`Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Комп’ютерна гра «Forgotten Island: Hero Artifact»
(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ--21-6
Михайло СКОРОХОД
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

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

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: доц. Наталія БОЛОГОВА
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ ______________ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

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

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри ________________
(підпис)
"____"________________ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Скороходу Михайло Михайловичу
(прізвище, ім'я, по батькові)

1. Тема роботи Комп'ютерна гра «Forgotten Island: Hero Artifact»

затверджена наказом по університету від " 26 " травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025р.

3. Вхідні дані до роботи 1) документація Unreal Engine 5; 2) документація Gameplay System; 3) середовище розробки Visual Studio; 4) графічний двигун Unreal Engine 5;

4. Перелік питань, що потрібно опрацювати у роботі
   1) аналіз предметної області та існуючих аналогів;
   2) аналіз та вибір програмного забезпечення;
   3) опис застосунку;
   4) програмна реалізація;
   5) висновки;

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп’ютерних ілюстрацій

Слайдів презентації - 20

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	27.05.25-30.05.25	
2	Вибір технології розробки та інструменталь-	31.05.25-03.06.25	
	них засобів		
3	Розробка концепції гри	04.06.25-05.06.25	
4	Розробка та відлагодження програмного	06.06.25-09.06.25	
	забезпечення		
5	Оформлення матеріалів кваліфікаційної роботи	10.06.25-11.06.25	
6	Подання кваліфікаційної роботи керівникові та	12.06.25-13.06.25	
	її попередній захист		
7	Подання кваліфікаційної роботи на рецензу-	14.06.25-16.06.25	
	вання		

Дата видачі завдання “ 26 ” травня 20 25 р.

Здобувач ____________
(підпис)

Керівник роботи ____________ доц. Наталія БОЛОГОВА
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 76 с., 16 рис., 0 табл., 2 дод., 16 джерел.

3-Д ГРА, ІНТЕРФЕЙС КОРИСТУВАЧА, ООП, С++, СТВОРЕННЯ ГРИ, ГРАФІЧНИЙ ДВИГУН, UNREAL ENGINE, ІГРОВИЙ ДОСВІД, ABILITY SYSTEM COMPONENT, VISUAL STUDIO.

Метою кваліфікаційної роботи є створення гри «Forgotten Island: Hero Artifact» з використанням графічного рушія під назвою Unreal Engine 5, з реалізацію ряду механік.

Мета вирішення задачі – використання мови програмування С++ та графічного двигуна Unreal Engine 5.

У ході виконання кваліфікаційної роботи було розроблено працездатну гру в жанрі рогалик, яка надає гравцям широкий вибір, цікаві завдання та незвичайне графічно оформлення.

Кодова база проекту побудована з використанням сучаних підходів та принципів проектування ігор. Це дозволяє легко розширювати проект та додатави нові механіки.

Гра написана на мові програмування С++ з використанням Unreal Engine API, проект побудований в середовищі Visual Studio.

ABSTRACT

Bachelor`s thesis 76 pages, 16 figures, 0 tables, 2 appendices, 16 sources.

3D GAME, USER INTERFACE, OOP, C++, GAME DEVELOPMENT, GRAPHICAL ENGINE, UNREAL ENGINE, GAME EXPERIENCE, ABILITY SYSTEM COMPONENT, VISUAL STUDIO.

The major goal of this thesis is to create the game "Forgotten Island: Hero Artifact" using the Unreal Engine 5 game engine, with the implementation of various mechanics.

The aim of solving this task is to utilize the C++ programming language and the Unreal Engine 5 game engine.

During the course of this thesis, a fully functional rogue-like game was developed, offering players a wide variety of choices, interesting tasks, and unique graphical design.

The project's codebase is built using modern approaches and game design principles. This allows the project to be easily expanded and new mechanics to be added.

The game is written in C++ using the Unreal Engine API, and the project is developed within the Visual Studio environment.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ 7
ВСТУП 8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ АНАЛОГІВ 11
1.1 Ігровий жанр «Рогалик» 11
1.2 Представники жанру «Рогалик» 12
1.3 Баланс ігор жанру «Рогалик» 16
1.4 Покращення ігор жанру «Рогалик» 18
2 АНАЛІЗ ТА ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 20
2.1 Графічний двигун Unreal Engine 20
2.2 Основна логічна частина двигуна Unreal Engine 25
2.3 Фреймворк Gameplay Ability System 29
3 ОПИС ЗАСТОСУНКУ 31
4 РЕАЛІЗАЦІЯ ЗАСТОСУНКУ 41
4.1 Реалізація базових класів застосунку 41
4.2 Реалізація здібностей. Gameplay Ability 44
4.3 Вплив здібностей на атрибути персонажів. Gameplay Effects 46
4.4 Gameplay Tags 47
4.5 Ability Tasks 49
4.6 Використання Gameplay Ability System в реалізації проектів 51
ВИСНОВКИ 52
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 53
ДОДАТОК А Графічний матеріал кваліфікаційної роботи 55
ДОДАТОК Б КОД ПРОГРАМИ 66
Б.1 Клас персонажу AFIHACharacterBase 66
Б.2 Клас компоненту UFIHAAbilitySystemComponent 71
Б.3 Клас атрибутів UFIHAAttributeSet 73

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ШІ – штучний інтелект (англ. Artificial Intelligence)

UI – користувацький інтерфейс (англ. User Interface)

UX – користувацький досвід (англ. User Experience)

API – інтерфейс прикладного програмування (англ. Application Programming Interface)

ВСТУП

Люди грали в ігри ще з давніх часів, але в сучасну епоху це захоплення вийшло на новий рівень – тепер вони освоюють комп’ютерні ігри різного ступеня складності. Вони поєднували в собі не тільки розваги, але і навчання, соціальну взаємодію та розвиток фізичних властивостей людини. Ігри бувають фізичними, інтелектуальними та цифровими, але всі вони мають в собі певні правила для досягнення цілей. Важливою характеристикою ігор є їх універсальність. Ігри не обмежуються віком, становищем та статтю людини, завдяки їх різноманітності кожному знайдеться якась гра по душі. Ігри також можуть мати освітній, спортивний або терапевтичний влив на гравця, сприяючи покращенню повних фізичних та психічних навичок.

В наш час ігри переросли в відеоігри. Сучасні відеоігри перетворити ігровий процес на інтерактивне мистецтво. Завдяки розвитку технологій ігри стали платформою для соціалізації і самовираження, і навіть професійної діяльності, зокрема в кіберспорті.

В ході розвитку ігрової індустрії з’явився новий напрямок в бізнесі під назвою «Game Development» або «Розробка ігор». Розробка ігор – це мистецтво створення ігор, що може включати створення концепції, проектування, створення, тестування та випуск. Коли ви створюєте гру, важливо подумати про ігрову механіку, винагороди, залучення гравців і дизайн рівнів. У цьому напрямку з’явилося безліч професій: розробник гри може бути програмістом, звуковим дизайнером, художником, дизайнером або багатьма іншими ролями, доступними в галузі. Розробкою ігор може займатися як велика студія розробки ігор так окрема особа. Поки гра дозволяє гравцеві взаємодіяти з вмістом і може маніпулювати елементами гри, ви можете називати це «грою». Щоб взяти участь у процесі розробки гри, вам не обов’язково потрібно писати код. Художники можуть створювати та проектувати ресурси, тоді як розробник може зосередитися на

програмуванні механік. Тестер може залучитися, щоб перевірити, чи гра працює належним чином та дати певний фідбек.

Спочатку виникли графічні бібліотеки та фреймворки, які включали в себе багато вбудованих функцій для реалізації графіки. Але дуже складні для людей, що не мають навиків програмування в минулому. Щоб вирішити проблеми, які виникли у ігрових фреймворків, були розроблені такі інструменти, як libGDX і OpenGL. Вони допомогли розробці ігор стати набагато швидшими та легшими, надаючи багато готових функцій і особливостей. Проте все одно було важко увійти в індустрію або зрозуміти структуру для людини, яка не є програмістом, що часто зустрічається в індустрії розробки ігор.

Тому для поліпшення та пришвидшення створення ігор були реалізовані ігрові рушії. Ігровий рушій (англ. Game engine) – програмний рушій, центральна програмна частина будь-якої відеогри, яка відповідає за всю її технічну сторону, дозволяє полегшити розробку гри шляхом уніфікації та систематизації її внутрішньої структури. Саме тоді були розроблені такі ігрові двигуни, як Construct, Game Maker, Unity та Unreal. Загалом, двигун має все, що має фреймворк, але з більш дружнім підходом завдяки використанню графічного інтерфейсу користувача (GUI) і допомозі з графічною розробкою гри.

У деяких випадках, наприклад у Game Maker і Construct, кількість готових функцій є настільки великою, що люди без попередніх навичок програмування можуть створювати гру з нуля, справді розширюючи сцену та роблячи розробку ігор доступною майже для всіх.

Багато розробників обирають розробляти гру за допомогою Game Development Engine. Ігрові двигуни можуть значно полегшити процес створення гри та дозволити розробникам повторно використовувати багато функцій. Він також піклується про рендеринг 2D і 3D графіки, фізику та виявлення зіткнень, звук, сценарії та багато іншого. Деякі ігрові движки мають дуже круту криву навчання, наприклад CryEngine або Unreal Engine.

Проте інші інструменти дуже доступні для початківців, а для деяких навіть не потрібно, щоб ви могли писати код для створення своєї гри, наприклад Constructor 2. Ігровий движок Unity знаходиться десь посередині, хоча він зручний для початківців, деякі популярні та комерційні ігри були створені з використанням Unity (наприклад, Overcooked, Superhot). Ігровий движок BuildBox в основному призначений для розробки гіперказуальних ігор.

Приклади ігрових движків:

- CryEngine;
- Unreal Engine;
- Ігровий движок Unity;
- Game Maker;
- Constructor 2/3;
- Godot Engine;
- Source;
- Frostbite;
- Buildbox.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ АНАЛОГІВ

1.1 Ігровий жанр «Рогалик»

Проект «Forgotten Island: Hero Artifact», належить до жанру рогаликів (roguelike) [1], який відзначається процедурною генерацією світу, постійною смертю персонажа (permadeath) та високим акцентом на тактичному мисленні й прийнятті рішень. Основна ідея рогаликів полягає у створенні унікального ігрового досвіду під час кожного проходження, що забезпечується випадковістю рівнів, подій та викликів. Гра використовує ключові принципи жанру, доповнюючи їх сучасними механіками, які дозволяють забезпечити високу реіграбельність, варіативність і динамічність геймплею.

Гра «Forgotten Island: Hero Artifact» розробляється з урахуванням класичних рис рогаликів, але водночас додає нові елементи, які роблять її цікавою для сучасного гравця. Однією з ключових особливостей гри є генерація мапи, де гравець має вибір який тип рівня будет наступним. Проект орієнтований на створення захопливого ігрового досвіду, де кожне нове проходження стає унікальною пригодою, що вимагає від гравця адаптивності, уважності та стратегічного підходу.

Жанр рогаликів має тривалу історію, коріння якої сягають класичної гри Rogue (1980) [2]. Ця гра задала основні принципи жанру, такі як процедурна генерація рівнів, постійна смерть персонажа (permadeath) та покроковий геймплей. Механіка permadeath може мати різні варіації, які впливають на складність і мотивацію гравців. Класичний варіант передбачає повну втрату прогресу після смерті, змушуючи гравця починати гру з нуля. Інший підхід — часткове збереження поліпшень, коли деякі елементи, такі як розблоковані здібності, зброя або бонуси, залишаються доступними в наступних спробах. Це допомагає гравцям поступово вдосконалюватися та

відчувати прогрес навіть після невдач. Деякі ігри пропонують альтернативні способи перезапуску, наприклад, можливість відродитися зі штрафами або змінами в світі гри, що додає тактичної глибини та варіативності проходження.

Сучасні рогалики розвивають ці ідеї, додаючи швидкий бойовий темп, глибшу систему прокачки персонажа, візуальну різноманітність та елементи сюжетної складової. У сучасній інтерпретації жанр часто поєднується з іншими стилями, такими як платформери чи екшен-RPG, створюючи піджанр рогалайт, який зберігає ключові механіки, але дозволяє легше адаптувати гру до ширшої аудиторії.

1.2 Представники жанру «Рогалик»

Одним із найвідоміших представників жанру є «The Binding of Isaac» [3] (рисунок 1.1). Гра, що була випущена у 2011 році, стала культовою завдяки своїй унікальній атмосфері, стилізованій під похмурий арт, що поєднує дитячу невинність із жахливими образами, натхненними біблійними мотивами.

Рисунок 1.1 – Зображення гри «The Binding of Isaac»

Геймплей побудований на класичній механіці двовимірного шутера з виглядом зверху, де гравець досліджує випадково згенеровані рівні, знищує ворогів, знаходить артефакти й посилює свого персонажа. Однією з ключових особливостей гри є система предметів, яка дозволяє комбінувати десятки бафів і здібностей, створюючи унікальні ефекти для кожного проходження. Деякі комбінації кардинально змінюють стиль гри, що змушує гравця експериментувати та знаходити нові підходи до проходження. Додатково гра має розгалужену систему секретів, альтернативних фіналів і розблоковуваного контенту, що додає мотивації грати знову й знову. Саме ці елементи зробили «The Binding of Isaac» класикою жанру рогаликів, яка вплинула на безліч майбутніх ігор.

«Dead Cells» [4] (рисунок 1.2) є ще одним знаковим проєктом у цьому жанрі, який вийшов у 2018 році. Гра поєднує швидкий і плавний бойовий геймплей у реальному часі з випадковою генерацією рівнів, що забезпечує динамічний і непередбачуваний ігровий процес.

Рисунок 1.2 – Зображення гри «Dead Cells»

Однією з ключових особливостей «Dead Cells» є її бойова система, яка містить елементи слешера з ухиленнями, парируваннями та можливістю змінювати зброю та здібності під час проходження. Це створює глибину боїв і вимагає від гравця швидкої реакції та стратегічного мислення. Також гра має систему поступового прогресу між спробами – хоча після смерті персонаж починає гру спочатку, деякі покращення залишаються, що дозволяє поступово розвивати героя. Візуально гра має яскравий піксель-арт у поєднанні з плавною анімацією, що надає їй унікального стилю. Завдяки такому поєднанню складності, динамічного бою та стильного оформлення «Dead Cells» здобула визнання як одна з найкращих рогалик-метроїдваній.

Ще одним прикладом сучасного рогалика є «Hades» [5] – гра, яка зробила справжній прорив у жанрі завдяки своїй глибокій сюжетній складовій і кінематографічній подачі (рисунок 1.3). Випущена у 2020 році студією Supergiant Games, вона поєднує високошвидкісний екшен, процедурну генерацію рівнів і захопливу історію, що розвивається під час кожного проходження.

Рисунок 1.3 – Зображення гри «Hades»

Гравець бере на себе роль Загрея, сина Аїда, який намагається вирватися з підземного царства, стикаючись із різними випробуваннями та перешкодами. Кожне нове проходження не лише впливає на механіку бою, а й змінює сюжет, дозволяючи персонажам взаємодіяти та відкривати нові деталі історії. Бойова система в «Hades» швидка та різноманітна, оскільки гравець може вибирати різні стилі бою завдяки широкому арсеналу зброї та божественним благословенням, що додають унікальні здібності. Окрім цього, гра відзначається якісною озвучкою, чудовим саундтреком і стильним візуальним оформленням, що створює атмосферу справжньої грецької міфології. Успіх «Hades» показав, що рогалики можуть мати не лише глибокий геймплей, а й насичену наративну складову.

Ще одним важливим представником жанру є «Spelunky» [6] – гра, яка робить акцент на дослідженні світу та високій складності. Випущена у 2008 році, вона стала однією з перших рогаликів, що популяризували концепцію «roguelike-платформерів». Основний ігровий процес зосереджений на проходженні випадково згенерованих рівнів, де гравець має досліджувати печери, шукати скарби та ухилятися від смертельних пасток (рисунок 1.4).

Рисунок 1.4 – Зображення гри «Spelunky»

Унікальність «Spelunky» полягає в складності та відсутності маркерів безпеки – будь-яка помилка може коштувати персонажу життя, змушуючи починати гру спочатку. Крім цього, гра містить глибоку систему фізики, що робить взаємодію з оточенням більш реалістичною. Наприклад, динаміт може підірвати шматок рівня, створюючи нові шляхи для проходження, або необережне використання бомби може спричинити руйнування, що зробить прохід неможливим. «Spelunky» є чудовим прикладом гри, яка балансує між випадковістю рівнів і можливістю гравця розвивати свої навички, що робить кожне проходження унікальним викликом.

Загалом аналіз цих ігор показує, що жанр рогаликів продовжує розвиватися, комбінуючи класичні механіки з новими інноваціями. «The Binding of Isaac» показав, як глибока система предметів може впливати на варіативність проходження, «Dead Cells» довів, що рогалики можуть мати динамічну бойову систему в стилі метроїдванії, «Hades» приніс у жанр сильний наративний елемент, а «Spelunky» розширив концепцію випадковості та дослідження світу. Всі ці проєкти вплинули на сучасні рогалики, формуючи основні принципи жанру та надихаючи нових розробників на створення унікальних ігрових механік. Таким чином, майбутнє жанру рогаликів виглядає перспективним, оскільки він має великий потенціал для подальшого розвитку та експериментів.

1.3 Баланс ігор жанру «Рогалик»

Ще одним важливим та ключовим аспектом всіх рогаликів є баланс між складністю та винагородою, оскільки саме він визначає, наскільки гра буде викликом для гравця і чи залишатиметься вона цікавою в довгостроковій перспективі. Якщо гра надто складна та не дає відчуття прогресу, гравці можуть швидко втратити мотивацію та бажання продовжити витрачати час на гру. Гравець повинен бути зацікавлений в якомусь прогресі. Але з іншого боку, якщо гра занадто легка, зникає основна привабливість жанру –

подолання труднощів і отримання задоволення від перемоги.

Різні рогалики використовують різні підходи для вирішення цієї проблеми. Наприклад, у «The Binding of Isaac» баланс складності досягається через систему випадкових предметів: одне проходження може бути легшим завдяки вдалим знахідкам, а інше – значно важчим, що змушує гравця адаптуватися. «Dead Cells» робить ставку на поступову еволюцію персонажа – навіть після поразки гравець може покращувати навички та відкривати нову зброю, що робить наступні спроби трохи легшими. «Hades» застосовує ще більш інноваційний підхід, вводячи «режим Бога», який після кожної смерті дає невеликі бонуси до виживання, дозволяючи навіть менш досвідченим гравцям рано чи пізно дійти до фіналу.

У «Forgotten Island: Hero Artifact» можна реалізувати подібний баланс через систему адаптивної складності – наприклад, після кількох невдалих спроб гравець міг би отримувати додаткові ресурси чи незначні бонуси для наступного проходження. Також важливо, щоб складність відчувалася справедливою – вороги та пастки не повинні здаватися надто непередбачуваними чи «дешево складними». Варіативність рівнів, можливість різних стратегій і поступове розширення доступних інструментів для виживання допоможуть підтримати інтерес гравців та забезпечити баланс між випробуванням і задоволенням від гри.

Проаналізувавши існуючі аналоги, можна побачити, що проєкт «Forgotten Island: Hero Artifact» має всі шанси стати унікальним представником жанру. Гра поєднує класичні механіки рогаликів із новими ідеями, такими як різні типи рівнів (рівні з босами, торговцями, бойові та зі знаком питання), які додають різноманітності ігровому процесу. Випадковість локацій, ворогів і подій забезпечує високу реіграбельність, а система прокачки персонажа дозволяє гравцям налаштовувати стиль гри відповідно до власних уподобань.

Унікальність «Forgotten Island: Hero Artifact» також полягає у поєднанні процедурної генерації зі структурованим підходом до рівнів, де

кожна локація має свій стиль, атмосферу та унікальні виклики. Система рівнів зі знаком питання додає несподіванки та ризику, що змушує гравців ретельно продумувати свої дії. Завдяки такому підходу гра може зацікавити не лише шанувальників класичних рогаликів, а й тих, хто цінує інновації та різноманітність у геймплеї.

1.4 Покращення ігор жанру «Рогалик»

Жанр рогаликів постійно еволюціонує, інтегруючи нові механіки та розширюючи свою аудиторію. У майбутньому можна очікувати, що ігри цього жанру дедалі більше комбінуватимуть випадкову генерацію з продуманими сюжетними лініями та глибшою взаємодією з ігровим світом. Такі проєкти, як «Hades», показали, що рогалики можуть мати сильний наративний компонент, який мотивує гравців на повторні проходження. Проект «Forgotten Island: Hero Artifact» може розширити цей підхід, інтегрувавши нелінійну розповідь, де вибір гравця впливатиме на розвиток подій, або унікальні сюжетні елементи, що відкриваються лише після певної кількості проходжень.

Окрім цього, розвиток жанру може йти шляхом більшої кастомізації персонажа та бойових механік. Наприклад, у майбутніх оновленнях «Forgotten Island: Hero Artifact» можна додати систему класів або змінні набори здібностей, які дозволять кожному проходженню відчуватися по-іншому. Також перспективним є введення кооперативного режиму, який би дозволяв гравцям взаємодіяти та проходити рівні разом, що могло б залучити ширшу аудиторію.

Ще одним напрямком розвитку може бути інтеграція процедурної генерації не лише рівнів, а й поведінки ворогів або механік босів, що зробить кожну битву ще більш унікальною та непередбачуваною. В цьому можуть допомогти нові технології та інтегрування ШІ в ігрові проекти. Зараз компанія NVIDIA [7] вже тестують таку систему під назвою «NVIDIA ACE»,

щоб в майбутніх іграх був реалізований розвинутий штучний інтелек, який буде постійно змінювати свою поведінку задля нового досвіду для гравців. Саме зараз NVIDIA розширює ACE від розмовних NPC до автономних ігрових персонажів, які використовують штучний інтелект, щоб сприймати, планувати та діяти як люди. Завдяки генеративному штучному інтелекту ACE створить живі динамічні ігрові світи з компаньйонами, які розуміють і підтримують цілі гравців, і ворогами, які динамічно адаптуються до тактики гравців.

Щодо потенційної аудиторії, гра може зацікавити не лише фанатів класичних рогаликів, а й тих, хто любить складні випробування та експериментує зі стратегіями. Завдяки трендам, таким як популярність ігор із високою реіграбельністю та змагальними елементами, можна зробити акцент на створенні унікальних комбінацій здібностей і викликах, які дозволять гравцям змагатися за найкращі результати. Крім того, сучасні геймери цінують естетично привабливі візуальні стилі, тому розширені анімації, детальне опрацювання середовища та різноманіття ефектів можуть стати важливими факторами успіху гри.

2 АНАЛІЗ ТА ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Графічний двигун Unreal Engine

В своєму проекті я обрав двигун Unrael Engine 5. Він чудово підходить як для розробки невеличких ігор, так і для великих ААА проектів з бюджетом в десятки а то і сотні мільйонів доларів. Зраз, Unreal Engine є одним із найпотужніших і найвідоміших ігрових двигунів у світі. Його створення почалося в середині 90-х років завдяки зусиллям американського програміста Тіма Суїні, засновника компанії Epic MegaGames (згодом перейменованої в Epic Games). Суїні, маючи досвід розробки простих ігор на мові програмування Pascal, поставив перед собою амбітне завдання — створити рушій, який би дозволяв розробляти високоякісні тривимірні ігри з просунутою графікою та фізикою.

Робота над Unreal Engine розпочалася в 1995 році, коли Суїні почав експериментувати з 3D-графікою для гри, яка пізніше отримала назву Unreal. Головною ідеєю було розробити рушій, який би мав гнучкі можливості для створення як однокористувацьких, так і багатокористувацьких ігор. Важливими особливостями рушія стали покращене освітлення, текстурування та система рівнів, яка дозволяла дизайнерам створювати складні світи без необхідності писати код. Одним із ключових нововведень було використання мови UnrealScript, яка давала змогу програмістам і дизайнерам легко створювати та змінювати ігрову логіку.

Першою грою, яка використала Unreal Engine, стала Unreal (1998) [8], що вразила гравців своєю на той час реалістичною графікою, динамічним освітленням і складним штучним інтелектом ворогів. Цей науково-фантастичний шутер від першої особи був розроблений компанією Epic MegaGames у співпраці з Digital Extremes і став одним із найбільших технологічних проривів свого часу.

Гравець опинявся в ролі в'язня космічного корабля, який зазнав аварії на чужій планеті, і змушений був виживати у ворожому середовищі, повному інопланетних істот, загадок і смертельних небезпек (рисунок 2.1).

Рисунок 2.1 – Гра Unreal (1998)

Гра Unreal [9] відзначалася неймовірною на той час візуальною якістю: рушій підтримував кольорове освітлення, динамічні тіні, об'ємне середовище, а також просунуту систему частинок і відбиттів. Середовище гри було деталізованим і масштабним, що створювало ефект повного занурення. Особливу увагу привертала система освітлення, яка змінювалась у режимі реального часу, створюючи атмосферу напруги й загадковості. Усе це вивело гру на абсолютно новий рівень у порівнянні з іншими шутерами того періоду, такими як Quake II чи Doom.

Ще однією перевагою гри був складний штучний інтелект супротивників. Вороги в Unreal не просто йшли на гравця напролом — вони маневрували, ховалися за укриттями, відступали й координували свої дії, що створювало динамічні й непередбачувані бої. Це робило геймплей захопливим і змушувало гравця обдумано підходити до кожної сутички. Крім

того, у грі був представлений потужний редактор рівнів – UnrealEd, що дозволив користувачам самостійно створювати нові карти та модифікації, чим започаткував активну моддерську спільноту.

Успіх Unreal був зумовлений не лише технічною перевагою, а й високою якістю проєктування ігрового процесу, що поєднував атмосферність, напруження та свободу дослідження. Гра отримала численні нагороди, визнання критиків і гравців, а також заклала фундамент для популяризації Unreal Engine як комерційного продукту, який у майбутньому використовували десятки провідних ігрових студій.

Величезний успіх гри сприяв популяризації двигна, і незабаром Epic Games почала ліцензувати його іншим розробникам. Наступні версі двигуна, зокрема Unreal Engine 2 (2002), Unreal Engine 3 (2006) і Unreal Engine 4 (2014), значно вдосконалювали технології, додаючи підтримку передових графічних ефектів, фізичних симуляцій та мультиплатформеність.

Unreal Engine 5, випущений у 2021 році, став черговим проривом завдяки технологіям Nanite (віртуалізованої геометрії) і Lumen (реалістичного глобального освітлення в реальному часі). Сьогодні двигун використовується не лише в ігровій індустрії, а й у кіновиробництві, архітектурі, віртуальній реальності та багатьох інших сферах. Завдяки своїй гнучкості та потужності Unreal Engine залишається одним із головних інструментів для розробки інтерактивного контенту в сучасному цифровому світі.

Двигун містить в собі багато плюсів та мінусів. Розберемося детальніше.

Основні переваги двигуна:

- Engine Source Code. При роботі з двигунами може виникнути момент, коли потрібно доопрацювати двигун під вашу задачу або спосіб використання. Хоча це трапляється рідко, але для великих компаній це дуже важливий параметр в виборі двигуна. Також це полегшує створення різних плагінів для пришвидшення роботи програмістів та дизайнерів;

- візуальний скриптинг і підтримка традиційного кодингу. Візуальний скриптинг чудово підходить для роботи художників і скорочує багато часу, який знадобиться, наприклад, для того, щоб прочитати C++ [10]. Код на C++ також чудовий, якщо ви, як і я, віддаєте перевагу потужності та читабельності мові програмування C++. Це дозволяє використовувати кілька методів для вирішення однієї і тієї ж проблеми. Більш того, ви можете комбінувати візуальний скриптинг (Blueprints) з програмуванням на C++;

- легко зрозуміти художника. Для людей, які не знайомі з програмуванням це великий плюс. Unreal дуже-дуже дружній до художників. Він має схожі робочі процеси для сіток і матеріалів, які можна використовувати за замовчуванням з Maya, blender або іншим 3D-програмним забезпеченням. Особливо з такими інструментами, як Live Link, це дуже корисно і легко для художників.

Основні недоліки двигуна:

- маркетплейс асетів це кількість ресурсів таких як 3д моделі та музики досить невеличка. Хоча вони можуть бути чудовими для початку проекту, у більшості випадків не слід покладатися на них, щоб створювати проект лише за допомогою купівлі ресурсів у інших художників. В деяких випадках люди надіються лише на них, це може погано вплинути на проект;

- документація в Unreal завжди була поганою, і такою і залишається. Зазвичай вона просто відсутня або зовсім коротка, не охоплює всю тему. Найкраще рішення, яке я знайшов, – це задавати запитання на форумах спільноти та сподіватися, що хтось на них відповість, але в більшості випадків мені, як правило, доводиться дивитися на код двигуна та намагатися знайти рішення в ньому. Це дійсно дає пасивну перевагу, оскільки ви розумієте більше концепцій внутрішньої роботи Unreals, але це не ідеально для людей, які мало розбираються в програмуванні;

- розмір спільноти та кількість навчального матеріалу. Зараз розмір спільноти все ще за малий, щоб полегшити навчання та адаптацію для нових користувачів, але за останній час приріст реально помітний та кількість

навчального матеріалу хоч і не швидко, але стабільно збільшується.

Unreal Engine 5 надає розробникам широкий спектр можливостей та вибору написання додатку. В цьому графічному двигуні присутні 2 способи написання логіки ігор: Blueprints та програмування на C++.

The Blueprint Visual Scripting у Unreal Engine – це мова візуального програмування, яка використовує інтерфейс на основі вузлів та нодів для створення елементів ігрового процесу. Робочий процес на основі вузлів надає дизайнерам широкий спектр концепцій та інструментів сценаріїв, які зазвичай доступні лише програмістам. Крім того, спеціальна розмітка Blueprint, доступна в реалізації C++ Unreal Engine. Вона дає програмістам можливість створювати базові системи, які дизайнери можуть розширювати. Як і у випадку з багатьма поширеними мовами скріптингу, розробник може використовувати систему для визначення об’єктно-орієнтованих класів або об’єктів в проекті. Цю систему разом із об’єктами часто називають просто «кресленнями».

Unreal Engine забезпечує надійну структуру для програмістів C++ [11], щоб допомогти втілити їх бачення в життя. Рушій має відкритий вихідний код, що дозволяє програмістам змінювати його на глибокому рівні, оптимізувати продуктивність і адаптувати рушій до специфічних потреб проєкту. Це особливо корисно для створення унікальних механік, вдосконалення рендерингу та оптимізації гри для різних платформ.

За допомогою C++ розробник може робити що завгодно: змінювати та корегувати код двигуна, писати свої плагіни, реалізовувати інтерфейси, класи, ігрову логіку та багато іншого. Unreal Engine використовує потужну систему класів та макросів, яка спрощує інтеграцію C++ із рушієм, дозволяючи програмістам взаємодіяти з внутрішніми механізмами через спеціальні макроси (наприклад, UCLASS, UFUNCTION, UPROPERTY). Це забезпечує ефективну роботу з об'єктами та сутностями гри, а також інтеграцію C++ коду з Blueprint для зручного керування логікою проєкту.

C++ – це потужний інструмент для створення проєкту будь якої

складності, оскільки він дає змогу працювати з пам'яттю, багатопотоковістю, оптимізацією ресурсів та створенням низькорівневих систем. Крім того, за допомогою C++ можна реалізовувати власні системи штучного інтелекту, фізичних взаємодій та мережевого коду, що критично важливо для складних багатокористувацьких ігор.

Завдяки потужному API та широкій документації Unreal Engine дозволяє програмістам створювати високоякісний код, який легко масштабувати та підтримувати. Це робить рушій одним із найкращих варіантів для розробників, які прагнуть мати повний контроль над своїм проєктом та досягати максимальної продуктивності.

Найкращій спосіб в реалізації проекту це комбінування C++ та Blueprints. Це дозволяє в декілька разів пришвидшити створення проекту, так як C++ дозволяють швидко та ефективно створювати ігрові механіки, в той час як візуальний скриптинг надає можливість зручно налаштовувати візуальну частину проекту, створювати матеріали та ініціалізувати параметри, що відповідають за графічну частину.

2.2 Основна логічна частина двигуна Unreal Engine

Unreal Engine має об'єктно-орієнтовану архітектуру [12], де всі елементи гри представлені у вигляді об'єктів із певними властивостями та поведінкою. Ця структура базується на ієрархічній системі класів, що дозволяє розробникам легко створювати, модифікувати та розширювати функціонал гри. Одними з ключових понять у рушії є Актор (Actor) та Світ (World), які визначають основну логіку існування об'єктів у грі. На русунку 2.2 зображена структурна схема базових класів двигуна. Кожен об’єкт у цій системі має життєвий цикл, який керується рушієм і визначає його взаємодію з іншими об’єктами та середовищем.

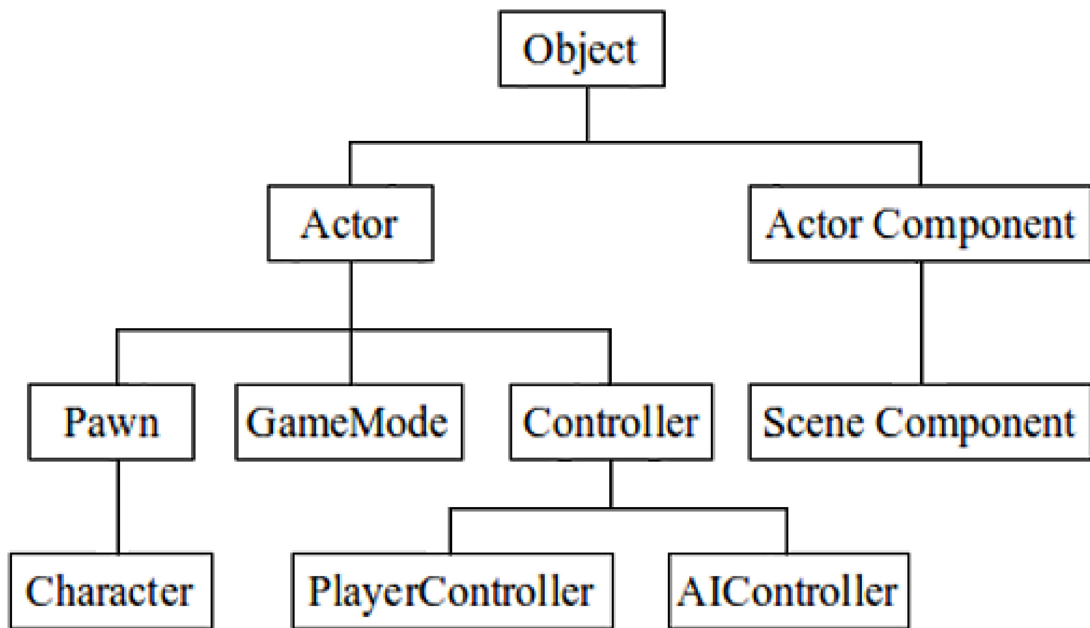

Рисунок 2.2 – Структурна схема ієрархії класів двигуна

Object – це базовий клас у системі Unreal Engine, з якого успадковуються всі інші класи рушія, включно з акторами. Об'єкти не можуть існувати безпосередньо в ігровому світі, але вони використовуються для зберігання даних, обробки логіки та взаємодії між компонентами гри. Прикладами об'єктів можуть бути компоненти акторів, керовані ресурси, такі як текстури або звукові ефекти, а також менеджери ігрових систем, що працюють у фоновому режимі.

Актор (Actor) – це базовий об'єкт в движкі Unreal Engine, з якого складаються всі інші об'єкти. Актори використовуються для створення будь-яких об'єктів у грі: від статичних елементів, таких як стіни або меблі, до складніших, наприклад, інтерфейсних тригерів чи взаємодіючих об'єктів. Вони можуть мати компоненти, наприклад, сітки (Meshes), колайдери чи світло, які додають їм фізичні властивості чи вигляд.

Світ (World) – це середовище, у якому знаходяться всі об'єкти гри. Світ в Unreal Engine відповідає за управління рівнем, розташуванням акторів і взаємодією між ними. У кожному світі є кілька ключових систем, наприклад, фізика, освітлення та управління часом. Він також містить точку входу (Player Start), з якої починається гра.

Персонаж (Character) – це спеціальний тип актора, створений для контролю руху і взаємодії з ігровим світом. Персонаж має вбудований компонент Character Movement, який спрощує реалізацію таких функцій, як ходьба, біг, стрибки та даші. В більшості проєктів персонаж пов’язаний із камерою для надання гравцю певної перспективи (від першої чи третьої особи). Character Movement – це вдубований зручний клас, який реалізує в собі параметри для переміщення персонажів в світі. Розробнику потрібно його лише налаштувати для отримання вже чудового робочого результату, витративши на все про все декілька хвилин.

Пішка (Pawn) – це базовий клас для об'єктів, якими може керувати гравець або штучний інтелект. У порівнянні з персонажем, пішка не має вбудованої логіки руху, що робить його більш гнучким для створення нестандартної механіки, наприклад, транспортних засобів, літаючих об’єктів і так далі.

Компоненти (Components) – це складові частини акторів, які визначають їх вигляд і поведінку. Вони надають акторам зручні можливості та реалізують в собі багато функцій, які по суті інкапсулюються в компонент і не засмічують самого актора. Наприклад, Static Mesh Component відповідає за візуальне представлення об'єкта, Box Collider – за область взаємодії, а Audio Component – за відтворення звуків. Компоненти можна комбінувати для створення складних об'єктів.

Контролер (Controller) – це базовий клас в архітектурі Unreal Engine, який виконує роль посередника між гравцем або штучним інтелектом і ігровим світом. Контролер не має фізичної присутності на рівні, тобто він сам не візуалізується і не взаємодіє з фізичним середовищем напряму, але він керує персонажем, яким "володіє" (possess). Основна мета контролера це керувати логікою поведінки: отримувати ввід, ухвалювати рішення й передавати їх відповідному актору. У структурі рушія контролер є ключовим елементом, що визначає, як саме об’єкт буде діяти у відповідь на події.

Із базового класу Controller успадковуються два головні типи: Player

Controller і AI Controller. Player Controller відповідає за обробку вводу від гравця — клавіатури, миші, геймпада чи сенсорного екрану. Цей клас перетворює сигнали керування на команди для персонажа: рух, стрибки, стрільбу, взаємодії з об'єктами. Також він відповідає за камеру, інтерфейс користувача, прив'язку клавіш і мультіплеєрну логіку. Він виконує роль "мозку" гравця в грі, ізольованого від фізичної моделі персонажа.

AI Controller (контролер штучного інтелекту) [13] виконує аналогічну роль, але для ботів та неігрових персонажів. Він ухвалює рішення на основі логіки, заданої розробником, наприклад через Behavior Tree або спеціальний скрипт. AI Controller може слідкувати за гравцем, патрулювати територію, реагувати на звуки або зміни в навколишньому середовищі. Його завдання — автономно керувати актором, забезпечуючи реалістичну поведінку без участі гравця.

Ігровий режим (Game Mode) – це клас, який визначає правила гри, наприклад, тип гравця, логіку спавну, поведінку при завершенні гри тощо. Game Mode також керує взаємодією між різними системами гри.

Рівень (Level) – це конкретний стан світу, який містить усі актори, ландшафт, освітлення та тригери. Рівні можуть бути завантажені окремо або разом із іншими (підрівні), що дозволяє створювати великі та деталізовані світи.

Камера (Camera) – це об'єкт, який визначає, що бачить гравець. Камери часто прикріплюються до персонажа або пешнеля для створення перспективи від першої чи третьої особи. З їх допомогою можна налаштувати різні ефекти, як-от глибина різкості чи кінематографічне зображення.

Ці елементи є базовими класами та об’єктами для створення гри і дозволяють побудувати логіку та механіку будь-якої складності. Від них розробник вже базує свою гру.

Саме за допомогою цих класів я і реалізував свій проект, створюючи від них похідні класи персонажів, контролерів і таке інше.

2.3 Фреймворк Gameplay Ability System

Unreal Engine надає широку підтримку плагінів і фреймворків. Щоб пришвидшити розробку проекту, я використовував фреймворк Gameplay Ability System, який був створений розробниками двигуна.

Gameplay Ability System [14] (Система ігрових здібностей, або коротко GAS) – це основа для створення атрибутів, здібностей і взаємодій, якими Актор може володіти та активувати їх. Система розроблена для адаптації до різноманітних проектів, орієнтованих на ігровий процес, таких як рольові ігри (RPG), пригодницькі ігри та багатокористувацькі онлайн-ігри Battle Arenas (MOBA).

Gameplay Ability System надає розробнику можливість швидко реалізувати автоматизовану систему атрибутів та здібностей, що пришвидшить створення проекту та зменшить кількість багів.

Компонент Ability System включає всі базові функції, які реалізує компонент Actor. GAS реалізує власний інтерфейс для доступу та взаємодії з системою системи здібностей ігрового процесу.

Розробник має можливість створити активні або пасивні ігрові здібності для акторів, які супроводжуються ігровими механіками, візуальними ефектами, анімацією, звуками та іншими елементами.

GAS використовує атрибути та набори атрибутів, які зберігають, обчислюють і змінюють значення, пов’язані з ігрою. Розробник має можливість змінювати атрибути за допомогою Gameplay Effects [15] (ігрових ефектів), які надають метод безпосередньої зміни значень атрибутів у дизайні проекту. Gameplay Effects містять компоненти , які визначають, як саме поводиться ці ігрові ефекти.

Ability Tasks (UAbilityTask) – це спеціалізована форма класу ігрових завдань, яка працює з ігровими здібностями. Ігри, які використовують Gameplay Ability System, зазвичай включають різноманітні користувацькі завдання здібностей, які реалізують їхні унікальні ігрові особливості. Вони

виконують асинхронну роботу під час виконання ігрової здатності та мають можливість впливати на потік виконання, викликаючи делегати у рідному коді C++ або переміщаючись крізь один або більше вихідних пінів виконання, як-от Blueprints.

Використовуючи цю систему, розробник може створювати такі атрибути та здібності різних складностей. Від простих атак з ефектами до важких та автоматизованих здібностей, як от наприклад здібності в моєму проекті: мій персонаж має ряд збідностей, які не тільки змінюють атрибути, а й можуть відштовхнути противника, вплинути на фізику в світі або створити звуковий та візуальний ефект.

3 ОПИС ЗАСТОСУНКУ

Гра «Forgotten Island: Hero Artifact» це аркадний проект, цільова аудиторія якого комп'ютери та ігрові консолі XBox. Ця гра відноситься до жанру рогаликів. Жанр рогалик (roguelike) - це тип відеоігор, основою яких є процедурно згенеровані світи, постійна смерть персонажа (permadeath) та значний акцент на дослідженні, тактиці й прийнятті рішень. Назва жанру походить від класичної гри Rogue (1980), яка задала багато ключових характеристик для подібних проєктів.

Основа ігрового процесу мого проекту полягає в нескінченному проходженні рівнів та прокачки персонажа, з метою отримання всіх досягнень та проходження всіх босів в грі. Гравець має можливість вибору ходів під час проходження кожного рівня. Самі ж рівні постійно випадкові та відрізняються лише по стилю локації. Вороги та боси на кожних рівнях теж постійно різні, в залежності від стилю самого рівня. Гравець має можливість прокачувати свого персонажа в ході проходження, отримуючи бали для покращення параметрів свого персонажа, цим самим посилюючи його.

В грі існують рівні 4 типів:

- рівні з босами – це рівень де є дуже сильний ворог з великою кількістю здібностей;
- бойові рівні – де гравцю просто потрібно зачистити локацію для проходження на наступний рівень та отримання ігрової валюти;
- локації з торговцем – це випадкові локації де гравець може за ігрову валюту поповнити припаси, купити нові властивості;
- локації зі знаком питання – це локації в яких може відбутися випадкова подія, вона може бути як і корисною для гравця так і навпаки – негативно вплинути на подальше проходження рівнів.

Також, під час проходження рівнів, одною із основних механік гри є отримання нових здібностей або прокачка існуючих за рахунок підвищення

рівня персонажа. Рівень персонажа підвищуються при знешкодженні ворожих юнітів. Після підвищення рівня, персонажу надається можливість обрати одну з трьох випадкових здібностей. Цим самим під час проходження, персонаж гравця постійно стає сильнішим та підвищує свої шанси на виживання.

Якщо ж гравець помирає, то гра починається спочатку, але атрибути та досягнення персонажів зберігаються – вони прокачуються назавжди.

Для створення атмосфери гри використано різноманітний музичний супровід та візуальні ефекти, які підкреслюють унікальність кожної локації. Гравцю пропонується вибір кількох персонажів, кожен із яких має свої унікальні здібності та стиль гри, що додає глибини ігровому процесу.

Також, в грі реалізована система досягнень, яка мотивує досліджувати кожен куточок світу та експериментувати з різними стратегіями.

В грі присутнє дуже просто та зрозуміле кожному меню (рисунок 3.1), в якому присутні 3 основні пункти: почати нову гру, налаштування гри, та вихід. Якщо ж гравець вже почав гру, то буде присутня четверта кнопка – «Продовжити гру». Вона буде першою в списку, та виділятися візуально дизайном та кольором.

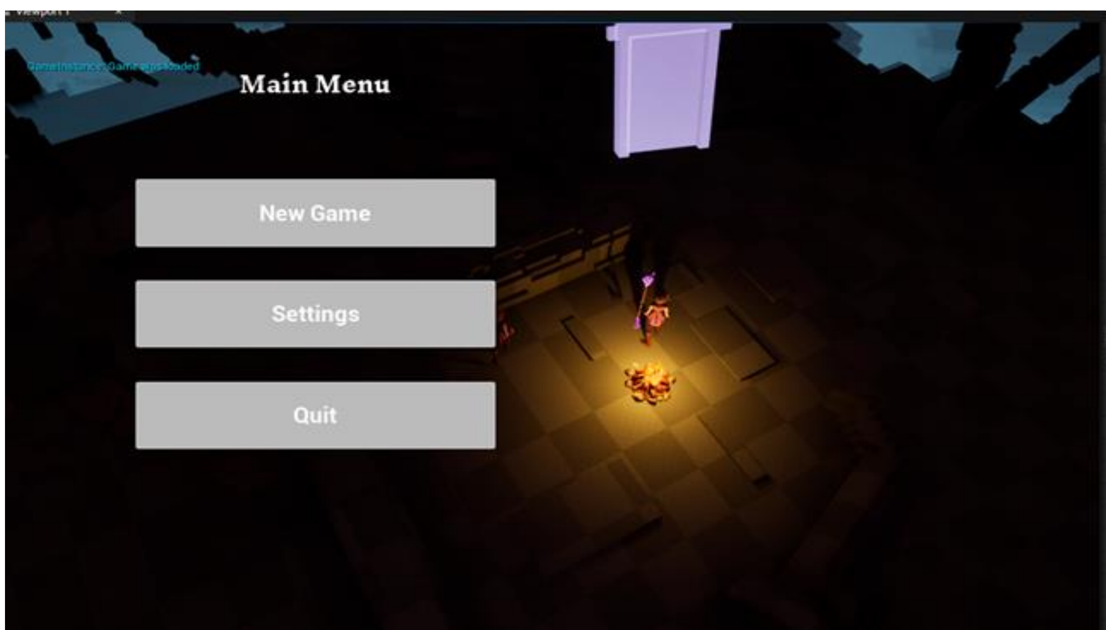

Рисунок 3.1 – Головне меню гри

Після натискання кнопки з текстом «Нова Гра», за допомогою анімації камера підлетить до персонажів, щоб їх було краще видно. На екрані з’являться кнопки інтерфейсу для перемикання між персонажами та кнопка «Вибрати Персонажа» відповідно для вибору (рисунок 3.2). Такий підхід дозволяє створити більш захопливу та кінематографічну подачу початку гри, забезпечуючи гравцеві перше занурення у візуальний стиль і атмосферу проєкту.

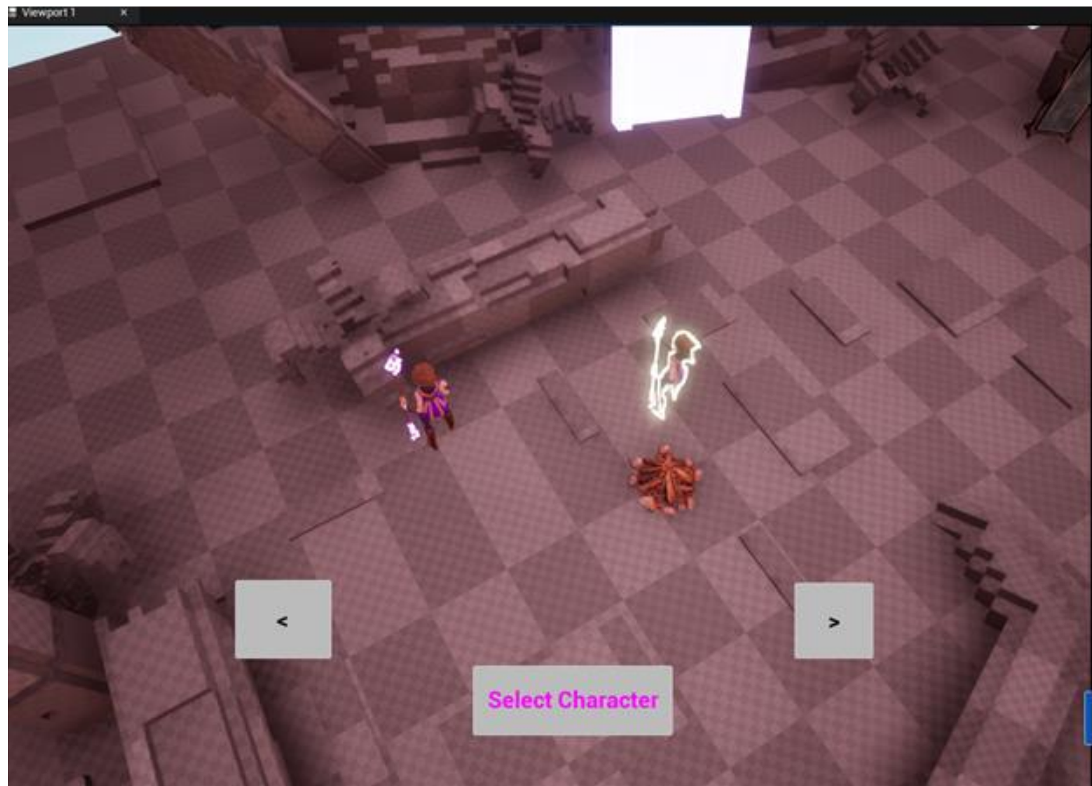

Рисунок 3.2 – Вибір персонажа

Перемикання між персонажами реалізоване з використанням інтерфейсних кнопок, які змінюють поточну фокусну точку камери та відображають інформацію про обраного героя — його ім’я, характеристики чи візуальні особливості. Це дає змогу гравцеві уважно ознайомитися з кожним доступним варіантом перед початком гри. Анімація камери, синхронізована з переходами між героями, створює плавний і приємний візуальний ефект.

Кнопка «Вибрати Персонажа» виконує ключову функцію підтвердження вибору. Після її натискання запускається логіка ініціалізації гри: завантажуються потрібні дані, обраний персонаж стає головним керованим об'єктом, а інтерфейс переходить у режим основного геймплею. Це дозволяє зробити процес вибору інтуїтивно зрозумілим і безперервним.

Крім цього, інтерфейс створений таким чином, щоб його можна було легко масштабувати — за потреби додати нових персонажів, оновити дизайн або змінити структуру взаємодії з користувачем. Завдяки гнучкій системі UI Unreal Engine, цей етап гри легко адаптувати як для ПК, так і для інших платформ за потреби.

Також в грі присутнє магічне дзеркало для перегляду та прокачки атрибутів персонажів (рисунок 3.3).

Рисунок 3.3 – Меню з переглядом та прокачкою атрибутів

Але переглядати одні атрибути та характеристики персонажів це замало. Так як в грі присутньо багато різних здібностей, гравцю потрібно

переглядати певну інформацію та бачити певні параметри цих здібностей. Для вирішення цього питання в гру була запроваджена магічна книга (рисунок 3.4), яка дозволяє переглядати всі здібності в грі.

Рисунок 3.4 – Книга для відображення спелів в грі

Також вона універсальна, тому дизайнер сам додає в неї здібності, які хоче показувати для гравця. Ця універсальність дуже важлива, адже в грі присутні певні здіності, які працюють весь час та моніторять стани персонажів, щоб зручно оновлювати анімації. Так як ці здібності суто технічні, гравець їх бачити в інтефрейсі не повинен.

Як зазначено раніше, в грі користувач сам обирає який рівень проходити. Для такої можливості було реалізовано 3-Д рівень для вибору наступних ходів (рисунок 3.5). В результаті користувач бачить, який тип рівня його чекає та вибирає шляхи проходження гри. Такий підхід забезпечує нелінійну структуру ігрового процесу, дозволяючи кожному гравцеві

формувати свій власний маршрут проходження, що значно підвищує реіграбельність та індивідуальність ігрового досвіду.

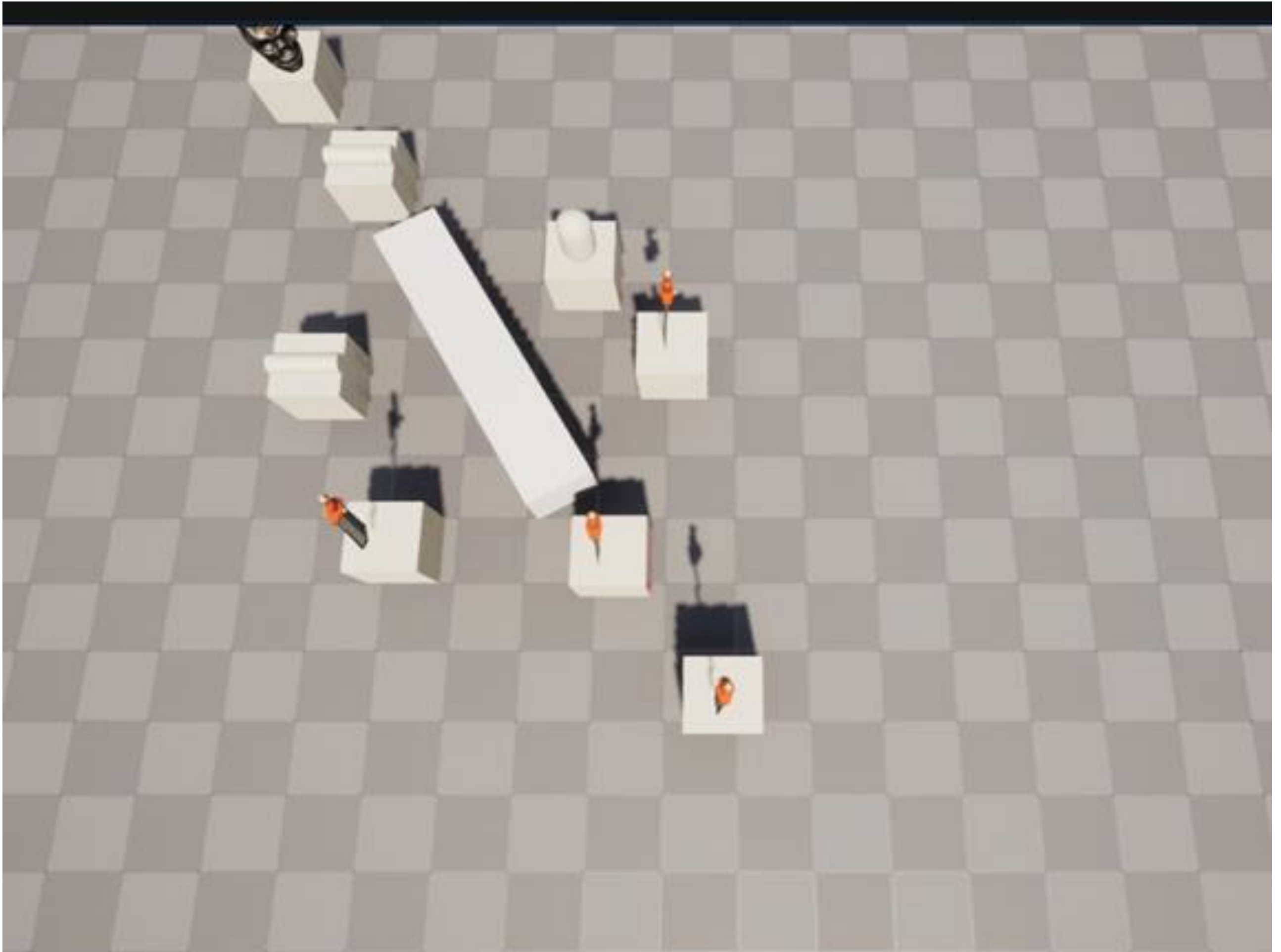

Рисунок 3.5 – Меню вибору рівнів

3D-рівень вибору виконано у вигляді інтерактивного простору, де генеруються кнопки, що відповідають певним рівням. Гравець у вигляді духа може вільно переміщатися або оглядати середовище камери, що створює відчуття дослідження та відкритості. Рівні позначені унікальними візуальними елементами, які натякають на їхній стиль, складність або тип завдань, що очікують попереду (наприклад, бойовий рівень, головоломка чи діалоговий сегмент з торговцем).

Як вже описувалося раніше, гравець, перемагаючи ворогів, отримує бали для покращення рівня персонажа. Під час отримання нового рівня, гравцю надається можливість вибрати 1 з 3 випадкових здібностей. Це може

бути як нова здібність, так і прокачка вже існуючої. На Рисунку 3.6 зображено результат отримання нового рівня та прокачку вже існуючої здібності.

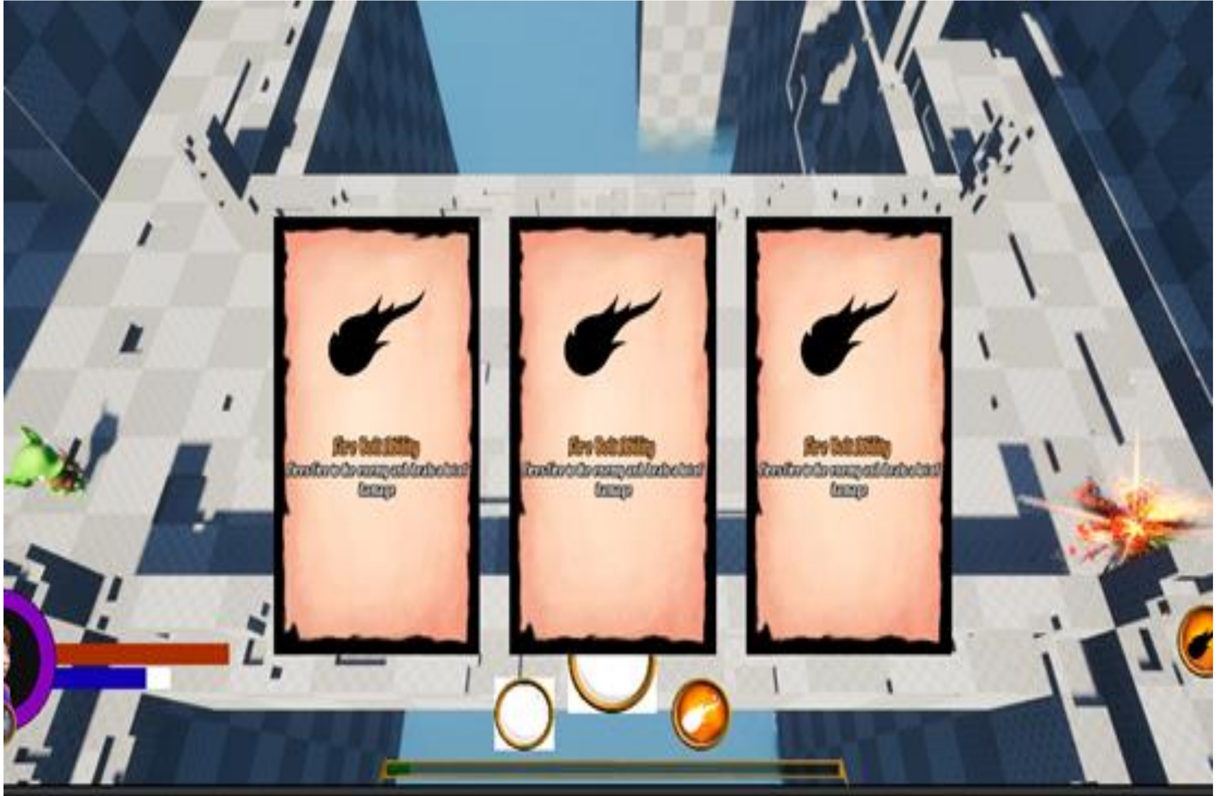

Рисунок 3.6 – Результат збільшення рівня та отримання випадкових здібностей

Випадковість у виборі здібностей додає елемент непередбачуваності, що робить кожен рівень захопливішим і не дозволяє гравцеві створити шаблонну стратегію. Гравець повинен зважено приймати рішення: обрати нову здібність, яка може кардинально змінити стиль бою, чи посилити вже наявну, підвищивши її ефективність.

Кожна здібність має кілька рівнів прокачування, і при кожному підвищенні вона може змінювати свої параметри – наприклад, збільшувати зону дії, силу впливу або зменшувати час перезарядки. Це дозволяє створювати сильні синергії між здібностями, якщо гравець будує розвиток персонажа в одному напрямку. Також передбачено візуальне та звукове

супроводження моменту вибору, що підкреслює важливість цього етапу та емоційно підсилює відчуття прогресу.

Так як в грі присутні різні типі здібностей, вони можуть поділятися на: пасивні, активні, звичайні та ультимативні. На рисунку 3.7 зображено результат використання ультимативної здібності на ворожому персонажі. Ультимативні здібності набагато сильніші звичайних, але вони мають довгу перезарядку, тому використовувати їх потрібно з розумом.

Рисунок 3.7 – Результат використання ультимативної здібності

З технічної точки зору, система здібностей реалізована через окремі класи, які можна легко додавати чи модифікувати. Це робить систему розширюваною та зручною для балансу. Крім того, вона інтегрована з інтерфейсом гравця, де за допомогою підказок гравець може швидко ознайомитись з ефектами кожної здібності перед вибором, що покращує користувацький досвід.

Також, гравець може спілкуватися з іншими неігровими персонажами. На рисунку 3.8 зображене діалогове вікно персонажа з іншим NPC – це неігровий персонаж (англ. NPC – Non-Playable Character) [13], який є

частиною ігрового світу, але не керується гравцем. В грі НПС зазвичай взаємодіють з гравцями, надають інформацію, завдання, товари чи інші елементи, які доповнюють сюжет гри. Це допомагає гравцям в ігровій формі пояснити якусь механіку гри, або надати підказку.

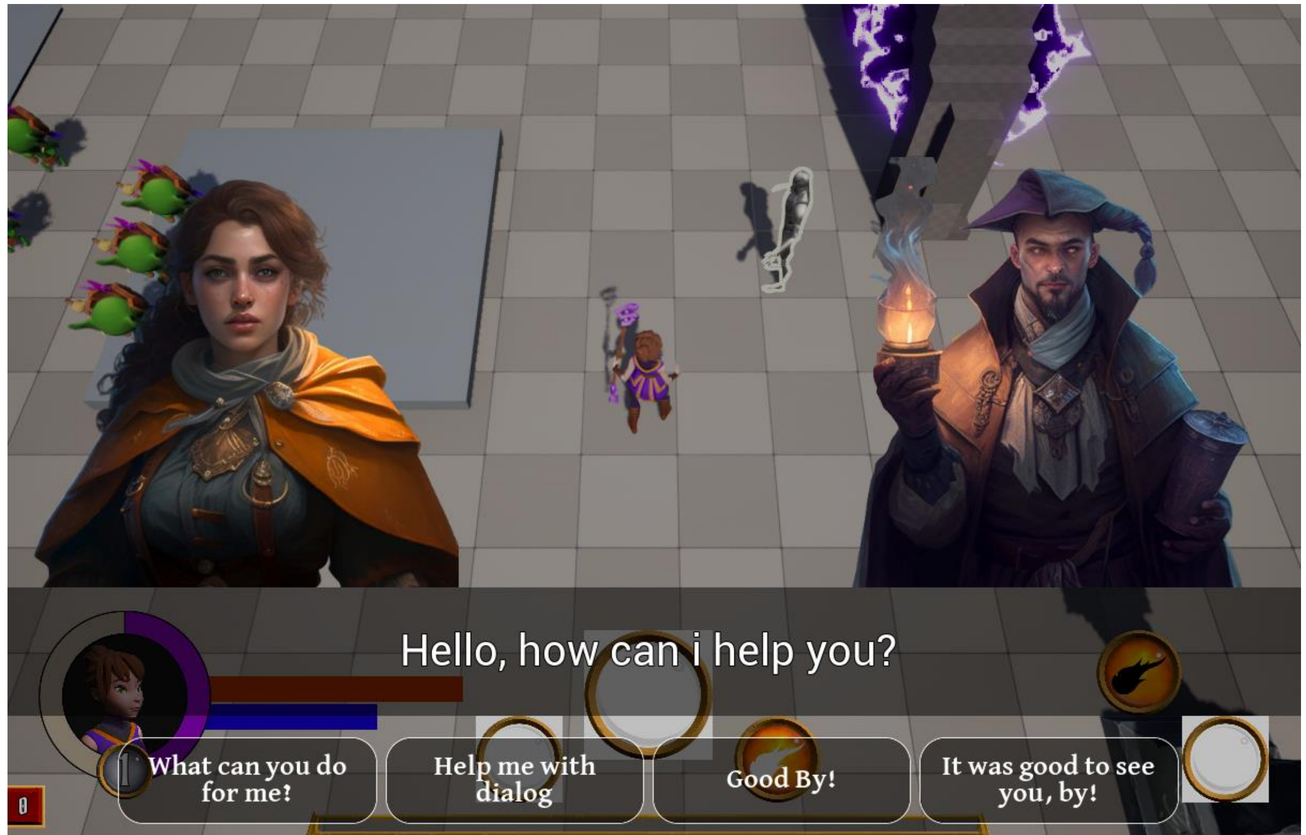

Рисунок 3.8 – Діалогове вікно персонажа з NPC

Діалогова система реалізована у вигляді інтерактивного вікна з варіантами відповідей, які гравець може обирати, впливаючи на хід розмови. Деякі NPC можуть мати гілки діалогу, які залежать від попередніх рішень гравця або його досягнень, що робить комунікацію живішою та більш варіативною. У більш складних випадках, відповіді NPC можуть змінювати ставлення до гравця або відкривати нові шляхи проходження місій.

Окрім функціонального навантаження, NPC відіграють роль живих «учасників» ігрового світу, які створюють враження справжньої, динамічної екосистеми. Вони можуть мати унікальні анімації, озвучення, поведінку та навіть підстраюватися під рівень, наякому знаходяться. Це робить світ більш

переконливим та цікавим.

З технічного боку, NPC реалізовані через акторів, якими керує AI Controller, що забезпечує їхню автономну поведінку. Поведінкові дерева (Behavior Trees), а також система діалогів, побудована на Blueprints та кастомних C++ класах дозволяють налаштовувати їх логіку без потреби у переписуванні основного коду. Кожен НПС має віртуальне дерево з діалогами та можливими варіантами відповідей нан них. Це дозволяє гнучко реалізовувати різні диалоги без потреби змінювати самих NPC – існує лише один параметр в класі неігрових персонажів, який і відповідає за діалогове дерево. Завдяки такій гнучкій архітектурі, NPC легко масштабуються й можуть бути адаптовані для будь-яких цілей – від простих торговців до ключових сюжетних фігур.

4 РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

4.1 Реалізація базових класів застосунку

Створення гри перш за все починається з реалізації базових функцій ігрового персонажу, налаштування камери, переміщення та створення ключових компонентів для персонажів. Отже, спочатку створимо базовий C++ клас AFIHACharacterBase, який має в собі реалізацію всіх базових функцій, які в собі мають всі персонажі гри. Від нього вже реалізуємо два похідних класи: AFIHACharacterPlayer, AFIHACharacterEnemy. Перший клас відповідає за реалізацію функцій гравця, а другий за ворогів та штучний інтелект. Але ці класи мають в собі багато спільного, наприклад атрибути, компонент для атрибутів, флаги для контролю стану пресонажу та багато іншого. Саме тому нам потрібно мати базовий клас для пришвидшення розробки та полегшення коду.

Далі, потрібно створити ще один важливий ключовий клас – Player State. Це клас в Unreal Engine, який використовується для зберігання інформації про гравця, незалежно від його контролера чи персонажа. Це особливо корисно в мережевих іграх, оскільки дозволяє зберігати дані гравця навіть у випадку зміни його персонажа або відключення від сервера. APlayerState успадковується від AInfo, що означає, що цей клас не має власної логіки оновлення кожен кадр (tick), а лише зберігає та передає важливі дані. APlayerState грає важливу роль, а саме створює компонент UFIHAAbilitySystemComponent – клас, який містить в собі всі атрибути персонажу, тобто кількість здоров'я, мани, швидкість руху персонажу, кількість броні та інші атрибути.

Ability System Component (ASC) [13] – це основний компонент у Gameplay Ability System (GAS), який використовується для управління здібностями персонажа, обробки ефектів та взаємодії з атрибутами. Він

дозволяє легко додавати, активувати та керувати здібностями персонажа, включаючи атаки, спеціальні навички, бафи, дебафи та інші ефекти, які можуть застосовуватися до нього під час гри. UFIHAAbilitySystemComponent є ключовим класом, без нього неможлива реалізація атрибутів та здібностей. Але цей клас не має в собі реалізацію атрибутів. Саме для цього потрібен другий не менше важливай клас під назвою Attribute Set.

UFIHAAttributeSet – це клас, який містить в собі всі атрибути персонажів, обробляє їх зміну та в залежності від різних значень які вже реалізує програміст, викликає різні функції на персонажі, такі як смерть, спрагу, усталість та багато іншого. Отже, в цьому класі нам потрібно реалізувати атрибути, в лістингу 4.1 зображено приклад реалізації атрибуту здоров'я персонажу.

Лістинг 4.1 – Приклад створення класу UFIHAAttributeSet (файл UFIHAAttributeSet.hpp)

```
UCLASS()
     class MYPROJECT_API UFIHAAttributeSet : public
UAttributeSet
     {
          GENERATED_BODY()

     protected:
          /** Атрибут здоров’я */
          UPROPERTY(EditAnywhere, BlueprintReadOnly)
          FGameplayAttributeData Health;

          //~ ... Інші атрибути тут ...

     public:
          //~ Допоміжні функції для атрибуту здоров’я
          GAMEPLAYATTRIBUTE_PROPERTY_GETTER(UMyAttributeSet,
Health);
          GAMEPLAYATTRIBUTE_VALUE_GETTER(Health);
          GAMEPLAYATTRIBUTE_VALUE_SETTER(Health);
          GAMEPLAYATTRIBUTE_VALUE_INITTER(Health);
     };
```

Для початку, потрібно створити сам атрибут типу FGameplayAttributeData та додати макрос UPROPERTY для того, щоб

атрибут було видно в самому двигуні з інтерфейсом. Це потрібно, щоб зручно його редагувати в інтерфейсі та змінювати в скриптах, якщо це потрібно. Також, далі за допомогою макросів потрібно створити допоміжні функції для читання та запису атрибуту, тобто для зручного змінення значень. Таким чином створюються всі інші антрибути, які перед цим потрібно детально продумати.

Для зміні атрибутів використовуються Ігрові ефекти (Gameplay Effects). Щоб програміст міг якось обробити данні для зміни атрибуту, існує спеціальна функція під назвою PostGameplayEffectExecute. В лістингу 4.2 зображено приклад обробки атрибуту за допомогою функції PostGameplayEffectExecute

В лістингу 4.2 також зображена перевірка, чи наш персонаж живий, за допомогою інтерфейсу UCombatInterface. Це яркий приклад, чому потрібно обробляти атрибути за допомогою функції PostGameplayEffectExecute, адже розробник має можливість контролювати, що саме відбувається з персонажом в момент зміни значень атрибутів, і чи можна їх змінювати взагалі.

Лістинг 4.2 – Обробка зміні атрибутів (файл UFIHAAttributeSet.cpp)

```
void UFIHAAttributeSet::PostGameplayEffectExecute(const
FGameplayEffectModCallbackData& Data)
{
	Super::PostGameplayEffectExecute(Data);

	FEffectProperties effectProps;
	SetEffectProperties(Data, effectProps);

	if (effectProps.TargetCharacter-
>Implements<UCombatInterface>())
	{
		if
(ICombatInterface::Execute_IsDead(effectProps.TargetCharacter))
		{
			return;
		}
	}

	if (Data.EvaluatedData.Attribute == GetHealthAttribute())
```

```
	{
		SetHealth(FMath::Clamp(GetHealth(), 0.f, 
GetMaxHealth()));
	}

}
```

PostGameplayEffectExecute – це спеціальний метод в системі Gameplay Ability System, який використовується в класі для обробки змін атрибутів після застосування Gameplay Effect. Його головне завдання – забезпечити коректну реакцію на зміну значень атрибутів, наприклад, оновлення здоров'я, манни, витривалості або інших параметрів персонажа.

Коли на персонажа накладається Gameplay Effect (наприклад, отримання урону, лікування, зміна швидкості), система спочатку застосовує зміни до атрибутів, а потім викликає PostGameplayEffectExecute. Це надає розробнику можливість виконати додаткову логіку після оновлення атрибутів. Наприклад, якщо значення здоров'я стало нижчим за нуль, у цьому методі можна викликати функцію смерті персонажа. Також тут можна реалізувати логіку обмежень, щоб атрибути не виходили за допустимі межі (наприклад, щоб здоров'я не було меншим за 0 або більшим за максимальне значення).

4.2 Реалізація здібностей. Gameplay Ability

Для того, щоб якось змінювати атрибути, та взагалі мати візуальний результат для гравця, потрібно створити здібностей для персонажів. Саме завдяки здібностям відбувається їх взаємодія в світі один з одним. Gameplay Ability – це ігрова дія, яку об’єкт може мати у своєму розпорядженні та повторно активовувати. Поширеними прикладами є заклинання, спеціальні атаки або ефекти, що активуються предметами. Ця концепція дуже поширена у відеоіграх і часто сприймається як належне, хоча процеси, що відбуваються під час виконання здібностей, можуть бути досить складними та вимагати точного таймінгу.

Наприклад, хоча програмування активації атаки саме по собі є досить простим, у довгостроковому проєкті складність розробки здібностей може значно зрости, коли додаються витрати ресурсів, бафи чи дебафи, що впливають на гравців, системи комбо та інші механіки. Від базового клаву вже створюються інші типи здібностей. Здібності мають в собі еффект, що впливає на атрибути інших об’єктів.

Через це в Gameplay Ability System Unreal Engine закладено три ключові аспекти, що визначають її структуру та роботу. Здібності та їхні ефекти повинні зберігати концепцію власника. Коли ефект виконує розрахунок, йому потрібно знати, хто є його власником, щоб використовувати його атрибути. Так само, коли здібність виконує дію, яка приносить очки гравцю, вона має знати, якому гравцю вона належить, щоб правильно зарахувати результат.

В лістингу 4.3 зображено створення класу здібності, яке містить в собі два поля: персонаж, що активує здібність, та контролер персонажа, що керує персонажем. Від базового клаву вже створюються інші типи здібностей. Здібності мають в собі еффект, що впливає на атрибути інших об’єктів.

Лістинг 4.3 – Створення базового класу здібності (файл UFIHAGameplayAbilityBase.hpp)

```
UCLASS()
class FIHA_API UFIHAGameplayAbilityBase : public
UGameplayAbility
{
	GENERATED_BODY()


protected:
	UPROPERTY(BlueprintReadWrite, Category = "Variables")
	TObjectPtr<APlayerController> OwnerPlayerController;

	UPROPERTY(BlueprintReadWrite, Category = "Variables")
	TObjectPtr<ACharacter> OwnerCharacter;
}
```

4.3 Вплив здібностей на атрибути персонажів. Gameplay Effects

Gameplay Ability System використовує Gameplay Effects [15] для зміни атрибутів акторів, які є цілями здібностей. Gameplay Effects складаються з бібліотек функцій, які можна застосовувати до атрибутів актора. Вони можуть бути миттєвими ефектами, такими як нанесення урону, або постійними ефектами, наприклад отруєння, яке завдає шкоди персонажу протягом певного часу.

Gameplay Effects можна використовувати як для бафів, так і для дебафів. Вони дозволяють робити персонажів сильнішими чи слабшими залежно від дизайну гри. Оскільки Gameplay Effects є окремими ресурсами (assets), вони є незмінними під час виконання гри.

Однак існують винятки, наприклад, коли Gameplay Effect створюється під час виконання гри, але його дані не змінюються після створення та налаштування. Для роботи з ефектами в реальному часі використовуються Gameplay Effect Specs – це версії Gameplay Effects, які містять інстансовані дані ефекту.

Під час роботи з Gameplay Effects у Blueprint-системі зазвичай використовується саме Gameplay Effect Specs, а не самі Gameplay Effects. Наприклад, бібліотека Ability System Blueprint Library активно працює з Gameplay Effect Specs, що відображається в її функціональності.

Gameplay Effect має параметр Duration (тривалість), який можна встановити в один із трьох режимів: Instant (миттєвий), Infinite (нескінченний) або Has Duration (із заданою тривалістю). Параметри тривалості зображені на рисунку 4.1.

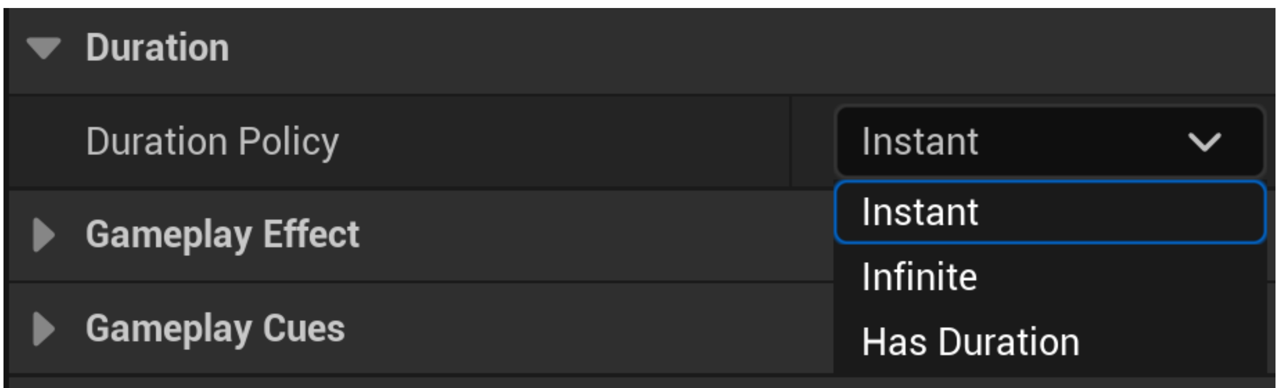

Рисунок 4.1 – Зображення параметрів тривалості ефекту

Gameplay Effects, які мають тривалість, додаються до Active Gameplay Effects Container – спеціального контейнера активних ефектів. Цей контейнер є частиною Ability System Component і відповідає за управління ефектами, які діють на персонажа протягом певного часу.

Також, ефекти мають параметри Modifiers (модифікатори), що визначають, як Gameplay Effect взаємодіє з атрибутами. Вони включають математичні операції над атрибутами, наприклад, збільшення показника броні на 5% від його базового значення, а також використання Gameplay Tags як умови для активації ефекту.

4.4 Gameplay Tags

Gameplay Tags – це користувацькі рядки, які виконують функцію концептуальних ієрархічних міток. Вони можуть бути застосовані до об’єктів у проєкті та використовуватися для перевірки станів і реалізації ігрової логіки, подібно до булевих змінних або прапорців.

Gameplay Tags відіграють ключову роль у Gameplay Ability System (GAS), забезпечуючи гнучкий механізм керування здібностями, ефектами та взаємодіями між об'єктами у грі. Вони працюють як мітки (теги), які допомагають системі визначати, коли здібності можуть бути активовані, які ефекти можуть бути застосовані та як вони впливають на персонажа або об'єкти в грі.

Основні ролі Gameplay Tags у GAS:

- контроль активації здібностей Gameplay Tags можуть бути використані для обмеження чи дозволу активації здібностей. Наприклад, якщо у персонажа є тег State.Stunned, він не зможе активувати певні здібності, такі як атака або ухилення. Це дозволяє легко керувати станами гравця без складних перевірок у коді;

- зміна ефектів та їх взаємодія – теги використовуються для визначення взаємодії між Gameplay Effects. Наприклад, якщо персонаж отримав баф Buff.SpeedBoost, який збільшує швидкість руху, можна зробити так, щоб він не міг отримати одночасно дебаф Debuff.Slow, або щоб вони нейтралізували один одного;

- фільтрація подій та здібностей – теги можна використовувати для виклику подій або перевірки умов виконання здібностей. Наприклад, якщо у грі є система комбо-ударів, можна перевіряти, чи у персонажа є тег Combo.Active, щоб вирішити, чи можна активувати наступний удар у серії;

- оптимізація роботи системи – замість того, щоб зберігати купу булевих змінних для різних станів персонажа, Gameplay Tags дозволяють зберігати та перевіряти інформацію більш ефективно, використовуючи єдину систему. Це полегшує розширення гри та керування складними механіками.

Gameplay Tags підтримують багаторівневу ієрархію, де рівні розділяються символом ".". Наприклад, тег Event.Movement.Dash має три рівні:

- Event – найзагальніший рівень, що позначає категорію подій;
- Movement – підкатегорія, яка уточнює, що подія пов’язана з рухом;
- Dash – найконкретніший рівень, що визначає саме ривок (Dash).

Ця структура дозволяє ефективно організовувати та використовувати Gameplay Tags для керування станами об'єктів, здібностями персонажів і обробки подій у грі.

В лістингу 4.4 зображено приклад створення тагів в C++. Ви повинні додати модуль GameplayTags до файлу Build.cs вашого проєкту, щоб

отримати доступ до функціоналу Gameplay Tags у C++.

Лістинг 4.4 – Створення базового класу здібності (файл UFIHAGameplayAbilityBase.hpp)

```
// In .h file
UE_DECLARE_GAMEPLAY_TAG_EXTERN(Movement_Mode_Walking);

// In .cpp file
UE_DEFINE_GAMEPLAY_TAG_COMMENT(Movement_Mode_Walking,
"Movement.Mode.Walking", "Default Character movement tag");
```

4.5 Ability Tasks

Ability Tasks (C++ клас UAbilityTask) – це спеціалізована форма загального класу Gameplay Task, призначена для роботи із Gameplay Abilities. Ігри, що використовують Gameplay Ability System, зазвичай включають набір власних Ability Tasks, які реалізують унікальні ігрові механіки. Вони виконують асинхронні завдання під час роботи здібності та можуть впливати на потік виконання, викликаючи Delegates (у нативному C++ коді) або перемикаючись через вихідні піни виконання (у Blueprints).

Це дозволяє здібностям виконуватися протягом декількох кадрів, а також виконувати кілька окремих функцій в одному кадрі. Більшість Ability Tasks мають пін виконання, який спрацьовує відразу, дозволяючи Blueprint-скрипту продовжити виконання після запуску завдання. Крім того, існують специфічні для завдання піни, які можуть активуватися із затримкою або після настання певної події (яка може відбутися або ні).

Ability Tasks можуть завершуватися самостійно, викликаючи функцію EndTask, або чекати автоматичного завершення при закінченні Gameplay Ability, яка їх запустила. Це запобігає утворенню "примарних" завдань, які можуть марно витрачати ресурси процесора та пам’яті. Наприклад, одна Ability Task може відтворювати анімацію заклинання, тоді як інша – розміщувати приціл на точці наведення гравця. Здібність може завершитися, якщо гравець натисне кнопку підтвердження для запуску заклинання або

просто дочекається закінчення анімації.

Хоча Ability Tasks можуть завершуватися будь-коли, вони гарантовано закінчуються щонайпізніше в момент завершення основної Gameplay Ability. На рисунку 4.2 зображено приклад реалізації здібності ближньої атаки, що реалізована в Blueprints.

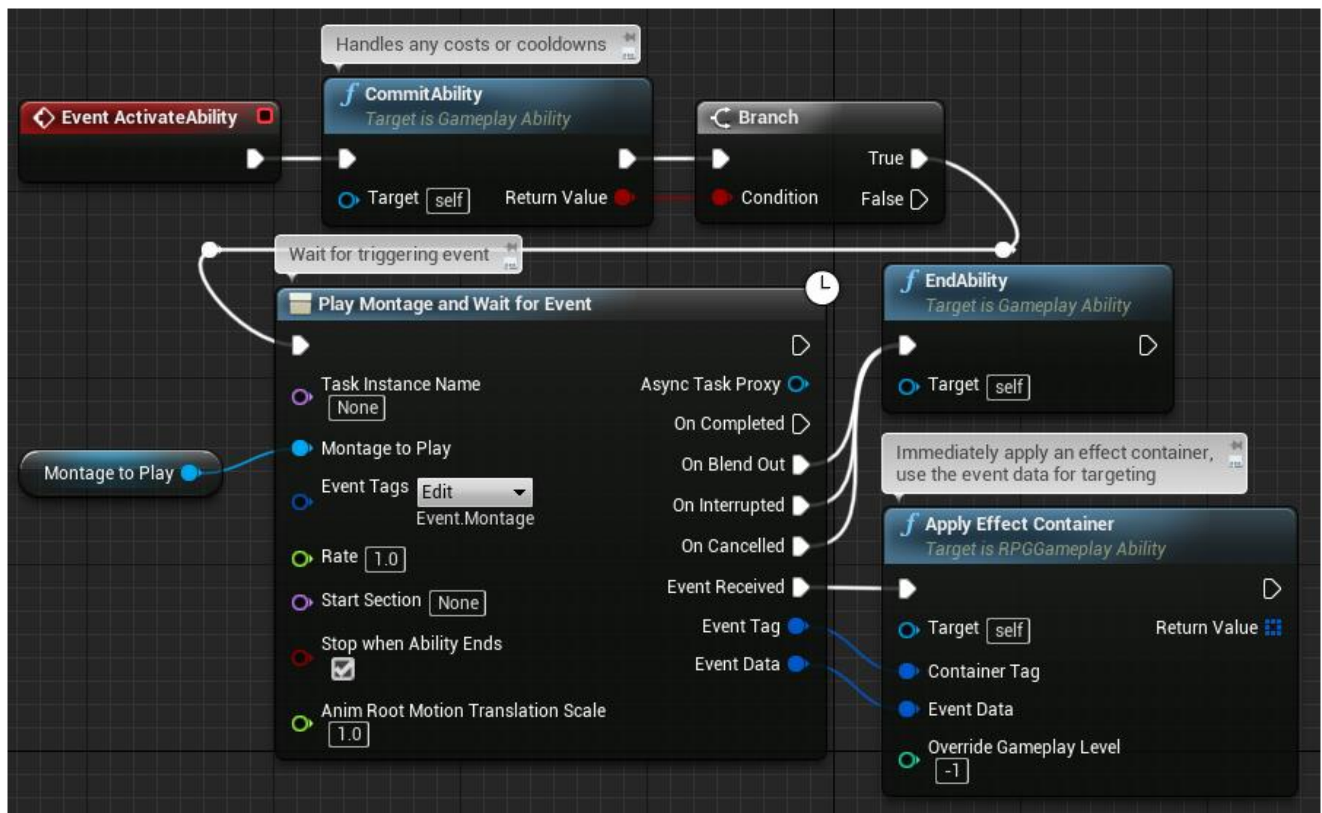

Рисунок 4.2 – Приклад використання Ability Task в програмі

У її центрі використовується Ability Task під назвою "Play Montage and Wait for Event" [16]. Ця Ability Task дозволяє запускати анімаційний монтаж атаки та очікувати певну подію, що відбудеться під час його відтворення. Це може бути, наприклад, момент, коли удар досягає цілі або коли завершиться певний кадр анімації. Подібний підхід дозволяє синхронізувати логіку гри з анімаціями, гарантуючи, що атака спрацьовує в потрібний момент, а не просто одразу при виклику здібності.

У цьому випадку, "Play Montage and Wait for Event" використана для визначення моменту, коли слід запустити перевірку попадання по ворогу,

застосувати ефекти урону або активувати інші механіки, пов'язані з атакою. Це забезпечує плавний і природний процес виконання здібності без необхідності вручну розраховувати затримки або покладатися на таймери.

4.6 Використання Gameplay Ability System в реалізації проектів

Підсумовуючи викладений матеріал, можна зазначити, що Gameplay Ability System (GAS) забезпечує гнучкий та модульний підхід до створення ігрових здібностей. Система дозволяє легко масштабувати механіки гри, спрощує керування ефектами, такими як бафи, дебафи, атаки та спеціальні здібності персонажів. Завдяки Gameplay Tags, Gameplay Effects та Ability Tasks, GAS дозволяє ефективно взаємодіяти з атрибутами персонажів та забезпечує контроль над виконанням здібностей у реальному часі.

Важливість використаних технологій полягає в тому, що GAS значно спрощує розробку складних бойових систем. Вона дозволяє централізовано керувати станами персонажів, анімаціями та подіями, що відбуваються під час виконання здібностей. Завдяки своїй архітектурі, система забезпечує оптимізацію використання ресурсів, запобігає зайвим обчисленням та забезпечує коректну взаємодію між різними механіками гри. Це робить GAS ідеальним рішенням та потужним інструментом для створення складних RPG та бойових систем. Вона дозволяє розробникам швидко додавати нові здібності та ефекти без значних змін у коді, що пришвидшує продуктивність та зменьшує затрати на розробку проекту.

ВИСНОВКИ

В наш час розробка ігор є як ніколи актуальна, адже багато користувачів по всьому світу граю в різни жанри та типи ігор. Ігри створюються під багато платформ: комп'ютери, ігрові консолі та приставки, телефони, веб-ігри. Більш того, це дуже прибутковий бізнес, адже кількість гравців постійно та стрімко збільшується. З кожним днем на світ виходить ще більше різноманітних проектів. Ігри не лише мають розважальну частину, а й несуть певну користь для користувача, якщо не зловживати ними. Ігри можуть поєднувати в собі навчання, соціальну взаємодію, розвиток фізичних властивостей людини.

Гра «Forgotten Island: Hero Artifact» не лише допомагає відпочити та розважитися, а й розвиває логічні здібності, пам'ять та увагу до оточення. Вона актуальна як ніколи, тому що розробляється під комп'ютери, ігрову консоль XBox та ігрові консолі. Гра працює на системах Windows та Linux, а також в майбутньому на MacOs. Одною з переваг цієї гри є невеликі системні вимогу, які дозволять користувачам насолодитися плавністю та приємною графікою.

Таким чином, гра «Forgotten Island: Hero Artifact» є чудовим вибором для користувачів, які хочуть не тільки відпочити, а й трішки подумати, щоб відкрити всі досягнення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Скороход М. М. Розробка комп’ютерної гри «Forgotten Island: Hero Artifact» / М. М. Скороход // Радіоелектроніка та молодь у XXI столітті : матеріали 29-го Міжнар. молодіж. форуму, 23–25 квітня 2025 р. – Харків : ХНУРЕ, 2025. – Т. 5. – С. 55–56. – УДК 004.946.

2. Legacy of Games. Rogue (1980). [Online]. Available: https://legacyofgames.com/2024/06/02/rogue-1980/

3. OPNoobs. Why The Binding of Isaac is the Greatest Game Ever (And You Can Too). [Online]. Available: https://opnoobs.com/opinions/why-the-binding-of-isaac-is-the-greatest-game-ever-and-you-can-too

4. IGN. Dead Cells Review. [Online]. Available: https://www.ign.com/articles/2018/08/11/dead-cells-review

5. Greenwood J. What Makes Hades so Special. [Online]. Available: https://cogconnected.com/feature/what-makes-hades-so-special/

6. Kestrel I. Why Spelunky is the Best Game I've Ever Played. [Online]. Available: https://medium.com/@iznaut/why-spelunky-is-the-best-game-ive-ever-played-50cc265e37ab

7. NVIDIA. NVIDIA ACE for Games: Bringing Digital Humans to Life with Generative AI. [Online]. Available: https://www.nvidia.com/en-us/geforce/news/nvidia-ace-autonomous-ai-companions-pubg-naraka-bladepoint/

8. Lee J. Learning Unreal Engine Game Development. Publisher: Packt Publishing; 1st edition. 2018. 400 p.

9. Habib L. Unreal — The Games of 1998. [Online]. Available: https://gamesof1998.com/home/unreal

10. Sherif W., Whittle S. Unreal Engine 4 Scripting with C++ Cookbook: Get the best out of your games by scripting them using UE4. Publisher: Packt Publishing; 1st edition. 2016. 472 p.

11. Li Z.G. Unreal Engine 5 Game Development with C++ Scripting:

Become a professional game developer and create fully functional, high-quality games. Publisher: Packt Publishing; 1st edition. 2022. 470 p.

12. Epic Games. Unreal Engine Terminology. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-terminology.

13. Newton P.L., Feng J. Unreal Engine 4 AI Programming Essentials. Publisher: Packt Publishing; 1st edition. 2016. 194 p.

14. Epic Games. Gameplay Ability System for Unreal Engine. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/gameplay-ability-system-for-unreal-engine.

15. Gameplay Effects for the Gameplay Ability System in Unreal Engine | Unreal Engine 5.5 Documentation | Epic Developer Community

16. Epic Games. Play Montage and Wait Node in Unreal Engine. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/ BlueprintAPI /Ability/Tasks/PlayMontageAndWait