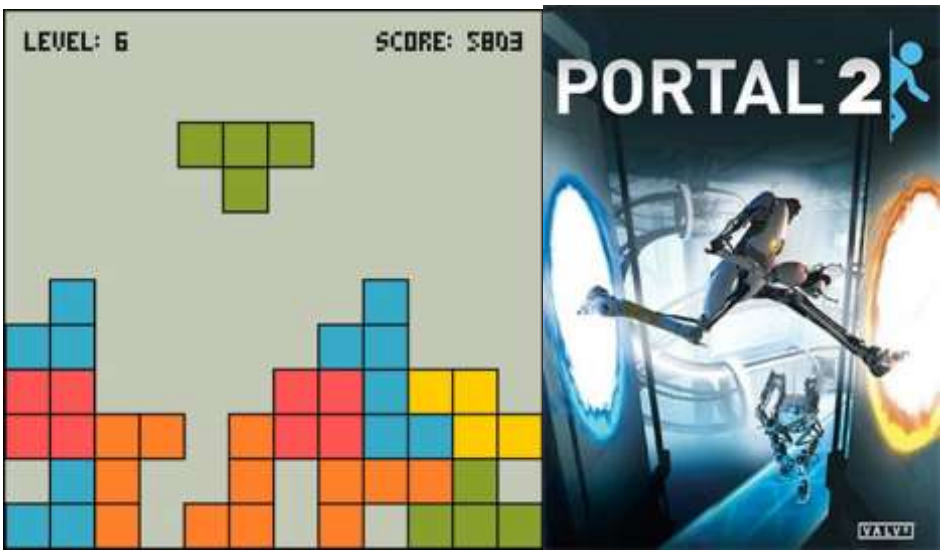

Рисунок 1.3 – Приклад головоломок "Tetris" та "Portal" [4]

Тактичні шутери. Поджанр шутерів, який приділяє особливу увагу стратегії, командній роботі та розгляду тактичних аспектів під час гри. У тактичних шутерах важливим аспектом є спланований підхід до боїв. Гравці повинні враховувати тактику, певні стратегії та плани дій, а не просто стріляти по всьому, що рухається. Часто прагнуть реалізму в зброї, екіпіровці, навколишньому середовищі та поведінці персонажів. Це може включати в себе реалістичну балістику, розслідування, вибір зброї і навіть симуляцію травм і пошкоджень. Приклади тактичних шутерів включають серії ігор, такі як "Tom Clancy's Rainbow Six", "Counter-Strike", і "Squad", приклад на рис. 1.4.

Рисунок 1.4 – Шутери "Tom Clancy's Rainbow Six", "Counter-Strike" [4]

Симулятори. Жанр відеоігор, які прагнуть моделювати певний аспект реального життя, події, процеси або дії. Вони розроблені для того, щоб дати змогу гравцям випробувати певні аспекти реальності або віртуального світу. Приклади симуляторів включають ігри, такі як "The Sims" (симулятор життя), "Microsoft Flight Simulator" (авіасимулятор), "Farming Simulator" (симулятор фермера) і безліч інших, приклад на рис. 1.5.

Рисунок 1.5 – Приклад симуляторів "The Sims" та "Farming Simulator" [4]

Хоррор. Жанр відеоігор, який створений для виклику у гравців почуття страху, тривожності та напруги. Вони зазвичай засновані на елементах жаху, містики, і психологічних ефектах, щоб створити атмосферу, яка змушує гравця боятися просуватися далі. Приклади хорор-ігор включають серії "Resident Evil", "Silent Hill", "Amnesia: The Dark Descent", "Outlast", приклад на рис. 1.6.

Рисунок 1.6 – Приклад хорор ігор "Resident Evil" та "Outlast" [4]

Стратегії. Жанр відеоігор, у яких гравці повинні розробляти плани, стратегії і тактики для досягнення певних цілей. Цей жанр включає різноманітні ігри, від глобальних стратегій з управління імперією до тактичних битв на полі бою. Прикладами стратегічних ігор є "Civilization" (глобальна стратегія), "StarCraft" (стратегія в реальному часі), "Total War" (стратегія в реальному часі і покрокова стратегія), "Age of Empires" (стратегія в реальному часі), і "XCOM" (тактична стратегія), приклад на рис. 1.7.

Рисунок 1.7 – Приклад ігор стратегій "Civilization" та "StarCraft" [4]

Візуальні романи. Особливий жанр комп'ютерних ігор, які являють собою суміш між інтерактивною книгою та грою. Візуальні романи зазвичай приділяють велику увагу сюжету, персонажам і наділяють гравця можливістю впливати на розвиток результату сюжету. Приклади популярних візуальних романів включають "Danganronpa" (детектив і жахи), "Steins;Gate" (наукова фантастика), "Clannad" (романтика і драма) (рис. 1.8), "Phoenix Wright: Ace Attorney" (юридична драма).

Рисунок 1.8 – Приклад візуальних романів [6]

Мультиплеєрні ігри. Вид відеоігор, у яких кілька гравців можуть брати участь в ігровому процесі одночасно, взаємодіючи один з одним у реальному часі. Ці ігри надають можливість змагатися, співпрацювати або просто взаємодіяти з іншими гравцями. Приклади мультиплеєрних ігор включають "Fortnite" і "PlayerUnknown's Battlegrounds" (шутери Battle Royale), "World of Warcraft" (MMORPG), "Minecraft" (пісочниця), "Dota 2" та "League of Legends" (моба-ігри з багатокористувацькими онлайн-аренками), приклад на рис. 1.9.

Рисунок 1.9 – Мультіплеєрні ігри "Dota 2" та "League of Legends" [6]

Характерними ознаками Action типу ігор є: динамічний геймплей, боротьба і битви, пригоди і дослідження, стрибки, перебирання перешкод, навички акробатики, наявність добре розроблених персонажів і захопливого сюжету, використання різноманітних спеціальних прийомів, магічних здібностей або умінь для урізноманітнення геймплею (рис. 1.10).

Рисунок 1.10 – Приклад action ігор [3]

Після огляду прикладів існуючого ігрового контенту маємо уявлення які типи ігор існують. Однак не буває чистих типів гри, усі ігри мають змішаний тип, що збільшує варіативність. У такий спосіб можна взяти основні механіки одного типу, наприклад аркад, і змішати з мультиплеєрною грою, у підсумку виходить онлайн ігри зі швидким гемплеєм. Наступним кроком розглянемо існуючи характеристики ігор та платформи їх реалізації.

1.3 Огляд характеристик ігор під існуючи платформи

Ігри доступні на різних платформах можуть відрізнятися за численними характеристиками, включаючи графіку, геймплей, аудіо та інші аспекти. Обираючи платформу для гри, важливо враховувати ваші особисті вподобання та можливості у власному обладнанні. Кожна платформа має свої переваги і недоліки, і вони можуть надавати різний досвід гри.

1. Вимоги для ігор на ПК (комп'ютерах) можуть значно відрізнятися залежно від конкретної гри, її графічної складності та оптимізації [9]. Однак основні елементи вимог зазвичай включають: операційну систему (ігри можуть бути призначені для різних операційних систем, таких як Windows, macOS або Linux), процесор (мінімальні та рекомендовані вимоги до швидкості ігрового процесора, або багатоядерний процесор для оптимальної продуктивності), оперативна пам'ять (мінімальна та рекомендована кількість оперативної пам'яті), графічна карта (Мінімальні та рекомендовані характеристики для якості графіки в іграх, що суттєво залежить від потужності графічної карти), місце на жорсткому диску (обсяг вільного місця, необхідного для встановлення гри), приклад на рис. 1.11.

Рисунок 1.11 – Приклад вимог ігри на персональні комп’ютери та консолі [3]

2. Вимоги для ігор на консолях (наприклад, PlayStation, Xbox, Nintendo Switch) зазвичай менше складні, ніж вимоги для ПК, оскільки консолі мають фіксовану апаратну архітектуру, і розробники можуть оптимізувати ігри для конкретної консолі. Однак загальні елементи вимог можуть включати: версія консолі (ігри призначені для певної версії ігрової консолі (наприклад, PlayStation 4, Xbox One, Nintendo Switch)), місце на жорсткому диску (обсяг вільного місця, необхідного для встановлення гри на внутрішній жорсткий диск консолі), контролери (ігри для консолей зазвичай оптимізовані для офіційних геймпадів конкретної консолі, і деякі функції гри можуть вимагати певних функцій контролера), інші периферійні пристрої (деякі ігри можуть підтримувати додаткові пристрої, такі як віртуальна реальність (VR) гарнітури або камери для спеціальних функцій гри).

3. Вимоги для ігор на мобільних пристроях, таких як смартфони і планшети, можуть варіюватися в залежності від конкретної гри та її графічної складності [17]. Однак загальні вимоги включають: операційна система (ігри зазвичай розробляються для певних мобільних операційних систем, таких як Android або iOS), процесор (мобільні ігри зазвичай оптимізовані для різних типів процесорів, включаючи ARM та Snapdragon), оперативна пам'ять (мінімальна кількість оперативної пам'яті, необхідна для гри), місце на

пристрої (обсяг вільного місця, необхідного для встановлення гри на внутрішній пам'яті пристрою), приклад на рис. 1.12.

Рисунок 1.12 – Приклад вимог на мобільні пристрої [3]

Після огляду вимог до ігор, маємо представлення основних характеристик пристроїв під різні платформи. Також дослідивши характеристики можна орієнтуватися на аудиторію людей, які можуть надалі грати в закінчений продукт. Наступним кроком розглянемо платформи, на яких можливо створювати ігрові додатки.

1.4 Огляд платформ для створення ігор на мобільні пристрої

Розробка ігор для мобільних пристроїв стала справжнім феноменом у світі геймдеву. Ігри на смартфонах та планшетах стали не просто розвагою, а повноцінною галуззю геймінгу зі своєю власною архітектурою, технологіями та інструментами для розробки [18]. Зараз розробка ігор для мобільних

пристроїв стала більш доступною, ніж коли-небудь, завдяки різноманітним платформам та фреймворкам. Основними платформами для розробки стали: Unity, Unreal Engine, Cocos2d-x, Godot Engine, GameMaker Studio, Construct [6].

1. Unity. Це потужний ігровий рушій, який широко використовується для створення відеоігор та інтерактивних додатків. Unity підтримує безліч платформ, включно з Windows, macOS, iOS, Android, консолями (PlayStation, Xbox, Nintendo), віртуальною реальністю та багатьма іншими. Це дає змогу розробникам створювати додатки для різних пристроїв. Надає інструменти для створення високоякісної 2D і 3D графіки. Він підтримує шейдери, освітлення, анімацію, системи частинок та інші засоби для створення візуально привабливих сцен. Має вбудовану фізичну модель, яка дає змогу створювати реалістичну поведінку об'єктів у грі. Це включає в себе симуляцію гравітації, зіткнень та інші фізичні ефекти. Підтримує інструменти для роботи зі звуком і музикою, включно з підтримкою 2D і 3D звукових ефектів, музичних треків та інтеграцією з різними форматами аудіофайлів. Unity використовує мову програмування C# для створення ігрової логіки. Розробники можуть писати скрипти, що керують поведінкою об'єктів і подіями в грі. Unity надає візуальні інструменти для створення сцен, редагування анімацій і налаштування компонентів без необхідності вручну кодувати кожну деталь. Присутні засоби для створення багатокористувацьких ігор і додатків з підтримкою мережевої гри.

2. Unreal Engine. Потужний і широко використовуваний ігровий рушій, розроблений компанією Epic Games. Unreal Engine славиться своїми вражаючими графічними можливостями. Він надає засоби для створення високоякісних 2D і 3D графічних сцен, включно з підтримкою фотореалістичної графіки, відображень, динамічного освітлення, тіней та інших візуальних ефектів. Містить вбудований фізичний рушій, що дає змогу створювати реалістичну поведінку об'єктів у грі. Це включає в себе симуляцію зіткнень, рідин, м'яких тіл та інших фізичних ефектів. Рушій надає потужні інструменти для роботи зі звуком, включно з підтримкою 3D звуку,

мікшуванням звукових ефектів і музики, а також інтеграцією з різними аудіоформатами. Unreal Engine підтримує безліч платформ, включно з PC, консолями (PlayStation, Xbox, Nintendo), мобільними пристроями, віртуальною реальністю та доповненою реальністю. Це забезпечує розробникам широкий спектр цільових пристроїв. Підтримує візуальну систему програмування під назвою "Blueprints", яка дає змогу створювати ігрову логіку без необхідності писати код. Це спрощує процес розробки для не-програмістів. Unreal Engine використовує мову програмування C++ для створення ігрової логіки та розширень. Розробники можуть використовувати C++ для глибшого налаштування та оптимізації гри. Надає доступ VR і AR розробку, що робить його популярним вибором для створення віртуальної реальності та доповненої реальності додатків. Unreal Engine має активну спільноту розробників, а також магазин активів і плагінів, де можна знайти готові ресурси та інструменти. Надає засоби для створення багатокористувацьких ігор і додатків з підтримкою мережевої гри.

3. Cocos2d-x - це безоплатний і відкритий ігровий рушій, який широко використовують для розроблення мобільних ігор і застосунків. Cocos2d-x дає змогу розробникам створювати ігри та додатки для різних платформ, включно з iOS, Android, Windows, macOS, Linux, HTML5, та інші. Це забезпечує максимальне охоплення аудиторії. Надає простий та інтуїтивно зрозумілий API, що робить його доступним навіть для початківців-розробників. Він також підтримує візуальний редактор для створення сцен і анімацій. Рушій підтримує 2D графіку й анімацію, включно зі спрайтами, частинками, текстурами, кадрами анімації та багатьма іншими графічними ефектами. Cocos2d-x містить інтеграцію з різними фізичними рушіями, такими як Box2D і Chipmunk, що дає змогу створювати реалістичну поведінку об'єктів у грі. Движок надає інструменти для роботи з аудіо, включно з відтворенням звукових ефектів і музики. Підтримує кілька мов програмування, включно з C++, JavaScript і Lua. Це дає розробникам можливість вибрати найбільш підходящу мову для своїх потреб. Cocos2d-x оптимізований для мобільних

пристроїв і надає інструменти для створення мобільних ігор, включно з підтримкою сенсорних екранів, багатозадачності та багатьох інших мобільних функцій. Рушій підтримує мережеву взаємодію, що дає змогу створювати багатокористувацькі ігри та додатки.

4. Godot Engine - це безкоштовний і відкритий ігровий рушій з чудовою репутацією серед розробників, особливо у сфері інді-ігор. Godot Engine підтримує різні операційні системи і платформи, включно з Windows, macOS, Linux, Android, iOS, HTML5 та інші, що дає змогу створювати ігри та додатки для різноманітних пристроїв. Пропонує видатну гнучкість у проектуванні ігрової логіки. Він використовує власну скриптову мову GDScript, яка схожа на Python, але також підтримує C# і VisualScript, візуальну систему програмування. Це дає змогу розробникам обирати найбільш підходящу мову для своїх потреб. Рушій підтримує як 2D, так і 3D графіку, включно зі спрайтами, анімацією, освітленням, частинками та іншими візуальними ефектами. Godot Engine містить вбудовану підтримку фізики, включно з рушієм Bullet для 3D-фізики і можливість створення фізичних ефектів у 2D. Також надає інструменти для створення багатокористувацьких ігор і додатків із підтримкою мережевої гри.

5. GameMaker Studio - це інтегроване середовище розробки та ігровий рушій, що дає змогу розробникам створювати 2D і 3D ігри та додатки. GameMaker Studio підтримує різні платформи, включно з Windows, macOS, Linux, Android, iOS, HTML5 і консолі (наприклад, PlayStation і Xbox), що дає змогу створювати ігри для різних пристроїв. Цей рушій орієнтований на розробників-початківців і надає зручний візуальний інтерфейс для створення ігрової логіки та сцен без необхідності писати код. GameMaker Studio використовує власну мову програмування GML (GameMaker Language), яка схожа на C. Розробники можуть використовувати GML для складніших і гнучкіших сценаріїв. GameMaker Studio має вбудовану підтримку фізичного моделювання, що дає змогу створювати реалістичну поведінку об'єктів у грі.

Рушій підтримує мережеву гру і надає інструменти для створення багатокористувацьких ігор.

6. Construct - це інструмент та ігровий рушій, призначений для створення 2D ігор та інтерактивних застосунків без необхідності писати код. Однією з головних особливостей Construct є його візуальна система програмування, яка дає змогу розробникам створювати ігрову логіку та взаємодію об'єктів, перетягуючи та з'єднуючи блоки в режимі "перетягни та кинь". Це робить інструмент доступним для тих, хто не має досвіду програмування. Construct дає змогу створювати ігри для різних платформ, включно з Windows, macOS, Linux, Android, iOS, HTML5 та іншими. Рушій підтримує 2D графіку, спрайти, анімацію, а також має інструменти для створення тайлових карт, анімованих спрайтів та інших графічних елементів. Construct надає можливості для керування аудіоефектами, музикою та звуковою інтеграцією.

Після огляду прикладів ігрових двигунів, які широко використовуються у індустрії, маємо величезний вибір сучасних рішень для розробки. Однак не всі рушії підходять під певні розробки і не всі у вільному доступі для розробки. У кожного ігрового рушія свої переваги перед іншими. Наступним кроком розглянемо платформи для створення 3D моделей під ігровий двигун.

1.5 Огляд платформ для створення 3D моделей

Створення 3D моделей стає все більш важливою складовою сучасного інтернет-простору, де вони використовуються для створення ігор, анімацій, віртуальної реальності та багатьох інших застосувань (рис. 1.13). Вибір правильної платформи для створення 3D моделей може суттєво вплинути на якість та ефективність проекту. Розглянемо різні платформи та інструменти, які допоможуть в цій творчій справі, відкриваючи нові можливості для ідей та концепцій у світі 3D моделювання. Основними засобами для створення 3D моделей будуть: Blender, Autodesk Maya, Cinema 4D, ZBrush.

Рисунок 1.13 – Приклад створеної 3D моделі [2]

1. Blender 3D - це відкритий, безкоштовний та дуже потужний інструмент для 3D моделювання, анімації, рендерингу та багатьох інших графічних задач (рис. 1.14). Він має ряд особливостей та характеристик, які роблять його популярним серед професіоналів і початківців. Blender є вільною та відкритою програмою, що означає, що ви можете використовувати його безкоштовно та мати доступ до вихідного коду. Це дозволяє користувачам адаптувати програму до своїх потреб та розвивати додаткові функції. Має вражаючий набір інструментів для 3D моделювання, включаючи моделювання об'єктів, текстури, анімацію, рендеринг, симуляції фізики та багато інших функцій. Підтримується на багатьох операційних системах, включаючи Windows, macOS та Linux, що робить його доступним для широкого кола користувачів. Інтерфейс Blender може здатися складним на перший погляд, але він пропонує широкі можливості для налаштування та дозволяє користувачам персоналізувати робочий простір під свої потреби.

Рисунок 1.14 – Приклад програми Blender 3D [2]

2. Autodesk Maya - це один із найпопулярніших та потужних інструментів для комп'ютерної графіки, 3D моделювання, анімації та візуальних ефектів (рис. 1.15). Він широко використовується в галузі анімації, кінематографії, відеоігор, архітектурного дизайну та інженерії. Maya надає великий набір інструментів для створення 3D об'єктів та моделей, включаючи полігональне, під-дівізійне та NURBS моделювання. Дозволяє створювати скелетну анімацію для персонажів, включаючи фейс-ріги та інші деталізовані анімаційні функції.

Рисунок 1.15 – Створення моделі у Autodesk Maya [3]

Маya використовується для створення візуальних ефектів у фільмах та відеоіграх, включаючи вибухи, вогонь, симуляцію частинок та інші спеціальні ефекти. Інтерфейс Маya досить складний, але він надає велику кількість налаштувань та можливостей для персоналізації робочого середовища.

3. Cinema 4D - це популярна програма для комп'ютерної графіки і 3D моделювання, розроблена німецькою компанією Maxon (рис. 1.16).

Рисунок 1.16 – Створення візуального ефекту за допомогою Cinema 4D [3]

Вона володіє декількома важливими характеристиками, які роблять її популярною серед художників, дизайнерів і візуальних ефектів. Cinema 4D славиться своєю дружньою та інтуїтивно зрозумілою інтерфейсом, що робить її ідеальною для початківців і досвідчених користувачів. Це особливо корисно для тих, хто швидко хоче навчитися створювати 3D моделі та анімацію. Має ряд інструментів для створення 3D моделей, включаючи полігональне та NURBS моделювання. Текстурування і нанесення текстур також виконуються зручно та швидко. Програма має потужні інструменти для створення рухливих об'єктів, включаючи скелетну анімацію, керування кадрами і анімацію частинок. Cinema 4D використовується для створення рухомої графіки та візуальних ефектів у відео, телебаченні та фільмах.

4. ZBrush - це потужний інструмент для комп'ютерного моделювання та скульптури в 3D-просторі. Він розроблений компанією Pixologic і використовується для створення високодеталізованих 3D-моделей, особливо в області художнього скульптурування (рис. 1.17).

Рисунок 1.17 - Високодеталізована 3D-модель за допомогою ZBrush [4]

Однією з основних особливостей ZBrush є можливість створювати дуже докладні 3D-моделі. Ви можете додавати деталі, текстури та рельєфи до своїх об'єктів без видимих геометричних обмежень. Хоча ZBrush в першу чергу призначений для створення 3D-моделей, він також має інструменти для створення анімації і анімаційних об'єктів. Підтримує роботу з текстурами та матеріалами, дозволяючи створювати реалістичні текстури та матеріали для вашого 3D-моделі. ZBrush використовується в ігровій індустрії та кіно для створення персонажів, монстрів, реквізитів та інших об'єктів.

Огляд та аналіз аналогів показав широкий спектр існуючих жанрів ігор з різними функціональними можливостями та технічними характеристиками. Це свідчить про розвиненість ігрової індустрії та багатство варіантів для вибору. Розглянувши багато жанрів ігор в цій роботі ми обрали жанр аркади у

поєднанні з механіками тактичних шутерів. Це додає прототипу варіативності, та дозволить гравцю не засумувати під час матчу. Поєднавши два типи ігор, отримуємо швидку гру з гнучкою варіативністю дій залежно від ситуації під час матчу. Розглянувши велику кількість ігрових движків і характеристик під ігри, обирають розробку прототипу під мобільну платформу Android, це дасть велику аудиторію для користування готовим ігровим додатком. Ігрових движків, які можуть впоратися з поставленим завданням і щоб був відкритий код, залишається небагато. Наступним етапом буде обиратися необхідна для розробки платформа, яка буде мати доступність і мультиплатформеність. Також ігровий рушій повинен підтримувати 3D-моделі, і здатний компілювати з ними під мобільні платформи. Щоб заповнити прототипами рівень і реалізувати 3D моделі розглянуто основні програми для розробки тривимірної графіки. Після аналізу стало зрозуміло, що найдоступнішим додатком є Blender за його гнучку систему моделінгу та великий набір інструментів. Він універсальний і вивантаження моделей відбувається швидко, також можна одразу текстурувати моделі в самому додатку. Провівши огляд і обравши програми для подальшої роботи, переходимо до аналізу обраних додатків для розробки та розгляду їхніх функцій.

2 АНАЛІЗ ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ ТА ОБҐРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТІВ

2.1 Обґрунтування вибору основної платформи для ігрового додатку

Першочерговим етапом при розробці ігрового додатку визначається платформа та вимоги до апаратного забезпечення. Як платформу під ігровий додаток було обрано Android, за його широкий спектр пристроїв, які працюють на цій платформі. Від дешевих смартфонів до потужних планшетів і флагманських смартфонів, Android покриває велику частину ринку, що дозволяє досягти більшої аудиторії.

Android пропонує ігрові можливості, такі як Google Play Games, які полегшують інтеграцію мережевого взаємодії, досягнень та інших функцій. Також, Android дозволяє використовувати різні механізми монетизації, такі як внутрішні покупки та реклама, приклад на рис. 2.1.

Рисунок 2.1 – Приклад додатків з Google Play Games [3]

Платформа надає можливість тестування ігор безпосередньо на реальних пристроях, що є важливим етапом в розробці для мобільної платформи. Це дозволяє виявити і виправити потенційні проблеми, які можуть виникнути на різних пристроях.

2.2 Обґрунтування вибору Unreal Engine, як основної платформи розробки

Вибір Unreal Engine як основної платформи для розробки ігрового додатку під Android обумовлений його потужністю та високоякісним графічним двигуном. Unreal Engine володіє передовими технологіями рендерингу, освітлення та обробки текстур, що дозволяє створювати вражаючі візуальні ефекти та реалістичні ігрові світи.

Окрім вражаючої роботи з графікою, ігровий двигун підтримує кросплатформеність. Unreal Engine є кросплатформеним інструментарієм, що робить його ідеальним вибором для розробки ігор на Android. Відкриті API Unreal Engine легко інтегруються з Android SDK, що дозволяє безперешкодно взаємодіяти з характеристиками та функціоналом платформи.

Важливим фактором для вибору ігрового двигуна були широкі можливості програмування, використовуючи мову C++ або графічний інтерфейс Blueprints (рис. 2.2). Це дає розробникам велику свободу при створенні складних геймплейних механік, штучного інтелекту та інших елементів гри.

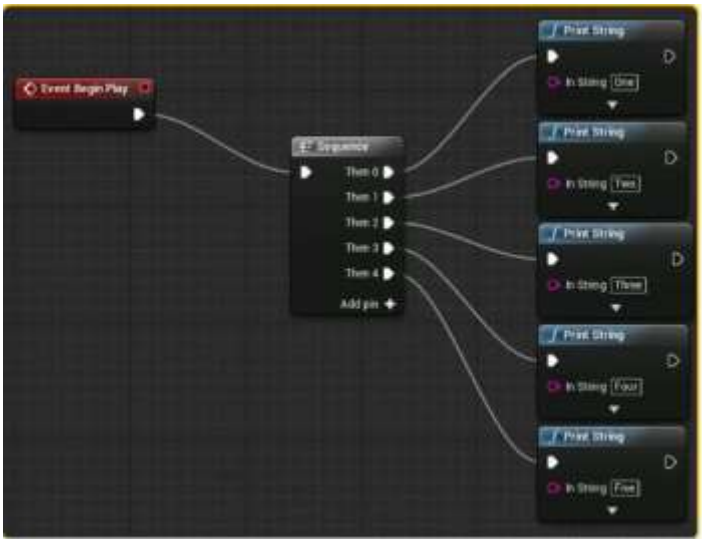

Рисунок 2.2 – Приклад візуального скриптингу Blueprints [1]

Unreal Engine інтегрується з різноманітними інструментами для 3D моделювання та анімації, такими як Blender чи Autodesk Maya (рис. 2.3). Це забезпечує ефективну роботу з графікою та анімацією об'єктів у грі.

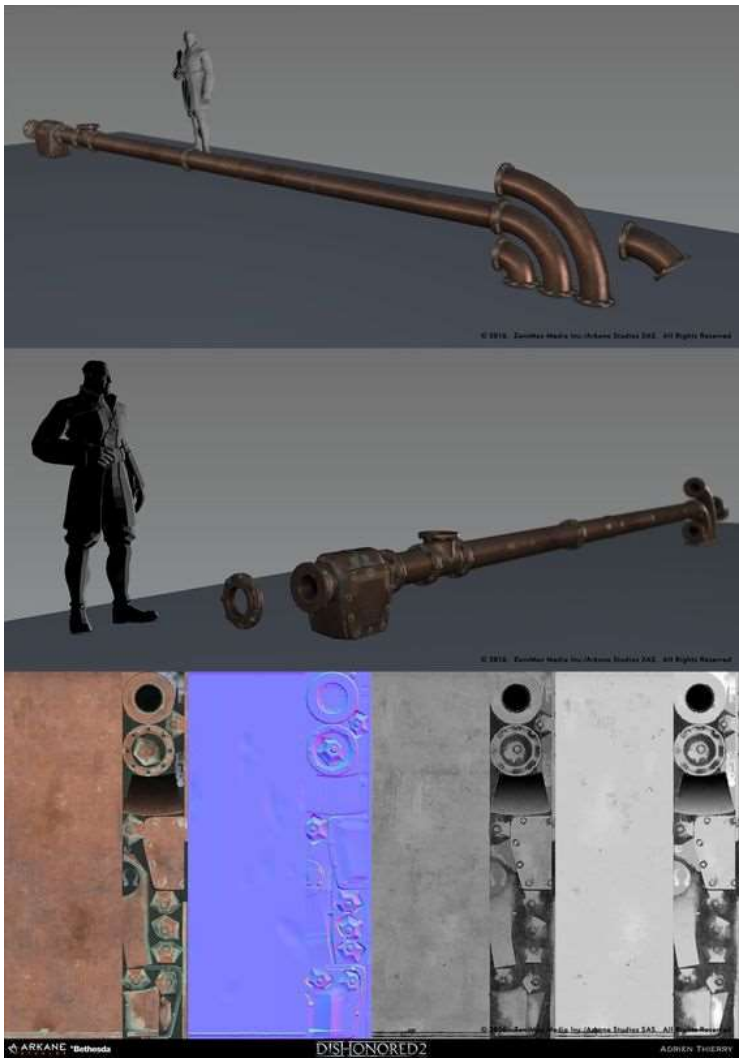

Рисунок 2.3 – Реалізації 3D об’єкту під ігровий двигун Unreal Engine [1]

Можливості налаштування графіки та коду дозволяють забезпечити високий рівень продуктивності на різних пристроях (рис. 2.4).

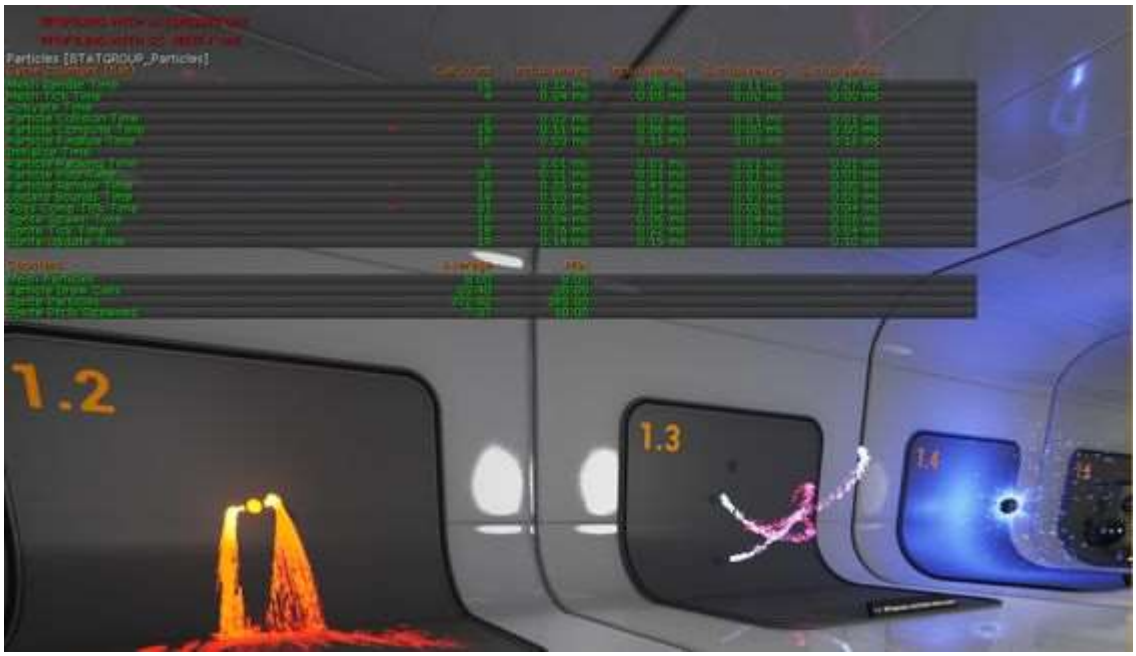

Рисунок 2.4 – Основні статистики оптимізації під час гри [1]

Ігровий двигун обраний в цій роботі надає інструменти для ефективної оптимізації гри під різні пристрої Android. Оскільки у програму вбудовано потужний графічний редактор, який дає змогу гнучко працювати з матеріалами та текстурами. Однак крім графічного ядра, є потужна платформа програмування та її доступність для всіх користувачів-початківців.

2.3 Обґрунтування вибору інструментів для 3D моделювання та візуалізації

Для створення 3D об'єктів та їх візуалізації в грі було обрано інструмент моделювання Blender. Це універсальний інструмент для 3D моделювання з великим спектром функціоналу, що дозволяє задовольняти потреби як початківців, так і досвідчених графічних дизайнерів чи аніматорів.

Blender надає різноманітні інструменти для створення та редагування мешів, такі як Extrude, Loop Cut, Bevel, і багато інших (рис. 2.5). Включає в себе різноманітні пензлі та інструменти для скульптування 3D об'єктів, що дозволяє долучити деталі та текстури.

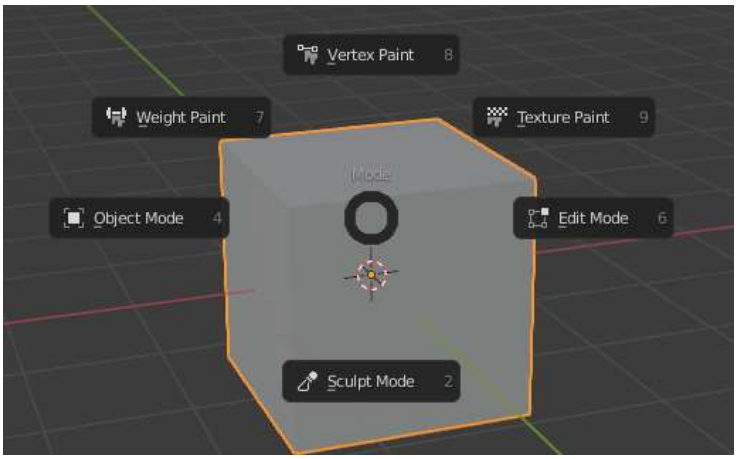

Рисунок 2.5 – Основні інструменти редагування мешів

У двигун включається інструмент для текстурування та налаштування матеріалів. Для цього наявні інструменти для створення та редагування UV-розгортань для текстур. Material Editor: Дозволяє визначати властивості матеріалів, включаючи текстури, колір, відбивання світла тощо.

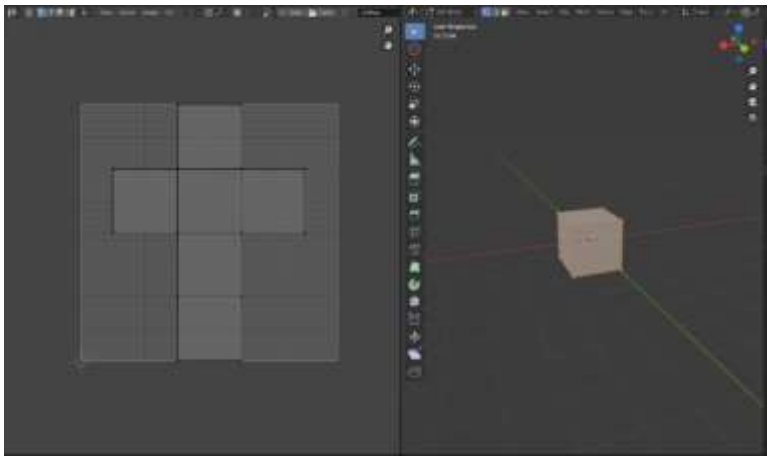

Рисунок 2.6 – Вікно UV-розгортань для текстур

Blender підтримує широкий спектр форматів файлів для імпорту та експорту, що полегшує співпрацю з іншими програмами. Для ігрового двигуну Unreal Engine підходить один із форматів, а саме .fbx. Цей формат містить дані про матеріали, анімації, налаштування нормалей об'єкта, внутрішні зміни об'єкта, створені колізії для об'єкта. З додаткових можливостей містить призначення кольору вершин і спрайтові анімації.

2.4 Аналіз етапів проектування гри та вибір концепції

Перед початком розробки ігрового додатку необхідно чітко визначити його концепцію. В кваліфікаційній роботі, обрано концепцію аркади гри з елементами тактичного шутеру. Гравець отримає можливість переміщуватись по рівню, взаємодіяти з об’єктами на рівні та ботами, які з’являються у певний проміжок часу. Ціллю буде назбирати максимальну кількість очків за виділений час.

Основні геймплейні механіки включатимуть в себе елементи бойових взаємодій, систему збору очків, чи інших предметів за необхідністю гри. Гравець матиме можливість вибору взаємодії з ботами, та впливу на події в рівні. Уся взаємодія обмежена часом гри, розширити його можливо завдяки взаємодії, однак гравцю потрібно слідкувати за особистим здоров’ям та ботами, що патрулюють рівень (рис. 2.7).

Рисунок 2.7 – Реалізація рівню з геймплеєм

Архітектура проекту буде побудована на основі принципів модульності та розширюваності. Основні компоненти включатимуть графічний двигун Unreal Engine, систему штучного інтелекту для не персонажів, систему

управління об’єктами на рівні, а також модулі для обробки анімацій та фізики гри. Бойові механіки реалізуються у компоненті, щоб мати гнучку систему реалізації та додавання до будь-яких додаткових персонажів. Також це дозволяє реалізувати відмінність взаємодії бойовими навичками у різних персонажів (шкоду, попадання і тощо).

Після визначення реалізації, геймплею, дизайну, розробляємо етапи розробки:

1. Концептуалізація та дизайн: детальне визначення сценарію гри, находження концептуальних малюнків та дизайну рівня.
2. Моделювання та текстурування: виготовлення 3D моделей об'єктів та оточення, а також створення текстур для них. Персонажів та анімацію береться з сайту Mixamo (рис. 2.8) [5].

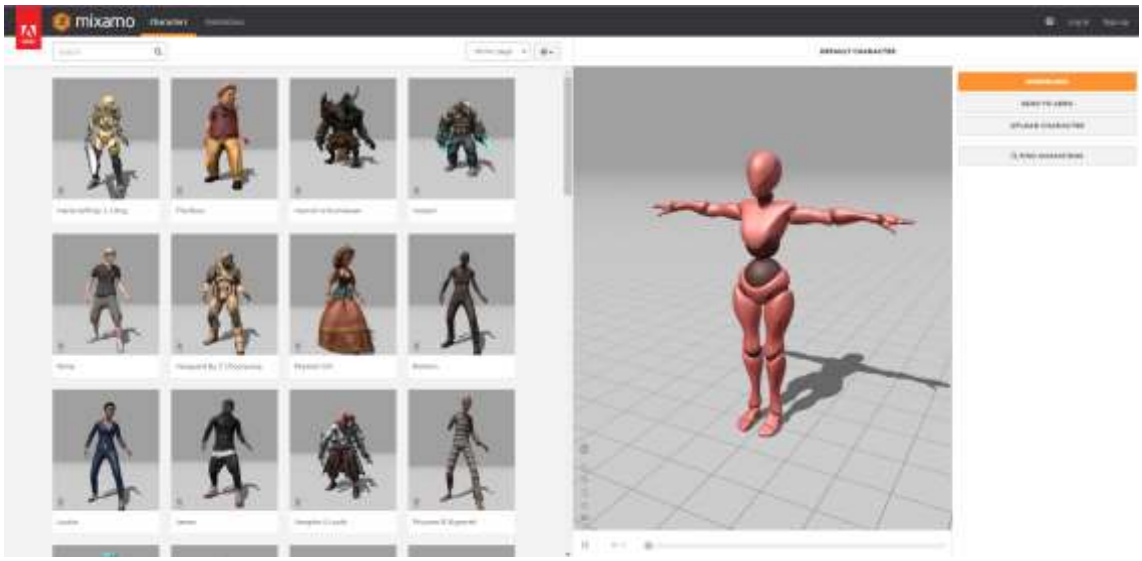

Рисунок 2.8 – Сайт Mixamo з персонажами та анімаціями [5]

3. Програмування: розробка геймплейних механік, системи штучного інтелекту, управління завданнями та інших ключових елементів гри.
4. Тестування та оптимізація: використання тестування для виявлення та виправлення помилок, а також оптимізація гри для забезпечення стабільної роботи.

Під час огляду інструментів та технічного забезпечення були виявлені певні етапи розробки, які залишаються малодослідженими або можуть потребувати більш детального аналізу, а саме: основи та принципи 3D моделювання, текстурування, розгортка моделей, анімація об’єктів. Розглянуто основні функції обраних програм, під мобільну платформу. Android платформа має велике коло користувачів через їхню доступність у світі, також пристрої мають різні характеристики, а отже застосунок можна зробити під велике коло користувачів, якщо врахувати мінімальні вимоги до ігрового застосунку. Розглянуто внутрішній функціонал ігрового рушія Unreal Engine, де показано приклади його використання. Крім прототипування ігрових застосунків у рушії є можливість прототипувати рівні за допомогою створеної 3D геометрії з використанням Blender, як головного тривимірного редактора. У розділі розглянуто етапи роботи з тривимірною графікою перед використанням їх в ігровому рушії, що дає змогу розширити функціонал ігрових застосунків і механік.

3 МОДЕЛЮВАННЯ ТА ТЕКСТУРІНГ 3D ОБ’ЄКТІВ ДЛЯ ІГРОВОГО ДОДАТКУ НА ANDROID

Розглянемо процес 3D моделювання та текстурування в контексті Unreal Engine, зосереджуючись на використанні популярного інструменту Blender. Розглянуто роль цих елементів у створенні ігрових об'єктів, їх текстуруванні, а також розглянемо, як ці компоненти взаємодіють для створення неповторних іммерсивних ігрових світів [2].

3.1 Аналіз основних етапів 3D моделювання у Blender

Основною будівельною одиницею у Blender є mesh - сітка, що визначає форму об'єкта. Робота з mesh є важливою частиною процесу 3D моделювання. Mesh визначає структуру об'єкта, складаючись із вершин, ребер та поверхонь. Створення та редагування вершин, ребер та поверхонь відбувається за допомогою додавання та видалення, це можна реалізувати при переході у режим: Edit Mode (рис. 3.1).

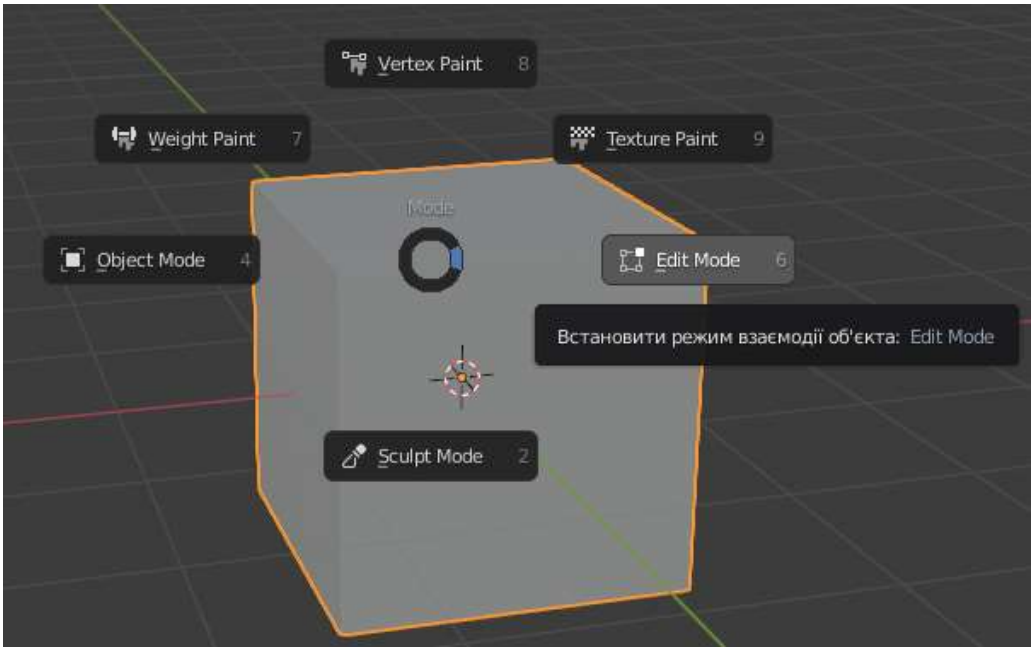

Рисунок 3.1 – Перехід до режиму Edit Mode

Розглянемо які інструменти надає Blender для додавання нових вершин до mesh, що дозволяє точно визначати форму об'єкта. Видалення вершин також може використовуватися для модифікації форми. Ребра з'єднують вершини та визначають грані об'єкта. Редагування грані дозволяє контролювати геометрію та форму об'єкта. Поверхні та грані утворюють області між вершинами та ребрами. Редагування граней дозволяє контролювати структуру та форму поверхні об'єкта (рис. 3.2).

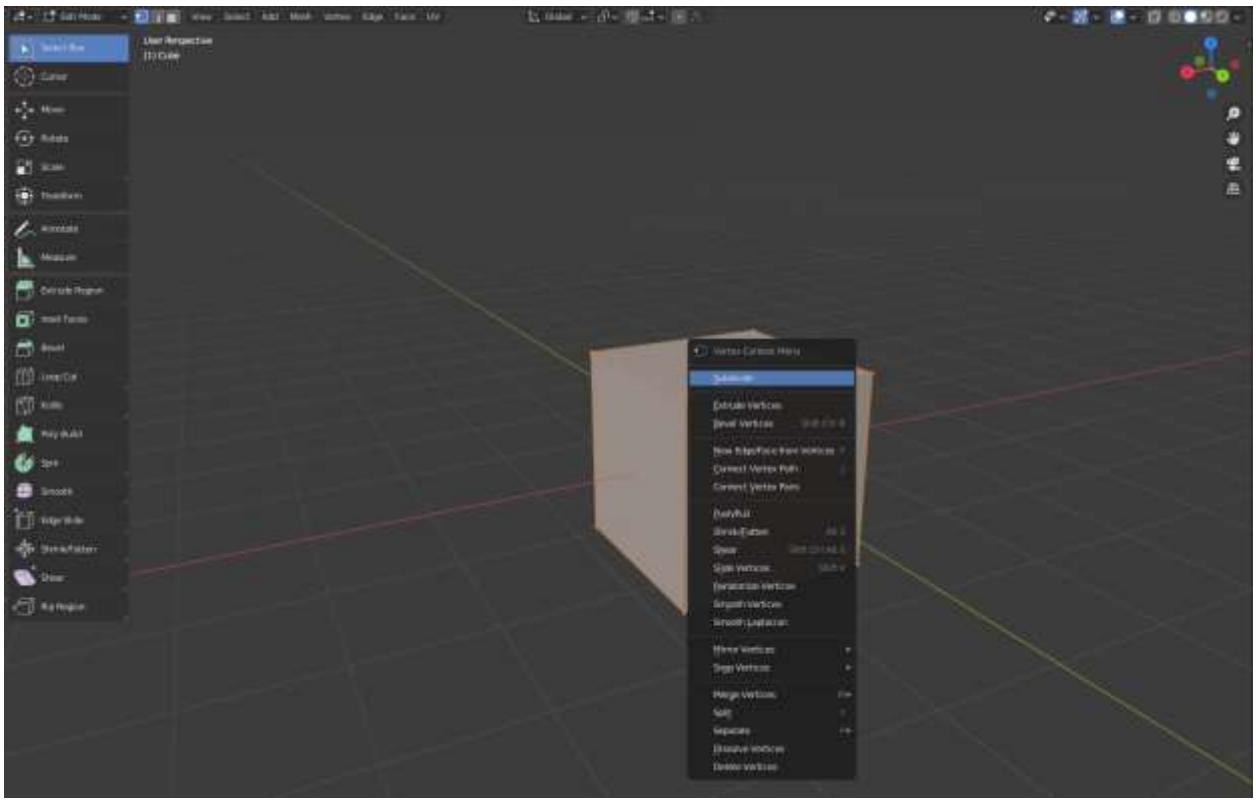

Рисунок 3.2 – Інструменти редагування у режимі Edit Mode

Окремо від інструментів редагування існують модифікатори, які допомагають досягти потрібної форми з меншими маніпуляціями геометрії у режимі Edit Mode. Використання модифікаторів розширює можливості моделювання, основними модифікаторами є: Subdivision Surface (збільшує кількість граней, створюючи більш гладку поверхню. Корисний для створення округлених форм), Boolean (дозволяє об'єднувати або відокремлювати об'єкти,

щоб швидко створювати складні форми), Array (дозволяє додати об’єкт на якому модифікатор, назначену кількість разів на певній відстані), Mirror (віддзеркалює об’єкт згідно нульової точки у об’єкті), Solidify (нарощує товщину об’єкту). Самі модифікатори діляться на: modify, generated, deform, physics (рис. 3.3). За категоріями відбувається вибір потрібного модифікатора для mesh, категорія відповідає на спосіб взаємодії модифікатора з об'єктом.

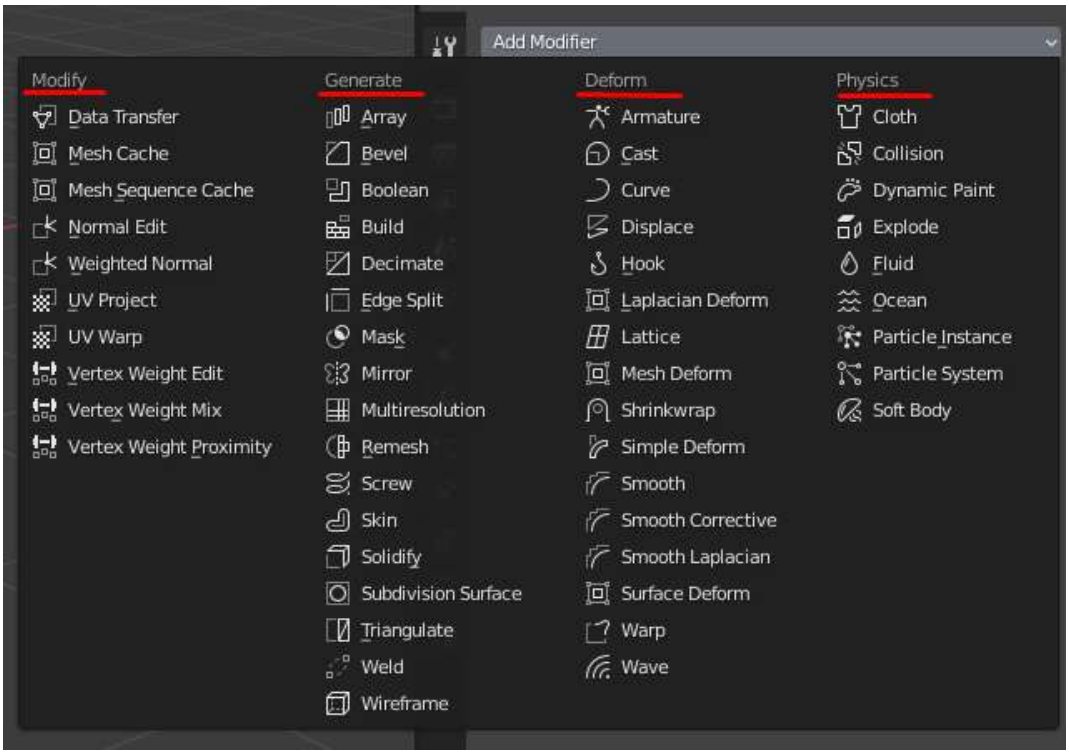

Рисунок 3.3 – Лист модифікаторів

Оскільки при моделінгу проходить деформація сітки, тому важливим фактором є збереження правильної топології та оптимізація моделі. Тому існують правила для правильної топології, основні правила: збереження полігону який складається з чотирьох вершин, оминати ситуацій коли створюється одна вершина в яку входить багато ребер (її називають зіркою у моделювані), приклад правильної топології на рис. 3.4.

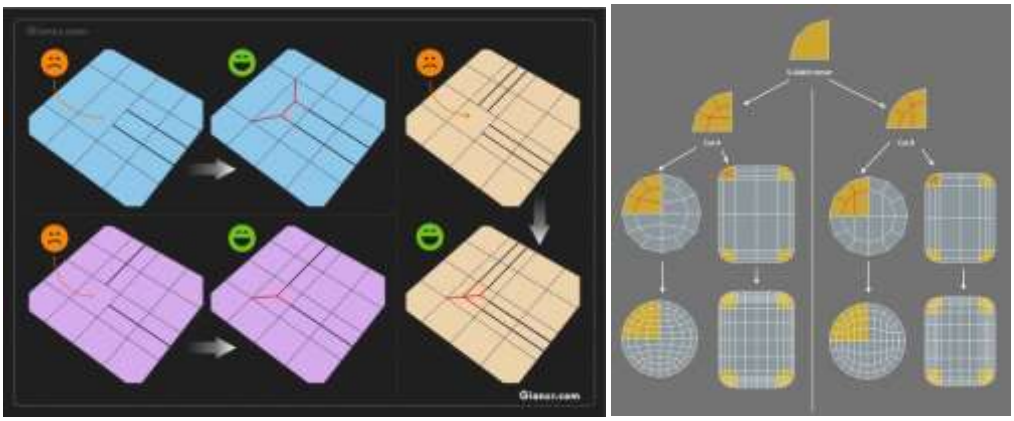

Рисунок 3.4 – Приклад роботи з топологією у об’єкта [3]

Оскільки правильна топологія дозволяє об'єктові гарно виглядати при будь-якому освітленні та анімації також окрім структури важливим явищем складає оптимізація, де потрібно стежити за кількістю вершин та граней, оптимізувати модель для забезпечення оптимальної продуктивності у грі, приклад відображення статистики mesh у вікні редагування рис. 3.5.

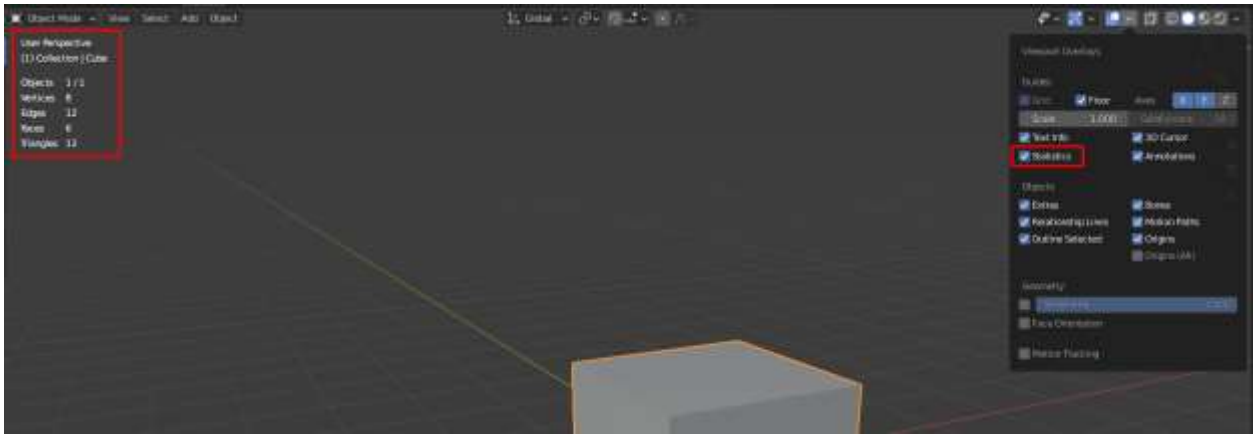

Рисунок 3.5 – Відображення статистики mesh у вікні редагування

Розглянуто етапи роботи з тривимірною графікою та виокремлено основні правила під час побудови моделей. Показано детально інструменти для редагування геометрії, і варіації використання цих інструментів, що розширює можливості Blender. Продемонстровано основний спосіб, за допомогою якого можна відстежувати складність топології в тривимірних об'єктах.

3.2 Аналіз етапу роботи з UV-Розгорткою для текстурингу

Підготовка моделі до текстурування включає в себе ключовий етап - роботу з UV-розгорткою (рис. 3.6). Цей процес визначає, як текстури будуть проектуватися на поверхню 3D об'єкта, і є важливим елементом для досягнення реалістичного вигляду у грі. Основні кроки роботи з UV-розгорткою у Blender: розділення об'єкта (розділення об'єкту на його складові частини, які зручно розгортати окремо. Це спростить процес текстурування), робота з матеріалами (прив'язка матеріалу до різних частин об'єкта, щоб кожна частина мала відмінні текстури).

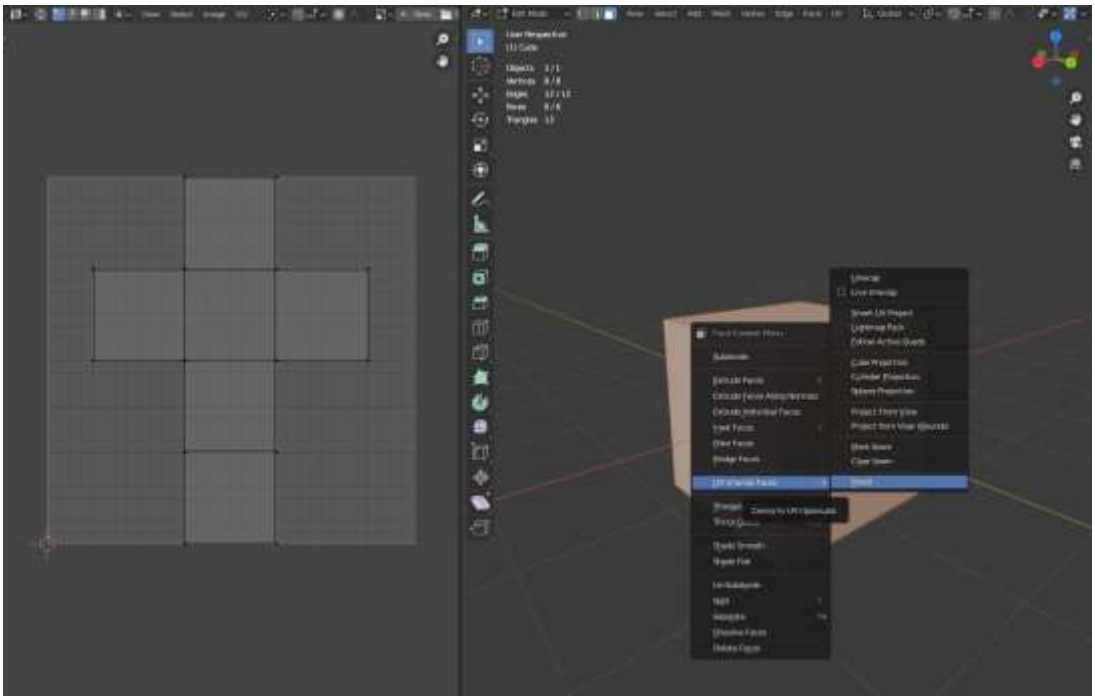

Рисунок 3.6 - UV-розгортка куба

Для розгортки об’єкту увімкнення режиму редактора UV та розгортання mesh. Використання інструменту розтягування, обертання та масштабування використовується для оптимального розташування UV-елементів. Наступним кроком оптимізування, щоб розташувати елементи для ефективного використання текстурного простору. Інструментів пакування

використовуються для уникнення розривів та максимізації роздільної здатності.

Після розгортання пов'язуються текстури з відповідними UV-координатами. Розуміння взаємодії між UV та текстурою визначає, як точно зображення буде відображатися на об'єкті.

Робота з UV-розгорткою в Blender - це не лише необхідний етап для текстурування, але й важливий для створення ефективних та реалістичних текстур на поверхні 3D об'єктів у грі.

3.3 Розробка 3D об'єкту під гру

Опишем розробку та моделінг об'єктів, що будуть використовуватись в цій роботі. Для цього імпортуємо в Blender фігурку людини з Unreal Engine, це робиться для правильного моделювання форми об'єкта, щоб створити потрібну висоту і розміри (рис. 3.7).

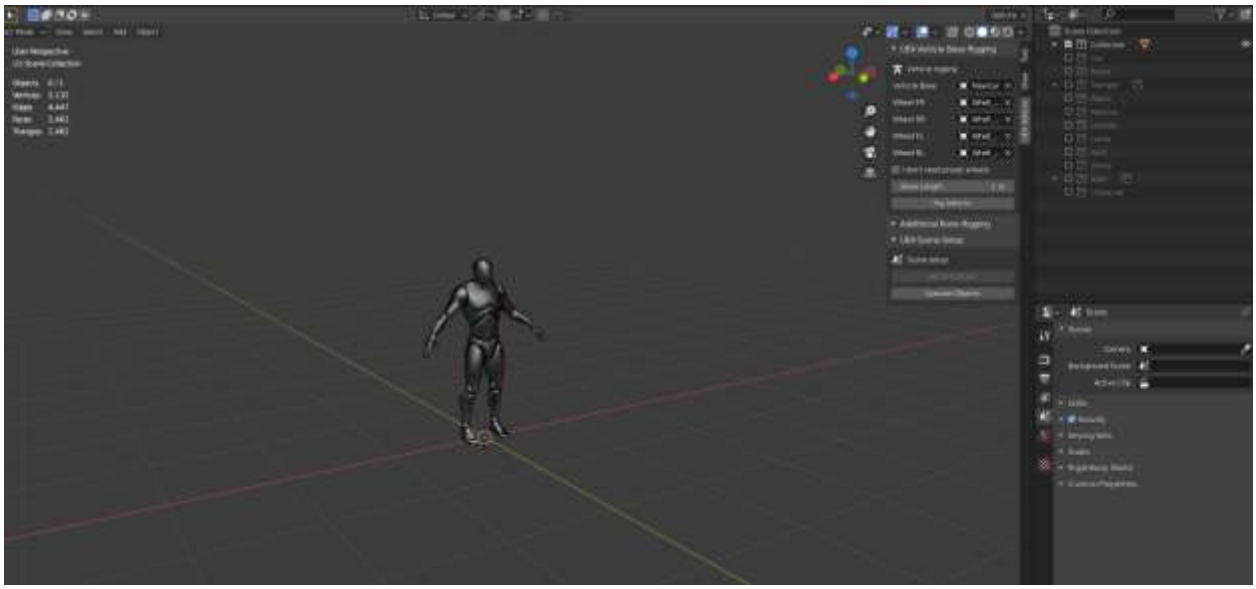

Рисунок 3.7 – Імпорт фігурки з Unreal Engine та додавання у колекцію окремо

Створимо нову колекцію, яку назвемо "Trample", щоб відокремити безліч 3D моделей і створити текстурний атлас. Створення будь-якого об'єкта

починається з примітивів, таких як куб, циліндр, конус, куля, площини. Для проекту, щоб створити трамплін, краще почати з площини (рис. 3.8).

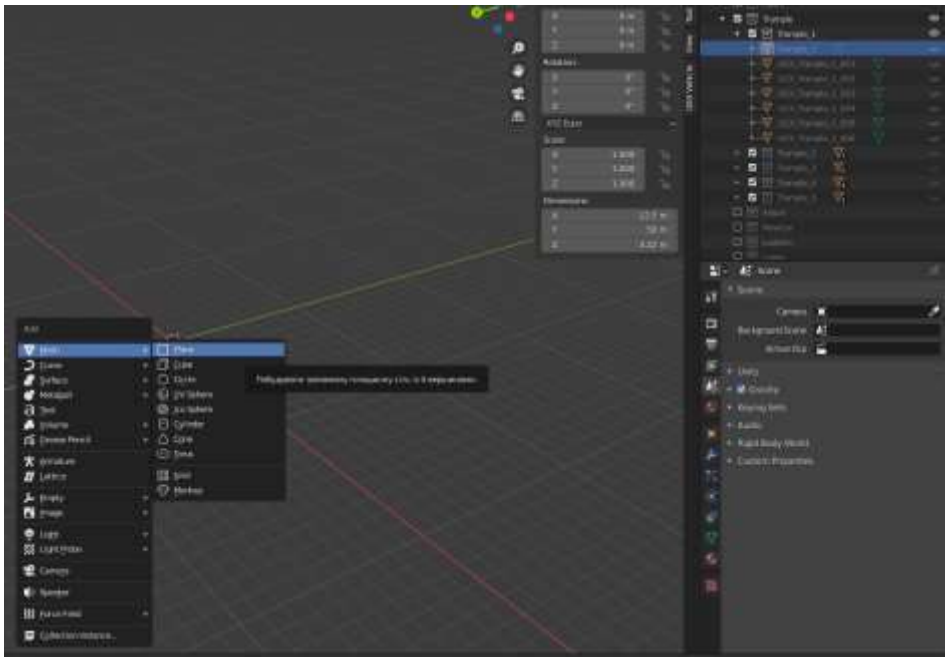

Рисунок 3.8 – Створення трампліну з площини

Переходимо в режим Edit mode, і за допомогою маніпуляцій полігонами, ребрами, вершинами створюємо трамплін. З маніпуляцій було використано нарощування геометрії, масштабування, з'єднання точок, створення фасок, додавання ребер, видалення непотрібних полігонів (рис. 3.9).

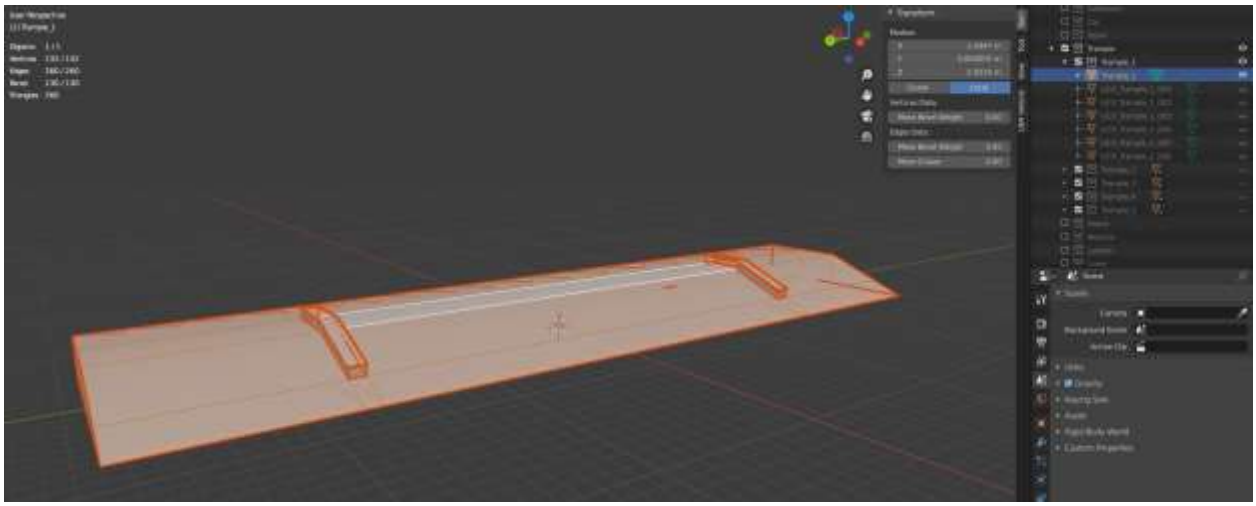

Рисунок 3.9 – Модель трампліну після маніпуляцій з геометрією

Після створення моделі потрібно створити розгортку для трампліна, для цього виділяємо ребра під 90 градусів і призначаємо розріз. Виділивши необхідні ребра, створюємо розгортку на UV-координатах (рис. 3.10).

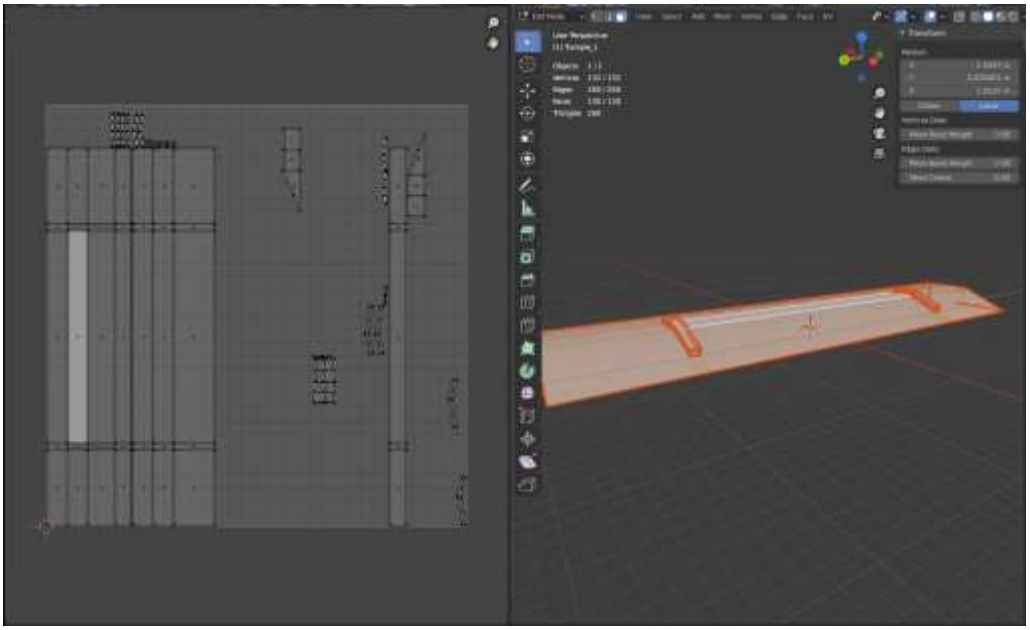

Рисунок 3.10 – UV- розгортка трампліну

Створивши 3D-об'єкт, необхідно створити колізію, щоб персонаж міг взаємодіяти з трампліном у грі. Колізію створюють із простих примітивів, і кожен примітив має підписуватися певним позначенням, що декларується в спеці Unreal Engine (рис. 3.11).

Collision meshes are identified by the importer based on their name. The collision naming syntax should be:

Mesh Prefix and Name	Description
UBX_[RenderMeshName]_##	A **Box** must be created using a regular rectangular 3D object. You cannot move the vertices around or deform it in any way to make it something other than a rectangular prism, or else it will not work.
UCP_[RenderMeshName]_##	A **Capsule** must be a cylindrical object capped with hemispheres. It does not need to have many segments (8 is a good number) at all because it is converted into a true capsule for collision. Like boxes, you should not move the individual vertices around.
USP_[RenderMeshName]_##	A **Sphere** does not need to have many segments (8 is a good number) at all because it is converted into a true sphere for collision. Like boxes, you should not move the individual vertices around.
UCX_[RenderMeshName]_##	A **Convex** object can be any completely closed convex 3D shape. For example, a box can also be a convex object. The diagram below illustrates what is convex and what is not:

Рисунок 3.11 – Правило підписів з документації Unreal Engine [1]

Для трампліна створення колізії краще робити з примітивів куба, і самих примітивів буде кілька, підписом будуть позначення UCX_(назва трампліна)_(цифра примітива), приклад на рис. 3.12.

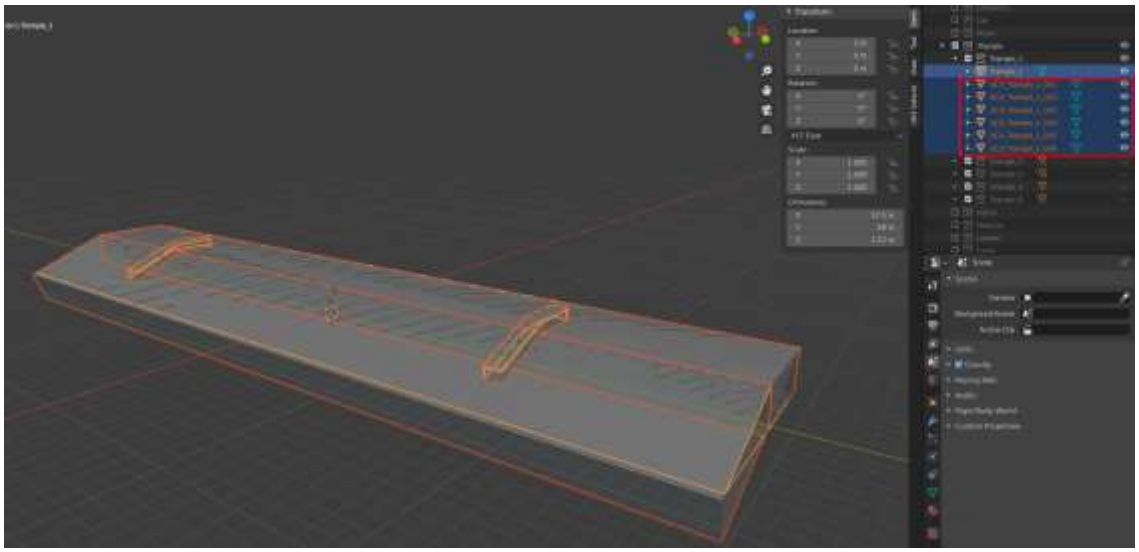

Рисунок 3.12 – Створення колізії трампліну

Останнім кроком для експорту в ігровий рушій потрібно налаштувати правило експорту. Для цього виділяємо колізії та об'єкт, у налаштуваннях позначаємо експорт виділених об'єктів, спрямованість згладжування нормалей і безліч інших параметрів (рис. 3.13).

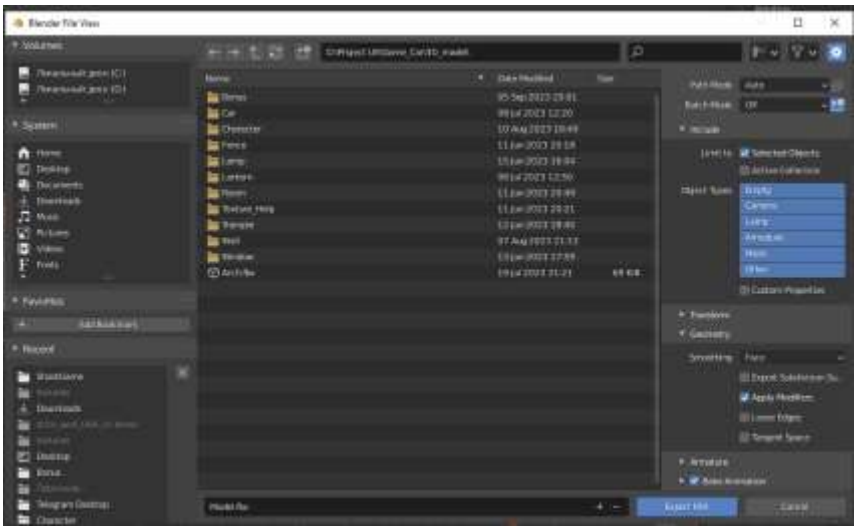

Рисунок 3.13 – Приклад налаштування експорту у формат .fbx

У процесі розробки було розглянуто етапи моделювання тривимірної графіки з використанням програми для моделювання. Винесено основні правила топології під час моделювання та показано спосіб спостереження за складністю геометрії. Розглянуто етап роботи з UV-розгорткою, робота і правила створення розгорток на тривимірних об'єктах. Показано приклад розгортки на одному з розроблених прототипів для застосунку. Наприкінці показано приклад розроблений дотримуючись етапів моделювання, створено колізію з багатьох примітивів, як куб, під ігровий рушій. Кожну колізію підписано згідно документації рушія: UCX_(назва об’єкту)_(цифра примітива). Згідно документації показано налаштування імпорту моделі в Unreal Engine, але такий імпорт можливо використовувати і в інших рушіях, формат імпортованої моделі обов’язково повинен бути .fbx, тому що він зберігає дані усіх налаштувань та текстури з матеріалом у 3D моделі.

4 РЕАЛІЗАЦІЯ ПРОТОТИПУ ІГРОВОГО ДОДАТКУ У UNREAL ENGINE

Розглянемо процес розробки ігрового додатку для платформи Android, використовуючи середовище Unreal Engine. Покажемо практичні аспекти, такі як особливості кодування, взаємодія з API Android та інші технічні деталі, які є ключовими для успішної реалізації гри [1].

4.1 Реалізація налаштування проекту під Android

Перед налаштуванням проекту під мобільні платформи, необхідно завантажити SDK і NDK компоненти. Щоб їх встановити спочатку потрібно завантажити AndroidStudio, попередньо вибравши потрібну версію під ігровий рушій (рис. 4.1).

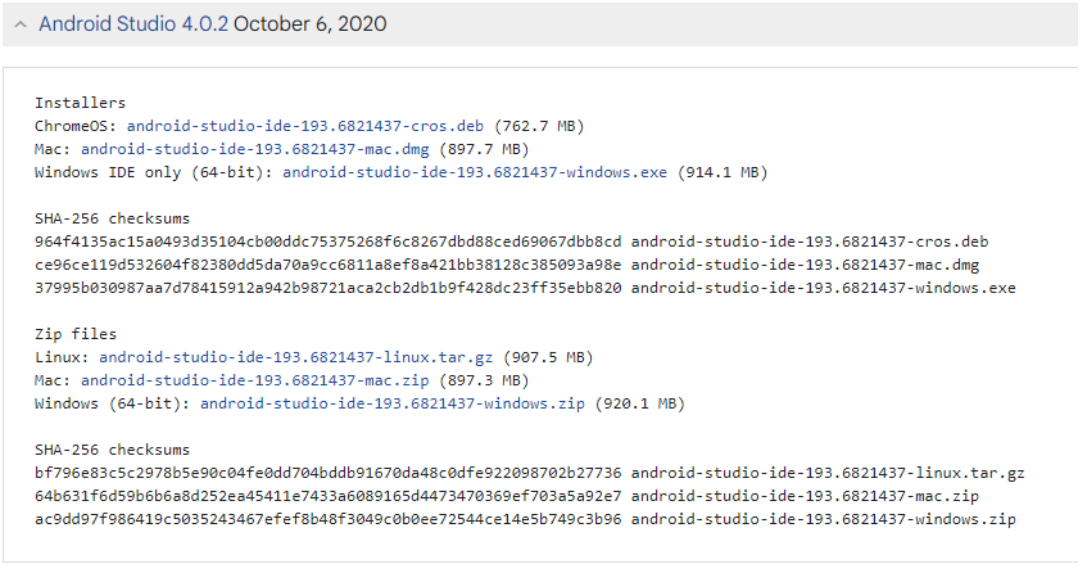

Рисунок 4.1 – Версія AndroidStudio під Unreal Engine 4.27 [1]

Закінчивши скачування AndroidStudio, потрібно налаштувати SDK і NDK компоненти. Для цього потрібно зайти в розкривний список "Configure"

і натисніть "SDK Manager". У налаштуваннях системи Android SDK перейти на вкладку "SDK Tools" , де з'явиться список додаткових компонентів (рис. 4.2).

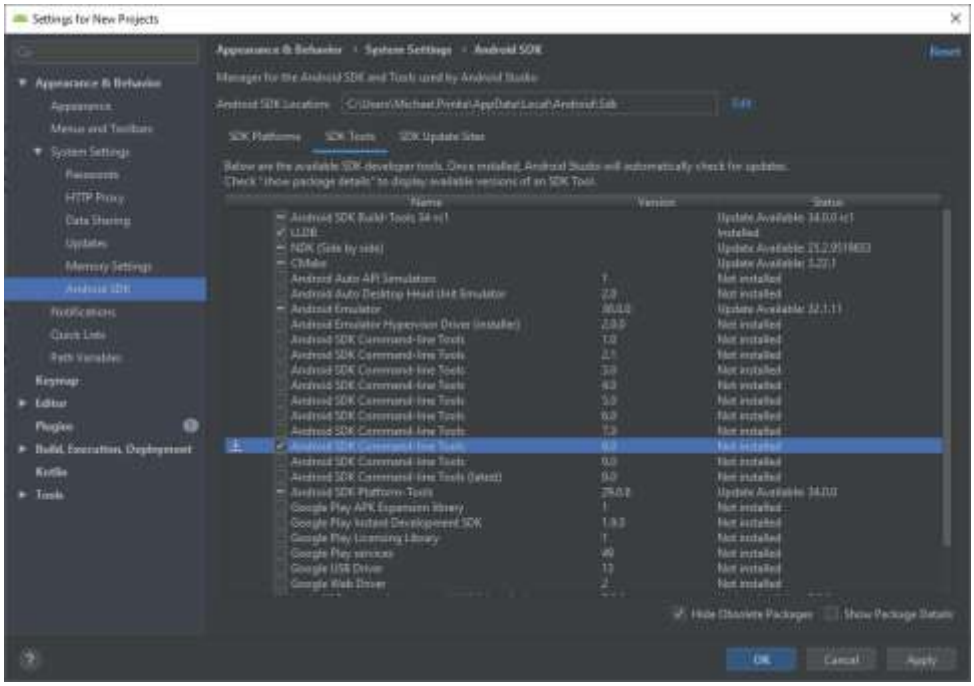

Рисунок 4.2 – SDK Manager [1]

Встановивши необхідні компоненти Android SDK, переходимо до встановлення відповідної версії Android NDK, використавши SetupAndroid сценарій для завантаження. Для цього в командній панелі потрібно прийняти ліцензійну угоду командою "Y" (рис. 4.3).

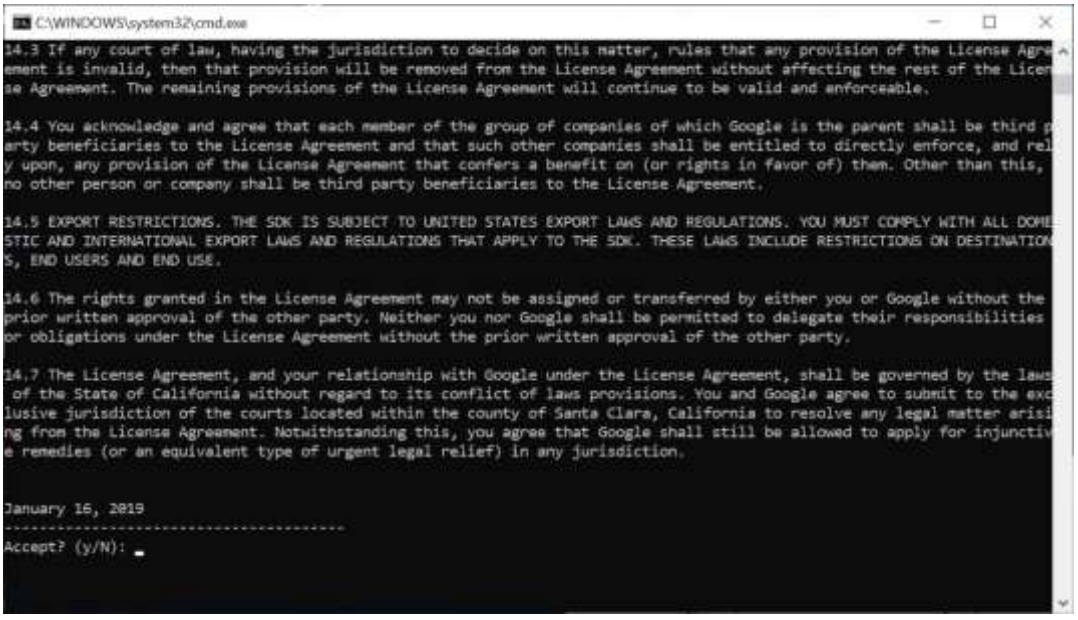

Рисунок 4.3 – Командна панель для установки NDK [1]

Після встановлення всіх компонентів потрібно зайти в ігровий рушій і перейти в розділ "Edit > Project Settings", де вручну встановити шлях до компонентів "Platforms > Android > Android SDK (Android NDK)" (рис. 4.4).

Рисунок 4.4 – Ручне налаштування шляху до компонентів [1]

У розділі розглянуто налаштування ігрового рушія Unreal Engine під мобільні додатки Android. Показано ручне налаштування рушія та які компоненти потрібні для кінцевої компіляції продукту для тестування, описано де завантажити компоненти під ігровій рушій.

4.2 Розробка рівня на Unreal Engine

Створення проекту у Unreal Engine - це перший та ключовий крок у розпочатку роботи над ігровою концепцією. При входженні в Unreal Engine з’явиться вікно "Launcher", де вибирається "New Project" для створення нового проекту. Після вибору "New Project", обирається тип проекту з різноманітних шаблонів, які Unreal Engine надає. Вони включають шаблони для ігор, архітектурних візуалізацій, анімацій та інших застосувань. Далі необхідно обрати шаблон для проекту. Наприклад, "First Person", "Third Person", "Top Down", тощо. Вибір шаблону залежить від типу гри чи застосування, яке створюється (рис. 4.5).

Рисунок 4.5 – Шаблони проекту, для проекту обирався Blanc

Наступний етап – визначення параметрів проекту, таких як назва проекту, розташування на диску, мови програмування (C++ або Blueprints), та рівня деталізації графіки. Рівні деталізації графіки: "Maximum Quality" для проектів з високою якістю графіки або "Scalable 3D or 2D" для менш ресурсомістких проектів (рис. 4.6).

Рисунок 4.6 – Налаштування параметрів проекту

Останній крок натиснути кнопку "Create Project". Unreal Engine розпочне створення нового проекту з обраними параметрами та шаблоном. Після створення проекту потрапляємо в робоче середовище Unreal Engine. Тут взаємодіють з об'єктами, матеріалами, світлом, кодом та багатьма іншими елементами для реалізації своїх ідей.

4.3 Огляд моделювання рівня та імпорт 3D моделей

Зазвичай доводиться покладатися на контент, який створюється в окремих сторонніх додатках та імпортується у проект за допомогою інструментів, що надаються в редакторі Unreal (рис. 4.7). Наприклад, ці типи контенту зазвичай включають 3D-геометрію (звану статичними сітками в Unreal Engine), текстури, матеріали, аудіофайли тощо.

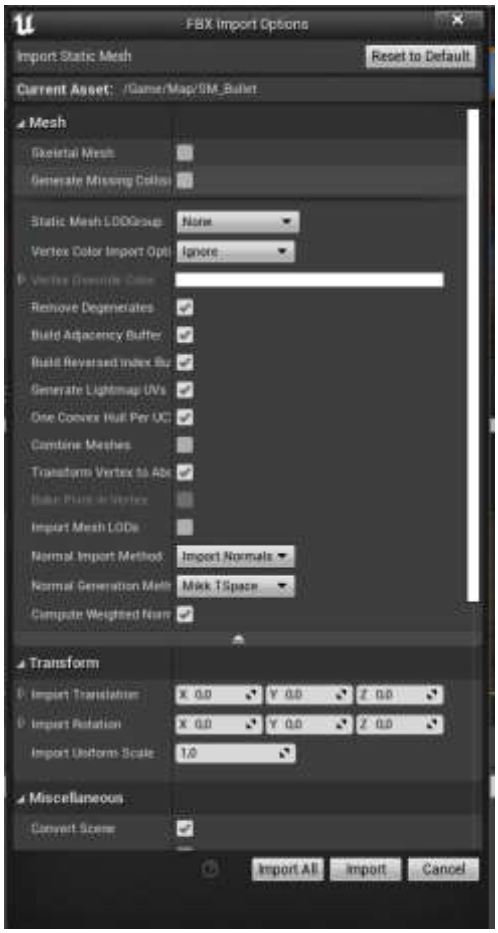

Рисунок 4.7 – Вікно імпорту моделі у двигун

Після імпортування всіх моделей, створюються матеріали. У редакторі матеріалів налаштовуємо текстури, що імпортували разом зі static meshes (рис. 4.8).

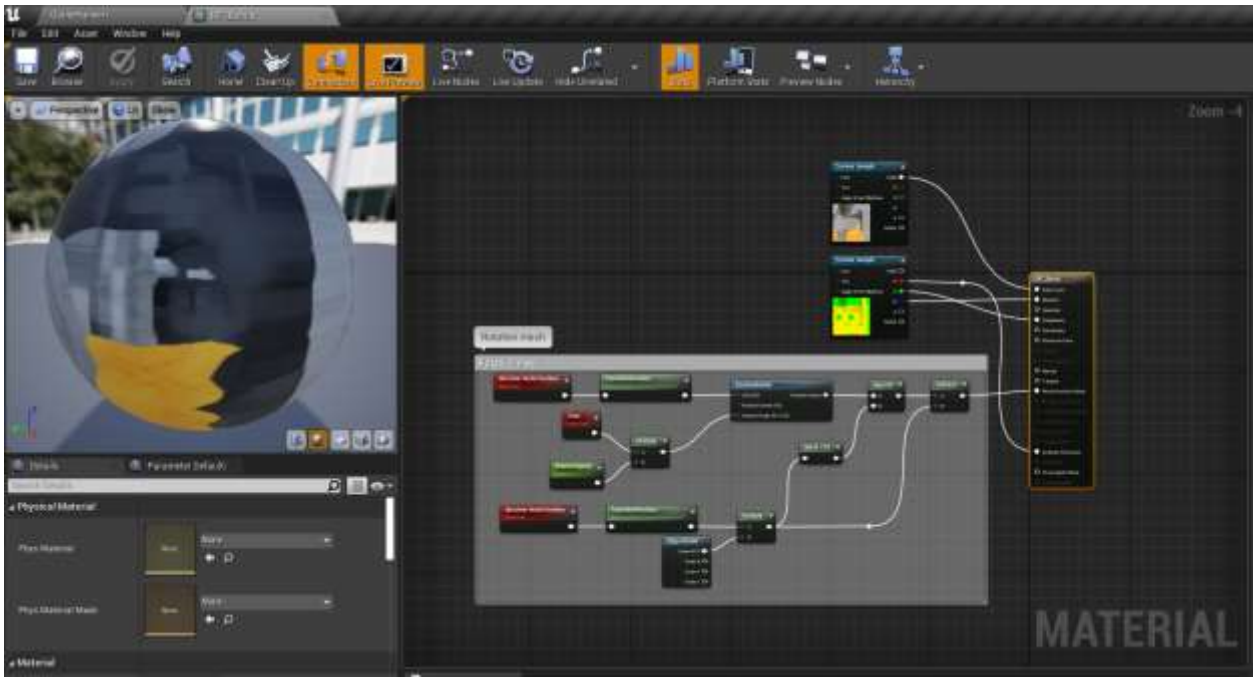

Рисунок 4.8 – Приклад налаштування матеріалів

Після налаштування матеріалів відбувається налаштування освітлення на сцені. Для цього у вкладці "World Settings" у Unreal Engine надаються зручні інструменти для налаштування освітлення та атмосфери, дозволяючи створювати потрібний настрій та вигляд геймплею (рис. 4.9). Основні налаштування: сонце та штучне світло, атмосферні ефекти, тіні та скайлайт.

Кожен елемент освітлення має свої характеристики, наприклад інтенсивність освітлення, вибір типу освітлення (статичній, динамічний) та інші налаштування. Основні налаштування світла на сцені: Sun Height (регулює висоту сонця на небі, визначаючи його напрямок та інтенсивність освітлення), Sky Light (включає або виключає "Sky Light" для створення динамічного освітлення від неба), Fog Density (визначає густоту туману в грі. Збільшення цього параметра створить більший ефект туману), Atmosphere (дозволяє налаштовувати параметри атмосфери, такі як колір та щільність, для досягнення бажаного ефекту), Cascaded Shadow Maps (CSM) (обирайте тип

тіней для сонця. CSM найчастіше використовується для реалістичних тіней великого масштабу), Lightmass Importance Volume (створює об'єм, в якому важливо розраховувати освітлення для покращення продуктивності), Global Light Color (змінює колір освітлення у всьому світі гри), Global Atmosphere (налаштовує глобальні параметри атмосфери для створення унікальної атмосфери).

Рисунок 4.9 – Налаштування освітлення на сцені

У розділі розглянуто та показано імпорт тривимірних моделей у Unreal Engine. Для імпорту показано налаштування правильного відображення об'єкта після завантаження. Продемонстровано приклад матеріалу та його відображення у потужному графічному редакторі. Розглянуто інструменти роботи з глобальним освітленням та показано приклад відображення освітлення без матеріалів для коректного налаштування.

4.4 Розробка алгоритмів та логіки механік гри у Unreal Engine

Для розробки логіки гри було обрано скриптинг за допомогою візуального редактора Blueprints, який є потужним інструментом у Unreal Engine, що дозволяє створювати логіку гри без необхідності писати код. Blueprint - це візуальна система програмування, що базується на графічних блоках, які представляють різні функції та операції.

4.4.1 Розробка пересування і маніпуляцій взаємодії гравця

У контейнері гри створюємо "Character", який відповідатиме персонажу, яким керуватиме гравець. Цей Blueprints складається з основної логіки, де реалізовано логіку реплікації, реалізовано клас, що працює зі скелетом моделі, і останній компонент пересування та взаємодії у світі (рис. 4.10).

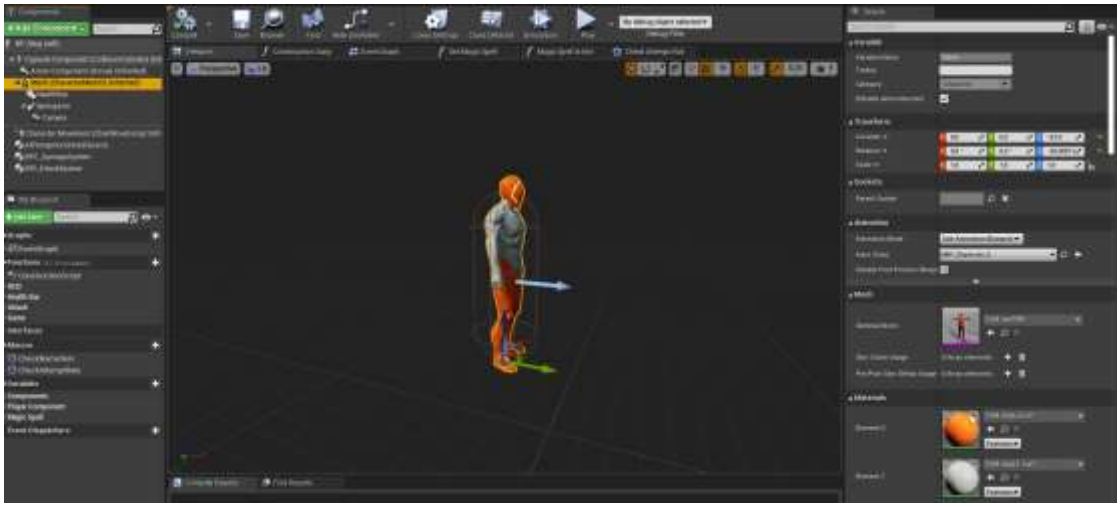

Рисунок 4.10 – Налаштування класу "Character"

Для керування цим персонажем потрібно створити "Player Controller", який є частиною архітектури керування персонажем у грі. Player Controller керує введенням від гравця, обробляє події введення, і керує персонажем або іншими компонентами гри (рис. 4.11).

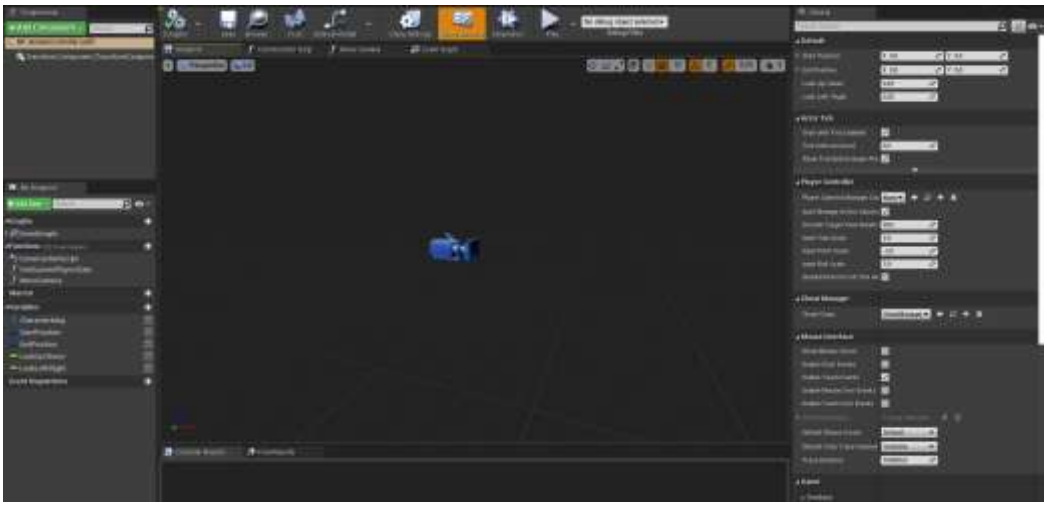

Рисунок 4.11 – Створення класу "Player Controller"

Щоб почати грати за персонажа, потрібно ще створити "Game Mode Base", це клас, який керує основною логікою гри. Він визначає правила гри, початкові умови та взаємодію між різними частинами ігрового світу. У нього перезначаємо створені Player Controller і Character, щоб рівень починався, використовуючи їх (рис. 4.12).

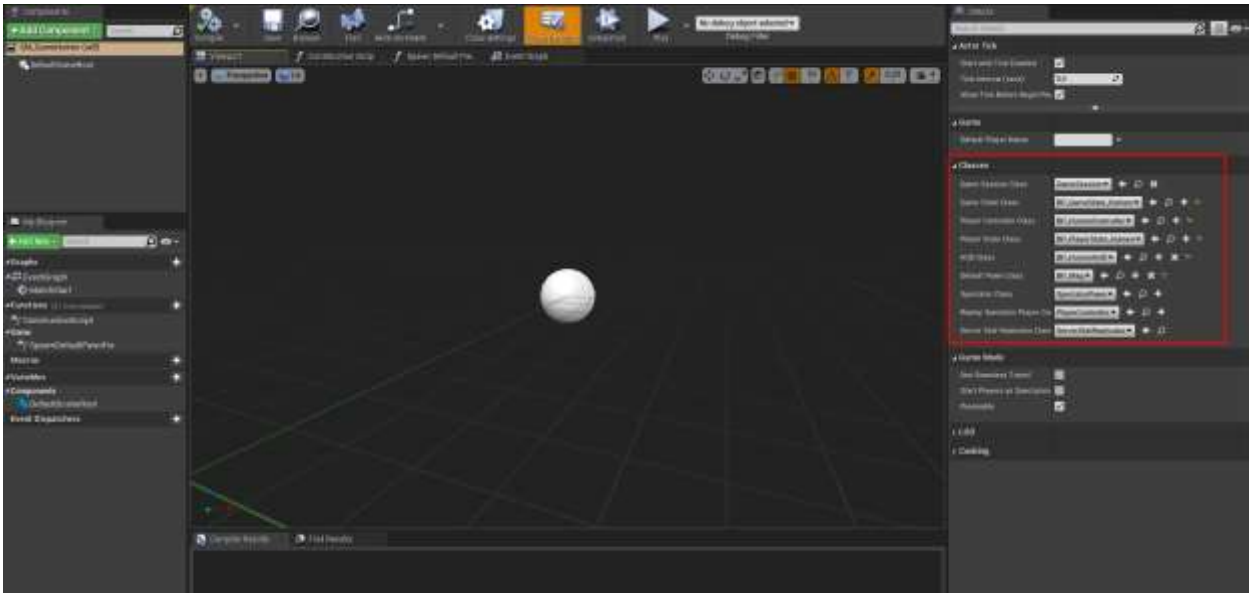

Рисунок 4.12 – Створення та переіначення класів у "Game Mode Base"

Після перших створінь класів персонаж зможе тільки з'явиться у світі, але руху для нього ще не прописано. Для пересування персонажа в

налаштуваннях "Project settings" у вкладці "Input", задати клас для джойстика і додаткові кнопки для стрибка та ініціалізації магічного снаряда (рис. 4.13).

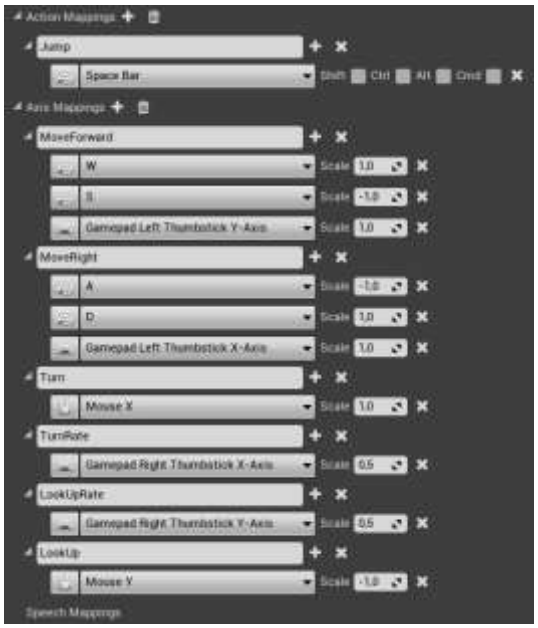

Рисунок 4.13 – Створення "Input" команд для маніпуляцій

Проініціалізувавши кнопки пересування, переходимо до класу Player Controller, де створюємо логіку взаємодії, для цього дістанемо Event пересування, прописані в Input, і отримаємо через функцію "Get Controlled Pawn" персонажа, яким володіє контролер, оскільки контролер може володіти безліччю персонажів одночасно, наприклад, в онлайн грі, алгоритм праці на рис. 4.14. З функції отримання персонажа потрібно отримати функцію пересування, в неї передамо з Event змінну напрямку (рис. 4.15).

Рисунок 4.14 – Алгоритм праці пересування гравця

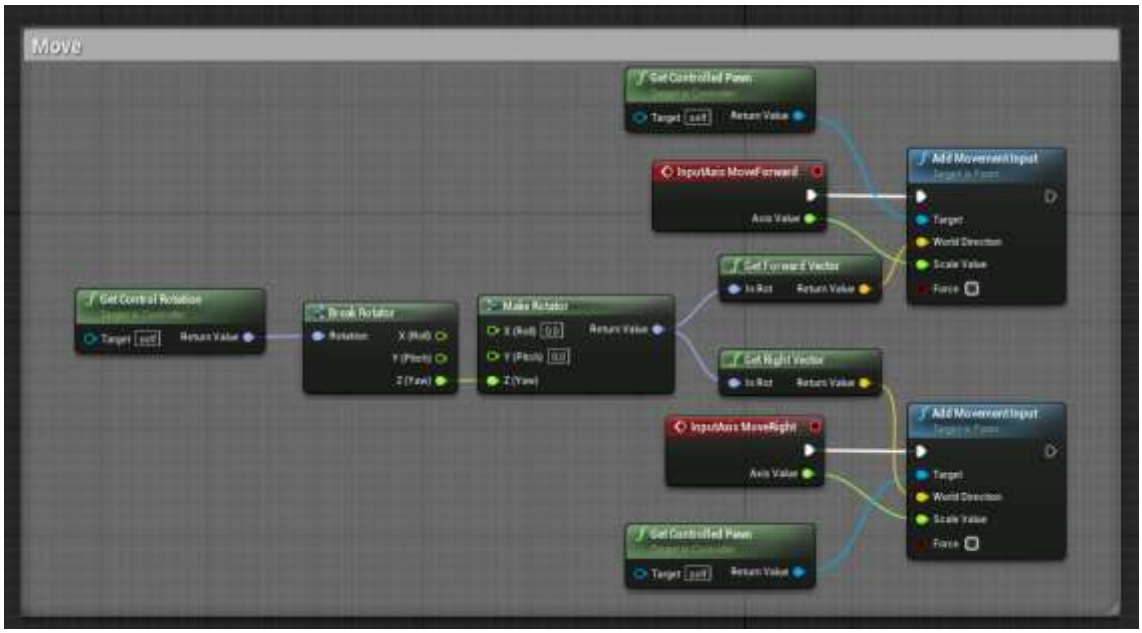

Рисунок 4.15 – Створення механіки руху персонажа у Player Controller

Проініціалізуємо розворот камери персонажа в тому ж Player Controller, все так само потрібно дістати Event, і функцію отримання персонажа, з якого дістаємо "Pitch" і "Yaw", проте додатково потрібно прорахувати пересування

пальця по екрану, щоб здійснити це, потрібно створити дві змінні 2D-вектора, які зберігатимуть і перезаписуватимуть початкове положення і напрямок пальця по екрану (рис. 4.16), алгоритм праці на рис. 4.17.

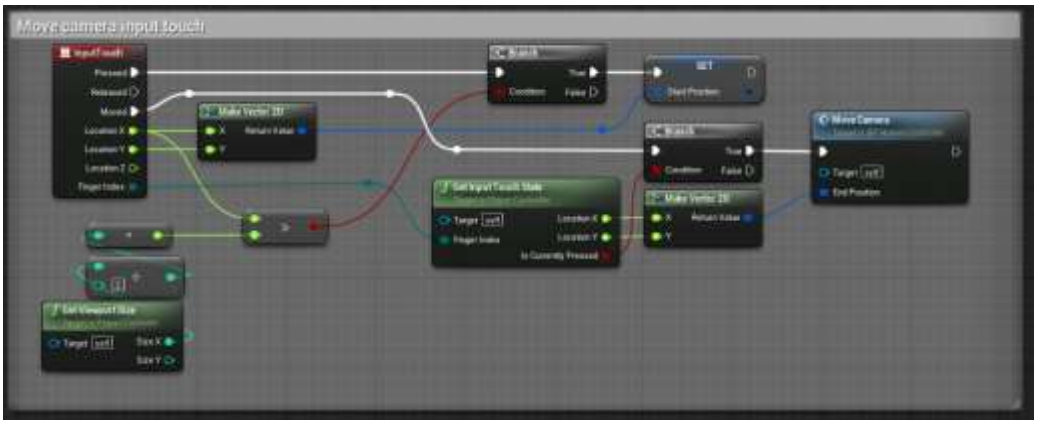

Рисунок 4.16 - Створення механіки руху камери у Player Controller

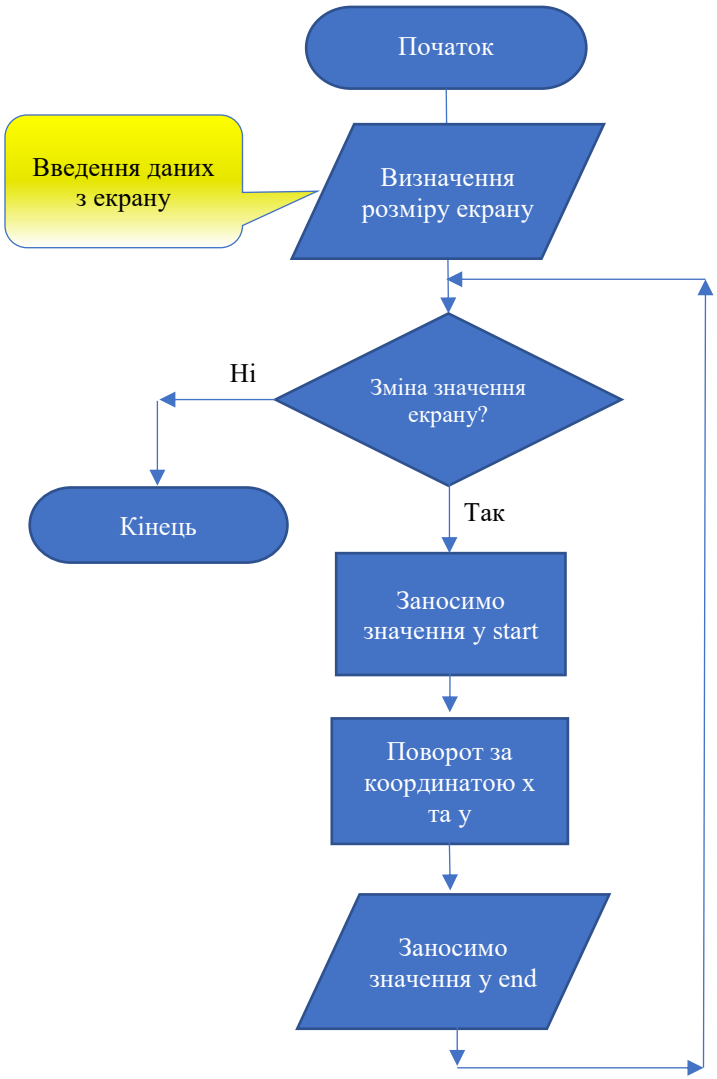

Рисунок 4.17 – Алгоритм праці переміщення обзору за допомогою екрану

Робота стрибка і стрільби магічною кулею схожі, і майже однаково реалізуються. Для цього дістаємо Event кнопки з віджета екрана, перевіряємо, чи існує персонаж у контролера, і викликаємо необхідну функцію (рис. 4.18), алгоритм праці на рис. 4.19.

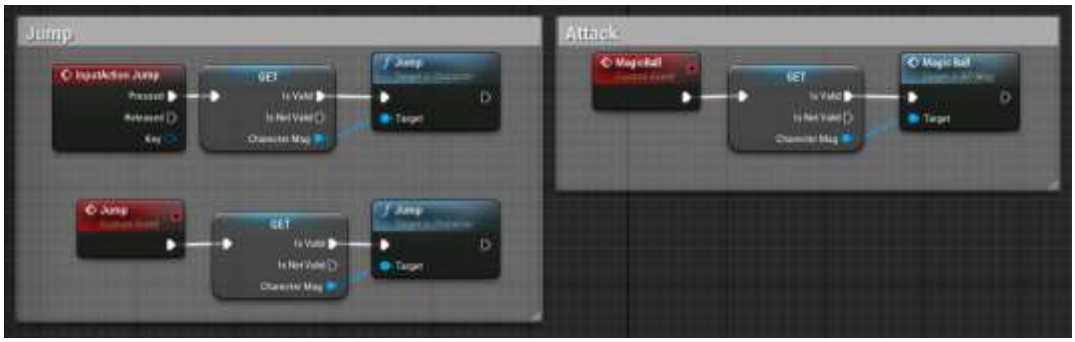

Рисунок 4.18 - Створення механіки стрибку та стрільби персонажа у Player Controller

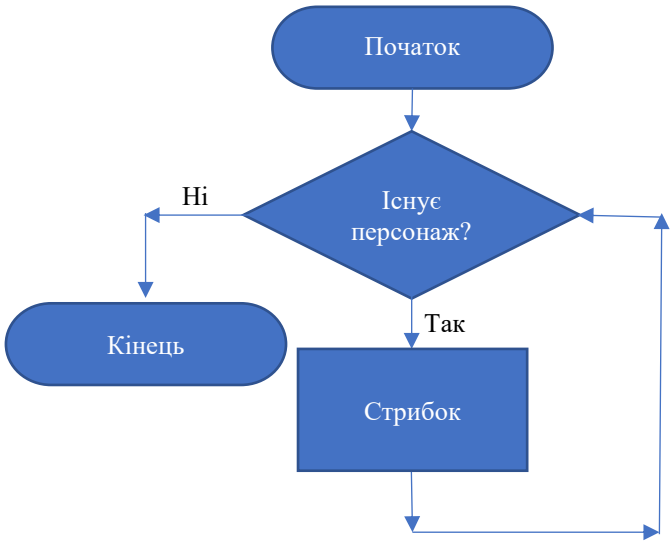

Рисунок 4.19 – Алгоритм праці стрибку, атаки гравця

Персонаж уміє рухатися, стрибати і стріляти магічними кулями (реалізації стрільби ще немає, але кнопка стрільби працює), однак не вистачає анімації щоб оживити маріонетку. Для цього в ігровий рушій завантажуються анімації взяті з Mixamo. Будуть потрібні анімації ходьби, стану спокою і стрибка на перший час (рис. 4.20).

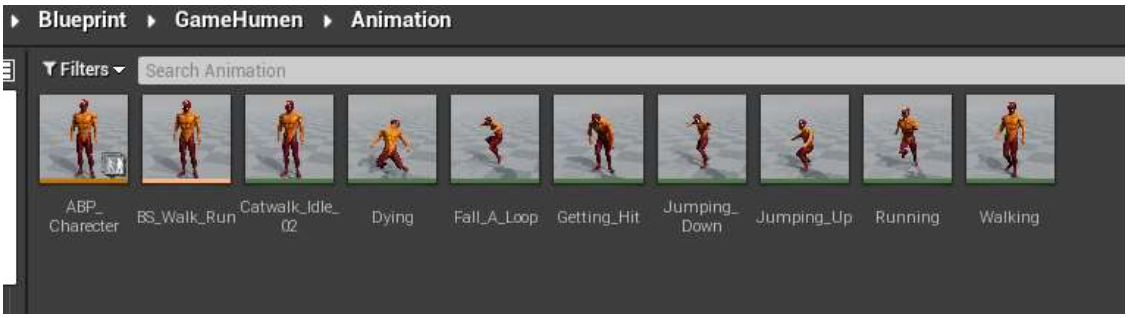

Рисунок 4.20 – Завантажені анімації з Mixamo

Далі потрібно створити спеціальний Blueprints, що відповідає за змішування і перехід з анімації. У ньому створюємо спеціальну "State machine", в якій створюємо умови переходів від анімації до іншої (рис. 4.21).

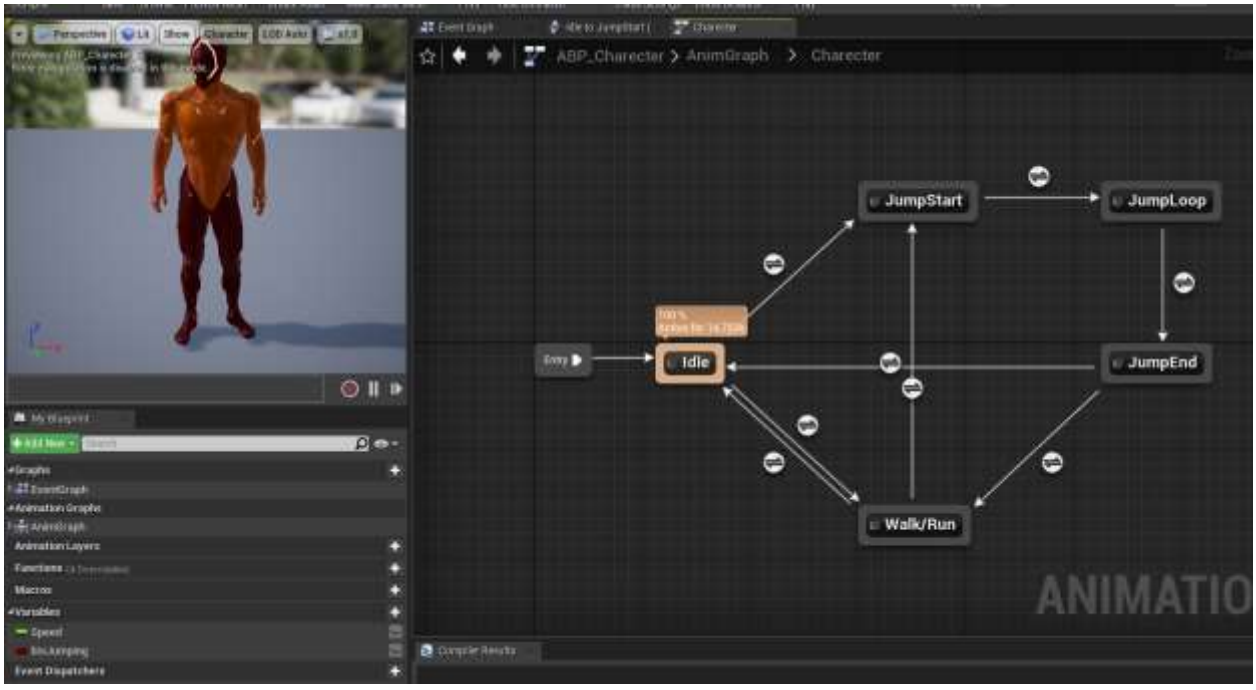

Рисунок 4.21 – Умови переходів анімації

У зв'язках переходів вказується логічна взаємодія змінних, за яких має виконатися умова. Оновлення змінних відбувається в основному графі анімаційного Blueprints (рис. 4.22), алгоритм праці на рис. 4.23.

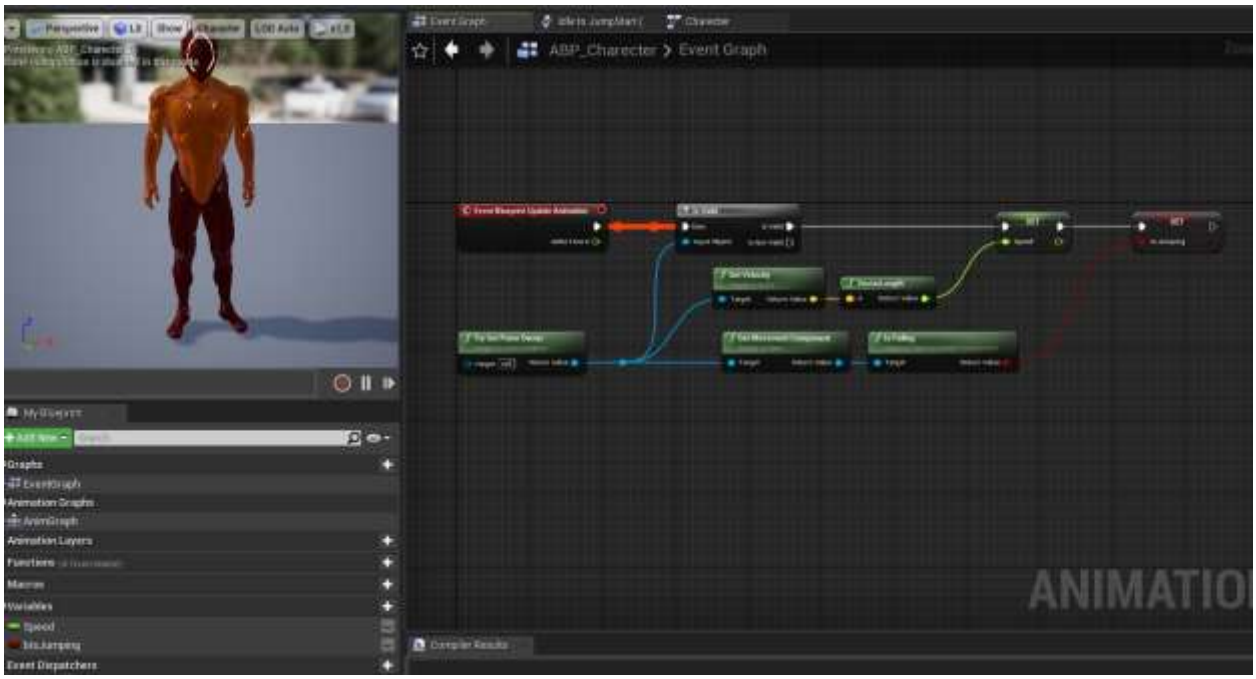

Рисунок 4.22 – Граф змін значення перемінних у анімаційного Blueprints

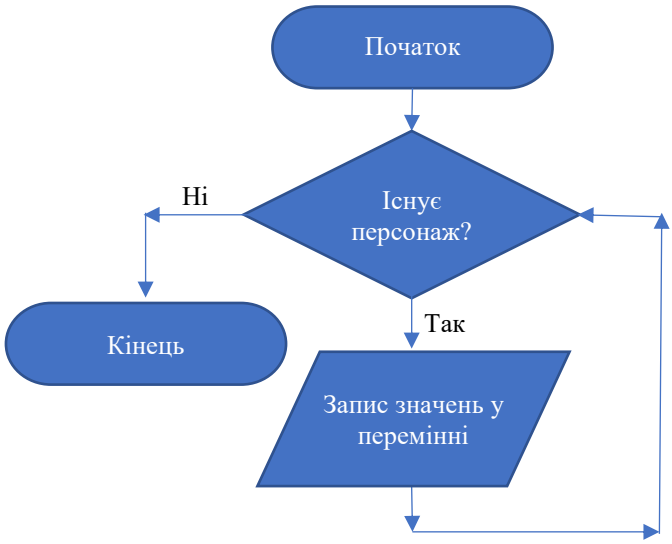

Рисунок 4.23 – Алгоритм праці анімаційного Blueprints

Встановивши послідовність перемикання анімацій та їхню взаємодію між собою, переходимо до створення атаки гравця, що надалі може розширити анімаційні можливості персонажа.

4.4.2 Створення атаки гравця

Щоб створити атаку гравця і надалі передати схоже штучному інтелекту, створюється "Actor Component". Він являє собою перевикористовувану частину функціональності, яку можна додати до акторів (Actors). Дозволяє створювати модульні та повторно використовувані блоки логіки, які можна додавати до різних акторів у грі.

Перш ніж починати реалізовувати перевикористовувану частину, потрібно створити основні функції, які викликатимуться у персонажів, але щоб інтерпретація функцій відрізнялася на відміну від назв. У цьому допоможе інтерфейс (Blueprint Interface), це спосіб забезпечення зв'язку між різними Blueprints без необхідності використання успадкування. Це дає змогу об'єктам різних типів взаємодіяти один з одним, не будучи спадкоємцями одного й того самого класу (рис. 4.24).

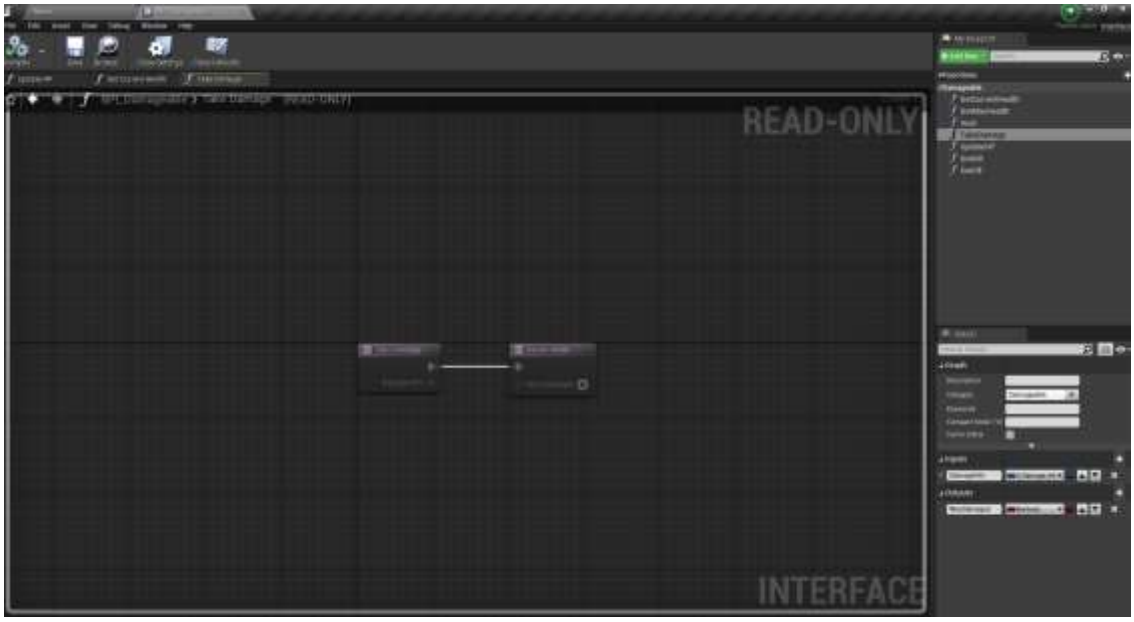

Рисунок 4.24 – Створення інтерфейсу для атаки

Також створюється структура, що складається з типу завданої шкоди, стану після завданої шкоди, кількості шкоди, чи може магічна куля переслідувати тощо (рис. 4.25).

Рисунок 4.25 – Створення структури даних

Реалізувавши структуру та інтерфейс, переходимо до реалізації логіки взаємодії. Реалізуємо основні функції отримання здоров'я, нанесення шкоди та перевірки. Створимо в компоненті два основних виклики, нанесення шкоди і смерть персонажа, коли здоров'я переходить позначку 0 (рис. 4.26), алгоритм праці нанесення шкоди персонажу в алгоритмі на рис. 4.27.

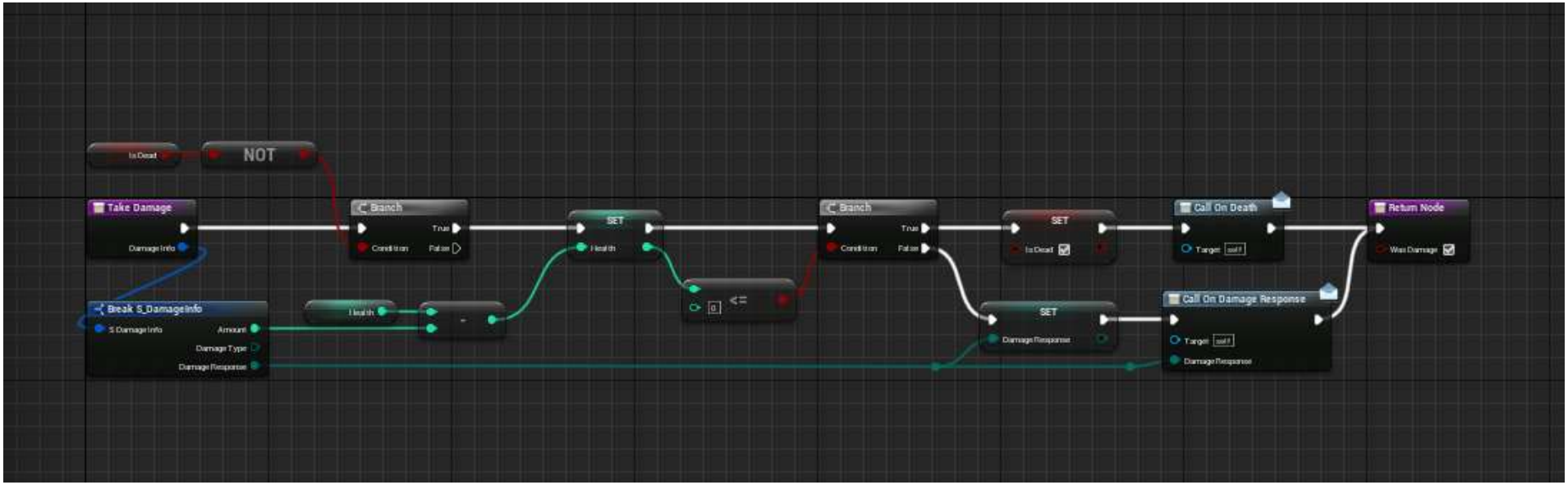

Рисунок 4.26 – Реалізація функцій з інтерфейсу у компоненті

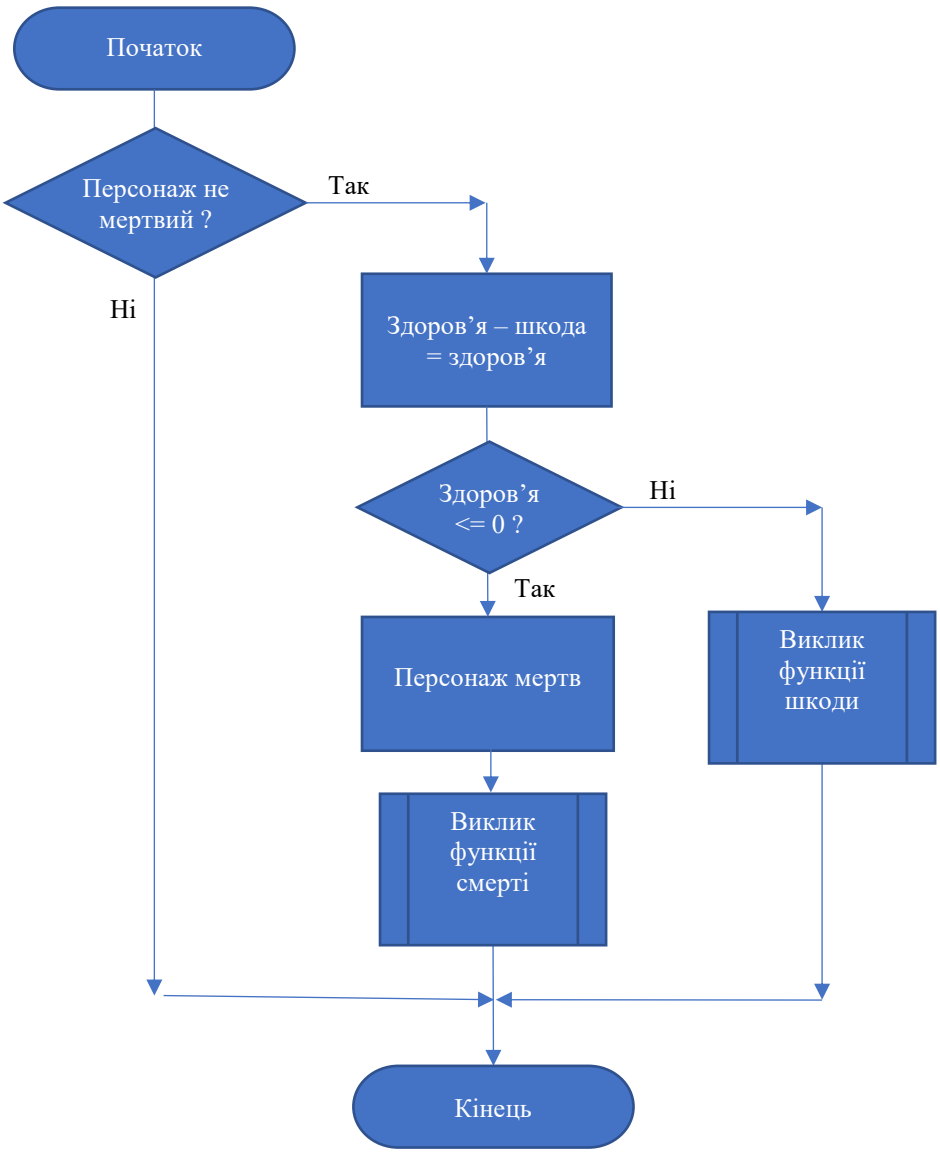

Рисунок 4.27 – Алгоритм нанесення шкоди персонажу

Отриманий компонент додаємо до основного персонажа гри, і додаємо раніше створений інтерфейс. У персонажі ініціалізуємо інші функції з інтерфейсу, що не були в компоненті (рис. 4.28).

Рисунок 4.28 – Ініціалізація функцій інтерфейсу у персонажі

Персонаж вміє наносити й отримувати шкоду, проте йому поки нічим її наносити, щоб реалізувати цей момент, потрібно створити "Actor". Він є базовим класом для всіх об'єктів у рівні. Сюди входять ігрові персонажі, об'єкти оточення, світло і багато іншого. Створивши базовий клас для магічної кулі, потрібно додати компоненти: статичної сітки, системи частинок, компонент руху снаряда (рис. 4.29).

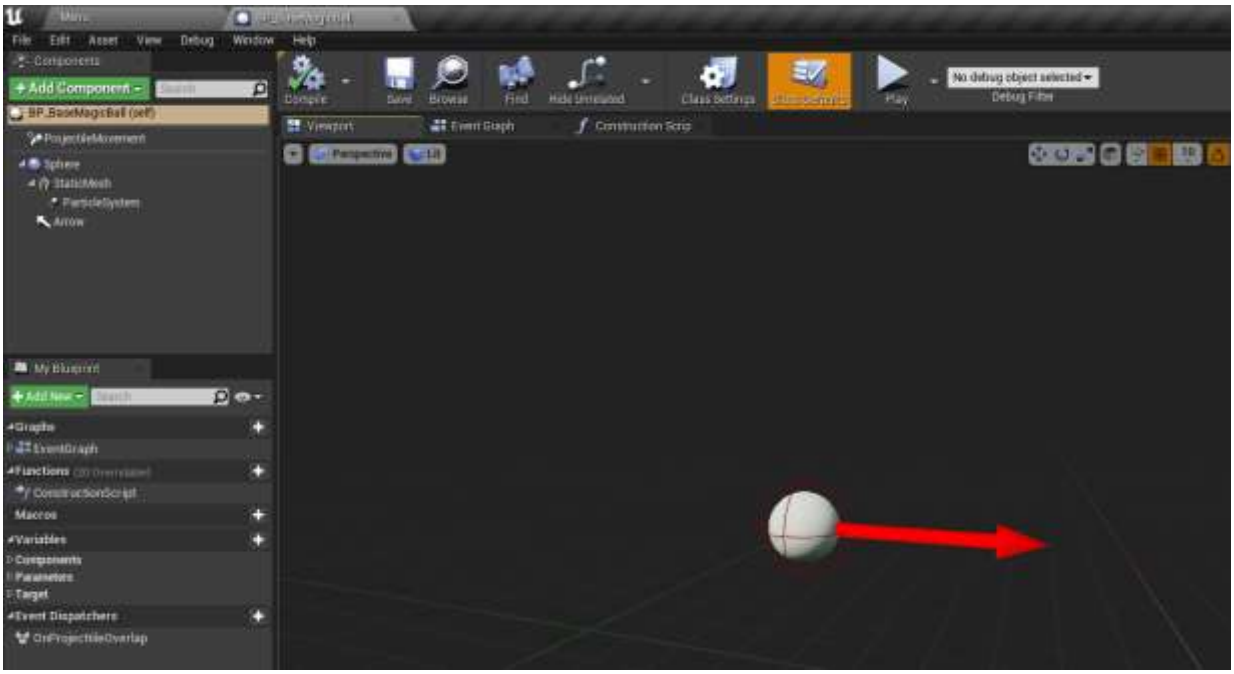

Рисунок 4.29 – Створення базового класу магічного снаряду

У конструкторі класу реалізуємо швидкість снаряда, шкоду, рух за ціллю, тип заряду, тип шкоди (рис. 4.30). В основному графі реалізується логіка пострілу, і створюється виклик у разі зіткнення з предметом, щойно зіткнення відбулося, додається ефект. Також під час пострілу магічною кулею, снаряд запам'ятовує персонажа, що вистрілив, щоб не завдати шкоди надалі під час руху (рис. 4.30).

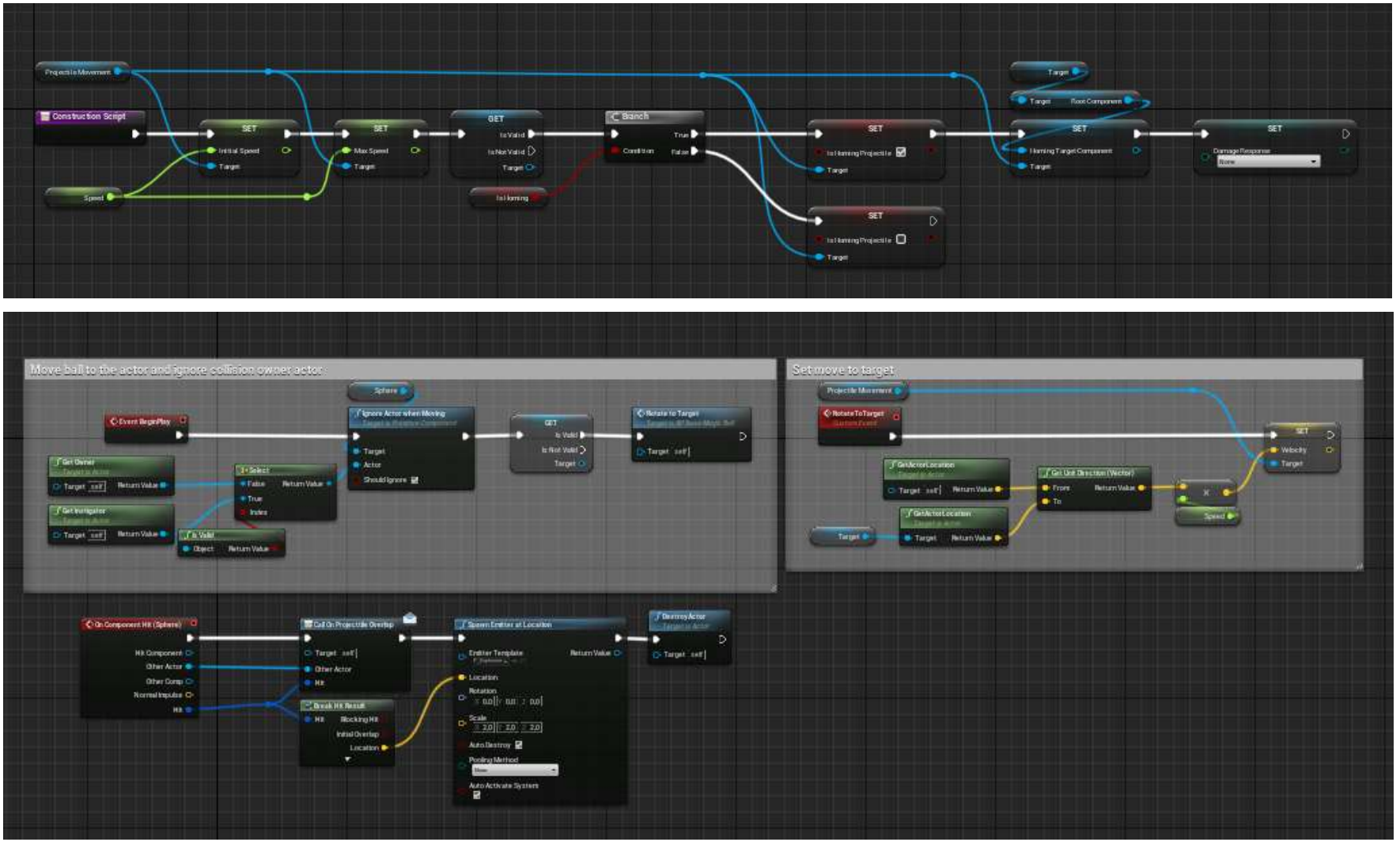

Рисунок 4.30 – Реалізація снаряду у конструкторі та графі класу

Від основного магічного снаряда створюємо "дочірні" класи, в яких реалізуються окремо вигляд і основні властивості (рис. 4.31).

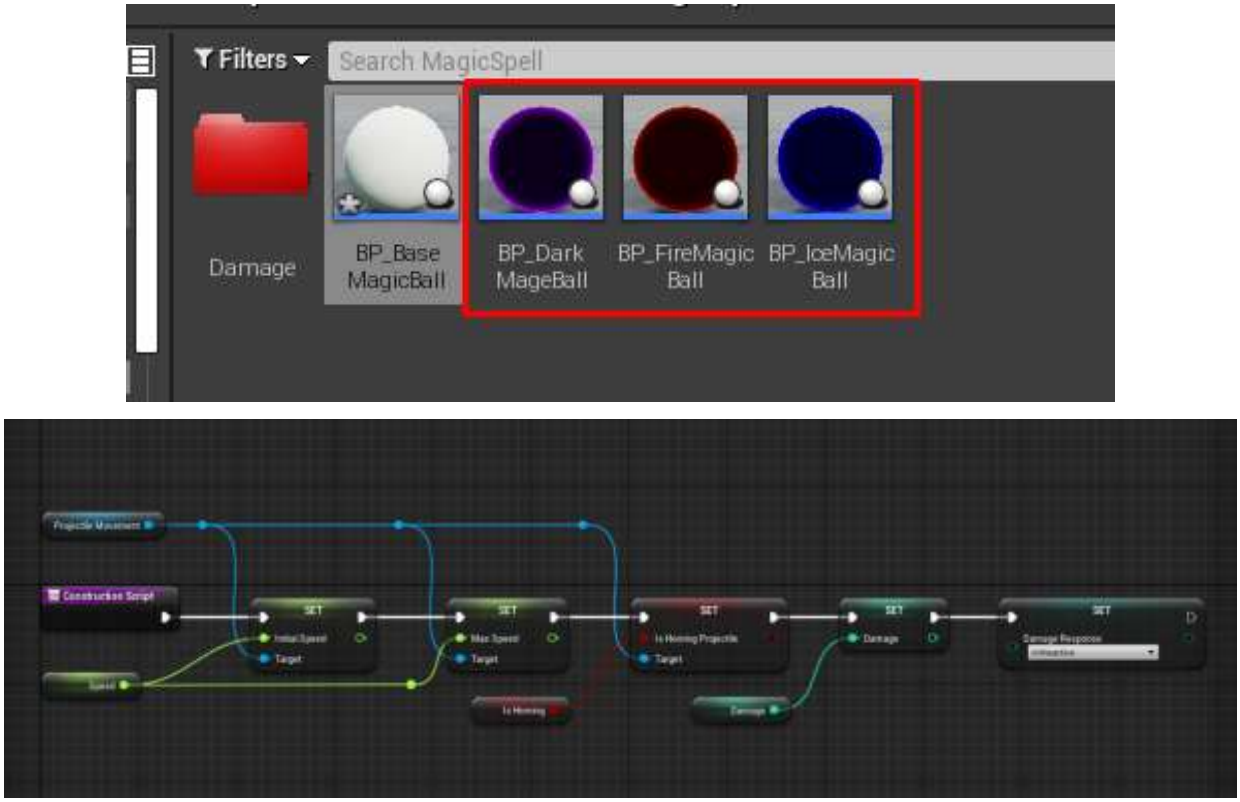

Рисунок 4.31 – Успадкування від базового класу снаряда

Реалізувавши снаряди, додаємо логіку пострілу його у персонажа, для цього використовуємо кнопку з віджету, що реалізували під час розробки маніпуляції персонажем (рис 4.32). Для цього потрібно взяти точку появи снаряда, вектор напрямку, і точку напрямку камери, також передати акторів, які ігноруватимуться при утворенні магічної кулі, і останнім параметром передати який снаряд буде утворений (рис. 4.33).

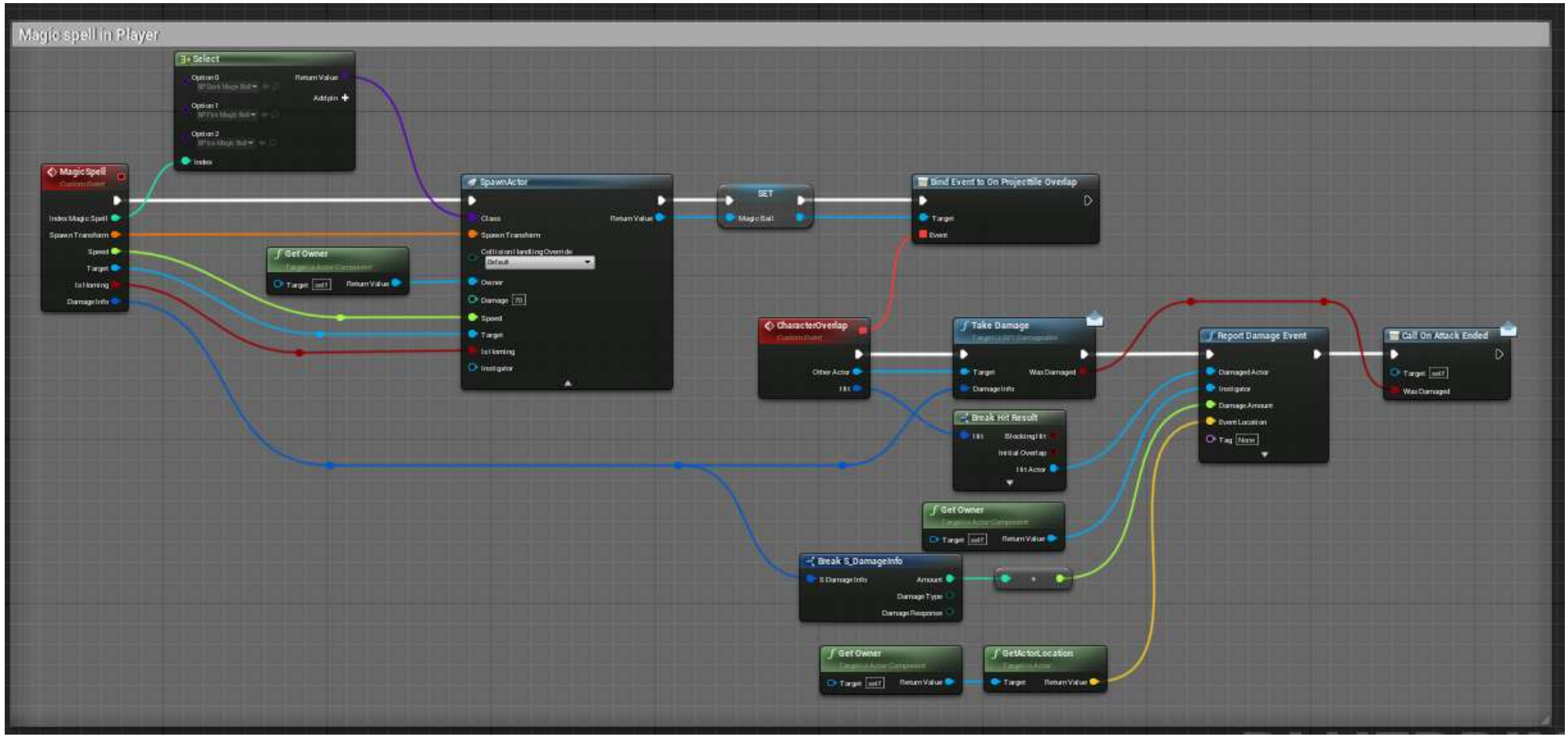

Рисунок 4.32 – Реалізація утворення снаряду при атаці

Рисунок 4.33 – Алгоритм створення снаряду та його взаємодія

Провівши всі маніпуляції, одержуємо величезну систему можливостей створення снарядів із різними параметрами, величезну систему нанесення шкоди та вибір завданої шкоди, що надалі дасть змогу обирати стан персонажа під час отримання певного типу шкоди. А головне цю систему можна розширювати і доповнювати новим функціоналом, і вона буде працювати на кожному персонажі так, як це реалізує програміст.

4.4.3 Створення підбору об’єктів на рівні гравцем

Реалізація підбору предметів починається зі створення базового класу "Actor", де додаються об'єкти спавна. Об'єкти мають з'являтися рандомно і через певний час змінюватися, і кожен об'єкт має мати свої властивості під час підбору.

Найкращим рішенням є створити таблицю з даними про об'єкти, дані зберігаються в структурі та змінних. Така таблиця називається "Data table", до неї заносяться дані про всі об'єкти, які підніматиме персонаж (рис. 4.34). Це збільшить можливість гнучко додавати нові компоненти для підняття.

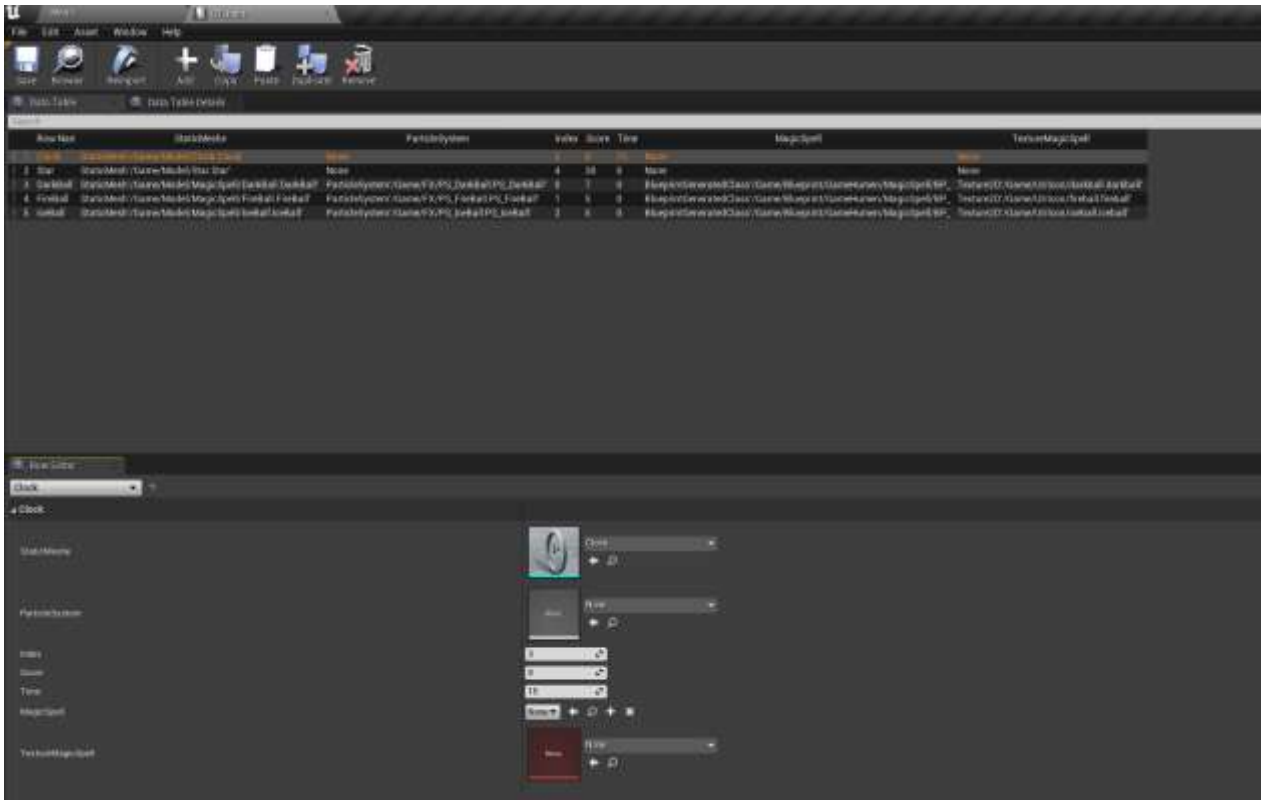

Рисунок 4.34 – Реалізація "Data table"

Після реалізації таблиці даних, ініціалізуємо логіку появи моделей, підняття і перевизначення об'єкта через час (рис. 4.35). Створимо пару Event для рандомної появи і перезаповнення. Після задаємо таймер і підключимо Event появи об'єктів, приклад праці у алгоритмі на рис. 4.36.

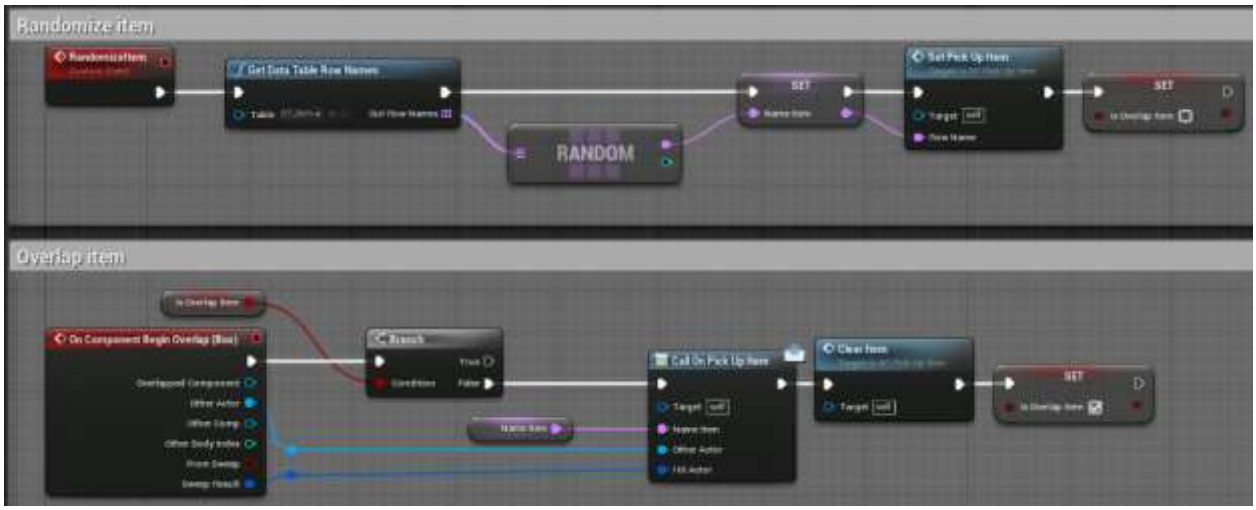

Рисунок 4.35 – Реалізація "Actor" для підбору предметів

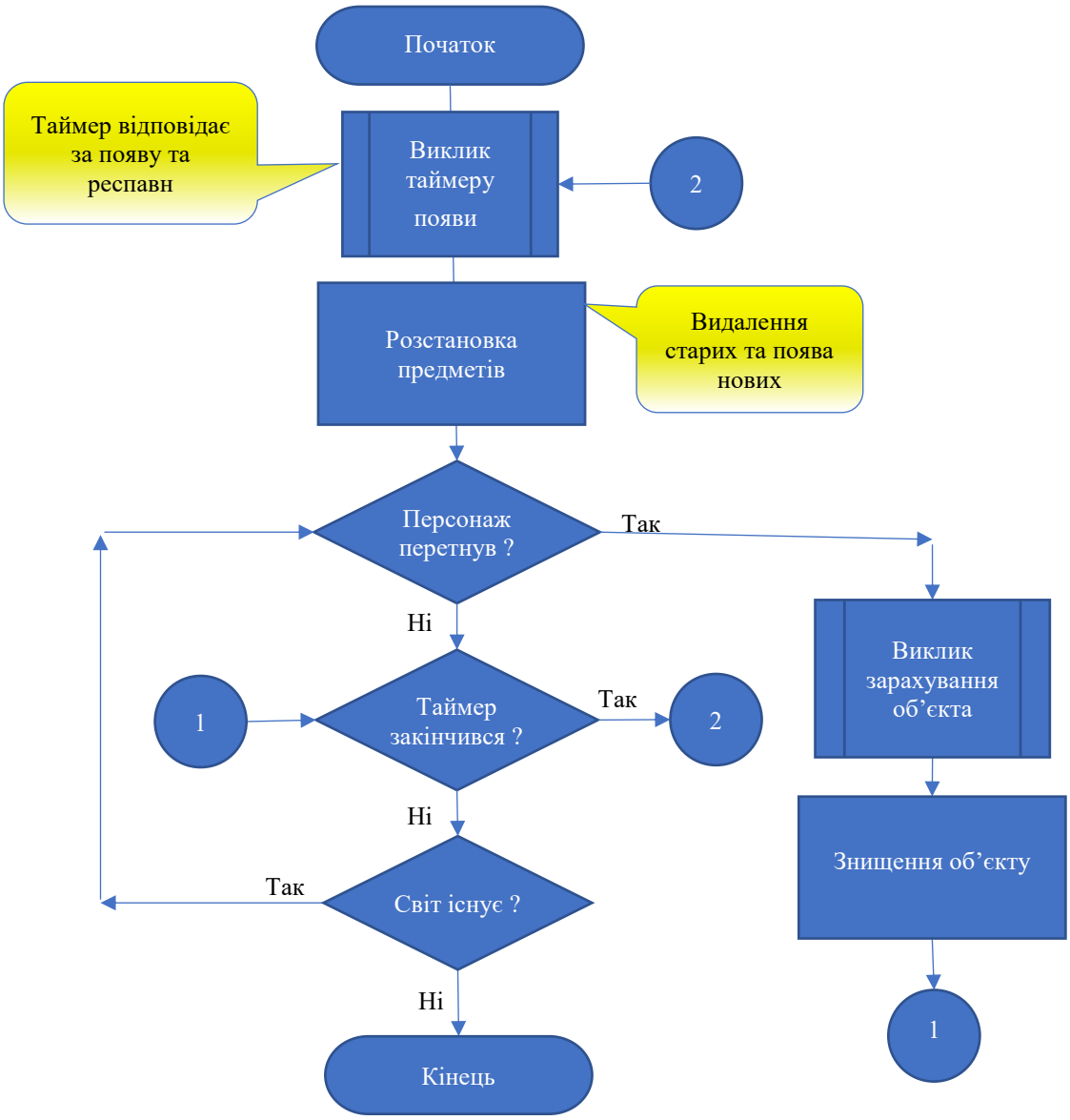

Рисунок 4.36 – Алгоритм появи та взаємодії з світом

Реалізувавши таку систему підбору, отримуємо гнучку систему додавання та отримання об'єкта персонажем, також є можливість впливати на гравців, які можуть отримувати певний об'єкт, і які дані отримуємо від отримання.

4.4.4 Створення штучного інтелекту на рівні

Щоб створити в Unreal Engine штучний інтелект, насамперед потрібно реалізувати спеціальний елемент "AI Player Controller". Це компонент, що керує поведінкою персонажів, керованих штучним інтелектом, у грі. Він є частиною ширшої структури штучного інтелекту, що надається Unreal Engine, і відповідає за процес ухвалення рішень, навігацію та взаємодію AI-персонажів [12].

Наступний крок, це створити персонажа, яким керуватиме AI Player Controller, для цього в Controller поміщаємо спеціальний компонент. У компоненті вибираємо спосіб сприйняття оточення, у грі вибираємо зір. Для зорового сприйняття існує Event, за якого відбувається сповіщення при знаходженні певного об'єкта (рис. 4.37).

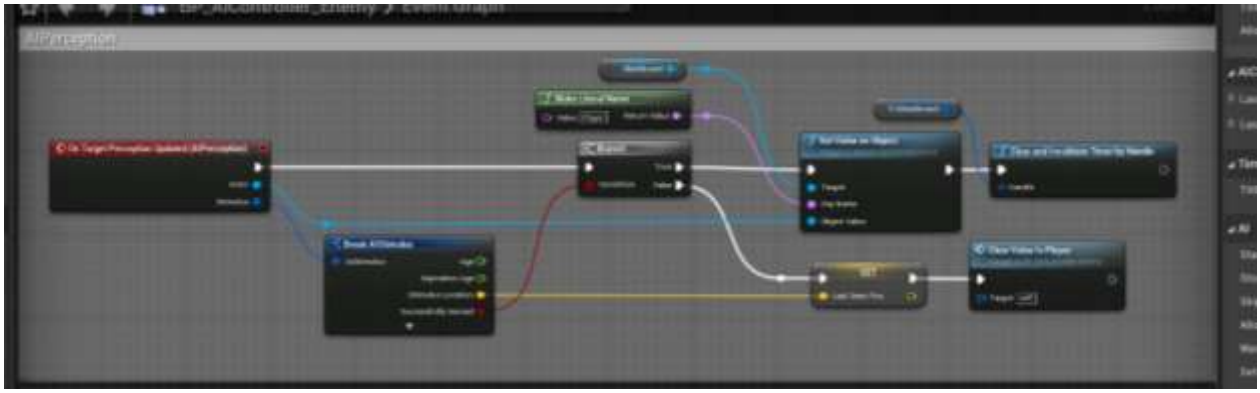

Рисунок 4.37 – Event зорового сприйняття

Щоб AI Player Controller зміг керувати персонажем, скопіюємо основного і в Class default перепризначаємо Controller, а в основному персонажі додаємо компонент, щоб AI Player Controller міг бачити головного героя, принцип схожий з датчиком, який ми вішаємо на героя (рис. 4.38).

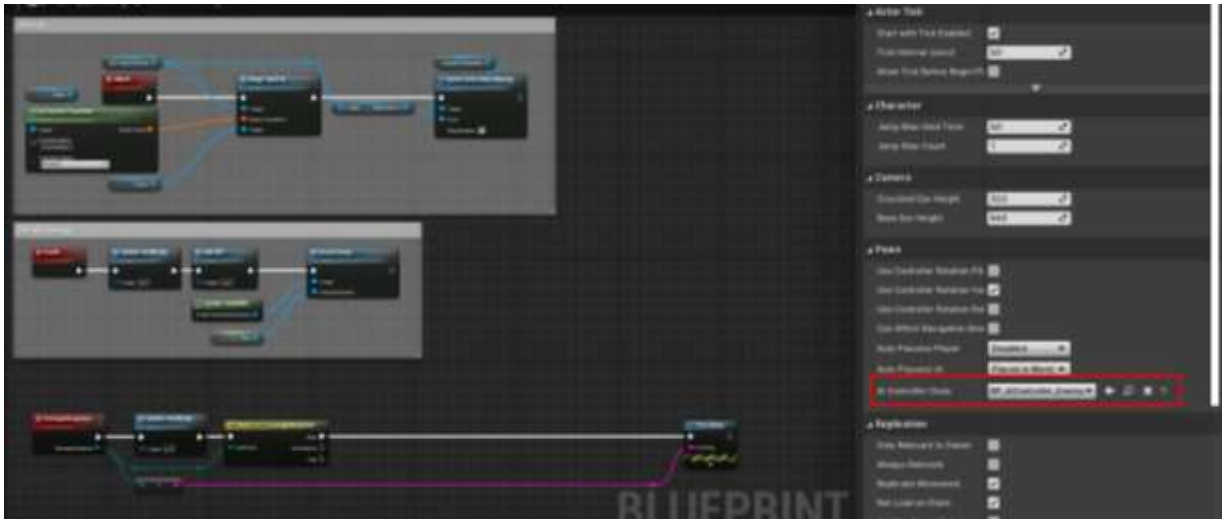

Рисунок 4.38 – Призначення AI Player Controller персонажу

Штучний інтелект тепер може керувати персонажем, однак він не знає, як це робити і які дії він повинен здійснювати в певних умовах. Щоб розв'язати таку проблему в ігровому рушії існує "Behaver Tree", його називають ще деревом поведінок. Це візуальний скриптовий інструмент, призначений для ієрархічної та модульної реалізації поведінки ШІ. Це потужна і гнучка система, що дає змогу розробникам і дизайнерам створювати складні моделі поведінки ШІ шляхом розташування вузлів у деревоподібній структурі (рис. 4.39).

Дерево поведінки працює в парі з дошкою "Blackboard". Вона є фундаментальним компонентом системи штучного інтелекту і зазвичай асоціюється з "деревами поведінки" (Behavior Trees). Дошка служить загальним сховищем даних, що забезпечує зв'язок і обмін інформацією між різними компонентами ШІ (рис. 4.40).

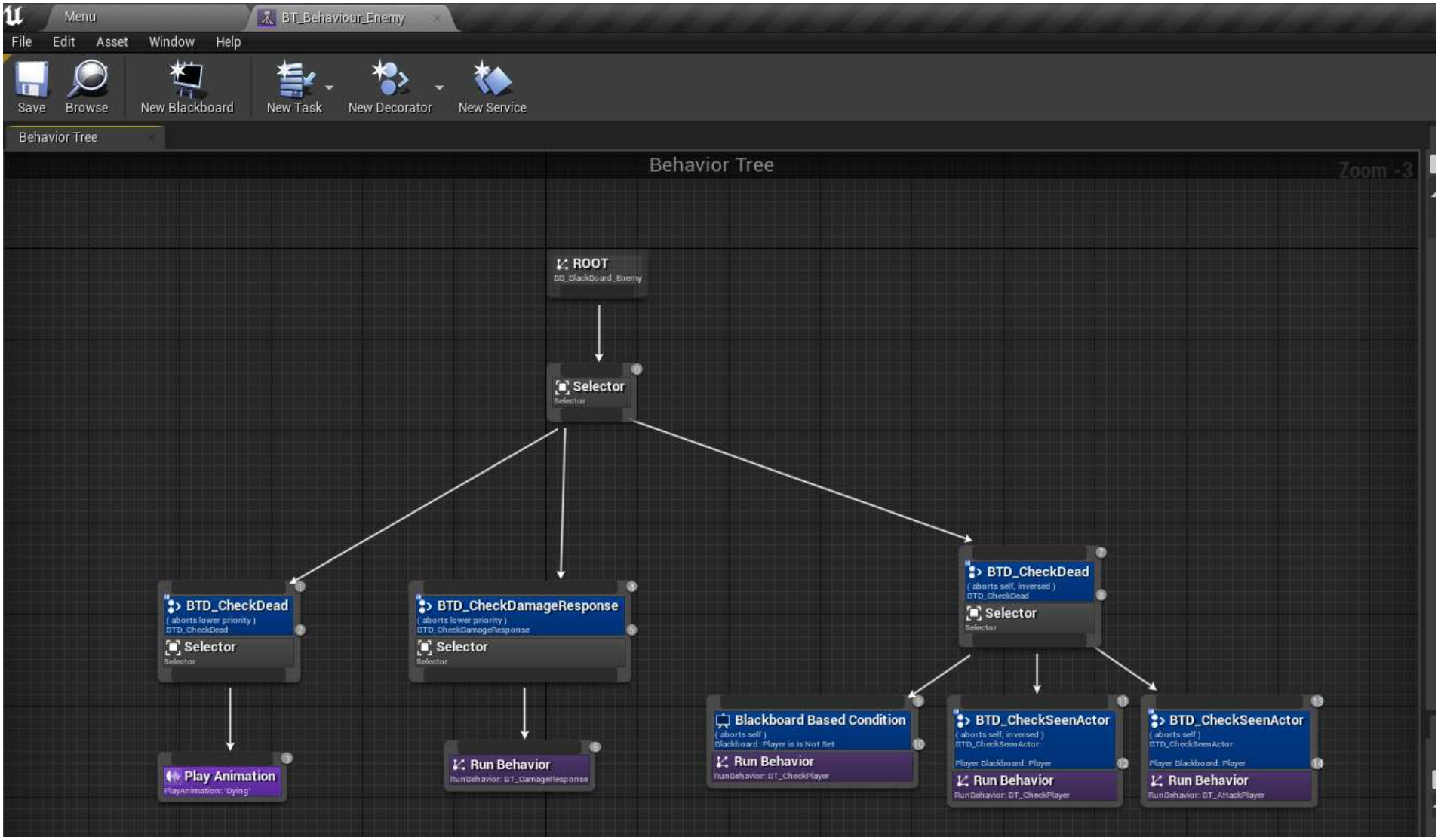

Рисунок 4.39 – Дерево поведінки "Behaver Tree"

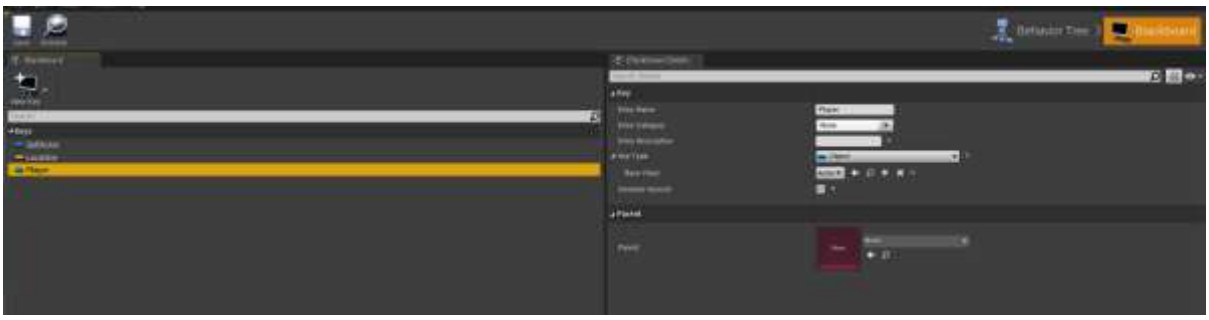

Рисунок 4.40 – Дошка "Blackboard"

Основними функціями ШІ будуть пошук, атака, переслідування головного героя. Для пошуку реалізується додаткове завдання генерації рандомних точок за рівнем, які можуть бути наближені до героя, рандомну точку записують у Blackboard, її передають у функцію пересування ШІ. Щойно персонаж підходить до випадково згенерованої точки він зупиняється на кілька секунд для огляду території (рис. 4.41).

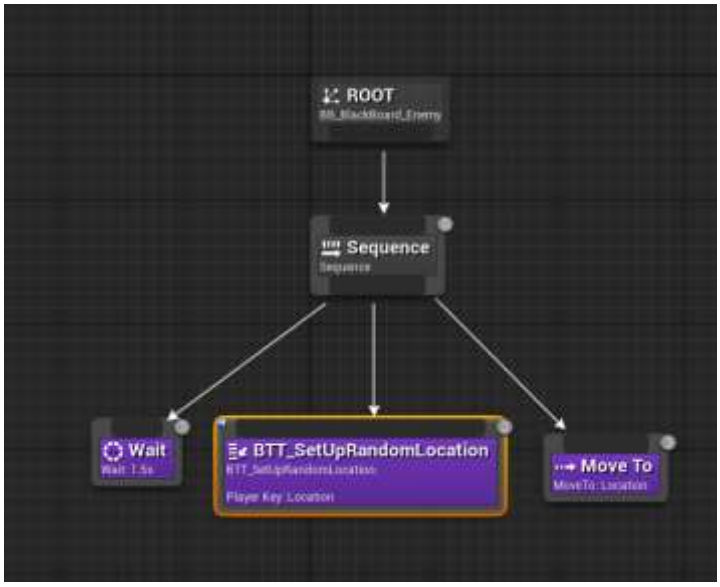

Рисунок 4.41 – Випадково генерована точка

Для функції пошуку потрібно реалізувати перевірку, якщо в Blackboard записаний герой, то пошук не відбувається, і другою умовою, якщо персонаж, загубив з поля зору головного героя і на останньому побаченим місці немає героя, починається "пошук" (рис. 4.42).

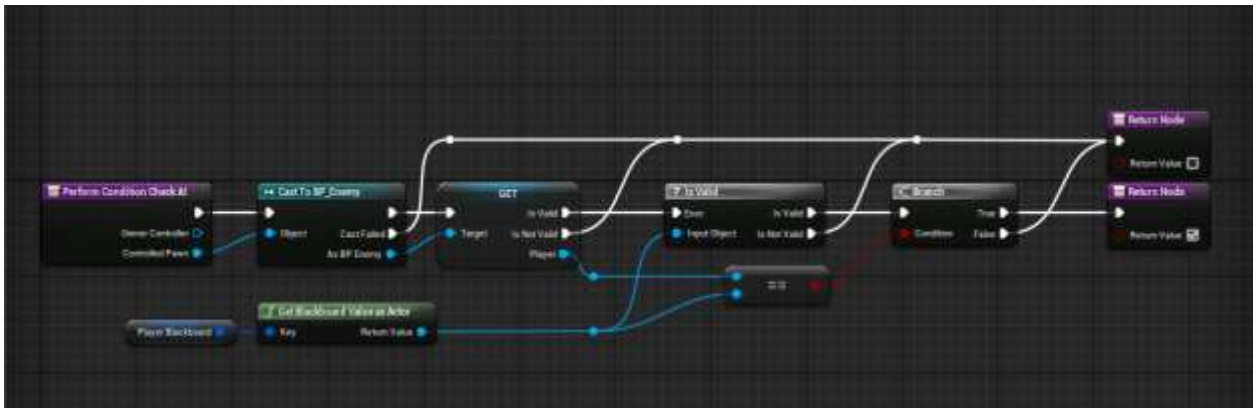

Рисунок 4.42 – Перевірка наявності персонажа у Blackboard

Функція пошуку обривається, щойно в Event приходить сигнал про знаходження героя, герой записується в Blackboard і починається функція переслідування. Під час переслідування відбувається дві дії: атака і ходіння за ним. Щойно ШІ втрачає з поля зору героя, Controller передає останнє місце розташування, де бачив, і підходить до цього місця (рис. 4.43). Якщо не знаходить героя, повертається до функції пошуку випадкових точок на карті.

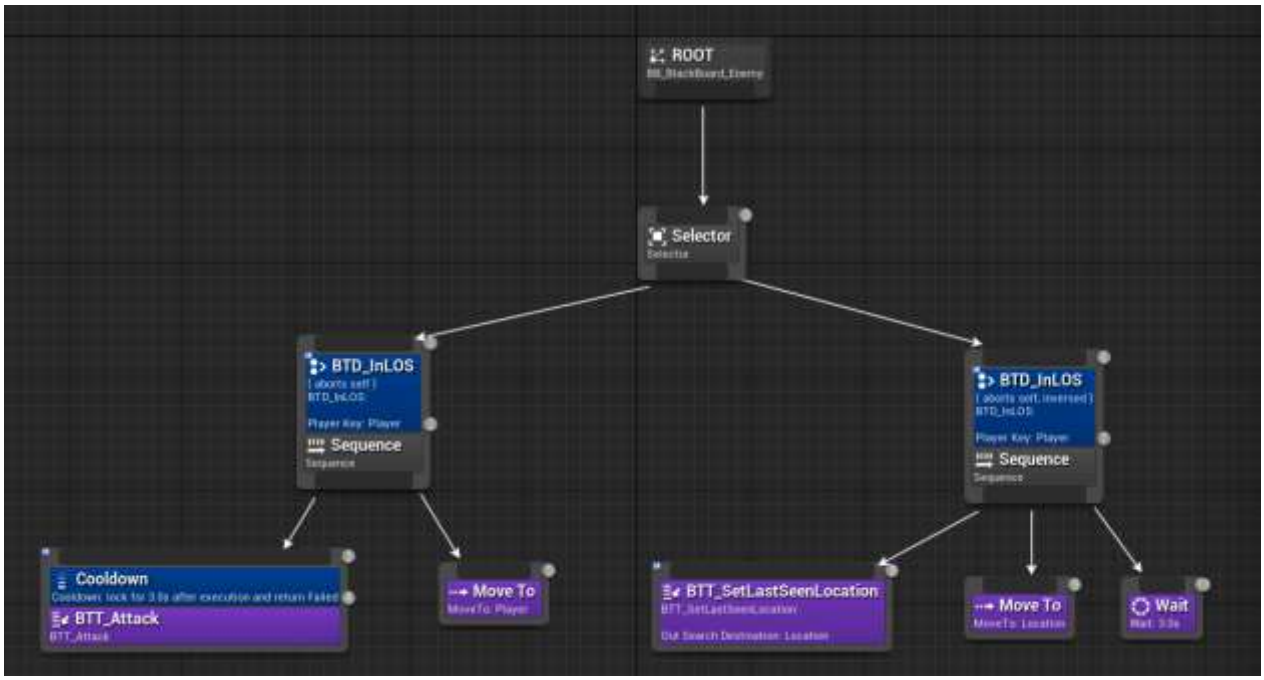

Рисунок 4.43 – Переслідування і атака гравця

Останньою функцією, що обриває всі інші, є функція смерті. Щойно в персонажа спрацьовує смерть в інтерфейсі, спрацьовує тригер, який вимикає всі функції Behavior Trees і знищує персонажа з рівня та прибирає зв'язки з AI Player Controller (рис. 4.44).

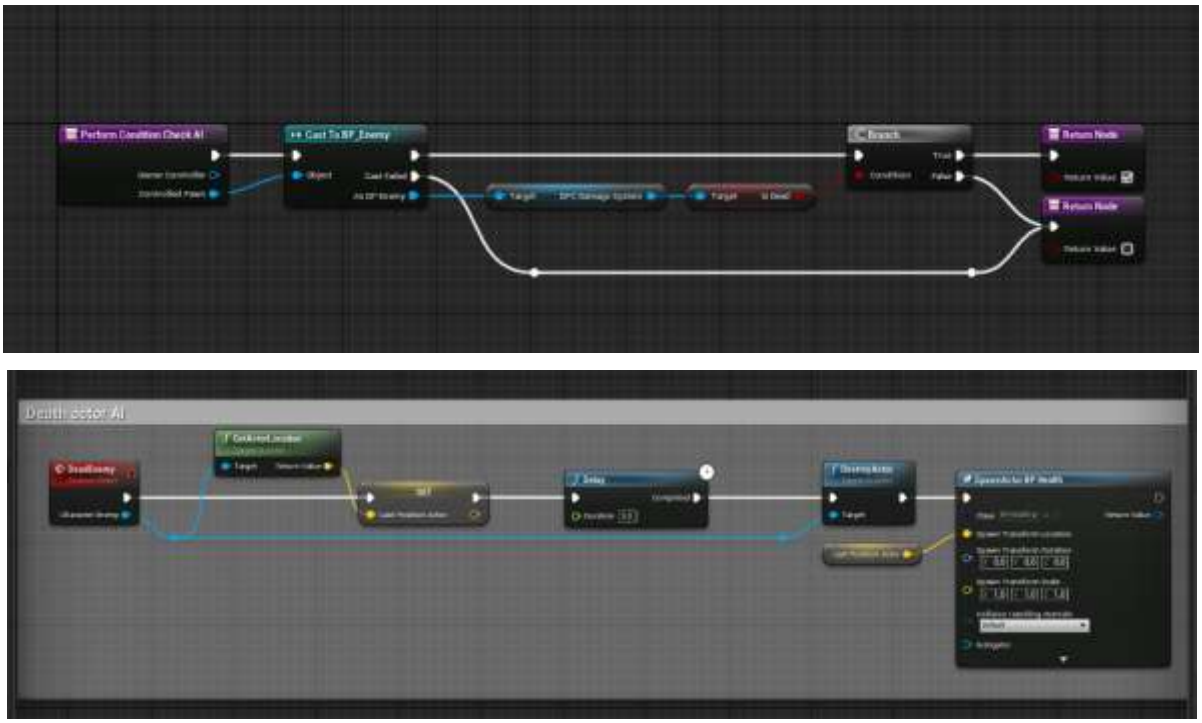

Рисунок 4.44 – Реалізація смерті персонажа

4.4.5 Створення таймеру та початку гри на рівні

Початок гри відбуватиметься як головний гравець з'явиться на рівні. Для цього реалізується перевірка появи героя на рівні, і щойно перевірка дає позитивне значення, запускається функція таймера (рис. 4.45).

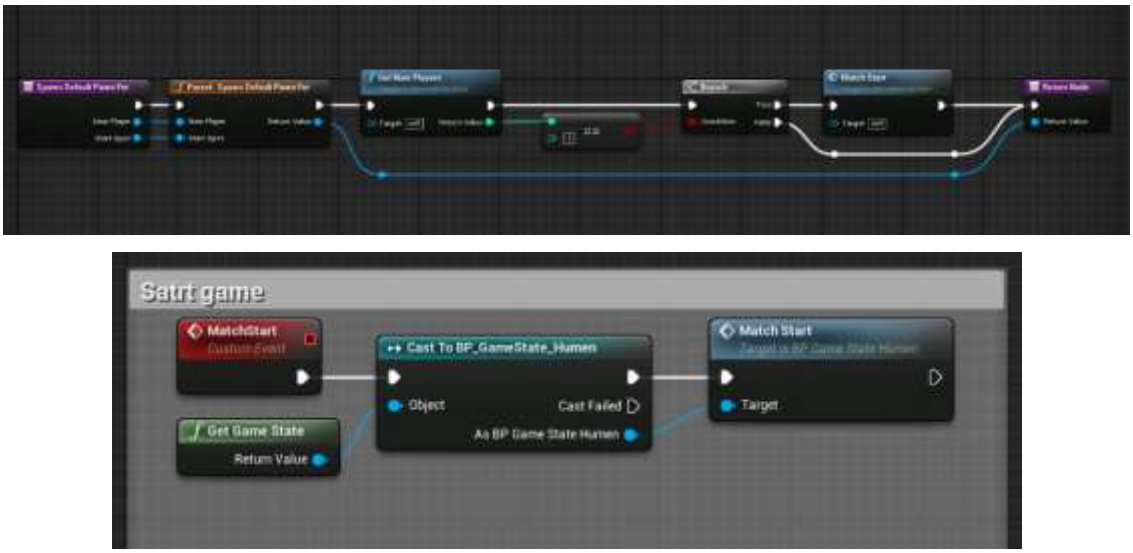

Рисунок 4.45 – Початок матчу гри

Таймер реалізується в класі "Game State", це найважливіший компонент, який допомагає керувати станом гри та синхронізувати його для всіх підключених гравців, алгоритм таймеру на рис. 4.46. Він слугує центральним вузлом для зберігання та тиражування критично важливої ігрової інформації, забезпечуючи узгодженість дій усіх клієнтів у багатокористувацькій грі (рис. 4.47).

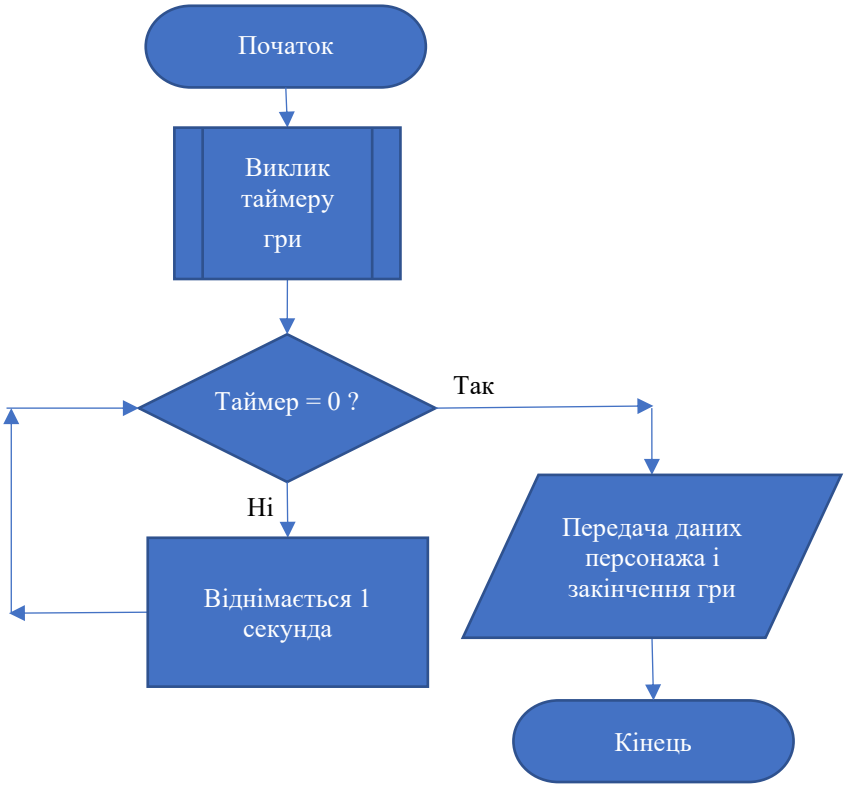

Рисунок 4.46 – Алгоритм таймеру гри

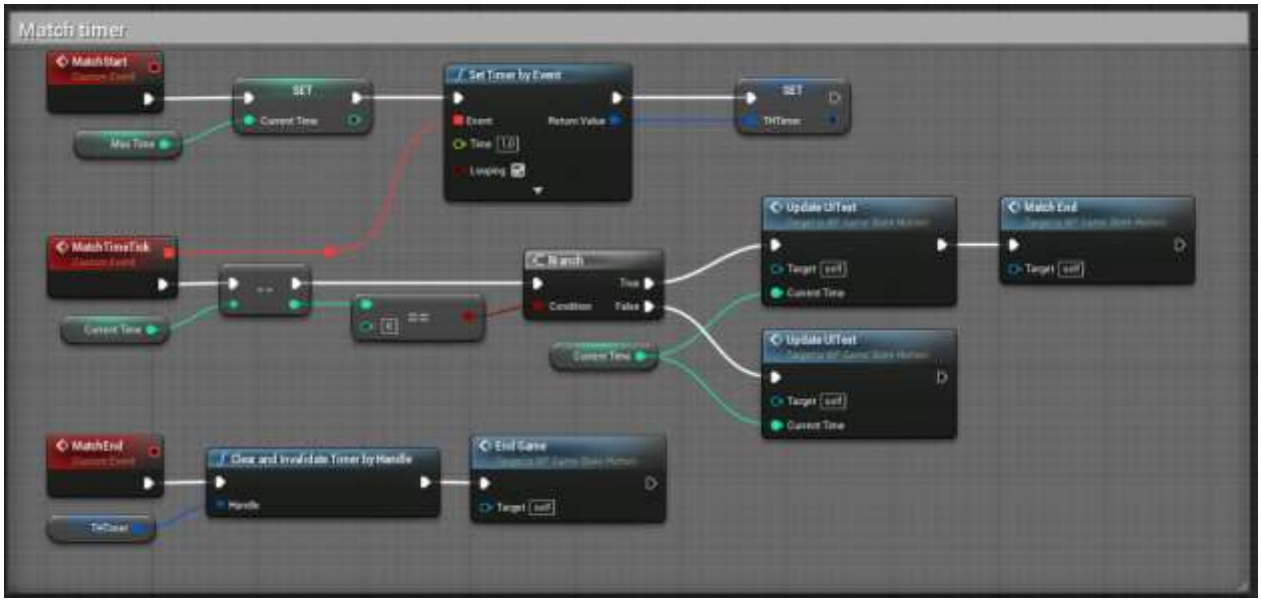

Рисунок 4.47 – Реалізація таймеру у "Game State"

Додатковою функцією реалізовано додавання часу, коли гравець підбирає предмет із бонусів, для збереження механіки додається 15 секунд часу до гри (рис. 4.48).

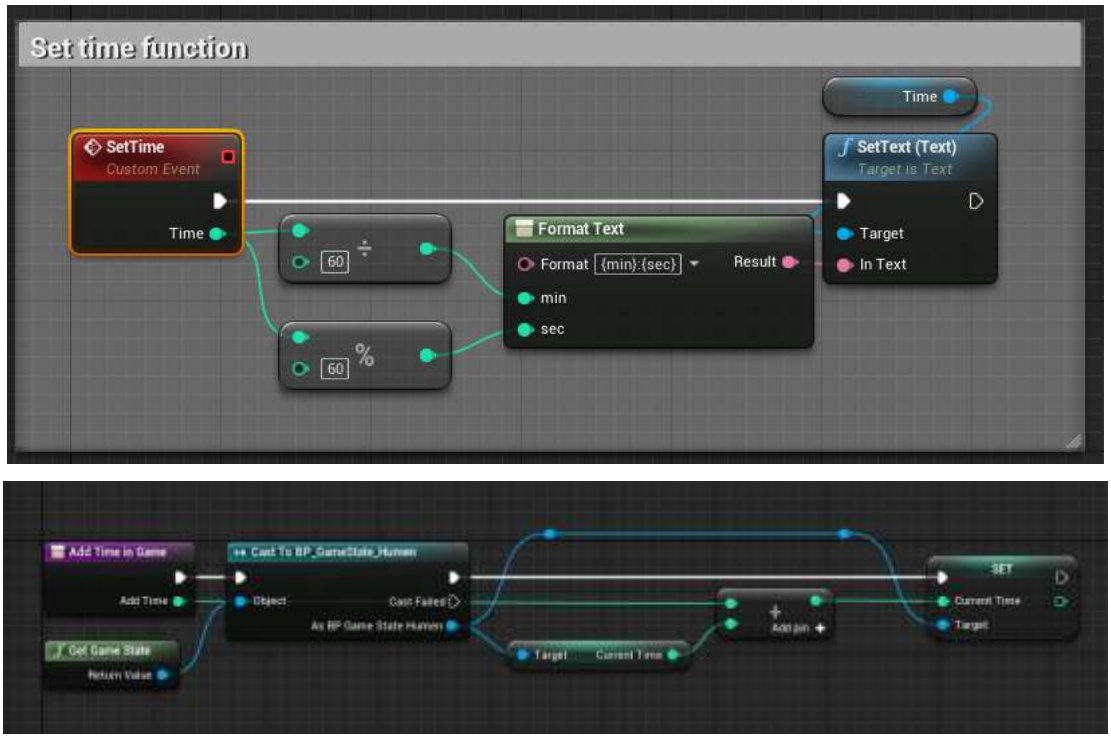

Рисунок 4.48 – Додавання додаткового часу

4.4.6 Реалізація віджетів гри

Для того щоб гравець зміг бачити дані про гру, а саме рівень життя, кількість зібраних очок, кількість вбитих персонажів, кількість і вигляд снарядів, переходи в грі та інші візуальні частини існує клас "User Widget Blueprint". Це клас візуальних сценаріїв, який використовується здебільшого для створення та оформлення елементів користувацького інтерфейсу (UI) в іграх, приклад інтерфейсу прототипу на рис. 4.49, а логіка взаємодії на рис 4.50.

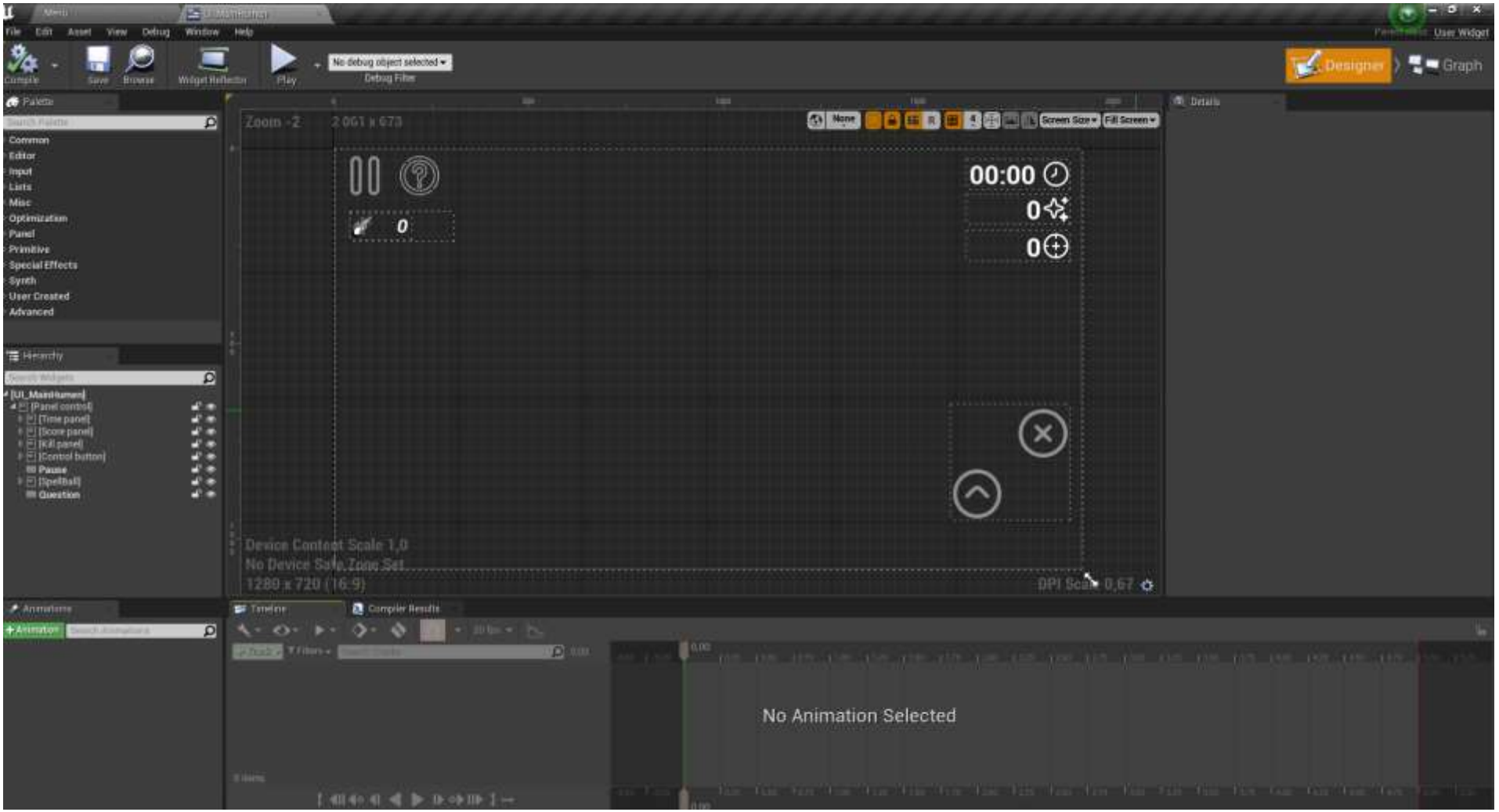

Рисунок 4.49 – Інтерфейс прототипу гри

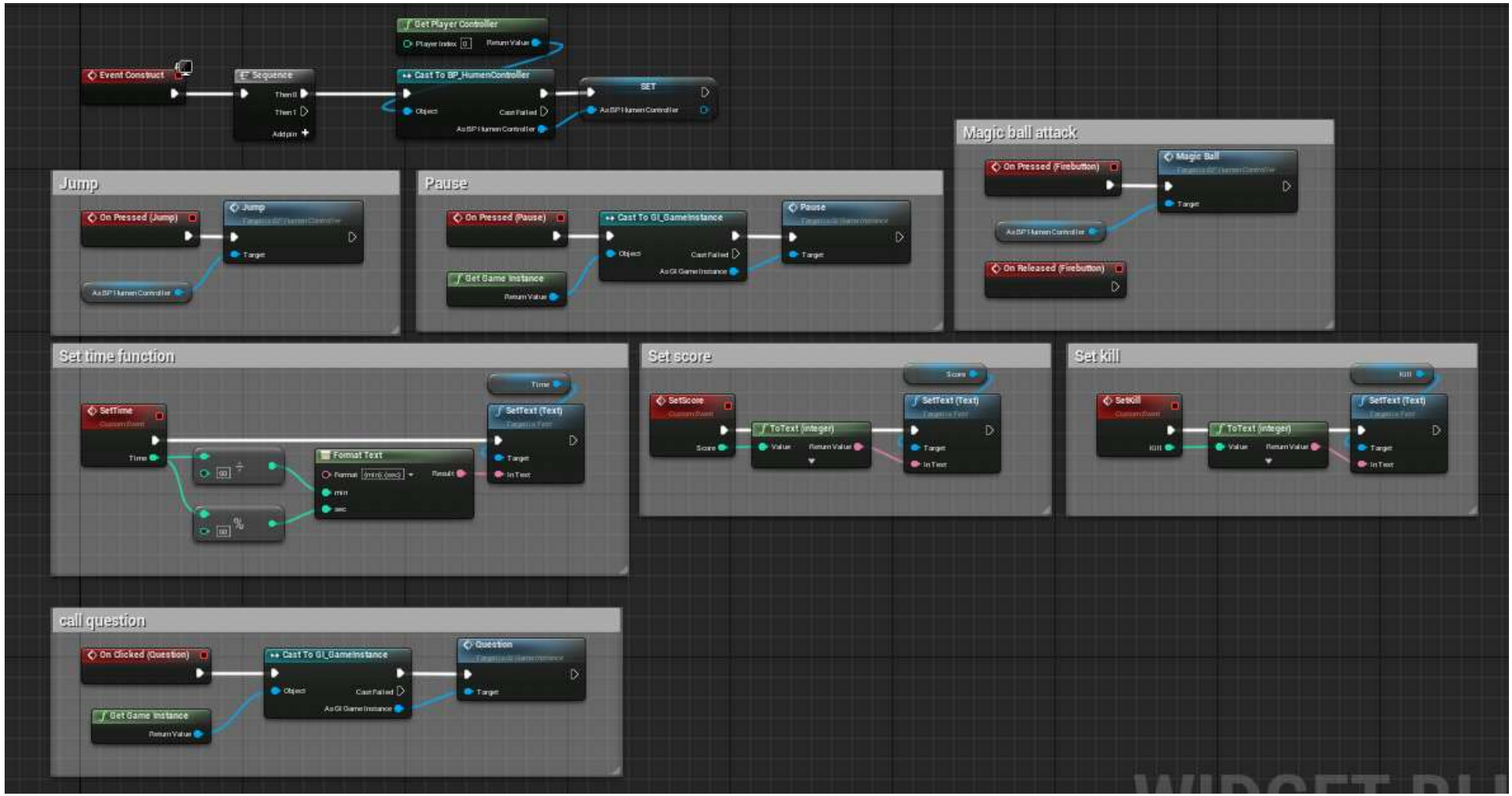

Рисунок 4.50 – Логіга взаємодії віджету гравця

У UI реалізується безліч функцій, однак, щоб керувати їхніми перемиканнями, існує клас "HUD", це компонент, що являє собою графічну накладку, яка відображається на екрані для передавання важливої інформації гравцеві під час гри. HUD зазвичай використовується для відображення таких важливих елементів, як смужки здоров'я, лічильники патронів, очок та інших важливих деталей. У Unreal Engine HUD часто реалізується за допомогою комбінації класів Blueprint і UMG (Unreal Motion Graphics) для створення візуальних елементів (рис. 4.51).

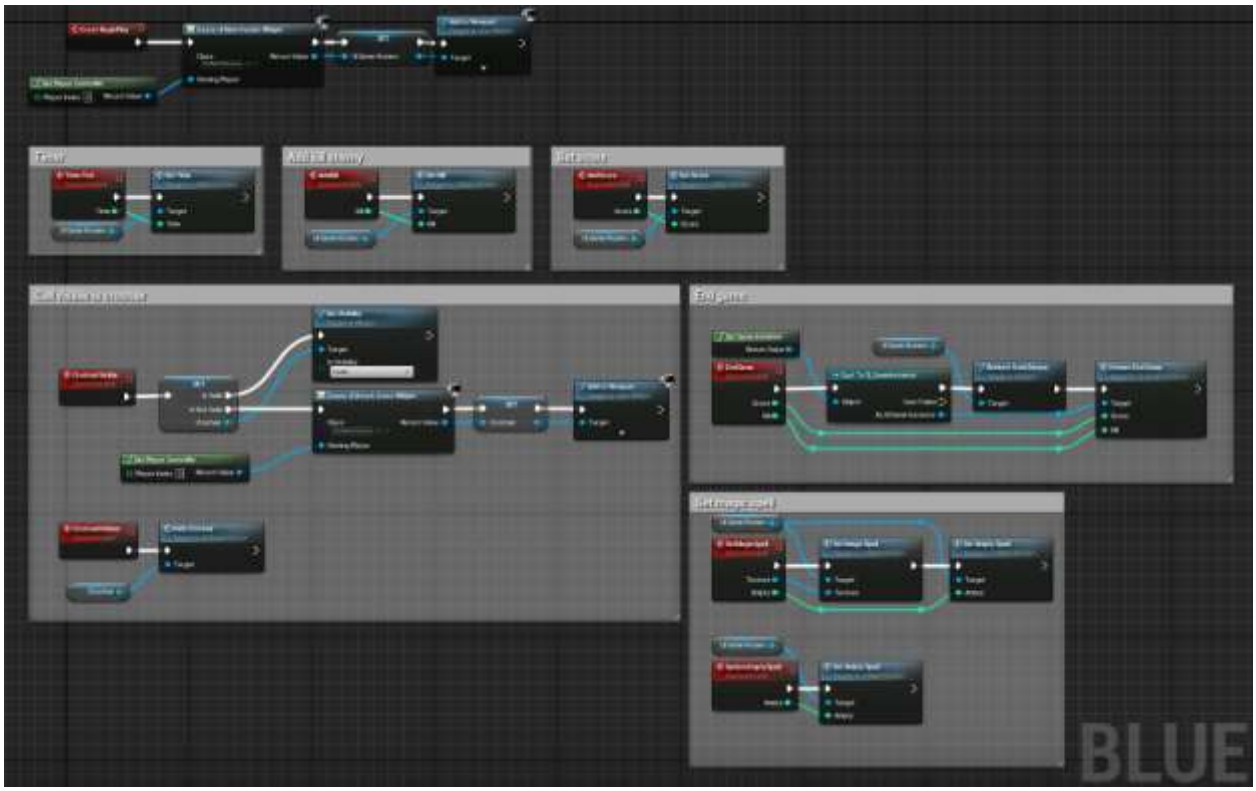

Рисунок 4.51 – Реалізація переходу між віджетами

Усі віджети та переходи між рівнями реалізуються в "Game Instance", це постійний об'єкт, що існує протягом усього часу існування гри. Це глобальний клас, схожий на синглтон, який зберігає свій стан під час переходів між рівнями, що робить його придатним для зберігання та управління даними в грі, які повинні зберігатися між різними частинами гри (рис. 4.52), алгоритм переходу логіки віджетів на рис. 4.53.

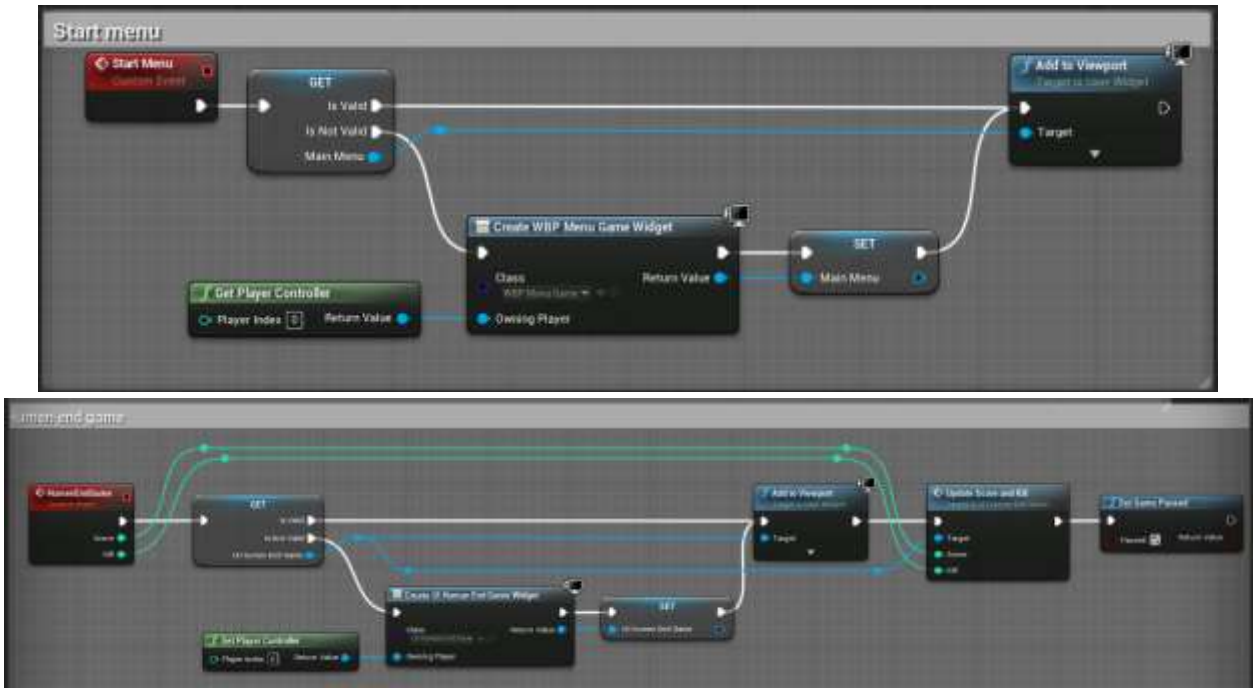

Рисунок 4.52 – Реалізація переходу через "Game Instance"

Рисунок 4.53 – Алгоритм переходу та створення віджетів

У процесі розробки прототипу гри під мобільні пристрої реалізовано прототип ігрової локації з використання 3D моделей, зроблених у програмі Blender. Показано налаштування матеріалу 3D моделі, налаштування імпорту. Реалізовано пересування гравця на рівні, встановлено анімації. Розроблено універсальну систему атаки гравця, яка може застосовуватися на будь-яких персонажах, даючи змогу гнучко змінювати можливості у різних гравців. Зроблено реалізацію появи чарівного снаряда і взаємодії його зі світом. Створено клас для появи об'єктів, які має підбирати гравець. Появу реалізовано випадковим методом, що дає змогу гнучко змінювати поведінку персонажа під час гри. Метод дає змогу гнучко налаштовувати та доповнювати об'єкти на рівні. Щоб гравець міг взаємодіяти з рівнем реалізовано ШІ з гнучкими можливостями взаємодії. Розроблено логіку початку матчу, його тривалість, і виведено статистику гравця. Щоб гравець міг стежити за своєю статистикою, розроблено візуальну логіку відображення, вона допомагає переходити за рівнями і візуально працювати з прототипом.

5 ТЕСТУВАННЯ ТА ОПТИМІЗАЦІЯ

5.1 Методи тестування та оптимізації ігрових додатків

Тестування є ключовою складовою для виявлення потенційних проблем та забезпечення високої якості гри перед її випуском на ринок. Оптимізація, з свого боку, дозволяє адаптувати гру до різних пристроїв та забезпечує її стабільну та ефективну роботу на різних апаратних платформах.

Оптимізація гри починається з освітлення, оскільки більша частина причин падіння fps (це вимірювання частоти зміни кадрів в грі або відео, яка визначає, скільки зображень або кадрів відтворюється пристроєм за одну секунду. Високий показник FPS вказує на більш плавну та зручну геймплей або перегляд відео, тоді як низький показник може викликати розриви або нерівномірне відтворення) являється перенавантаження рівню джерелами освітлення. Основні підходів до оптимізації освітлення:

- динамічне освітлення. Використання динамічних джерел світла може створювати більш реалістичні ефекти, але відоме також за свою велику обчислювальну вартість;
- статичне освітлення. Обчислення освітлення для статичних об'єктів у грі може зменшити обчислювальні витрати під час гри;
- зменшення кількості джерел світла або їхніх параметрів для зниження навантаження на систему;
- використання різних рівнів деталізації для освітлення на різних відстанях може допомогти знизити навантаження на систему;
- зменшення роздільної здатності тіней або використання менш вимогливих на ресурси методів рендерингу тіней.

Ще фактором який впливає на оптимізацію гри є складна логіка та неправильне використання Blueprints. Оптимізація Blueprints у Unreal Engine може значно підвищити продуктивність гри, зменшити навантаження на

процесор та поліпшити швидкість роботи. Кілька способів оптимізації Blueprints:

- уникання дублювання функціональності в різних частинах Blueprints;
- розділення великих функцій на менші, більш чіткі, що дозволить оптимізувати виклики та відлагодження коду;
- уникання складних обчислень у циклах, оскільки вони можуть значно збільшити навантаження на процесор;
- використання вбудованих інструментів Unreal Engine для профілювання та виявлення місць зайвого використання ресурсів допоможе знайти оптимізаційні можливості.

Оптимізація сітки (mesh) в Unreal Engine може значно покращити продуктивність гри та її візуальну якість. Способи оптимізації сіток:

- уникання зайвої деталізації моделей, особливо там, де вона не помітна для гравця;
- використання оптимізовані інструменти для спрощення геометрії моделей без втрати візуальної якості. Unreal має інструменти для редагування сіток в самому движку;
- використання різних рівнів деталізації для об'єктів залежно від відстані до гравця. На великій відстані можна використовувати менш деталізовані версії сіток, що допоможе зменшити обсяг обробки;
- оптимізація текстури з відповідним розміром та форматом для мінімізації використання ресурсів;
- простіші колізії для сіток, що не потребують деталізованої обробки фізики;
- використання інстансів (успадкування) сіток для повторюваних об'єктів, що може значно зменшити навантаження на систему.

Важливим інструментом оптимізації є профайлінг (profiling) в Unreal Engine. Це процес аналізу продуктивності гри, включаючи вимірювання часу, використання пам'яті та інших ресурсів для виявлення можливих проблем

продуктивності. Unreal Engine має кілька вбудованих інструментів для профайлінгу, які допомагають розібратися з найбільш ресурсомісткими частинами:

- “Stat Commands”: набір команд Stat, які дозволяють отримувати різноманітну інформацію про продуктивність гри, таку як FPS (кадри в секунду), використання пам'яті, час виконання окремих частин коду тощо. Команди Stat можна використовувати безпосередньо в консолі движка під час виконання гри [7];
- “Memory Profiler” (профайлер пам'яті): Даний інструмент дозволяє візуально аналізувати використання пам'яті під час виконання гри, щоб виявити утечки пам'яті або об'єкти, які споживають занадто багато ресурсів;
- “GPU Profiler” (профайлер GPU): Для аналізу використання графічного процесора (GPU) Unreal Engine має інструменти для визначення, які складові графіки споживають більше всього ресурсів, що дозволяє оптимізувати візуальні ефекти та матеріали для поліпшення продуктивності [8].

Розглянемо основну команду (State Commands), яка складається з розрахунку часу показу кадру - це загальний час, витрачений на створення одного кадру гри. Оскільки обидва потоки Game і Draw синхронізуються перед завершенням кадру, час кадру часто близький до часу, який показується в одному з цих потоків.

Відтворення кожного кадру в грі займає певний час, який називається, часом відображення кадру, або, коротко, часом кадру (frame time). Обчислюється час кадру зазвичай у мілісекундах (мс), тобто тисячних частках секунди. Однак в ігрових індустрії замість цієї характеристики набагато частіше використовують частоту зміни кадрів або, коротко, частоту кадрів (frame rate), що дорівнює кількості кадрів, намальованих за одиницю часу. Вимірюється частота кадрів у кількості кадрів за секунду (frames per second,

fps), і для стислості частоту кадрів також дуже часто називають абревіатурою від назви її одиниці виміру, тобто FPS.

Між часом кадру і частотою кадрів є математичний зв'язок: значення FPS, підраховане безпосередньо після відмальовування чергового кадру, іменоване зазвичай миттєвим *FPS*, є величиною, зворотною часу цього кадру:

$$FPS_i = {}^{1}/_{t_i} \tag{5.1}$$

Де: t_i – час кадру і обчислюється в мілісекундах.

Необхідно врахувати, що час кадру зазвичай обчислюється в мілісекундах, а частота кадрів – в одиницях на секунду, тому підсумкова формула для миттєвого *FPS* буде такою:

$$FPS_i = {}^{1000}/_{t_i} \tag{5.2}$$

Але головний показник продуктивності, це середній *FPS* по всій ігровій сцені, який є кількістю кадрів n, відмальованих за весь час бенчмарка *t*:

$$FPS_{avg} = {}^{n}/_{t} \tag{5.3}$$

Де: *n* – кількість кадрів.

Середній *FPS* можна обчислити як величину, обернену до середнього часу кадру:

$$t_{avg} = {}^{\sum t_n}/_{n} \tag{5.4}$$

$$FPS_{avg} = {}^{1000}/_{t_{avg}} \tag{5.5}$$

Основний показник часу показу кадру (Frame), розраховується як сумма середніх *FPS* від потоків Game і Draw:

$$FPS_{frame} = {\sum FPS_{avg}}/{n} \quad (5.6)$$

Де: *n* – кількість потоків.

Усі математичні операції проходять без участі програміста, дизайнера або технічного художника. До розрахунків немає можливості потрапити в коді, це зроблено щоб не порушити цілісність ігрового рушія і відображення всіх результатів. Усі результати Unreal Engine проводить сам і виводить у вигляді таблиць або візуальних обчислень, які проходять у симуляції гри. Для обчислень він збирає дані з буферів і відправляє в закладений код, де надалі перетворює на звіти. Такий спосіб полегшує роботу з оптимізацією та прискорює розуміння і розв'язання проблеми з помилками в ігрових світах.

5.2 Тестування ігрового додатку

Тестування становить ключовий етап у процесі розробки та вдосконалення будь-якого продукту чи системи. Його значення полягає у проведенні детального аналізу, випробуванні функціональності та пошуку можливостей для підвищення ефективності та якості.

В оптимізації гри було обрано робота над освітлення, перевірка та скорочення логіки у Blueprints, праця з 3D моделями. Використано кілька варіантів для оптимізації освітлення в Unreal Engine:

- перехід з динамічного до статичного освітлення, щоб зберегти ресурси на логіку гри. Але це збільшить пам’ять додатку при компіляції гри (рис. 5.1);

Рисунок 5.1 – Рівні оптимізації освітлення

– Зменшення кількості джерел світла та вибір spotlight для економії ресурсів ігрового двигуна (рис. 5.2).

Рисунок 5.2 – Використання spotlight на рівні

Другим етапом тестування стало оптимізація Blueprints у Unreal Engine що підвищує продуктивність гри. Із способів оптимізації Blueprints обрано:

– зменшення дублювання функціональності у різних частинах Blueprints (рис. 5.3);

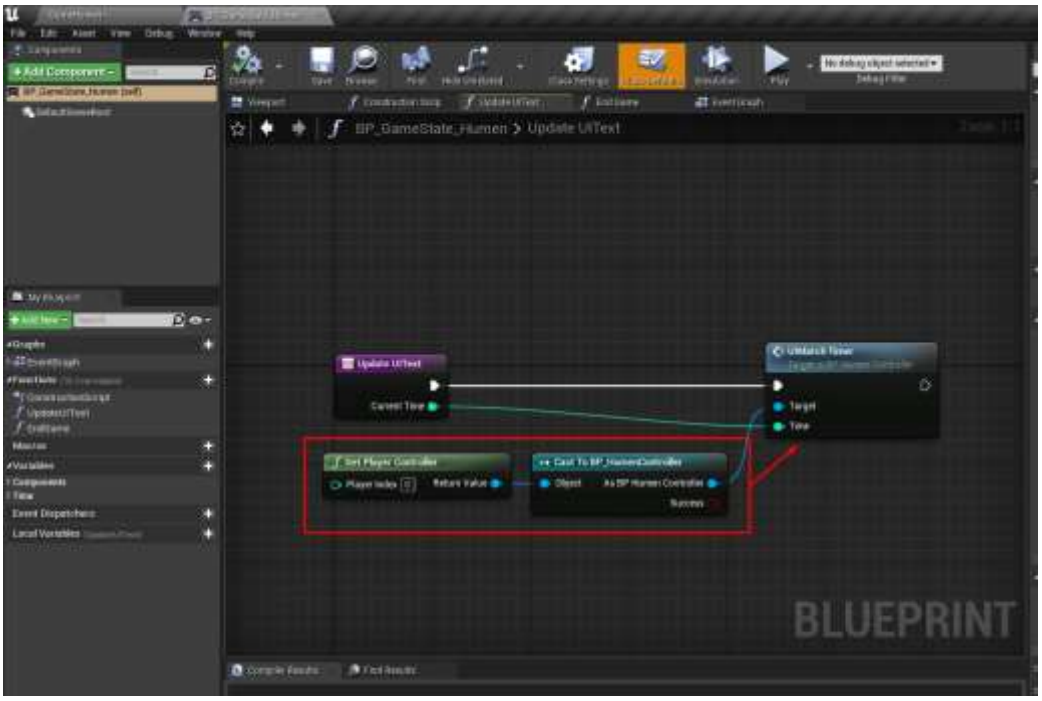

Рисунок 5.3 – Використання готової логіки у інших реалізованих класах

– розділення великих функцій на менші, що дозволило підвищити оптимізування та читабельність вузлів (рис. 5.4).

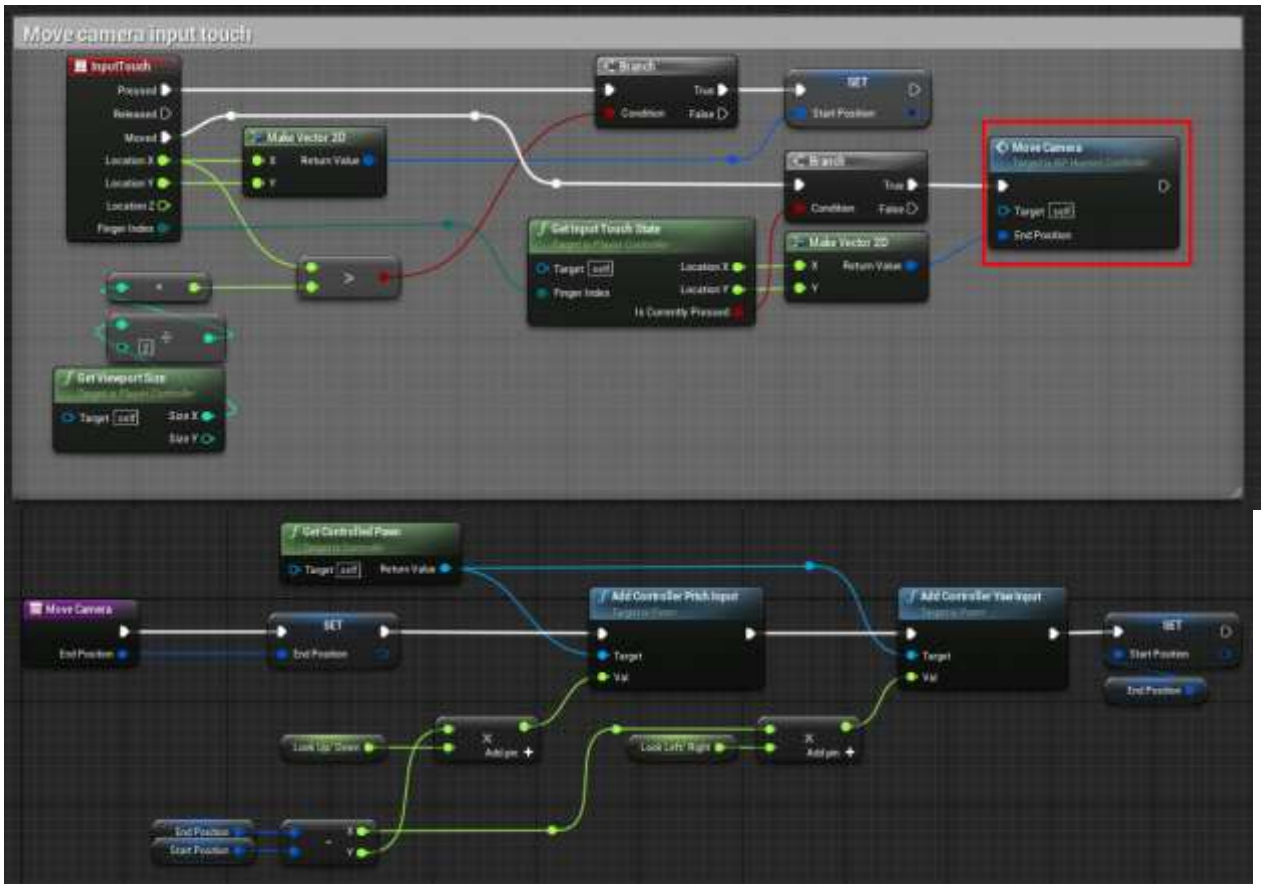

Рисунок 5.4 – Розділення великих функцій у Blueprints

Використано робота з сіткою (mesh) в Unreal Engine що покращує продуктивність гри та її візуальну якість (рис. 5.5). Зі способів оптимізації сіток обрано:

- деталізацію моделей перенесено на UV-карти які називаються normal;
- використання з’єднання декількох текстур у одну, за допомогою занесення їх по каналам rgb, з відповідним розміром та форматом для мінімізації використання ресурсів;
- використання простих колізій для сіток, що не потребують деталізованої обробки фізики;
- використання інстансів (успадкування) мешів для повторюваних об'єктів.

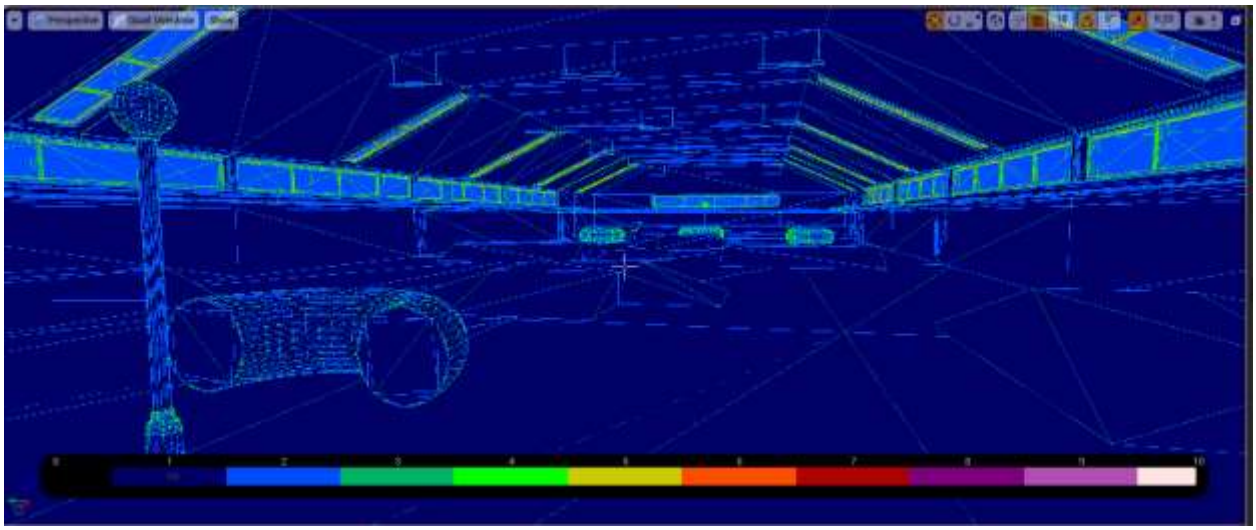

Рисунок 5.5 – Відображення рівня деталізації на рівні

Для перевірки потоку ресурсів у ігри є використання інструмента оптимізації профайлінг (profiling) в Unreal Engine. У процес аналізу продуктивності гри, отримали високі результати ефективності, пов’язані з кількістю кадрів у секунду, якість відмолювання текстур, та стабільність переходів логіки у грі (рис. 5.6).

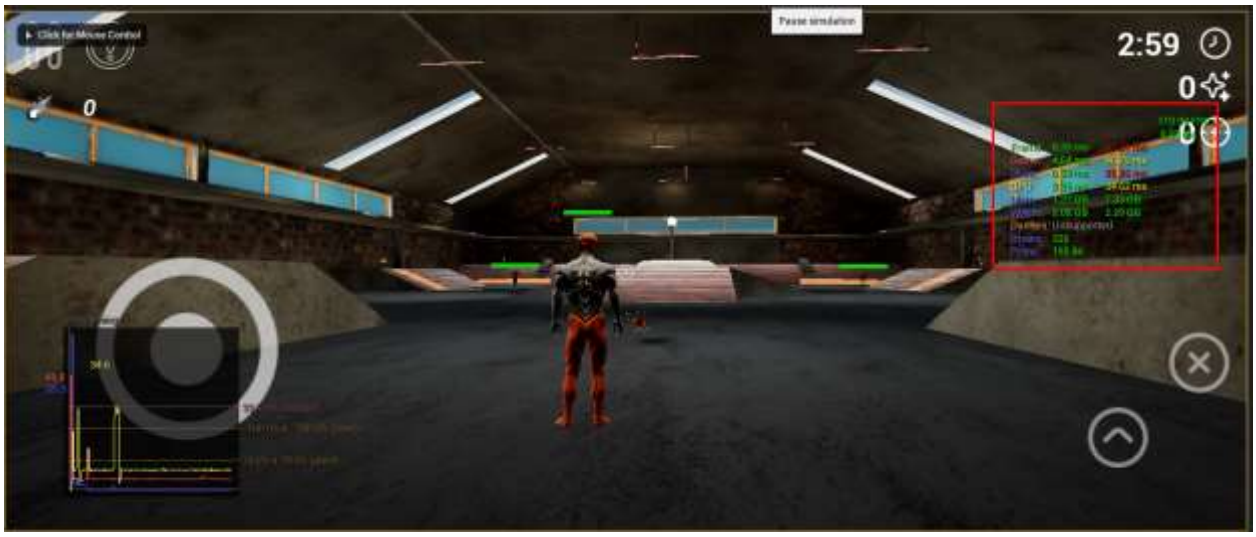

Рисунок 5.6 – Потік ресурсів гри з використанням профайлінгу

У процесі розглянуто основні принципи оптимізації з використанням вбудованих інструментів ігрового рушія Unreal Engine. Розглянуто процес розрахунку частоти кадрів з використанням додаткових буферів рендеру рушія. Проведено оптимізацію прототипу, на якій налаштовано джерела освітлення, зменшено їхню кількість та перевизначено тип. Зменшено кількість повторюваних функцій у проєкті, для повторних функцій зроблено переходи для використання в інших класах. Зменшено кількість змінних, що звільнило частину пам'яті та збільшило час розрахунку логіки. Великі ланцюжки логіки переведені у функції, щоб підвищити читабельність коду, і збільшити швидкість переходу для запису значень, що дає змогу оптимізувати розрахунки на буфері, за які відповідає процесор і відеокарта. Зроблено оптимізацію тривимірних об'єктів, щоб зменшити час розрахунку буфера рендерінгу, проведено роботу над текстурами і матеріалами для менших витрат пам'яті. Показники оптимізації в процесі гри видають високу кількість кадрів, що дає змогу розширити прототип надалі та додати нових об'єктів у сцену. Також ці результати дають можливість реалізації прототипу навіть на малопотужні пристрої.

ВИСНОВКИ

Основна мета даної магістерської роботи полягала у вивченні можливостей Unreal Engine для створення високоякісної та ефективної ігрової платформи для мобільних пристроїв.

Протягом роботи проаналізовані ключові жанри ігор і їх відомі приклади, розглянуто основні ігрові двигуни для реалізації та платформи і налаштування для збиту ігор. Переглянуті основні аспекти роботи з Unreal Engine, включаючи його інтерфейс, можливості з розробки графіки, роботи з 3D моделями, налаштування фізики та логіки гри. Основною частиною було використання 3D моделювання для створення візуально привабливих та реалістичних об'єктів на сцені, що забезпечило збільшення іммерсивності та привабливості гри. В якості програмою для 3D моделювання було обрано Blender, за його гнучкі можливості та доступність у користувачів.

В ході розробки були реалізовані етапи: від створення прототипу гри до оптимізації продукту для забезпечення його стабільної роботи на мобільних пристроях з оптимальною продуктивністю. Процес оптимізації включав в себе роботу з різними аспектами гри, від текстур і освітлення до роботи зі сценами та моделями, з метою забезпечення плавності геймплею на різних пристроях.

Результатом цієї роботи стало створення прототипу ігрового додатку, який поєднує в собі тривимірну графіку, мультифункціональний геймплей та стабільну роботу на емуляторі Android.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ресурс з комп'ютерної графіки [UnrealEngine]. – 2012. Режим доступу до ресурсу: https://www.unrealengine.com/en-US (дата звернення: 10.10.2023)

2. Ресурс з комп'ютерної графіки [Blender]. – 2015. Режим доступу до ресурсу: https://www.blender.org/ (дата звернення: 15.10.2023)

3. Ресурс з комп'ютерної графіки [Habr]. – 2010. Режим доступу до ресурсу: https://habr.com/ua/post/451266/ (дата звернення: 15.10.2023)

4. Ресурс з комп'ютерної графіки [GameDevDOU]. – 2020. Режим доступу до ресурсу: https://gamedev.dou.ua/articles/ (дата звернення: 9.11.2023)

5. Ресурс з комп'ютерної графіки [Mixamo]. – 2018. Режим доступу до ресурсу: https://www.mixamo.com/ (дата звернення: 15.11.2023)

6. Ресурс з комп'ютерної графіки [Render]. – 2010. Режим доступу до ресурсу: https://render.ua/ua/ (дата звернення: 15.11.2023)

7. Ресурс з комп'ютерної графіки [GitHub]. – 2010. Режим доступу до ресурсу: https://github.com/Allar/ue5-style-guide (дата звернення: 18.11.2023)

8. A Comparative Example Between The Use Of Pca And Mds For Image Classification / Hernandez, W., Mendez, A., Flor-Unda, O., Camejo, I.M., Kolendovska, M.// IEEE International Symposium on Industrial Electronics, 29th IEEE International Symposium on Industrial Electronics, ISIE 2020; Delft; Netherlands; 17 June 2020 до 19 June 2020; Volume 2020-June, June 2020, № 9152565, Pages 1353-1358

9. Algorithm For Generating Refined Frequency Estimates In Atmospheric Radio Sounding Systems / Kartashov V., Hernandez W., Hernandez-Balbuena D., M. Kolendovska, Konovalenko O., Melnyk V.// IEEE International Symposium on Industrial Electronics, 29th IEEE International Symposium on Industrial Electronics, ISIE 2020; Delft; Netherlands; 17 June 2020 до 19 June 2020; Volume 2020-June, June 2020, № 9152562, Pages 79-82

10. Application of Fast Frequency Shift Measurement Method for INS in Navigation of Drones / D. Avalos-Gonzalez, D.H. Balbuena, V. Tyrsa, V.M. Kartashov, M. Kolendovska, S. Sheiko, O. Sergiyenko, V. Melnyk, F.N. Murrieta-Rico // IECON 2018 – 44th Annual Conference of the IEEE Industrial Electronics Society. – P. 3159–3164.

11. Avalos-Gonzalez, D., Sergiyenko, O., Hernandez-Balbuena, D., Tyrsa,V., Kartashov V.M., V.,Rivas-Lopes, M., Murrieta-Rico, F.N. Constraints definition and application optimization based on geometric analysis of the frequency measurement method by pulse coincidence// Measurement: Journal of the International Measurement Confederation (USA). 2018, V.126. P. 184-193.

12. Book “Control and Signal Processing Applications for Mobile and Aerial Robotic Systems”, Hardback - Advances in Computational Intelligence and Robotics English. Edited by Oleg Sergiyenko, Moises Rivas-Lopez, Wendy Flores-Fuentes, Julio Cesar Rodríguez-Quiñonez, Lars Lindner. Editorial IGI Global, Hershey, United States, January 2020, 340 páginas. ISBN10 152259924X, ISBN13 9781522599241

13. Cesar Sepulveda-Valdez ; Oleg Sergiyenko ; Vera Tyrsa ; Wendy Flores-Fuentes ; Julio César Rodríguez-Quiñonez ; Fabian Natanael Murrienta-Rico ; Jesús Elías Miranda-Vega ; Paolo Mercorelli ; Marina Kolendovska. "Geometric analysis of a laser scanner functioning based on dynamic triangulation," 2020 IEEE 29th International Symposium on Industrial Electronics (ISIE), Delft, Netherlands, 17-19 of June 2020, pp. 1398-1403, doi: 10.1109/ISIE45063.2020.9152268. https://ieeexplore.ieee.org/abstract/document/9152268

14. Cuauhtémoc Mariscal-García; Wendy Flores-Fuentes; Daniel Hernández-Balbuena; Julio C. Rodríguez-Quiñonez ; Oleg Sergiyenko. "Classification of Vehicle Images through Deep Neural Networks for Camera View Position Selection," 2020 IEEE 29th International Symposium on Industrial Electronics (ISIE), Delft, Netherlands, 17-19 of June 2020, pp. 1376-1380, doi: 10.1109/ISIE45063.2020.9152440.https://ieeexplore.ieee.org/abstract/document/9152440

15. Developing and Applying Optoelectronics in Machine Vision/ O. Sergiyenko, J.C. Rodriguez-Quiñonez, IGI Global, 2016; 341p.

16. Experimental estimation of direction finding to unmanned air vehicles algorithms efficiency by their acoustic emission, /Oleynikov, V., Zubkov, O., Kartashov, V., ...Sheiko, S., Babkin, S.//2019 IEEE International Scientific-Practical Conference: Problems of Infocommunications Science and Technology, PIC S and T 2019 - Proceedings, 2019, стр. 175-178, 9061337

17. Features of acoustic noise of small unmanned aerial vehicles / Semenets, V.V., Kartashov, V.M., Leonidov, V.I. //Telecommunications and Radio Engineering (English translation of Elektrosvyaz and Radiotekhnika), 2020, 79(11), стр. 985-995

18. I. Y. A. Corpus, L.Lindner, O.Sergiyenko. "Transimpedance Amplifier for Laser Scanning System Range Extension," 2020 IEEE 29th International Symposium on Industrial Electronics (ISIE), Delft, Netherlands, 17-19 of June 2020, pp. 1421-1426, doi: 10.1109/ISIE45063.2020.9152487. https://ieeexplore.ieee.org/abstract/document/9152487