НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Однокористувацький ігровий застосунок у жанрі Action RPG з використанням ігрового штучного інтелекту на рушії Unreal Engine

Виконав студент IV курсу, групи ІП-12
(шифр групи)

Стецун Дмитро Олександрович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник ст. викл, д-р філософії, Сарнацький В. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант ст. викл, д-р філософії, Головченко М. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___" _____2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Стецун Дмитро Олександрович
(прізвище, ім'я, по батькові)

1. Тема проєкту — Однокористувацький ігровий застосунок у жанрі Action RPG з використанням ігрового штучного інтелекту на рушії Unreal Engine

керівник проєкту — Сарнацький Владислав Віталійович, д-р філософії, ст. викл
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «23» травня 2025 р. №1705-с

2. Термін подання студентом проєкту «16» червня 2025 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Передпроєктне обстеження предметної області: постановка завдання дипломного проектування, аналіз предметної області, аналіз існуючих рішень, аналіз відомих програмних продуктів, аналіз відомих алгоритмічних та технічних рішень, опис бізнес-процесів
2) Розроблення вимог до програмного забезпечення: варіанти використання програмного забезпечення, аналіз системних вимог, розроблення функціональних вимог, розроблення нефункціональних вимог, аналіз економічних показників програмного забезпечення, постановка завдання на розробку програмного забезпечення

3) Конструювання та розроблення програмного забезпечення: архітектура програмного забезпечення, архітектурні рішення та обґрунтування вибору засобів розробки, аналіз безпеки даних.
4) Аналіз якості та тестування програмного забезпечення: аналіз якості програмного забезпечення, опис процесів тестування, опис контрольного прикладу.
5) Розгортання та супровід програмного забезпечення: розгортання програмного забезпечення, супровід програмного забезпечення.

5. Перелік графічного матеріалу
1) Схема структурна варіантів використань
2) Схема структурна блок-схеми алгоритму планування дій
3) Схема структурна бізнес-процесу взаємодії з продавцем

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «15» березня 2025 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	15.03.2025	
2	Аналіз існуючих методів розв’язання задачі	20.03.2025	
3	Постановка та формалізація задачі	30.03.2025	
4	Розробка інформаційного забезпечення	10.04.2025	
5	Алгоритмізація задачі	15.04.2025	
6	Обґрунтування вибору використаних технічних засобів	20.04.2025	
7	Розробка програмного забезпечення	15.05.2025	
8	Налагодження програми	20.05.2025	
9	Виконання графічних документів	22.05.2025	
10	Оформлення пояснювальної записки	31.05.2025	
11	Подання ДП на попередній захист	02.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	16.06.2025	

Студент _____ Дмитро Стецун
(підпис) (ініціали, прізвище)

Керівник _____ Владислав САРНАЦЬКИЙ
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 41 таблицю, 28 рисунків та 11 джерел – загалом 73 сторінки.

Дипломний проєкт присвячений розробці однокористувацького ігрового застосуноку у жанрі Action RPG з використанням ігрового штучного інтелекту на рушії Unreal Engine.

Метою розробки є підвищення рівня занурення та реіграбельності гри шляхом впровадження адаптивного штучного інтелекту, який реагує на стан світу, коригує поведінку ворогів у реальному часі та забезпечує різноманітні бойові сценарії, при цьому зберігаючи високу продуктивність та масштабованість проєкту

У розділі «Передпроєктне обстеження предметної області» розглядається предметна область розробки, подібні продукти до розробки та можливі алгоритмічні та технічні рішення, що можуть бути використані у розробці.

Розділ «Розроблення вимог до програмного забезпечення» присвячений аналізу варіантів використання програмного забезпечення, за результатом якого було розроблено функціональні та нефункціональні вимоги. Розроблені вимоги були використані для аналізу економічних показників, такі як тривалість та вартість розробки.

У розділі «Конструювання та розроблення програмного забезпечення» обговорюються прийняті рішення щодо архітектури програмного забезпечення, зокрема детальний розгляд прийнятих архітектурних рішень та обґрунтування вибору засобів розробки.

Розділ «Аналіз якості та тестування програмного забезпечення» присвячений аналізу якості програмного забезпечення за допомогою обраних методів тестування, опис процесу тестування та наведення прикладу роботи програмного забезпечення.

У розділі «Розгортання та супровід програмного забезпечення» йде мова про процес розгортання програмного забезпечення та його супроводу.

КЛЮЧОВІ СЛОВА: ІГРОВИЙ ЗАСТОСУНОК, UNREAL ENGINE, GOAP

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 41 tables, 28 figures and 11 sources – in total 73 pages.

The purpose of the diploma project is to increase the level of immersion and replayability by implementing adaptive artificial intelligence that responds to the state of the world, dynamically adjusts enemy behavior in real time, and provides diverse combat scenarios while maintaining high performance and project scalability.

The section «Pre-project analysis of the subject area» examines the development domain, similar existing products, and potential algorithmic and technical solutions that could be applied in the project.

The section «Development of software requirements» focuses on the analysis of possible software use cases, based on which functional and non-functional requirements were developed. These requirements were then used to analyze economic indicators such as development time and cost.

In the section «Software design and development», decisions regarding the software architecture are discussed, including a detailed examination of the selected architectural solutions and the rationale behind the choice of development tools.

The section «Software quality analysis and testing» is devoted to evaluating the software's quality using selected testing methods, describing the testing process, and providing a working example of the application.

The section «Software deployment and maintenance» covers the process of software deployment and its subsequent maintenance.

KEYWORDS: GAME APPLICATION, UNREAL ENGINE, GOAP

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-0222.045480.01.91	Технічне завдання	11	
3	А4	КПІ.ІТ-0222.045480.02.81	Пояснювальна записка	75	
4	А4	КПІ.ІТ-0222.045480.03.12	Текст програми	58	
5	А4	КПІ.ІТ-0222.045480.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІТ-0222.045480.05.34	Керівництво користувача	13	
7	А3	КПІ.ІТ-0222.045480.06.99	Графічний матеріал	4	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-0222.045480.00.90		
Розроб.		Стецун Д. О.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Сарнацький В. В.					
Н.контр.		Головченко М. М.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-12	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2025 р.

ОДНОКОРИСТУВАЦЬКИЙ ІГРОВИЙ ЗАСТОСУНОК У ЖАНРІ ACTION RPG З ВИКОРИСТАННЯМ ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ НА РУШІЇ UNREAL ENGINE

Технічне завдання

КПІ.ІТ-0222.045480.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Владислав САРНАЦЬКИЙ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Дмитро СТЕЦУН

Київ – 2025

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ 3
2 ПІДСТАВА ДЛЯ РОЗРОБКИ 4
3 ПРИЗНАЧЕННЯ РОЗРОБКИ 5
4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 6
4.1 Вимоги до функціональних характеристик 6
4.1.1 Користувацького інтерфейсу 6
4.1.2 Для користувача: 6
4.2 Вимоги до надійності 6
4.3 Умови експлуатації 6
4.3.1 Вид обслуговування 7
4.3.2 Обслуговуючий персонал 7
4.4 Вимоги до складу і параметрів технічних засобів 7
4.5 Вимоги до інформаційної та програмної сумісності 7
4.5.1 Вимоги до вхідних даних 7
4.5.2 Вимоги до вихідних даних 8
4.5.3 Вимоги до мови розробки 8
4.5.4 Вимоги до середовища розробки 8
4.5.5 Вимоги до представленню вихідних кодів 8
4.6 Вимоги до маркування та пакування 8
4.7 Вимоги до транспортування та зберігання 8
4.8 Спеціальні вимоги 8
5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ 9
5.1 Попередній склад програмної документації 9
6 СТАДІЇ І ЕТАПИ РОЗРОБКИ 10
7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ 11

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Однокористувацький ігровий застосунок у жанрі Action RPG з використанням ігрового штучного інтелекту на рушії Unreal Engine.

Галузь застосування: організація дозвілля.

Наведене технічне завдання поширюється на розробку однокористувацького ігрового застосунку Descendance Hero [DH], котра використовується для розваг та призначена для зняття стресу серед великого кола користувачів, що грають в комп’ютерні ігри.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки Descendance Hero є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для розваг та зняття стресу серед великого кола користувачів, що грають в комп’ютерні ігри.

Метою розробки є підвищення рівня занурення та реіграбельності гри шляхом впровадження адаптивного штучного інтелекту, який реагує на стан світу, коригує поведінку ворогів у реальному часі та забезпечує різноманітні бойові сценарії, при цьому зберігаючи високу продуктивність та масштабованість проєкту

.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– відображення рівня здоров’я, ресурсу, досвіду та рівня персонажа користувача та показувати обрані користувачем навички та їх стан перезарядки;
– відображення меню взаємодії з продавцем;
– відображення меню взаємодії з інвентарем;
– відображення меню вибору навичок;
– відображення ігрового рівня;

4.1.2 Для користувача:

– можливість почати гру;
– можливість вийти з гри;
– можливість взаємодії з продавцем (купівля та продаж предметів);
– можливість екіпірування предметів;
– можливість використання навичок;
– можливість вбити ворога;

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i3-6100 або аналог за продуктивністю;
– об‘єм ОЗП: 6 Гб;
– Графічний процесор: Nvidia GeForce GTX980 або аналогічний за продуктивністю;
– Місце на диску: 10 ГБ HDD.

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i5-8400 або аналог за продуктивністю;
– об‘єм ОЗП: 8 Гб;
– Графічний процесор: Nvidia GeForce GTX980 або аналогічний за продуктивністю;
– Місце на диску: 10 ГБ SSD.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN64 (Windows 10, Windows 11) або Unix.

4.5.1 Вимоги до вхідних даних

Вимоги до вхідних даних не висуваються..

4.5.2 Вимоги до вихідних даних

Вимоги до вихідних даних не висуваються..

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування: C++

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі Unreal Engine та з використанням середовища розробки Rider

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді програмного коду в окремому текстовому документі.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати інсталяційну версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;

– технічне завдання;

– текст програми;

– програма та методика тестування;

– керівництво користувача;

– графічні матеріали

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;

– схема структурна блок-схеми алгоритму роботи штучного інтелекту;

– схема структурна необхідних бізнес-процесів

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проєкту	15.03	
2.	Розробка технічного завдання	15.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проєктування структури програмного забезпечення, проєктування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	15.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	20.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проєкту	25.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проєкту	29.05	Графічний матеріал проєкту
9.	Оформлення технічної документації проєкту	31.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: **Однокористувацький ігровий застосунок у жанрі Action RPG з використанням ігрового штучного інтелекту на рушії Unreal Engine**

КПІ.ІТ-0222.045480.02.81

Київ – 2025

ЗМІСТ

1. ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ..... 5
1.1. Постановка завдання дипломного проектування..... 5
1.2. Аналіз предметної області..... 5
1.3. Аналіз існуючих рішень..... 7
1.3.1. Аналіз відомих програмних продуктів..... 8
1.3.2. Аналіз відомих алгоритмічних та технічних рішень..... 11
1.4. Опис бізнес-процесів..... 17
Висновки до розділу..... 18
2. РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ..... 19
2.1. Варіанти використання програмного забезпечення..... 19
2.2. Аналіз системних вимог..... 26
2.3. Розроблення функціональних вимог..... 27
2.4. Розроблення нефункціональних вимог..... 32
2.5. Аналіз економічних показників програмного забезпечення..... 33
2.6. Постановка завдання на розробку програмного забезпечення..... 35
Висновки до розділу..... 36
3. КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 38
3.1. Архітектура програмного забезпечення..... 38
3.2. Архітектурні рішення та обґрунтування вибору засобів розробки..... 41
3.3. Конструювання програмного забезпечення..... 44
3.4. Аналіз безпеки даних..... 47
Висновок до розділу..... 47
4. АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ... 49
4.1. Аналіз якості ПЗ..... 49
4.2. Опис процесів тестування..... 50
4.3. Опис контрольного прикладу..... 55
Висновок до розділу..... 63
5. РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ..... 64
5.1. Розгортання програмного забезпечення..... 64
5.2. Супровід програмного забезпечення..... 66
Висновок до розділу..... 68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 71
ДОДАТКИ..... 72
ДОДАТОК А ЗВІТ ПОДІБНОСТІ..... 72

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ДП — Дипломний проект.

ПЗ — Програмне забезпечення.

ОС — Операційна система.

IDE — Integrated Development Environment — інтегроване середовище розробки.

ВСТУП

На даний момент існує купа різноманітних ігрових застосунків у різних жанрах. Їх створюють як великі ігрові компанії, так і самостійні розробники. Найбільша проблема таких застосунків у тому, що більшість з них є однотипними. Якщо ж це продовження певної серії, то не рідко трапляється, що нова розробка майже не відрізняється від попередніх. Звісно, змінюється сюжетна складова, візуальна, однак з точки зору саме ігрового процесу зміни мінімальні. Часто можна почути скарги від гравців про те, що поведінка неігрових персонажів є ненатуральною. Це стосується зазвичай з ігровими застосунками в жанрі Action Role-Play, де гравці зтикаються з великою кількістю ворогів.

Більшість компаній-лідерів індустрії перевикористовують свої попередні напрацювання з певними адаптаційними змінами в нових розробках. У зв’язку з цим поведінка неігрових персонажів є однотипною, часто прослідковуються одні й ті ж шаблони поведінки, такі як бездумні атаки при очевидних ризиках померти. Справді, деякі ігрові серії набули свою популярність саме завдяки попередньо запрограмованих комбінацій атак, які гравці повинні запам’ятовувати й вчасно помічати. Однак для решти це є часто недолік. Причиною цього є система ігрового штучного інтелекту неігрових персонажів. Іншими ж елементами, що часто визначають вірогідність успіху ігрового застосунку є можливість перепроходження такого застосунку інакшим шляхом від попереднього. На щастя, це не є проблемою у більшості випадків, однак способи досягнення можливості перепроходження є різні.

У рамках даного дипломного проєкту буде реалізовано ігровий застосунок «Descendance Hero» з використанням поглибленого штучного інтелекту неігрових персонажів, системою атрибутів, інвентара та навичок. Розробка виконується з використанням сучасних технологій, а саме ігрового рушія Unreal Engine. Ігровий застосунок є однокористувацьким та матиме можливість зрозумілої модифікації програмного коду для додавання нового функціоналу.

1. ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Постановка завдання дипломного проектування

Дипломне проєктування передбачає виконання наступних завдань:

– аналіз предметної області та опис її ключових бізнес-процесів, визначення загального завдання розробки у рамках ДП;

– аналіз існуючих рішень (як програмних продуктів, так і технічних рішень) обраного завдання розробки у рамках ДП;

– аналіз та моделювання бізнес-процесів;

– розроблення функціональних, нефункціональних та системних вимог до програмного забезпечення;

– аналіз економічних показників програмного забезпечення ДП;

– постановка завдання на розробку програмного забезпечення ДП;

– розроблення архітектури програмного забезпечення;

– розроблення архітектурних рішень та обґрунтування вибору засобів розробки програмного забезпечення;

– конструювання та розроблення програмного забезпечення;

– аналіз безпеки даних програмного забезпечення;

– аналіз якості та тестування програмного забезпечення;

– розгортання та супровід програмного забезпечення;

– створення супроводжувальної документації до розробленого програмного забезпечення.

1.2. Аналіз предметної області

Ігровий застосунок — це програмний продукт, створений для реалізації інтерактивного ігрового процесу на комп’ютері, ігровій консолі, мобільному пристрої або іншій платформі. Основною метою ігрового застосунку є забезпечення користувачу (гравцю) розважального або навчального досвіду через візуальну, аудіо та геймплейну взаємодію.

Такий застосунок поєднує в собі графіку, логіку гри, системи керування персонажами та об’єктами, механіки взаємодії, а також інтерфейс користувача. Залежно від жанру, ігровий застосунок може включати сюжет, систему прогресу, багатокористувацький режим, штучний інтелект тощо. Його розробка зазвичай вимагає використання ігрового рушія і охоплює як технічну реалізацію, так і художнє оформлення.

Перед початком розробки ігрового застосунку потрібно визначити основні компоненти, які будуть використані під час проектування.

Найголовніше це ігровий рушій. Це набір інструментів, що використовується для програмної реалізації ігрового застосунку [1]. Він надає розробникам базовий набір функцій для візуалізації графіки, відтворення аудіо, симуляцію фізики, управління вводом та відтворення анімацій. Відповідно, ігровий рушій зазвичай має наступні компоненти:

– графічний рушій — відповідає за роботу з текстурами, освітленням, тінями, візуальними ефектами;

– фізичний рушій — відповідає за симуляцію фізики;

– система управління ресурсами — відповідає за завантаження, зберігання та використання ресурсів;

– мережевий рушій — відповідає за управління мережевого підключення для реалізації багатокористувацьких проектів;

– рушій штучного інтелекту — надає інструментарій для програмування поведінки неігрових персонажів.

Вибір ігрового рушія залежить від потреб проекту. Тому, якщо візуальна складова ігрового застосунку не вимагає фотореалістичного вигляду, варто звернути увагу на ті рушії, що використовують більш прості методи візуалізації, оскільки їхня продуктивність у такому випадку є дещо кращою. Таким чином можна знизити мінімальні системні вимоги.

Також, популярні ігрові рушії є комерційними, тому при публікації проекту потрібно пам’ятати про умови ліцензії. Більшість комерційних рушіїв не вимагають плату за використання поки кількість користувачів не перевищить

певне значення. Відповідно, якщо ціллю розробки є створення безкоштовного ігрового застосунку, варто звернути увагу на існуючі рушії з відкритим програмним кодом. Розробники таких рушіїв не стягують плату за використання, однак інструментарій, що пропонується разом з рушієм може бути дещо обмеженим.

Іншим важливим компонентом при розробці ігрового застосунку є методологія ігрового штучного інтелекту, що буде використана для програмування поведінки неігрових персонажів [2][3][4]. Звісно, їх поведінку можна прописати з використанням розгалужень за умовою, однак подібні рішення вважаються поганим тоном серед розробників.

Вибір методології ігрового штучного інтелекту залежить від того, яка поведінка очікується від неігрових персонажів. Якщо очікується складна поведінка, що активно реагує на зміни навколишнього світу, варто звернути вагу на ті методології, що підтримують велику кількість можливих станів персонажів та дозволяють легко вносити зміни за потреби. Однак недоліком подібних рішень є нижча продуктивність, порівняно з більш простими методологіями.

Таким чином можна виокремити інший фактор, який також потрібно враховувати при виборі методології ігрового штучного інтелекту, а саме очікувана продуктивність. Якщо передбачається взаємодія з великою кількістю неігрових персонажів одночасно, варто обирати методологію, що пропонує високу продуктивність, адже в зворотному випадку є ризик зниження частоти оновлень, що призводить до погіршення умов ігрового процесу для користувача.

1.3. Аналіз існуючих рішень

Наразі існує досить багато ігрових застосунків, кожен з яких використовує різні ігрові рушії та методології штучного інтелекту. У цьому підрозділі буде проведено аналіз існуючих ігрових застосунків у жанрі Action Role Play,

альтернативні до обраних методології ігрового штучного інтелекту та ігрових рушіїв.

1.3.1. Аналіз відомих програмних продуктів

Жанр Action Role Play зародився на початку 1980-х років. З того часу відбулося багато технічних проривів для технологічних рішень, однак основні атрибути цього жанру не змінилися. У межах цього підрозділу буде йти мова про існуючі ігрові застосунки цього жанру, їхні особливості та сильні сторони.

На даний момент, найбільш популярними ігровими застосунками у жанрі Action Role Play є:

– серія «Diablo»;
– серія «Dark Souls»;
– серія «The Witcher»;

Ігрові застосунки серії «Diablo» [5] є типовими представниками жанру, з положенням камери згори. Тобто користувач бачить все навколо свого ігрового персонажа під кутом згори, що дає можливість спостерігати за більшою частиною ігрового світу, порівняно з іншими ігровими застосунками.

Щодо особливостей ігрових застосунків цієї серії, то можна відзначити просторовий інвентар персонажа, тобто внутрішньо-ігрові речі мають не лише вагу, а й займають простір. У зв’язку з цим, задачею користувача є не лише керуванням кількості речей у інвентарі свого персонажа, а й їх розміщенням.

Достовірно не відомо, яка саме методологія ігрового штучного інтелекту була використана для ігрових застосунків цієї серії, однак є припущення, що було використано суміш скінченних автоматів та утилітарного підходу. Про методології ігрових штучних інтелектів буде йти мова у наступному підрозділі.

Загалом сильними сторонами ігрових застосунків цієї серії є варіативність у способах проходження ,оскільки існує декілька класів персонажів та кожен з них має декілька дерев навичок, що змінюють стиль гри. Також існує велике різноманіття противників, завдяки великій кількості їх класів та додаткових

можливих атрибутів для кожного противника. Через те, таку велику кількість різних комбінацій навичок та атрибутів як для ігрового, так і неігрових персонажів, користувачі дають позитивні відгуки ігровим застосункам серії “Diablo” та проходять їх знову і знову багато разів.

Ігрові застосунки серії «Dark Souls» [6] є представниками піджанру Souls-Like. Особливостями цього піджанру є надзвичайно висока складність ігрового процесу, що зумовлена акцентом на вміння користувача реагувати на комбінації рухів неігрових персонажів. Наприклад, якщо персонаж користувача не відстрибне вбік під час атаки згори від противника, він, імовірно, загине. У зв’язку з цим кількість потенційних користувачів є меншою за попередню згадану серію ігрових застосунків.

Окрім жанрових особливостей, варто зазначити положення ігрової камери від третьої особи, тобто користувач спостерігає за ігровим процесом зі спини свого ігрового персонажа.

Щодо інвентара та системи навичок, то вони є дещо схожими до наявних у ігрових застосунках серії «Diablo». Інвентар є просторовим та існує велика кількість можливих комбінацій навичок.

Згідно інформації, отриманої експертами у сфері ігрового штучного інтелекту на основі програмних файлів, ігрові застосунки серії «Dark Souls» використовують методологію планування, про яку йтиме мова у наступному підрозділі. Завдяки цьому, неігрові персонажі попри не велику кількість можливих рухів мають можливість будувати різноманітні комбінації рухів, що робить ігровий процес значно складніше.

Сильними сторонами ігрових застосунків цієї серії є варіативність у способах проходження та, як не дивно, висока складність ігрового процесу. Користувачі намагаються постійно покращити свої навички і досить часто транслюють свій процес гри на стрімінгові платформи. Це призводить до постійного інтересу людей з ігрового суспільства до ігрових застосунків серії «Dark Souls».

Ігрові застосунки серії «The Witcher» [7] є представниками дещо іншого піджанру. Основний акцент ігрового процесу робиться на дослідження відкритого світу та сюжетну складову. Подібно до ігрових застосунків серії «Dark Souls», положення ігрової камери є позаду ігрового персонажа.

Як було сказано раніше, особливістю ігрових застосунків цієї серії є саме дослідження світу та сюжет. Оскільки це не оригінальний всесвіт, а створений на основі однойменних творів, поціновувачі оригінальної історії витрачають купу часу на дослідження всесвіту в межах ігрового процесу. Користувачам доступні незмінний перелік заклять та можливість їх покращувати за допомогою різних віток дерева навичок. Це призводить до варіативності, схожої до попередньо згаданих серій ігрових застосунків. Щодо інвентара, то речі мають лише вагу, тобто просторових обмежень він не має.

Щодо ігрового штучного інтелекту використаного в ігрових застосунках серії «The Witcher», є припущення серед експертів, що було використано суміш скінченних автоматів та дерев прийняття рішень.

Сильними сторонами в ігрових застосунках серії «The Witcher» є глибокий сюжет та великий відкритий світ. Користувачі мають свободу переміщення по карті, в межах якої можна зустріти незліченну кількість персонажів, з якими можна взаємодіяти. Користувачам доступна варіативність у ігровому процесі, схожа до тих, що у попередньо згаданих серіях ігрових застосунків.

Для порівняння існуючих рішень та визначення якостей, які дана розробка повинна мати, щоб бути конкурентоспроможною, було створену таблицю 1.1.

Таблиця 1.1 — Порівняння з аналогами

	Diablo	Dark Souls	The Witcher	Дипломний проект «Descendance Hero»
Положення камери	Згори	Позаду	Позаду	Згори

Таблиця 1.1 — Порівняння з аналогами

Інвентар	Просторовий	Просторовий	Не просторовий	Не просторовий
Штучний інтелект	Скінченні автомати + утилітарний підхід	Планування	Скінченні автомати + дерева прийняття рішень	Планування
Варіативність	Різні класи персонажів + користувацький вибір навичок	Користувацький вибір навичок	Користувацький вибір навичок	Різні класи персонажів + користувацький вибір навичок
Світ	Попередньо створений	Попередньо створений	Попередньо створений	Процедурно згенерований

Як видно, дана розробка бере найкращі елементи з кожного з аналогів, тому очікується, що це призведе до популярності розробленого ігрового застосунку серед гравців.

1.3.2. Аналіз відомих алгоритмічних та технічних рішень

У межах цього підрозділу буде йти мова про відомі алгоритмічні рішення у вигляді методологій ігрових штучних інтелектів та технічні рішення у вигляді ігрових рушіїв.

Почнемо з алгоритмічних рішень. У попередньому підрозділі під час аналізу існуючих програмних рішень було виокремлено три основних методології ігрового штучного інтелекту, а саме:

– скінченні автомати;
– дерева прийняття рішень;
– планування.

Попри те, що під час порівняння аналогів було згадано утилітарний підхід, його важко виокремити як окрему методологію ігрового штучного

інтелекту, оскільки це є концепцією вибору оптимальної дії за певного стану персонажу за допомогою евристичної функції в заданий момент часу без урахування попередніх чи наступних дій. Цей підхід використовується у більшості методологій як допоміжна функція, аніж основний спосіб прийняття рішення.

Методологія ігрового штучного інтелекту з використанням скінченних автоматів[2] полягає у виборі дії персонажа згідно його ігрового стану за принципом скінченних автоматів. Вузлами такого автомату є дія персонажа, а умовою переходу є ігровий стан. Ігровий стан персонажа створюється у вигляді певної структури даних та містить атрибути, що є важливими на думку розробника для прийняття рішень. Перехід з одного стану в інший є заздалегідь встановленим розробником, що є найбільшим недоліком цієї методології. Оскільки скінченний автомат у контексті ігрового штучного інтелекту не має термінального стану, всі вузли повинні бути зв’язані так, що з одного можна перейти до іншого та зміна стану має сенс. Це призводить до довгого процесу вибору параметрів переходу і переналаштування кожного переходу при зміні кількості вузлів та ігрових станів.

Сильною стороною цієї методології є те, що вона є найбільш ефективною з точки зору обчислювальної складності. Переходи є заздалегідь визначені розробником, що означає відсутність алгоритмів пошуку.

Методологія ігрового штучного інтелекту з використанням дерева прийняття рішень[3] використовує, як можна здогадатись, дерева прийняття рішень. Розробник створює дерево, вузлами якого є дія персонажа. Кожен вузол може мати умову входу і вузли-нащадки. Порівняно зі скінченним автоматом, ця методологія виконує обхід дерева після досягнення термінального вузла. Тобто прийняття рішень не залежить від попереднього обходу дерева. Однак проблема того, що потрібно переналаштовувати все дерево при зміні кількості вузлів, нікуди не зникла.

Сильною стороною цієї методології є обчислювальна ефективність у зв’язку з продуктивними алгоритмами обходу дерева, що є заздалегідь визначеним розробником.

Існує гібридний ігровий штучний інтелект з використанням суміші скінченних автоматів та дерева прийняття рішень. У такому випадку вузлами скінченного автомата є певний стан персонажа. Для кожного стану визначається окреме дерево прийняття рішень. Таким чином відбувається спрощення налаштування системи прийняття рішень, оскільки дії залежать тепер від того, в якому стані перебуває персонаж. При цьому продуктивність системи залишається дуже високою.

Методологія ігрового штучного інтелекту з використанням планування[4] є більш сучасною. Вона полягає в тому, що прийняття рішень відбувається шляхом планування комбінації дій, що призводять до певного результату. Основними сутностями в такій методології є ціль та дія. Прийняття рішень відбувається у контексті стану світу, поля якого обираються розробником.

Кожна ціль містить початкові значення полів стану світу, що спричиняють її активацію, кінцеві — ті, що будуть відповідати стану світу на момент досягнення цієї цілі, та пріоритет. Якщо виникає ситуація, коли стан світу задовільняє умови активації декількох цілей, буде обрано ту, що має найвищий пріоритет.

Для того, щоб змінити стан світу виконуються дії. Кожна дія, подібно до цілі, складається з початкових та кінцевих значень полів світу, значення вартості. У цьому випадку вартість є параметром, що показує скільки умовних одиниць ресурсу буде витрачено на виконання дії.

Планування дій на основі цілі забезпечує оптимальний упорядкований перелік дій, після виконання якого буде досягнуто встановлену ціль. Процес планування є алгоритмом пошуку.

Варто зазначити, що існують версії планування з розбиттям на підцілі, однак у такому випадку відмінність лише в тому, що перед плануванням дій обирається більш точна підціль, решта алгоритму залишається незмінною.

Сильною стороною цієї методології є те, що розробнику достатньо визначити, які цілі та дії існують, а під час налаштування системи змінити лише декілька значень, аніж переналаштовувати всю систему. Однак, використання алгоритму пошуку призводить до проблем з продуктивністю, оскільки час виконання алгоритму повинен бути достатньо малим, щоб частота оновлення кадрів ігрового застосунку була прийнятною.

Тепер перейдемо до технічних рішень. Існує досить велика кількість ігрових рушіїв, однак вони мають різне призначення. Сфокусуємо увагу на тих, що спеціалізуються на тривимірному просторі та реалістичній візуалізації. У такому випадку можна виділити три найпоширеніших рушія:

– Unreal Engine;
– Unity;
– Godot.

Unreal Engine[8]— це ігровий рушій, написаний мовою програмування C++, та має інструментарій для створення ігрових скриптів за допомогою графічного інтерфейсу з використанням вузлів та за допомогою коду, написаного мовою програмування C++.

Оскільки він написаний мовою програмування C++, створені ігрові застосунки можуть використовуватись на різних типах пристроїв, такі як персональні комп’ютери, консолі та смартфони.

Також, цей рушій існує вже давно, тому було проведено багато часу для оптимізації його роботи. Якщо розробник обирає створювати свій ігровий застосунок за допомогою написання коду мовою C++, рушій має власну бібліотеку структур даних та функцій для роботи з ними, що мають високу продуктивність з точки зору використання пам’яті. Як відомо, при розробці мовою C++ розробник сам повинен впроваджувати функціонал для менеджменту пам’яті, однак бібліотека Unreal Engine оптимізує використання пам’яті самостійно.

Він є розповсюдженим рішенням для розробки професійних високо-бюджетних ігрових застосунків, оскільки він є open-source, тобто його

програмний код можна знайти у вільному доступі. Це дає змогу розробникам змінити поведінку рушія за допомогою внесення змін до вихідного коду, що є значно ефективнішим рішенням, ніж використання надбудов.

Окрім ігрової індустрії, Unreal Engine активно використовується в кіноіндустрії, як середовище для створення візуальних ефектів. Причиною цьому є те, що цей рушій має оптимізовані алгоритми для обчислення фотореалістичної графіки.

Відповідно, сильними сторонами Unreal Engine як ігрового рушія є висока оптимізація, можливість прототипування за допомогою графічного середовища та можливість впровадження фотореалістичної графіки.

Unity[9] — це ігровий рушій, написаний мовами програмування C++ і C#, який надає широкий інструментарій для створення ігрових проєктів. Рушій підтримує візуальне створення скриптів через інспектор та компоненти, а також дозволяє розробникам писати код вручну за допомогою мови C#.

Оскільки Unity підтримує кросплатформенну розробку, створені проєкти можна легко експортувати на різні пристрої, такі як персональні комп’ютери, мобільні телефони, ігрові консолі та браузери. Завдяки цьому рушій активно використовується як для мобільних та інді-ігор, так і для великих комерційних проєктів.

Unity існує вже багато років, тому його архітектура оптимізована для високої продуктивності. Рушій має вбудовану систему управління пам’яттю, яка працює на основі Garbage Collector. Це полегшує розробку, оскільки розробник не повинен вручну контролювати розподіл пам’яті, але водночас може викликати потенційні проблеми з продуктивністю у великих проєктах.

Однією з головних переваг Unity є зручність у використанні. Рушій має гнучку систему компонентів, що дозволяє швидко створювати складні механіки, використовуючи модульний підхід. Крім того, Unity Asset Store містить величезну кількість готових рішень, плагінів, 3D-моделей та скриптів, що значно прискорює розробку.

Таким чином, сильними сторонами Unity є простота у використанні, гнучкість завдяки компонентній архітектурі, величезна кількість готових ресурсів та підтримка кросплатформеності. Це робить його універсальним рішенням як для початківців, так і для досвідчених розробників.

Godot[10] — це безкоштовний open-source ігровий рушій, написаний мовою C++, який надає розробникам гнучкі можливості для створення 2D та 3D ігор. Він підтримує візуальне створення ігрових механік через інтуїтивний інтерфейс, а також дозволяє писати код вручну за допомогою GDScript (власна скриптова мова, схожа на Python), C#, C++ або VisualScript (візуальне програмування).

Оскільки рушій є кросплатформним, створені проєкти можуть бути експортовані на різні пристрої, включаючи персональні комп’ютери, мобільні пристрої, веб-браузери та ігрові консолі. Його легка архітектура забезпечує швидкий запуск навіть на слабкому обладнанні, що робить його популярним вибором для мобільних та інді-ігор.

Однією з головних особливостей Godot є система сцени та вузлів, яка дозволяє будувати ігрові механіки у деревоподібній структурі, роблячи процес розробки зручним та модульним. Завдяки цьому навіть складні проєкти можуть бути легко масштабовані та оптимізовані.

Оскільки Godot є відкритим рушієм, розробники можуть вільно змінювати його вихідний код та адаптувати рушій під свої потреби. На відміну від багатьох інших рушіїв, Godot не вимагає ліцензійних відрахувань, що робить його ідеальним вибором для незалежних розробників та стартапів.

Завдяки мінімальним системним вимогам, простому кодуванню, потужним інструментам для 2D-ігор і повній відсутності ліцензійних обмежень, Godot є відмінним вибором як для новачків, так і для досвідчених розробників, які шукають гнучкий та легкий рушій.

1.4. Опис бізнес-процесів

Для опису бізнес-процесу програмного забезпечення використовуються BPMN моделі, зображені на рисунку 1.1 та графічному матеріалі КПІ.ІТ-0222.045480.06.99.ССВ.

Опис послідовності початку гри:

– користувач запускає гру;

– користувач тисне на кнопку початку гри;

– користувач обирає персонажа.

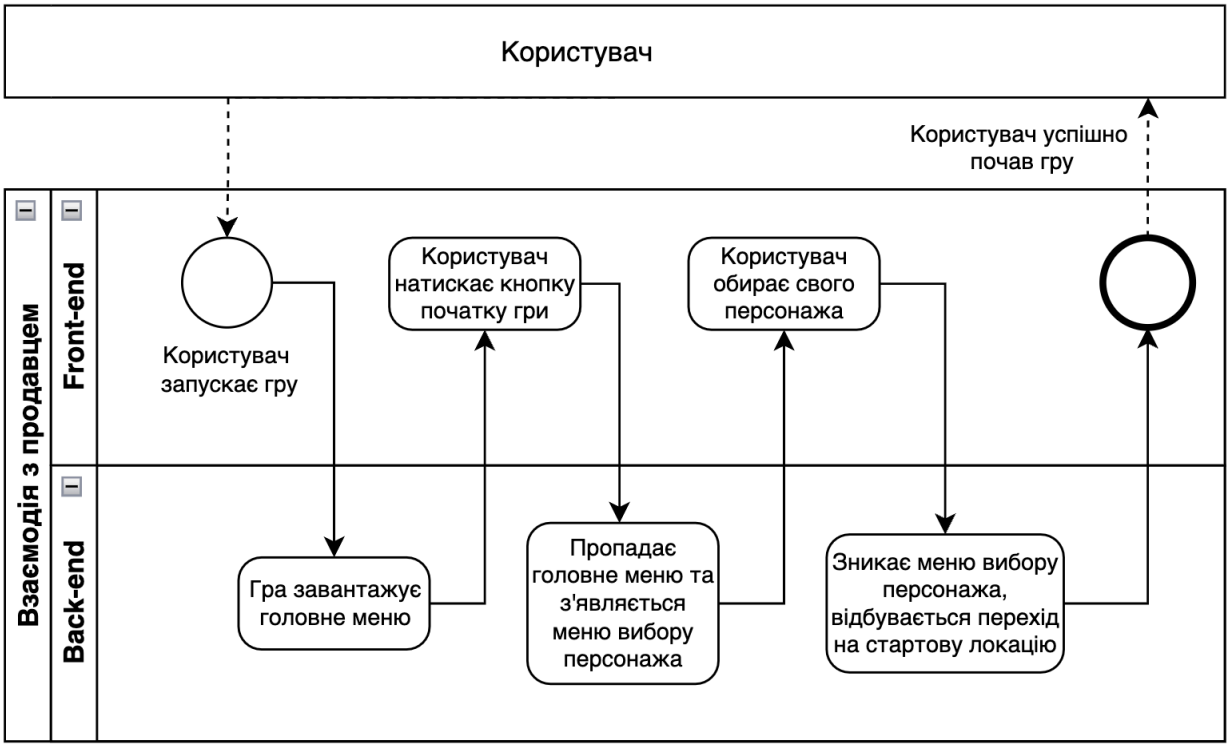


Рисунок 1.1 — Бізнес процес початку гри

Опис послідовності взаємодії з продавцем:

– користувач натискає кнопку взаємодії;

– користувач натискає кнопку купівлі предмету, якщо хоче купити обраний предмет;

– користувач натискає кнопку продажу предмету, якщо хоче продати предмет з інвентара;

– користувач натискає кнопку закриття меню, якщо хоче припинити взаємодію з продавцем.

Висновки до розділу

У результаті проведеного аналізу було визначено ключові складові ігрового застосунку, серед яких основною є ігровий рушій, що забезпечує графічну, аудіо, фізичну, мережеву та штучноінтелектуальну функціональність. Правильний вибір рушія суттєво впливає на продуктивність і можливості майбутнього проекту, а також на вимоги до системи та ліцензійні умови.

Додатково було розглянуто основні методології ігрового штучного інтелекту — скінченні автомати, дерева прийняття рішень та методи планування. Кожна з них має свої переваги та недоліки, які слід враховувати залежно від складності поведінки неігрових персонажів та вимог до продуктивності.

Аналіз існуючих популярних ігрових застосунків у жанрі Action Role Play, таких як серії «Diablo», «Dark Souls» та «The Witcher», продемонстрував різні підходи до побудови ігрової механіки, систем навичок, інвентаря та штучного інтелекту. Ці приклади стали основою для формування вимог та концепції розробки власного проекту, що поєднує найкращі елементи успішних рішень.

Отже, сформовані у цьому розділі теоретичні основи дозволяють перейти до практичної реалізації ігрового застосунку з урахуванням вибору відповідного ігрового рушія, алгоритмів штучного інтелекту та оптимізації продуктивності для забезпечення комфортного ігрового досвіду.

2. РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Варіанти використання програмного забезпечення

У даній грі користувач може почати гру та повинен обрати персонажа після цього. Далі йому доступні основні ігрові механіки, такі як переміщення, використання навичок, взаємодія з інвентарем і подібні. Всі варіанти застосування можна побачити на графічному матеріалі КПІ.ІТ-0222.045480.06.99.ССВ.

В таблицях 2.1-2.16 наведено детальний опис варіантів застосування даного програмного забезпечення.

Таблиця 2.1 — Варіант використання UC— 1

Use case name	Почати гру
Use case ID	UC-1
Goals	Почати ігровий процес
Actors	Користувач
Trigger	Користувач бажає почати грати в гру
Pre-conditions	Користувач у головному меню
Flow of Events	Після натискання на кнопку початку гри відкриється вікно вибору персонажа
Extension	Окрім можливої несправності програми, винятків нема.
Post-condition	Користувач бачить вікно вибору персонажа

Таблиця 2.2 — Варіант використання UC— 2

Use case name	Вийти з гри
Use case ID	UC-2
Goals	Гру закрито
Actors	Користувач
Trigger	Користувач бажає закрити гру

Продовження таблиці 2.2

Pre-conditions	Користувач знаходиться в головному меню або меню персонажа
Flow of Events	Після натискання на кнопку виходу з гри гра закриється
Extension	Окрім можливої несправності програми, винятків нема.
Post-condition	Гру закрито

Таблиця 2.3 — Варіант використання UC— 3

Use case name	Обрати персонажа
Use case ID	UC-3
Goals	Персонажа обрано
Actors	Користувач
Trigger	Користувач бажає обрати персонажа
Pre-conditions	Користувач перебуває у меню вибору персонажа
Flow of Events	Користувач читає інформацію про персонажів та обирає того, що йому подобається
Extension	Окрім можливої несправності програми, винятків нема.
Post-condition	Початок ігрового процесу

Таблиця 2.4 — Варіант використання UC— 4

Use case name	Змінити положення персонажа
Use case ID	UC-4
Goals	Персонаж змінив свою позицію в просторі
Actors	Користувач
Trigger	Користувач бажає змінити положення персонажа
Pre-conditions	Користувач перебуває у ігровому процесі і немає відкритих меню.

Продовження таблиці 2.4

Flow of Events	Користувач натискає кнопки управління і персонаж переміщується у обраному напрямку
Extension	Окрім можливої несправності програми, винятків нема.
Post-condition	Персонаж змінив свою позицію в просторі

Таблиця 2.5 — Варіант використання UC— 5

Use case name	Відкрити меню персонажа
Use case ID	UC-5
Goals	Відкрити меню персонажа
Actors	Користувач
Trigger	Користувач бажає побачити свої характеристики або внести зміни в інвентар чи екіпірування
Pre-conditions	Користувач перебуває у ігровому процесі і немає відкритих меню.
Flow of Events	Користувач натискає кнопку відкриття меню й меню відкривається
Extension	Окрім можливої несправності програми, винятків нема.
Post-condition	Користувач бачить меню персонажа

Таблиця 2.6 — Варіант використання UC— 6

Use case name	Покращити навичку
Use case ID	UC-6
Goals	Покращити навичку персонажа
Actors	Користувач
Trigger	Користувач бажає покращити навичку персонажа
Pre-conditions	Користувач перебуває у меню персонажа

Продовження таблиці 2.6

Flow of Events	Користувач обирає, яку навичку покращити, й натискає відповідну кнопку
Extension	Персонаж має очко покращення
Post-condition	Навичка покращена

Таблиця 2.7 — Варіант використання UC— 7

Use case name	Екіпірувати предмет
Use case ID	UC-7
Goals	Одягнути обраний предмет з інвентара
Actors	Користувач
Trigger	Користувач бажає екіпірувати предмет
Pre-conditions	Користувач перебуває у меню персонажа
Flow of Events	Користувач обирає предмет та натискає відповідну кнопку
Extension	Персонаж має предмет в інвентарі
Post-condition	Предмет екіпіровано

Таблиця 2.8 — Варіант використання UC— 8

Use case name	Зняти предмет
Use case ID	UC-8
Goals	Зняти екіріпований предмет й помістити його до інвентара
Actors	Користувач
Trigger	Користувач бажає зняти предмет
Pre-conditions	Користувач перебуває у меню персонажа

Продовження таблиці 2.8

Flow of Events	Користувач обирає предмет та натискає відповідну кнопку
Extension	Персонаж має екіпірований предмет
Post-condition	Предмет знято й поміщено до інвентара

Таблиця 2.9 — Варіант використання UC— 9

Use case name	Закрити меню персонажа
Use case ID	UC-9
Goals	Закрити меню персонажа
Actors	Користувач
Trigger	Користувач бажає закрити меню персонажа
Pre-conditions	Користувач перебуває у меню персонажа
Flow of Events	Користувач натискає кнопку закриття меню
Extension	Окрім можливої несправності програми, винятків нема.
Post-condition	Меню персонажа закрито

Таблиця 2.10 — Варіант використання UC— 10

Use case name	Використати навичку
Use case ID	UC-10
Goals	Використати навичку персонажа
Actors	Користувач
Trigger	Користувач бажає використати навичку
Pre-conditions	Користувач перебуває у ігровому процесі і немає відкритих меню.
Flow of Events	Користувач натискає кнопку активації обраної навички
Extension	Персонаж має достатньо ресурсу
Post-condition	Навичку використано, рівень ресурсу зменшився

Таблиця 2.11 — Варіант використання UC— 11

Use case name	Телепортуватись до підземелля
Use case ID	UC-11
Goals	Телепортуватись до підземелля зі стартової локації
Actors	Користувач
Trigger	Користувач бажає телепортуватись до підземелля
Pre-conditions	Користувач перебуває у ігровому процесі на стартовій локації і немає відкритих меню.
Flow of Events	Користувач натискає кнопку взаємодії з порталом
Extension	Персонаж знаходиться поруч з порталом
Post-condition	Персонажа телепортовано до підземелля

Таблиця 2.12 — Варіант використання UC— 12

Use case name	Телепортуватись до стартової локації
Use case ID	UC-12
Goals	Телепортуватись до стартової локації з підземелля
Actors	Користувач
Trigger	Користувач бажає телепортуватись до стартової локації
Pre-conditions	Користувач перебуває у ігровому процесі у підземеллі і немає відкритих меню.
Flow of Events	Користувач натискає кнопку взаємодії з порталом
Extension	Персонаж знаходиться поруч з порталом
Post-condition	Персонажа телепортовано до стартової локації

Таблиця 2.13 — Варіант використання UC— 13

Use case name	Відкрити меню взаємодії з продавцем
Use case ID	UC-13
Goals	Відкрити меню взаємодії з продавцем для продажу чи покупки предметів
Actors	Користувач
Trigger	Користувач бажає купити чи продати предмет
Pre-conditions	Користувач перебуває у ігровому процесі у стартовій локації і немає відкритих меню.
Flow of Events	Користувач натискає кнопку взаємодії з продавцем
Extension	Персонаж знаходиться поруч з продавцем
Post-condition	Відкрито меню взаємодії з продавцем

Таблиця 2.14 — Варіант використання UC— 14

Use case name	Купити предмет
Use case ID	UC-14
Goals	Купити предмет у продавця
Actors	Користувач
Trigger	Користувач бажає купити предмет
Pre-conditions	Користувач перебуває у меню взаємодії з продавцем
Flow of Events	Користувач обирає предмет та натискає кнопку для покупки
Extension	Персонаж має достатньо монет
Post-condition	Придбано предмет та поміщено його до інвентаря

Таблиця 2.15 — Варіант використання UC— 15

Use case name	Продати предмет
Use case ID	UC-15
Goals	Продати предмет продавцю
Actors	Користувач
Trigger	Користувач бажає продати предмет
Pre-conditions	Користувач перебуває у меню взаємодії з продавцем
Flow of Events	Користувач обирає предмет та натискає кнопку для продажі
Extension	Персонаж має предмет у інвентарі
Post-condition	Продано предмет та додано кількість монет

Таблиця 2.16 — Варіант використання UC— 16

Use case name	Закрити меню взаємодії з продавцем
Use case ID	UC-15
Goals	Закрити меню взаємодії з продавцем, продовжити ігровий процес
Actors	Користувач
Trigger	Користувач бажає закрити меню взаємодії з продавцем
Pre-conditions	Користувач перебуває у меню взаємодії з продавцем
Flow of Events	Користувач натискає кнопку закриття меню взаємодії з продавцем
Extension	Персонаж має достатньо монет
Post-condition	Придбано предмет та поміщено його до інвентаря

2.2. Аналіз системних вимог

Оскільки розробка ведеться на кросплатформенному рушії, вимог до операційної системи немає.

Мінімальна конфігурація технічних засобів:

– Процесор : Intel Core i3-6100 або аналог за продуктивністю

– ОЗУ: 6 ГБ

– Графічний процесор: Nvidia GeForce GTX980 або аналогічний за продуктивністю

– Місце на диску: 10 ГБ HDD

Рекомендована конфігурація технічних засобів:

– Процесор : Intel Core i5-8400 або аналог за продуктивністю

– ОЗУ: 8 ГБ

– Графічний процесор: Nvidia GeForce GTX1060 або аналогічний за продуктивністю

– Місце на диску: 10 ГБ SSD

2.3. Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.17 наведено загальну модель вимог, а в таблицях 2.18 – 2.28 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.29.

Таблиця 2.17 —Модель вимог у загальному виді

Назва	Код	Пріорітет	Ризик	Статус
Вивід меню	FR-1	Високий	Високий	Готово
Можливість почати гру	FR-2	Високий	Середній	Готово
Можливість вийти з гри	FR-3	Низький	Середній	Готово
Можливість взаємодії з продавцем	FR-4	Середній	Низький	Готово
Можливість взаємодії з порталом	FR-5	Середній	Низький	Готово
Система атрибутів	FR-6	Середній	Середній	Готово
Система інвентаря	FR-7	Середній	Середній	Готово
Система навичок	FR-8	Середній	Високий	Готово

Продовження таблиці 2.17

Робота штучного інтелекту	FR-9	Високий	Високий	Готово
Можливість померти	FR-10	Низький	Низький	Готово
Можливість вбити ворога	FR-11	Низький	Низький	Готово

Таблиця 2.18 —Функціональна вимога FR-1

Назва	Вивід меню
Опис	Ігровий застосунок повинен коректно відображати всі елементи графічного інтерфейсу, такі як меню взаємодії з продавцем та меню персонажа

Таблиця 2.19 —Функціональна вимога FR-2

Назва	Можливість почати гру
Опис	Ігровий застосунок повинен дати можливість користувачу почати гру

Таблиця 2.20 —Функціональна вимога FR-3

Назва	Можливість вийти з гри
Опис	Ігровий застосунок повинен дати можливість користувачу завершити гру

Таблиця 2.21 —Функціональна вимога FR-4

Назва	Можливість взаємодії з продавцем
Опис	Ігровий застосунок повинен дати коректно опрацьовувати взаємодію користувача та продавця

Таблиця 2.22 —Функціональна вимога FR-5

Назва	Можливість взаємодії з порталом
Опис	Ігровий застосунок повинен коректно опрацьовувати переміщення персонажа з одного рівня на інший

Таблиця 2.23 —Функціональна вимога FR-6

Назва	Система атрибутів
Опис	Ігровий застосунок повинен коректно реалізовувати атрибути персонажа

Таблиця 2.24 —Функціональна вимога FR-7

Назва	Система інвентаря
Опис	Ігровий застосунок повинен коректно опрацьовувати дії з інвентарем

Таблиця 2.25 —Функціональна вимога FR-8

Назва	Система навичок
Опис	Ігровий застосунок повинен коректно опрацьовувати використання та покращення навичків

Таблиця 2.26 —Функціональна вимога FR-9

Назва	Робота штучного інтелекту
Опис	Ігровий застосунок повинен коректно приймати рішення для поведінки NPC

Таблиця 2.27 —Функціональна вимога FR-10

Назва	Можливість померти
Опис	Ігровий застосунок повинен дати можливість користувачу бути вбитим

Таблиця 2.28 —Функціональна вимога FR-11

Назва	Можливість вбити ворога
Опис	Ігровий застосунок повинен дати можливість користувачу вбити ворога

Таблиця 2.29 —Матриця трасування вимог

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11
UC-1	+	+									
UC-2	+		+								
UC-3	+										
UC-4											
UC-5	+										
UC-6	+							+			
UC-7	+					+	+				
UC-8	+					+	+				
UC-9	+										
UC-10	+							+			
UC-11	+				+						
UC-12	+				+						
UC-13	+			+							
UC-14	+			+		+	+				
UC-15	+			+		+	+				
UC-16	+			+							

2.4. Розроблення нефункціональних вимог

Безсумнівно, нефункціональні вимоги відіграють важливу роль у якості даної розробки — гри, побудованої на основі Unreal Engine 5 із використанням GOAP. Під час розробки особливу увагу було приділено продуктивності, адже для гри критично важливо забезпечити плавний ігровий процес без затримок та погіршення швидкодії. Було оптимізовано логіку поведінки агентів та структурні компоненти гри з урахуванням продуктивності навіть на середньому апаратному забезпеченні.

Крім цього, значну роль відіграє масштабованість — дана розробка повинна легко розширюватися, підтримуючи додавання нових навичок, предметів та цілей без необхідності зміни існуючого функціоналу. Саме тому архітектура реалізована з використанням принципів модульності та об’єктно-орієнтованого програмування, що дозволяє ефективно впроваджувати нові елементи.

Ще однією ключовою нефункціональною вимогою була зрозумілість коду та підтримуваність. У межах даної розробки дотримано принципів чистого коду, використано зрозумілі назви класів та методів, а також здійснено документування логіки складних систем, таких як GOAP, система вмінь та атрибутів.

Не можна обійти стороною й безпеку — хоча дана розробка не працює в мережевому середовищі, все одно було реалізовано перевірки на коректність дій гравця та передбачено обробку виняткових ситуацій, щоб уникнути критичних збоїв у логіці гри.

Загалом, дана розробка враховує такі важливі нефункціональні вимоги як продуктивність, масштабованість, підтримуваність та безпека, що, на мою думку, позитивно впливає на її якість як повноцінного програмного продукту.

2.5. Аналіз економічних показників програмного забезпечення

Для розрахунку економічних показників програмного забезпечення було використано метод оцінки функціональних точок (Functional Points, FP).

Функціональні точки є одиницею вимірювання, що дозволяє визначити обсяг функціональності, яку надає програмне забезпечення користувачам. Вони відображають кількість реалізованих бізнес-функцій у системі та дають змогу оцінити функціональний розмір (FSM) програмного продукту.

Метод функціональних точок є ефективним підходом до кількісного оцінювання функціональності ПЗ у термінах, зрозумілих користувачу. Він ґрунтується на підрахунку функцій згідно з вимогами до системи і широко використовується в індустрії для планування, визначення витрат і обсягів робіт при створенні програмного забезпечення.

У таблиці 2.17 було наведено функціональні вимоги до програмного продукту, що моделюється. На їх основі здійснено підрахунок кількості функціональних вимог (ФВ) з урахуванням рівня складності. Для обчислення кількості нескоригованих функціональних точок (Unadjusted Functional Points, UFP) застосовуються вагові коефіцієнти: 3 — для ФВ низької складності, 4 — середньої та 6 — високої. Ці коефіцієнти є загальноприйнятим стандартом для обчислення UFP. Підсумки обчислень представлені в таблиці 2.30.

Таблиця 2.30 — Підрахунок UFP

Складність	**Кількість**	**Ваговий коефіцієнт**	**Результат**
Проста	3	3	9
Середня	4	4	16
Висока	4	6	24
		Загальна кількість UFP	49

Наступним етапом є обчислення значення коригуючого чинника (VAF). Це фактор, який використовується для коригування значення UFP у відношенні до факторів середовища. Він обчислюється за наступною формулою:

$$VAF = 0,65 + TDI/100,$$

де TDI — підсумкове значення оцінки факторів середовища (GSC). Кожен з 14 факторів середовища має оцінку в межах 0-5. Підрахунок TDI для даного програмного забезпечення наведена в таблиці 2.31.

Таблиця 2.31 — Підрахунок TDI

Фактор	**Оцінка**
Data Communications	0
Distributed Data Processing	0
Performance	5
Heavily Used Configuration	4
Transaction Rate	3
On-Line Data Entry	0
End-User efficiency	5
On-Line Update	0
Complex Processing	4
Reusability	3
Installation Ease	2
Operational Ease	3
Multiple Sites	0
Facilitate Change	2
Разом	22

На основі отриманого значення TDI тепер можна обчислити VAF:

$$VAF = 0,65 + 31/100 = 0,96.$$

Тепер можна обчислити підсумкове значення скоригованих функціональних точок за наступною формулою:

$$FP = VAF \times UFP = 0,96 \times 49 = 47,04.$$

Використовуючи отримане значення скоригованих функціональних точок тепер, можна провести оцінку часу розробки програмного забезпечення. Враховуючи, що розробка ведеться однією людиною, тривалість робочого тижня рівна 40 годин та значення середньої кількості годин для реалізації однієї функціональної точки для мови C++ приблизно дорівнює 13 годин/функціональну точку , отримаємо наступні значення:

$$Hours = 47,04 \times 13 \approx 612.$$

$$Weeks = 612/40 = 15,3.$$

$$Months = 15,5/4,5 \approx 3,4.$$

На основі отриманого значення часу розробки можна підрахувати вартість даної розробки. За грудень 2024 року медіана місячної заробітньої плати C++ розробника без досвіду роботи становить $850 (~35317,86 грн). Відповідно, вартість даної розробки буде становити:

$$Cost = Month \times Salary = 3,4 \times 850 = \$2890 \approx 120080,71UAH.$$

2.6. Постановка завдання на розробку програмного забезпечення

Призначення – ігровий застосунок, створений на рушії Unreal Engine 5, що використовує підхід GOAP (Goal-Oriented Action Planning) для побудови поведінки персонажів. Дана розробка реалізована як прототип інтелектуальної системи поведінки в грі з фокусом на модульність, розширюваність та дослідження системного дизайну. Основною метою застосунку є демонстрація та тестування взаємодії підсистем, зокрема: системи атрибутів, інвентаря, вмінь та поведінки, побудованої на основі цілей і дій.

Метою розробки є підвищення рівня занурення та реіграбельності гри шляхом впровадження адаптивного штучного інтелекту, який реагує на стан

світу, коригує поведінку ворогів у реальному часі та забезпечує різноманітні бойові сценарії, при цьому зберігаючи високу продуктивність та масштабованість проєкту, використовуючи оптимізовані алгоритми та ефективне управління ресурсами рушія Unreal Engine. Ця мета досягається завдяки реалізації таких підсистем:

– Система атрибутів, що складається з первинних і вторинних характеристик.

– Система інвентаря, яка зберігає інформацію про предмети та передає модифікатори атрибутів до системи атрибутів при екіпіруванні.

– Система вмінь, побудована на основі підкласів конкретних умінь, які об'єднані компонентом і використовуються персонажем в залежності від контексту.

– GOAP-система, яка відповідає за логічне планування дій на основі визначених цілей.

Таким чином, дана розробка дозволяє створити гнучке середовище для тестування інтелектуальних агентів та глибше зануритися в розробку поведінки NPC на основі цілей.

Висновки до розділу

Було виконано комплексний аналіз вимог до програмного забезпечення, необхідних для розробки ігрового проєкту на рушії Unreal Engine 5 із реалізацією системи GOAP (Goal-Oriented Action Planning).

По-перше, було розглянуто сценарії використання програмного забезпечення, зокрема взаємодію користувача з основними механіками гри, такими як керування персонажем, використання інвентара, застосування вмінь та взаємодії зі світом. Цей етап дав змогу сформувати перелік ключових сценаріїв, що є критичними для створення функціонально насиченої та зручної гри.

По-друге, було проведено аналіз технічних вимог до системи, включаючи апаратні обмеження та архітектурні особливості, з метою забезпечення стабільної та продуктивної роботи гри на сучасних комп’ютерних системах.

По-третє, були сформульовані функціональні вимоги до гри: коректна робота системи атрибутів, інвентаря та вмінь, взаємодія персонажа зі світом. Окрему увагу було приділено логіці прийняття рішень агентами в рамках GOAP — для досягнення адаптивної, гнучкої поведінки віртуальних істот.

Окрім цього, були розроблені нефункціональні вимоги, що стосуються продуктивності, масштабованості, підтримуваності, безпеки та надійності. Дана розробка побудована з урахуванням принципів чистого коду та використання шаблонів проєктування, що гарантує її гнучкість та зручність у подальшій розробці.

Також було проведено функціональний аналіз економічних вимог, за результатами якого було отримано оцінку часу та вартості розробки. Завдяки цим оцінкам можна ефективно визначити темп та графік реалізації обраних функціональних точок.

Таким чином, у цьому розділі було проведено всебічний аналіз і розробку вимог до програмного забезпечення, що є міцним підґрунтям для подальшої реалізації гри. Визначені вимоги забезпечують створення якісного, функціонального та орієнтованого на гравця продукту.

3. КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Архітектура програмного забезпечення

Ігрове програмне забезпечення побудоване за компонентною архітектурою, що базується на принципах об’єктно-орієнтованого програмування та реалізоване на рушії Unreal Engine 5.

Всі персонажі, а саме гравець та NPC, є похідними класами від базового персонажу. Для реалізації ігрових механік було обрано модульний підхід. Це означає, що кожна функціональна система є окремим модулем, що незалежно опрацьовує власні дані. У Unreal Engine це було імплементовано за допомогою ActorComponent. Ось опис створених компонентів:

– система атрибутів (Attribute Component) — обчислює ігрові атрибути персонажа;

– система вмінь (Ability Component) — опрацьовує використання вмінь;

– система інвентара (Inventory Component) — зберігає об’єкти предметів і модифікує атрибути при їх екіпіруванні.

Система атрибутів містить 3 первинних атрибути: сила, спритність та інтелект; та вторинні атрибути, які обчислюються на основі первинних згідно встановлених коефіцієнтів. Для вторинних атрибутів було вирішено обрати наступні: кількість одиниць здоров’я, кількість одиниць ресурсу, фізичний захист, магічний захист, шкода, підсилення магічної шкоди, швидкість переміщення та швидкість атаки.

Система вмінь містить список доступних для використання навичок. Кожна навичка це похідний клас від базової навички і має власну логіку при активації. Базова навичка містить поля про перезарядку, кількість ресурсу, що буде використано при активації.

Система інвентара містить інформацію про “рюкзак” персонажа та екіпіровані предмети. Кожен предмет має модифікатори, що впливають на атрибути. Наприклад, зброя може мати модифікатор “+20 до шкоди”, тобто до обчислених атрибутів персонажа буде додано 20 одиниць шкоди.

Похідні класи від BaseCharacter було створено за допомогою візуального скриптингу, що у Unreal Engine називається Blueprint. Вони не впроваджують нову логіку, а лише замінюють певні поля за замовчуванням, такі як 3D модель, значення основного первинного атрибуту та подібні. Діаграму класів, що були використані у BaseCharacter, наведено на рисунку.

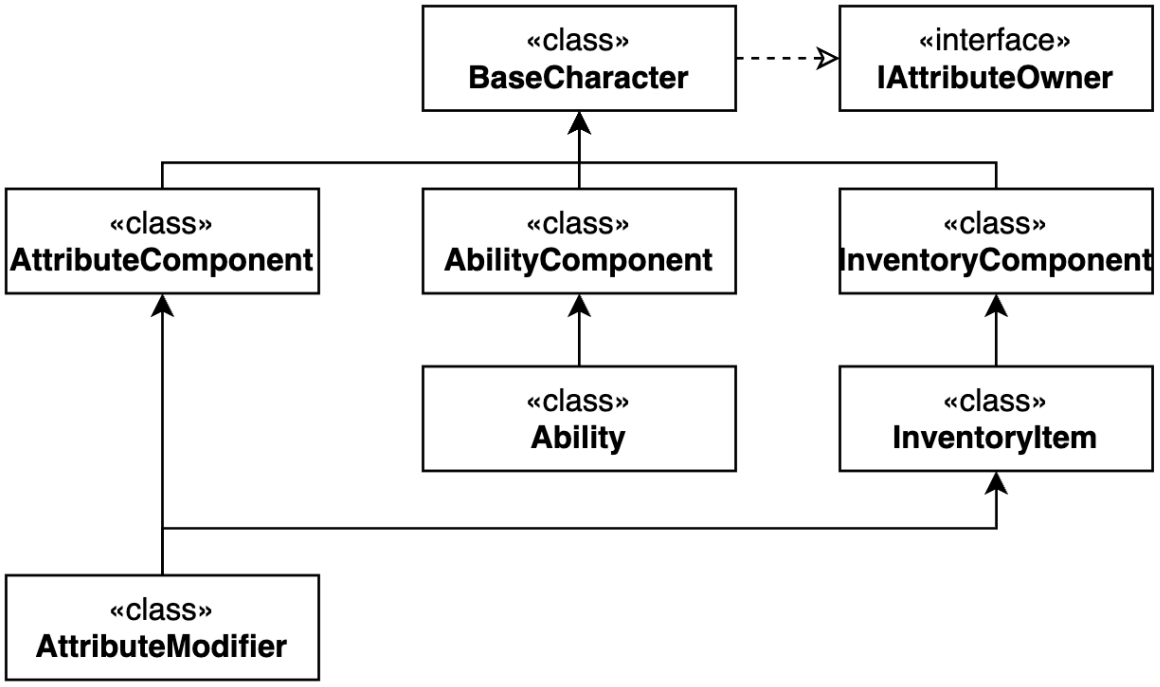


Рисунок 3.1 — Діаграма класів для BaseCharacter

У Unreal Engine для керування персонажем використовується контролер. Для тих, що керується гравцем — це Player Controller, для NPC — AI Controller. У випадку ігрового персонажу, Player Controller обробляє користувацький ввід, наприклад, пересування чи взаємодія з інтерфейсом. Для NPC AIController приймає рішення про поведінку персонажу. У даній розробці було використано систему Goal Oriented Action Planning у ролі AI Controller. Для використання такої системи також необхідно було визначити сутності Action та Goal.

GOAP AI Controller це клас, що зберігає важливу інформацію для та про планування. А саме, цей клас містить поля про доступні для персонажа цілі та дії, актуальну ціль та план дій для її досягнення. Решту даних, такі як нинішній стан світу, містить Blackboard, але про це пізніше.

Action — це логічна сутність, що описує потенційну дію в рамках системи GOAP. Вона містить інформацію про передумови для виконання, очікуваний результат (зміну стану світу), умовну вартість дії, а також посилання на відповідну реалізацію дії. Важливо підкреслити, що сама сутність Action не взаємодіє безпосередньо з ігровим світом — її роль полягає виключно в логічному плануванні. Реальна взаємодія з ігровим світом відбувається під час виконання дії, на яку вказує посилання.

Goal — це логічна сутність, яка представляє ціль поведінки GOAP-агента. Вона визначає бажаний стан світу, до якого агент прагне, а також містить пріоритет, що визначає важливість цієї цілі порівняно з іншими. Goal не містить логіки взаємодії зі світом і не виконується безпосередньо — її призначення полягає у спрямуванні планувальника на побудову послідовності дій (Action), необхідних для досягнення визначеного стану. Таким чином, Goal є стартовою точкою для формування поведінки агента в рамках системи планування.

Діаграма класів для GOAP AI Controller наведено на рисунку 3.2.

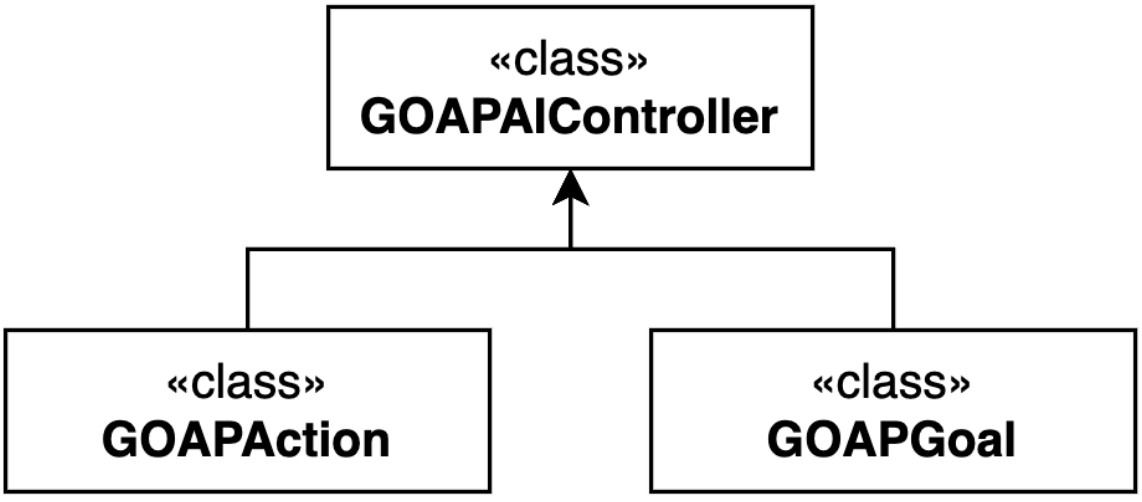


Рисунок 3.2 — Діаграма класів для GOAP AI Controller

3.2. Архітектурні рішення та обґрунтування вибору засобів розробки

Зараз поговоримо детальніше про архітектурні рішення цієї розробки. Основним способом для реалізації ШІ в Unreal Engine є дерева прийняття рішень, або ж Behaviour Tree, оскільки у цьому рушії уже створено весь функціонал для створення власного такого дерева. Оскільки ця система є оптимізованою, було прийняте рішення використати її сильні сторони у розробці своєї системи, а саме впровадити власний рушій для GOAP на основі функціоналу Behaviour Tree.

По-перше, Behaviour Tree використовує сутність Blackboard для спільного використання даних між всіма вузлами дерева. Також, Behaviour Tree мають сутність Service, що виконується з заданим інтервалом часу й записує результат свого виконання в Blackboard. Ще існують декоратори, які розширюють функціонал вузлів. Один з них використовує перевірку значення поля Blackboard у ролі такої умови. Цей функціонал є оптимізованим і дуже підходящим для створення GOAP.

По-друге, конструювання Behaviour Tree відбувається графічно, відповідно при проведенні тестування можна візуально побачити як виконується дерево. Також можна поставити на паузу весь ігровий цикл, щоб перевірити значення полів Blackboard.

Отже, на основі цього було прийнято рішення створити систему GOAP таким чином, що єдині вузли такого дерева це вузол з планування та вузол з виконання плану. Вузол з планування матиме декоратор з умовою на булеве поле, що зображає необхідність повторного планування. Таким чином, якщо ціль залишилась незмінною та запланована дія може бути виконана, необхідності в повторному плануванні немає і можна зекономити обчислювальні ресурси. Інший же вузол лише виконує заплановану дію. Також на одному з вузлів встановлено сервіс, який виконує оцінку стану світу та перевіряє чи запланована раніше ціль є актуальною. Сервіс виконується 2 рази на секунду, тому великого навантаження на обчислювальні ресурси немає.

У результаті було отримано наступне “дерево” системи GOAP, яке зображено на рисунку 3.3.

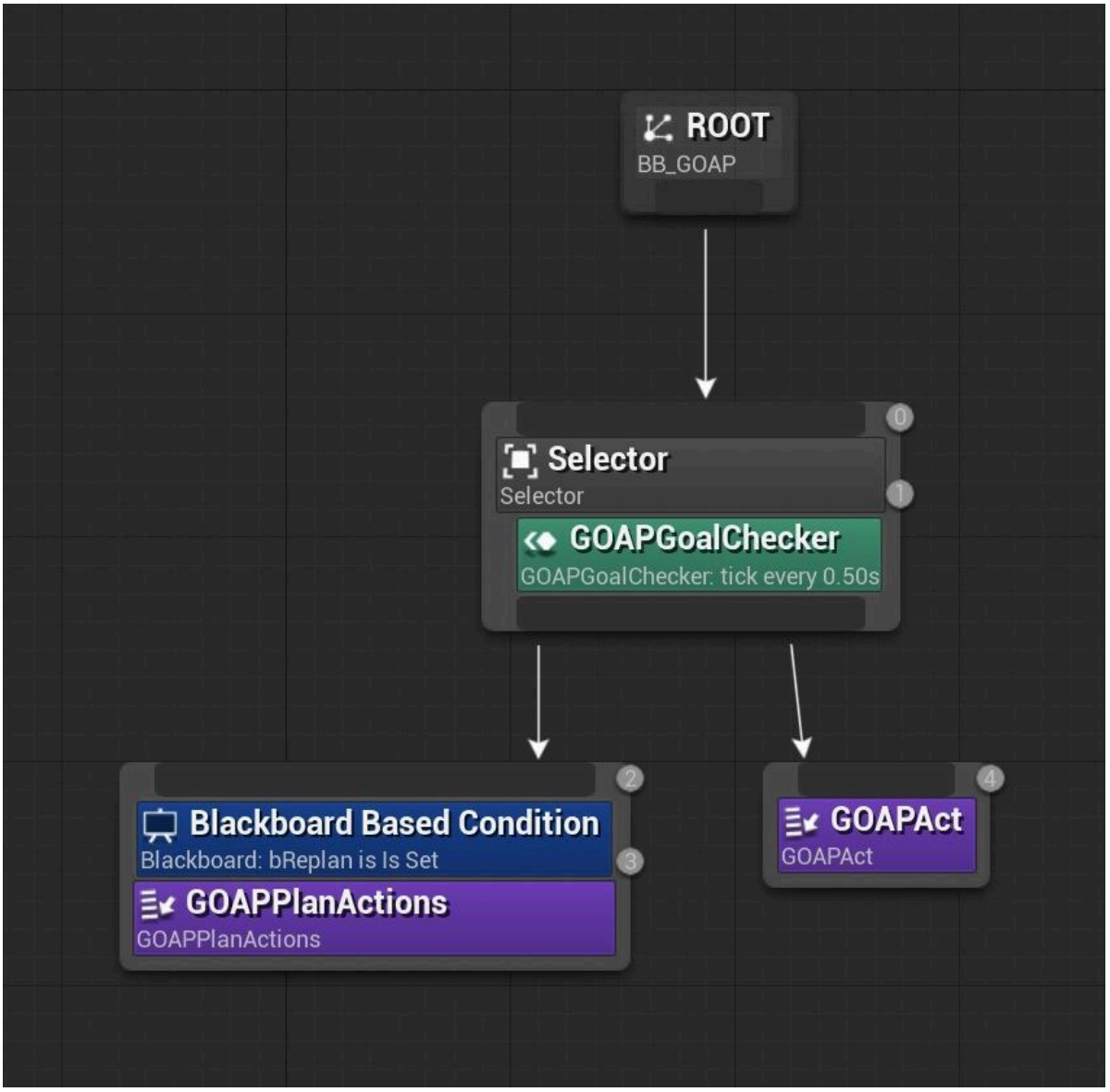


Рисунок 3.3 — Система GOAP

Іншим досить важливим рішенням було використовувати BitMask як структуру даних, що описує стан світу. Під час планування дій перевіряється велика кількість можливий станів світу, відповідно якщо вони описуються структурами даних, то відбувається постійне перевиділення пам’яті, що призводить до погіршення частоти кадрів. Наприклад, базовий стан світу містить інформацію про нинішню. та максимальну кількість одиниць здоров’я та ресурсу неігрового персонажа та гравця у змінних типу float (кожен 4 байти),

позиція гравця та неігрового персонажа у вигляді тривимірних векторів типу float (кожен вектор 12 байт), та декілька значень типу boolean (кожен 1 байт). У результаті отримуємо, що структура даних стану світу займатиме принаймні 64 байти пам’яті. У процесі пошуку кількість станів, що зберігаються тимчаово в пам’яті, може перевищувати 100. Таким чином для планування дій одного персонажа буде витрачатись понад 6400 байт, або ж приблизно 6 кілобайт. При плануванні дій 10 персонажів постійний перезапис 60 кілобайт призведе до значного погіршення швидкодії гри. Однак, якщо записувати стан в 32 біта int32, то у зв’язку з надзвичайно малим об’ємом пам’яті (4 байти), що менше однієї структури стану в 16 разів, то виділення пам’яті на 100 станів відбувається абсолютно непомітно. Звісно, обчислення значень бітів такої маски вимагає певні обчислення процесора, однак для встановлення значення кожного біта потрібно виконати перевірку декількох логічних умов та виконати одну бінарну операцію. Ці обчислення в машинному коді еквівалентні виконанню декількох інструкцій, тому таке навантаження не призводить до затримок інших обчислень. Таким чином було досягнуто хороші рівні швидкодії застосунку.

Для розробки даної гри було обрано рушій Unreal Engine 5, оскільки він забезпечує високоякісну графіку, потужні інструменти для створення складної логіки та підтримку систем на основі компонентів, що є критично важливим для реалізації GOAP. Unreal Engine 5 дозволяє ефективно працювати з Blueprints та C++, що дає змогу поєднувати візуальне програмування з низькорівневим контролем. Вибір рушія також зумовлений наявністю великої спільноти, документації та підтримкою сучасних стандартів оптимізації. Такий підхід дозволяє створити модульну, масштабовану архітектуру та реалізувати інтелект NPC з використанням GOAP.

Іншою перевагою для використання цього рушія є те, що він має підтримку Data-Driven Development. Це означає, що ігрова логіка може бути відокремлена від даних, що використовуються, такі як параметри певних сутностей. У випадку даної розробки, усі параметри для InventoryItem були описані у таблиці даних, на основі яких потім генерується об’єкт.

3.3. Конструювання програмного забезпечення

Поговоримо детальніше про реалізацію GOAP. Як було згадано вище, система складається лише з 3 основних етапів. Перший етап це формування стану світу й перевірка актуальності цілі. Блок-схеми роботи основних функцій наведено на рисунках 3.4-3.5.

Оскільки ціль це лише перелік передумов та пріоритет, то вибір актуальної цілі є простим: повертаємо ту ціль, передумови якої виконуються і що має найвищий пріоритет.

Після виконання цих функцій має значення поточного стану світу, актуальну ціль та вирішили, чи потрібно перепланувати дії. Ці дані записуються в Blackboard і використовуються на наступному етапі.

Другий етап GOAP це власне планування. Існують варіації з різними алгоритмами пошуку, однак дане програмне забезпечення реалізує алгоритм A*. Це є досить простий, водночас ефективний алгоритм. Блок-схему його роботи для цієї задачі наведено на графічному матеріалі КПІ.ІТ-0222.045480.06.99.ССА.

Отриманий план зберігається в GOAP AI Controller. Він використовується в останньому етапі, а саме виконанні плану.

Останній етап є найпростішим у виконанні, оскільки все, що він робить це викликає функцію присвоєну дії. Для цього було використано функціонал системи навичок, оскільки більшість дій пов’язані з або атаками на персонажа гравця, або є заміною ручного вводу гравця як, наприклад, переміщення. Тому створення декількох допоміжних навичок виглядає як оптимальне рішення.

Загальний принцип роботи описаного алгоритму наведено у вигляді блок-схеми на рисунку 3.7.

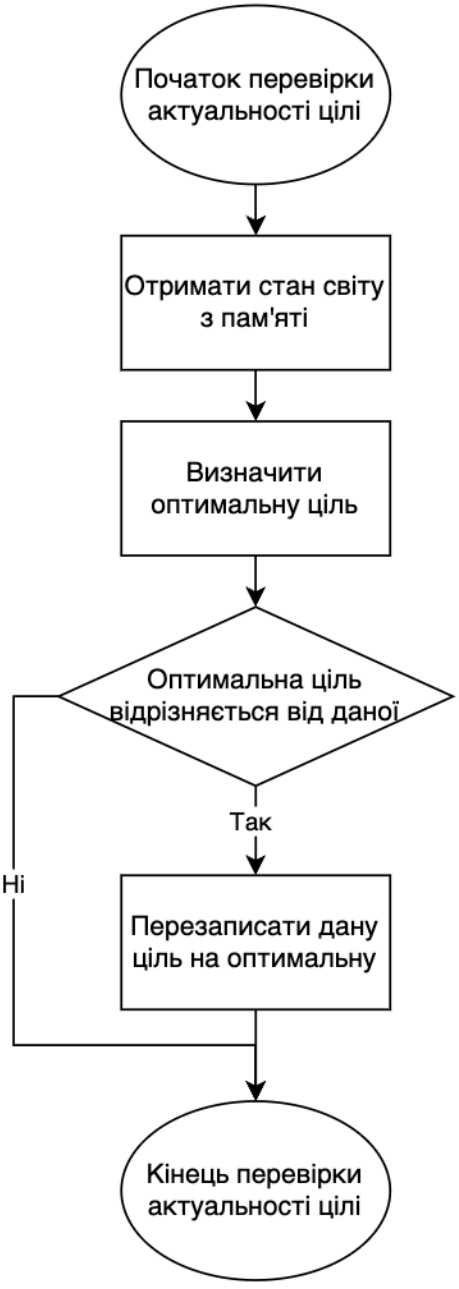


Рисунок 3.4 — Блок-схема перевірки актуальності цілі

Рисунок 3.5 — Блок-схема вибору актуальної цілі

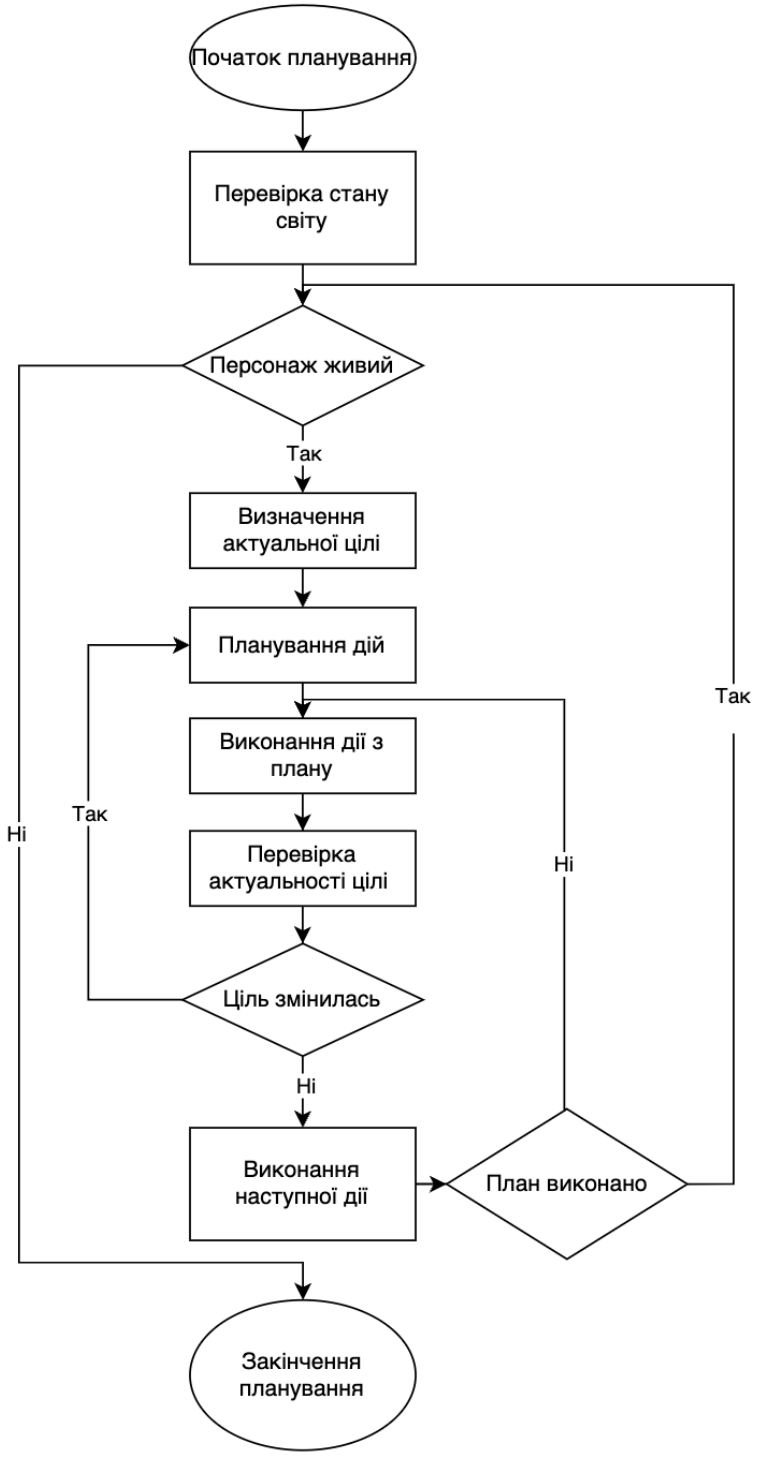


Рисунок 3.7 — Блок схема загального принципу роботи GOAP

Якщо ж так сталося, що відбулась зміна стану між перевірками актуальності цілі і певні вхідні дані відсутні, то відбувається зміна значення

мітки перепланування на true. Наприклад, система зору оновлює поля, пов’язані з виявленням ворога. Якщо ж його немає в полі зору, то вказівник на комірку пам’яті з об’єктом ворога відсутній. Припустимо, що це сталося в період часу між перевірками і наступна дія, яка виконуватиметься вимагає дані про ворога. У такому випадку відбудеться перепланування ще до перевірки актуальності цілі, оскільки Behaviour Tree починає виконання дерева з початку як тільки було виконано останній вузол. Таким чином NPC завжди може реагувати на зміну ситуації.

3.4. Аналіз безпеки даних

Дана розробка використовується локально, тобто загроза втрати чи пошкодження даних під час їх проходження в мережі відсутня. Також розробка не використовує дані користувача, відповідно загроза порушення конфіденційності даних користувача відсутня. Користувач може виконувати лише обмежений перелік дій, у той час як переважна більшість логіки не вимагає його втручання. На випадок виконання користувачем непередбачуваної операції відбувається валідація даних на у межах кожної функції. Таким чином можна стверджувати, що безпека даних гарантована.

Висновок до розділу

Було обґрунтовано вибір рушія Unreal Engine 5, який забезпечує високоякісну графіку, сучасні інструменти для розробки складної логіки, підтримку компонентно-орієнтованої архітектури, а також зручне поєднання візуального програмування через Blueprints і гнучкого контролю за допомогою мови C++. Вибір рушія також обумовлений широкою документацією, активною спільнотою розробників і підтримкою підходу Data-Driven Development, що сприяє гнучкості проєктування.

Для реалізації адаптивної поведінки NPC було впроваджено власну систему GOAP на базі функціоналу Behaviour Tree, що дозволило використати

вже наявні оптимізовані механізми, такі як Blackboard, Decorator та Service. Архітектурно програмне забезпечення побудовано за модульним принципом, де кожна функціональна підсистема (атрибути, вміння, інвентар) реалізована у вигляді окремого компонента. Це забезпечує високу масштабованість і спрощує тестування. Система GOAP оптимізована через використання BitMask для збереження стану світу, що мінімізує навантаження на пам’ять і покращує продуктивність.

Таким чином, обрані засоби розробки та архітектурні рішення дозволили створити гнучку, розширювану ігрову систему зі складним штучним інтелектом, здатним адаптуватися до змін навколишнього середовища в реальному часі.

4. АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Аналіз якості ПЗ

З метою забезпечення комфортного ігрового процесу для користувачів було проведено тестування коректності роботи ігрового застосунку. Перевірці підлягав ключовий функціонал програми, розробленої в жанрі Action RPG. Особливий акцент зроблено на перевірку взаємодії між ігровими компонентами та персонажами.

Метою тестування є:

– перевірка коректності роботи програмного. застосунку згідно до функціональних вимог;

– перевірка швидкодії програмного застосунку;

– знаходження проблем, помилок та недоліків з метою їх усунення.

Для перевірки якості ПЗ також було проведено статистичний аналіз програмного коду за допомогою вбудованого аналізатору коду Qodana в середовищі розробки Rider[11] на проблеми, зокрема:

– не коректне перетворення типів;

– великі частини дубльованого коду;

– не коректне використання вказівників.

Загалом було проведено 856 перевірок пов’язаних як з мовою програмування C++, так і Unreal Engine, згідно з офіційною документацією, що містить рекомендації щодо написання програмного коду. Результат статистичного аналізу наведено на рисунку 4.1.

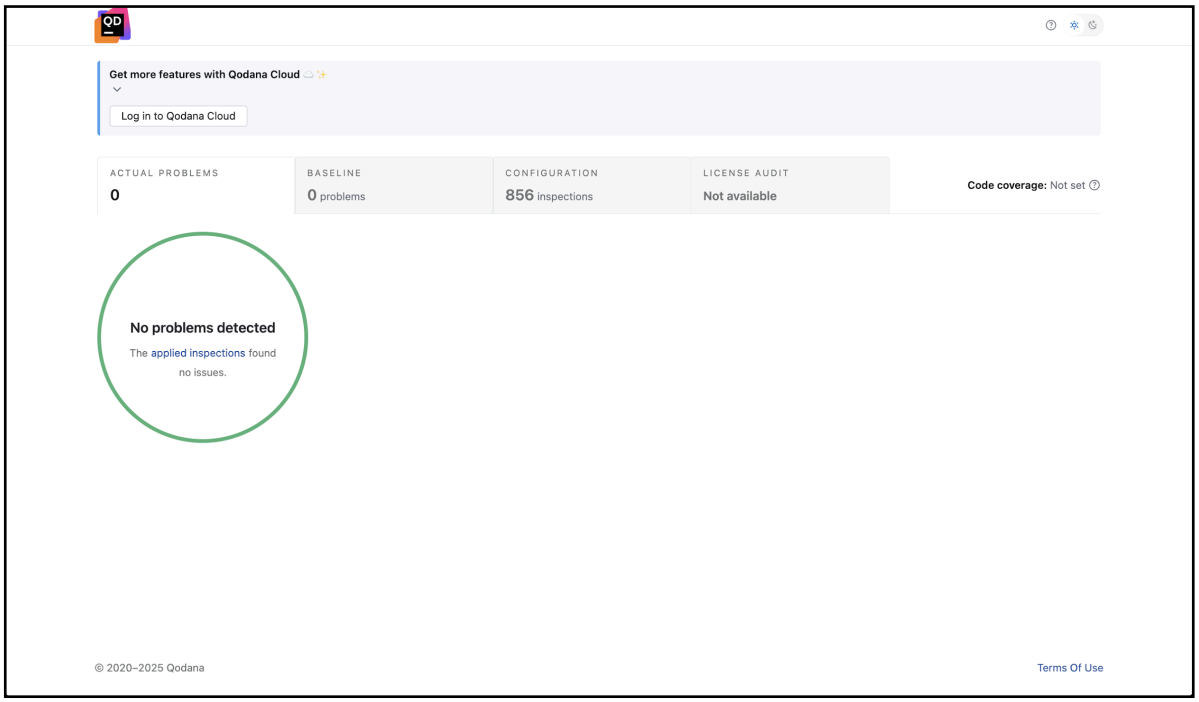


Рисунок 4.1 — Результат статистичного аналізу програмного коду

4.2. Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Таблиця 4.1 — Тест 1 Початок гри

Початковий стан системи	Користувач знаходиться в ігровому меню
Вхідні дані	Немає
Опис проведення тесту	Натискається кнопка початку гри, на етапі вибору персонажу натискається кнопка вибору відповідного персонажу

Оскільки розроблене програмне забезпечення є ігровим застосунком, то найефективнішим рішенням є проведення мануального тестування згідно функціональних вимог. Опис відповідних тестів наведено у таблицях 4.1 - 4.9.

Продовження таблиці 4.1

Очікуваний результат	Після натискання кнопки початку гри з’являється екран вибору персонажа з переліком Їх основних характеристик. Після натискання кнопки вибору відповідного персонажу користувач попадає в стартову локацію й керує обраним персонажем.
Фактичний результат	Збігається з очікуваним

Таблиця 4.2 — Тест 2 Пересування ігрового персонажа

Початковий стан системи	Користувач знаходиться в ігровій локації
Вхідні дані	Немає
Опис проведення тесту	Відбувається натискання на клавіші, що відповідають за пересування ігрового персонажу
Очікуваний результат	Після натискання на клавіші пересування ігровий персонаж переміщується в очікуваному напрямку, згідно керівництва користувача
Фактичний результат	Збігається з очікуваним

Таблиця 4.3 — Тест 3 Взаємодія з продавцем

Початковий стан системи	Користувач знаходиться в стартовій локації
Вхідні дані	Немає
Опис проведення тесту	За допомогою клавіш пересування наближуємо ігрового користувача до продавця. При досягненні необхідної відстані для взаємодії натискається клавіша взаємодії. У відкритому меню взаємодії натискається кнопку покупки предмету, на який персонаж має достатньо грошей. Натискається кнопка продажу купленого предмету

Продовження таблиці 4.3

Очікуваний результат	При досягненні необхідної для взаємодії відстані персонаж продавця підсвічується і з’являється підказка з клавішою для взаємодії. Після натискання клавіші відкривається меню продавця. Предмети продавця, на які не вистачає грошей підсвічуються червоним. Ті, на які достатньо грошей мають кнопку купівлі. Після натискання такої кнопки кількість грошей персонажа зменшилось на значення ціни предмету, куплений предмет показується в інвентарі персонажу та моє кнопку продажу. Після натискання цієї кнопки предмет пропадає з інвентаря, кількість грошей персонажу збільшилась на значення вартості предмету.
Фактичний результат	Збігається з очікуваним

Таблиця 4.4 — Тест 4 Взаємодія з інвентарем персонажа

Початковий стан системи	Користувач знаходиться в ігровій локації
Вхідні дані	Немає
Опис проведення тесту	Натискається клавіша відкриття меню персонажа. У відкритому меню у секції інвентаря натискаємо кнопку для екіпірування предмету. Після цього натискаємо кнопку для зняття екіпірованого предмету
Очікуваний результат	При натисканні клавіші відкриття меню персонажа відкривається меню персонажа з секціями про атрибути персонажа та інвентар. Після натискання кнопки екіпірування, предмет пропадає з інвентаря й з’являється в комірці екіпірованого предмету. Атрибути персонажа оновились згідно модифікаторів предмету. Після натискання кнопки зняття предмету, предмет пропадає з комірки екіпірованого й потрапляє до інвентаря. Атрибути персонажа повернулися до початкового стану.
Фактичний результат	Збігається з очікуваним

Таблиця 4.5 — Тест 5 Робота порталу

Початковий стан системи	Користувач знаходиться в ігровій локації
Вхідні дані	Немає
Опис проведення тесту	За допомогою клавіш пересування, персонажа наближено до порталу так, що він «входить» в нього
Очікуваний результат	При натисканні клавіш пересування так, що персонаж «входить» в портал, відбувається перехід на іншу локацію.
Фактичний результат	Збігається з очікуваним

Таблиця 4.6 — Тест 6 Активація навички

Початковий стан системи	Користувач знаходиться в ігровій локації, навичка доступна для активації
Вхідні дані	Немає
Опис проведення тесту	Натискання клавіші активації навички. Повторне натискання клавіші активації навички
Очікуваний результат	При натисканні клавіші активації навички відбувається використання відповідної навички. Ресурс персонажа зменшився згідно витрат ресурсу на активацію навички, навичка на перезарядці. Повторние натискання клавіші не активує навичку, а виводить повідомлення про її перезарядку
Фактичний результат	Збігається з очікуваним

Таблиця 4.7 — Тест 7 Вибір навичок

Початковий стан системи	Користувач знаходиться в ігровій локації
Вхідні дані	Немає

Продовження таблиці 4.7

Опис проведення тесту	Натискання клавіші відкриття меню навичок. Натискання кнопки вибору навички з переліку навичок. Натискання кнопки отримання навички з переліку навичок.
Очікуваний результат	При натисканні клавіші відкриття меню навичок показується меню з переліком навичок. Ті, що користувач не може отримати підсвічуються червоним. Ті, що можна отримати, мають кнопку «отримати». Ті, що вже отримані, мають кнопку «обрати». Обрані навички не мають кнопку вибору.
Фактичний результат	Збігається з очікуваним

Таблиця 4.8 — Тест 8 Нанесення шкоди ворогу

Початковий стан системи	Користувач знаходиться в підземеллі поруч з ворогом
Вхідні дані	Немає
Опис проведення тесту	Натискання клавіші активації навички
Очікуваний результат	Після активації навички, ворог у зоні ураження навички отримав шкоду. Шкала здоров’я ворога відображає зменшення здоров’я ворога.
Фактичний результат	Збігається з очікуваним

Таблиця 4.9 — Тест 9 Отримання шкоди від ворога

Початковий стан системи	Користувач знаходиться у підземеллі, ворог атакує користувача
Вхідні дані	Немає
Опис проведення тесту	Ворог використовує одну зі своїх навичок, щоб завдати шкоди персонажу користувача.

Продовження таблиці 4.9

Очікуваний результат	Персонаж користувача отримав шкоду. Школа здоров’я гравця відображає зменшення здоров’я. Щосекунди здоров’я персонажа відновлюється.
Фактичний результат	Збігається з очікуваним

4.3. Опис контрольного прикладу

У контрольному прикладі буде описано процес початку гри та основні частини проходження першого підземелля.

На початку гри користувача зустрічає головне меню з кнопками початку гри та виходу (рис 4.2).

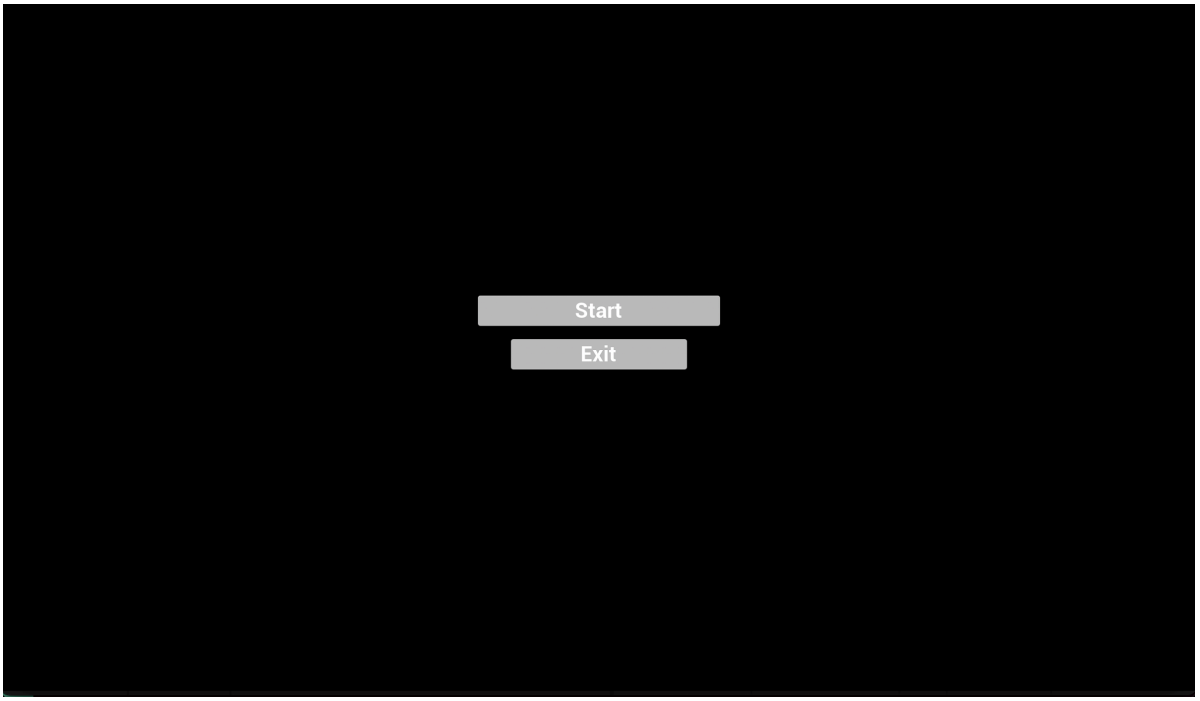


Рисунок 4.2 — Головне меню на початку гри

Після натиснення на кнопку початку відбувається перехід на меню вибору персонажа (рис 4.3).

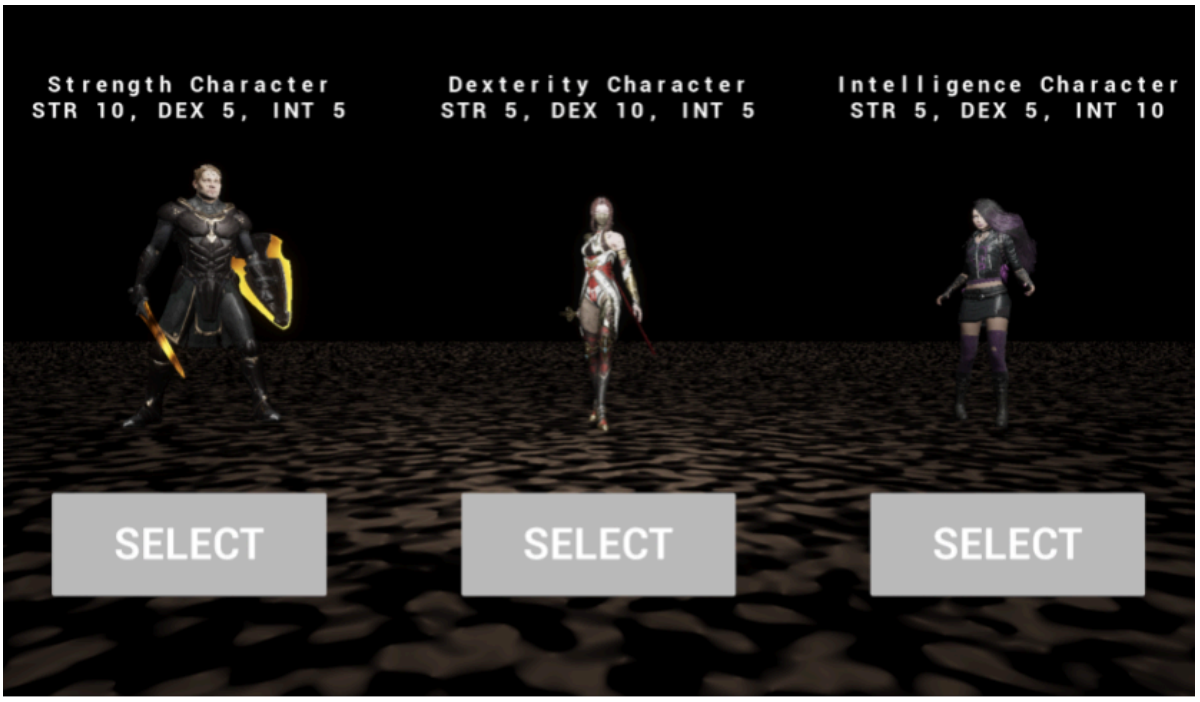


Рисунок 4.3 — Меню вибору персонажа

Після натискання кнопки вибору обраного персонажу відбувається перехід на стартову локацію (рис 4.4).

Рисунок 4.4 — Стартова локація

Тут можна придбати нові речі або продати старі, однак на даних момент у персонажу немає грошей та зайвих предметів для цього (рис 4.5).

Рисунок 4.5 — Меню продавця

Тому, зайдемо в портал і потрапимо в перше підземелля (4.5).

Рис 4.6 — Підземелля

У цьому лабіринті є вороги, яких потрібно знищити, однак головне завдання це знайти та вбити боса підземелля. При зустрічі ворога, щоб завдати йому шкоди потрібно використати одну з доступних навичок (рис 4.7).

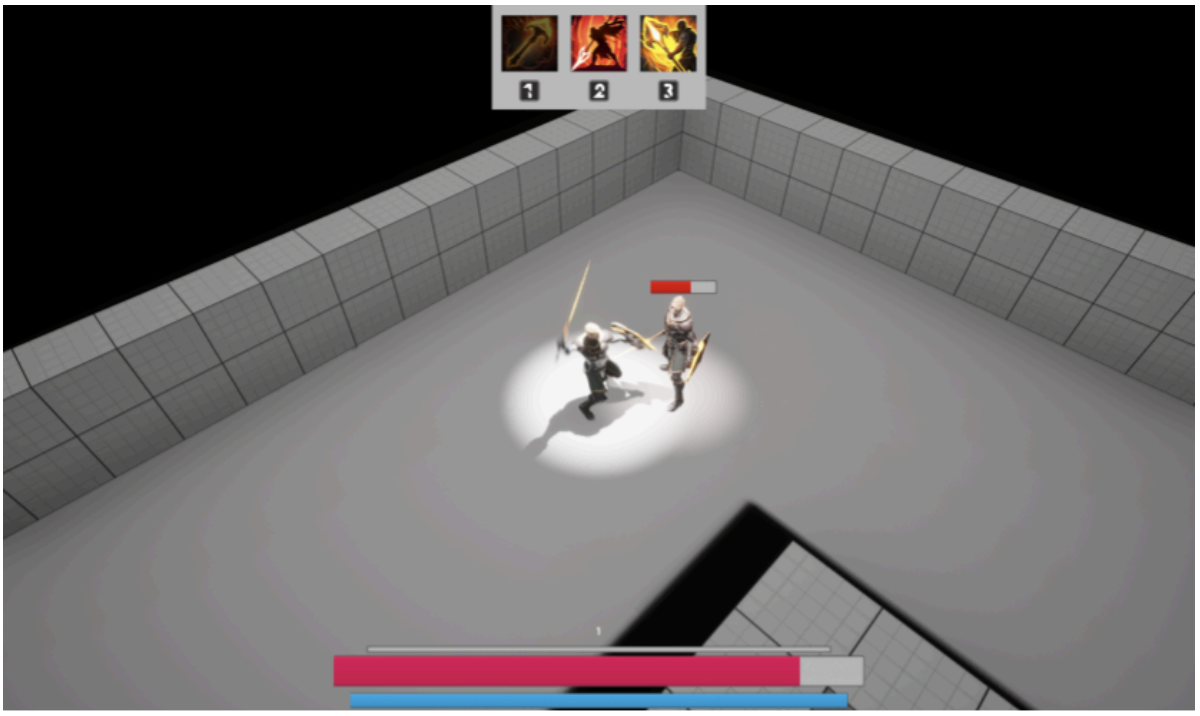


Рисунок 4.7 — Процес битви з ворогом

При смерті ворога з певною вірогідністю випадає предмет, який можна одягнути та покращити власні характеристики (рис. 4.8 - 4.10).

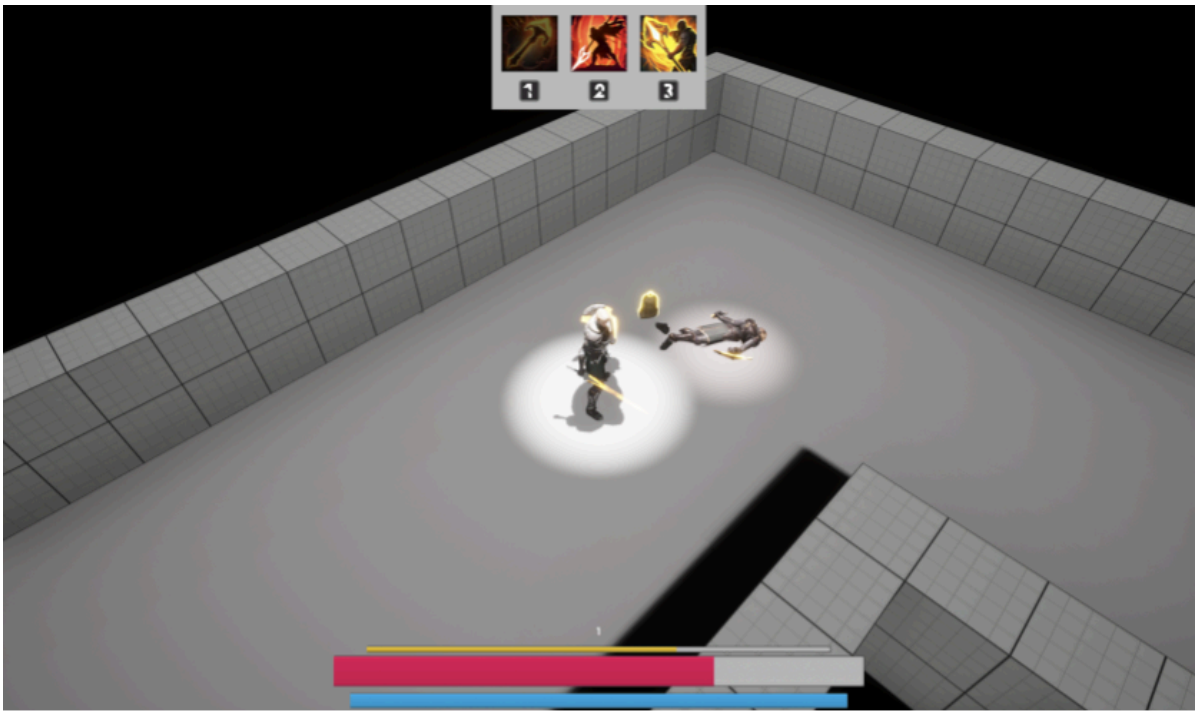


Рисунок. 4.8 — Предмет, що випав з ворога

Рисунок 4.9 — Предмет з ворога в інвентарі

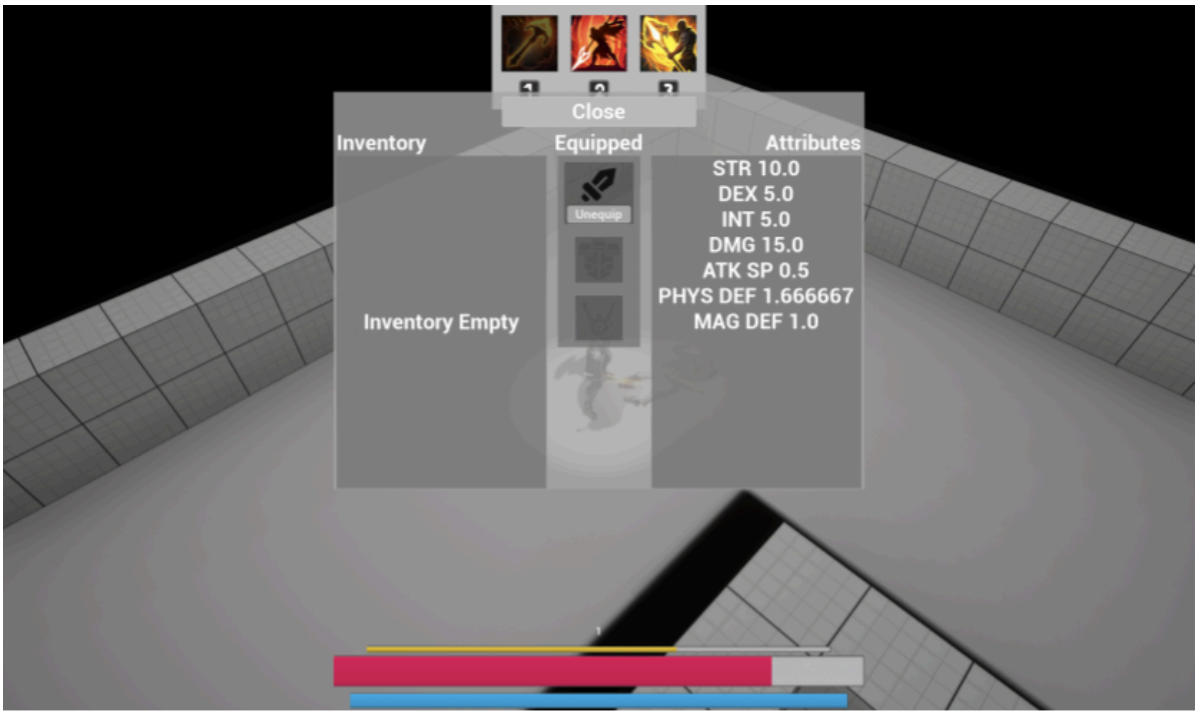


Рисунок 4.10 — Споряджений предмет

Після проходження певної частини лабіринту знайшли боса підземелля, що є дещо сильнішим за звичайних ворогів (рис 4.11).

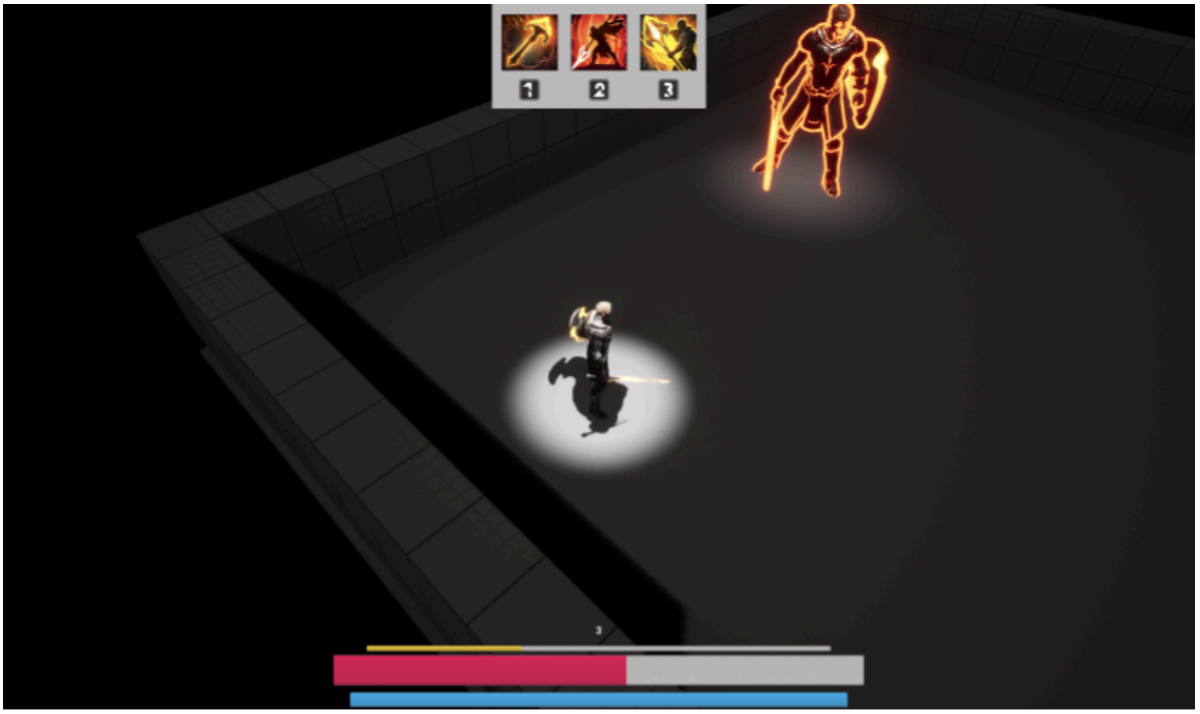


Рисунок 4.11 — Кімната боса підземелля

При вбивстві ворога персонаж отримує досвід, який використовується для отримання нових навичок, особливо багато досвіду приносить вбивство боса підземелля. Коли персонаж покращує свій рівень, він отримує одиниці прокачки навичок, що дають можливість використовувати покращені версії навичок (рис 4.12).

Рисунок 4.12 — Меню прокачки навичок

Вбивство боса призводить до появи порталу в стартову локацію, з якої потім потрапляємо в наступне підземелля (рис 4.13-4.14). Гра продовжується допоки персонаж гравця не помре. Після цього гравець починає гру з початку.

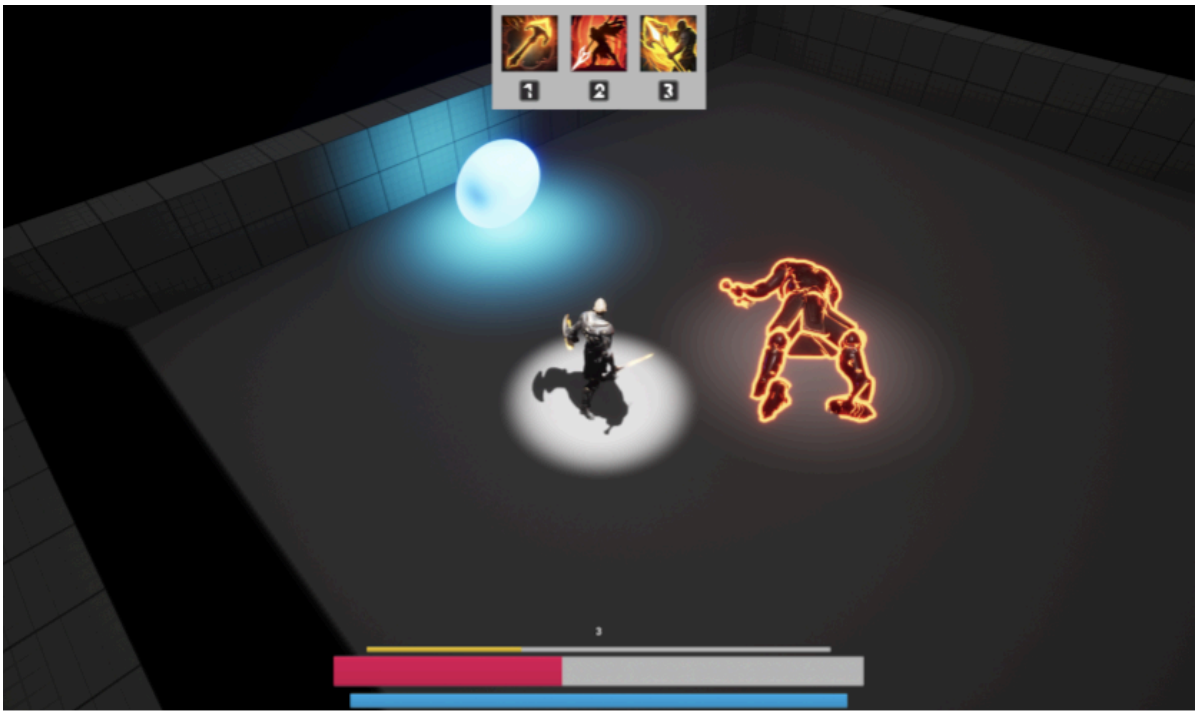


Рисунок 4.13 — Портал до стартової локації

Рисунок 4.14 — Перехід до стартової локації після підземелля

Висновок до розділу

Під час розробки даного ігрового застосунку було проведено тестування якості програмного коду, функціональних та нефункціональних вимог.

Обраним способом тестування є мануальне тестування, що дозволяє перевірити коректність роботи застосунку при реальній взаємодії з ним. Таким чином було перевірено, що всі функціональні вимоги працюють як очікувалось та обрані нефункціональні вимоги теж присутні.

Було задокументовано процес проходження першого рівня ігрового застосунку, включаючи взаємодію з різними реалізованими системами.

5. РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1. Розгортання програмного забезпечення

Для створення робочої версії застосунку для різних платформ було використано середовище розробки, а саме редактор Unreal Engine. Спершу потрібно перевірити та усунути наявні помилки в програмному коді. На стаді збірки та компіляції проекту при їх наявності редактор вкаже на них і не дасть продовжити створення.

Отже, для створення робочої версії ігрового застосунку для поширення, потрібно відкрити меню Platforms -> Project Launcher (рис. 5.1). У відкритому меню, потрібно створити новий профіль, який буде містити інформацію, на які платформи випускається даний ігровий застосунок. Для цього потрібно натиснути кнопку Add справа знизу та з випадаючого списку обрати пункт Create Custom Profile (рис. 5.2). У відкритому меню далі потрібно обрати проект та обрати платформи, для яких треба підготувати файли ігрового застосунку. (рис 5.3). Після цього повертаємось в попереднє меню і запускаємо створений профіль (рис 5.4)

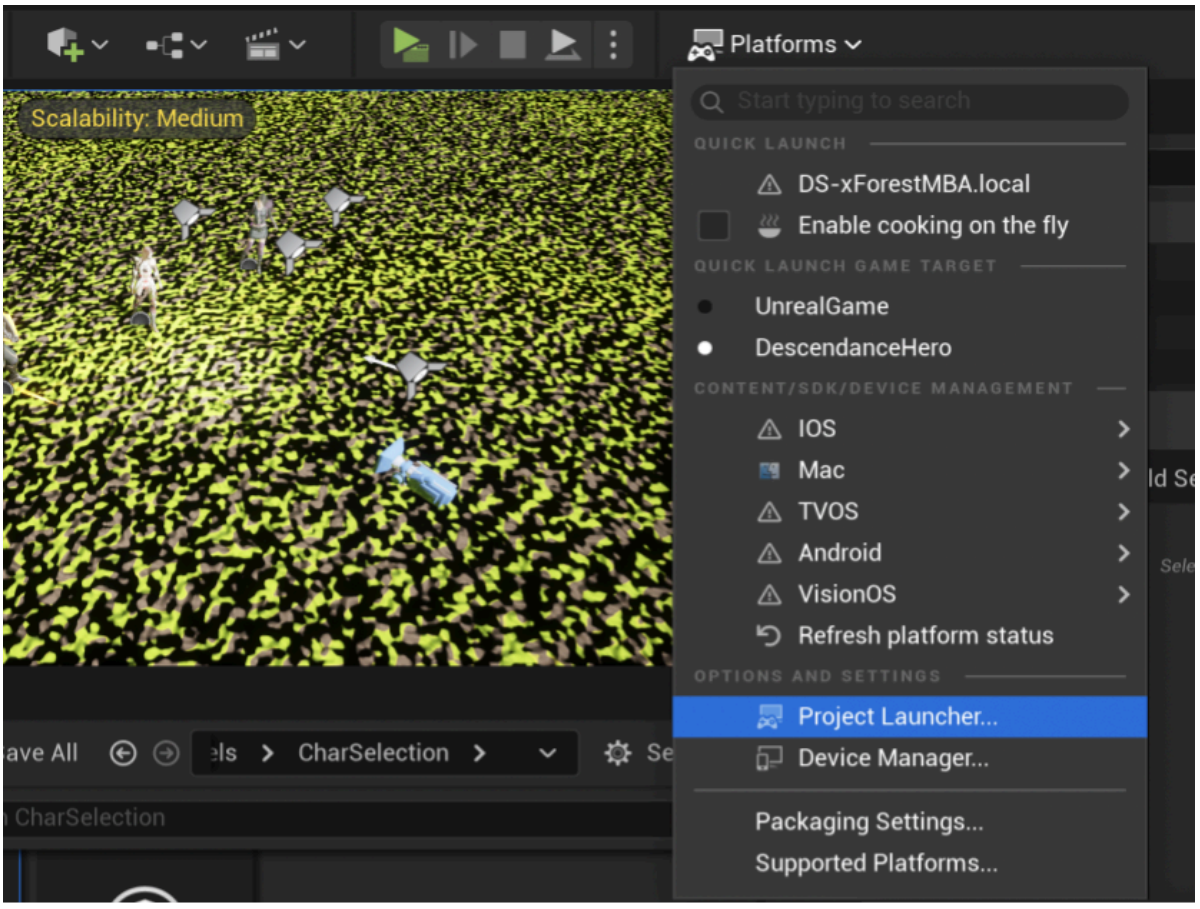


Рисунок 5.1 — Шлях до Project Launcher

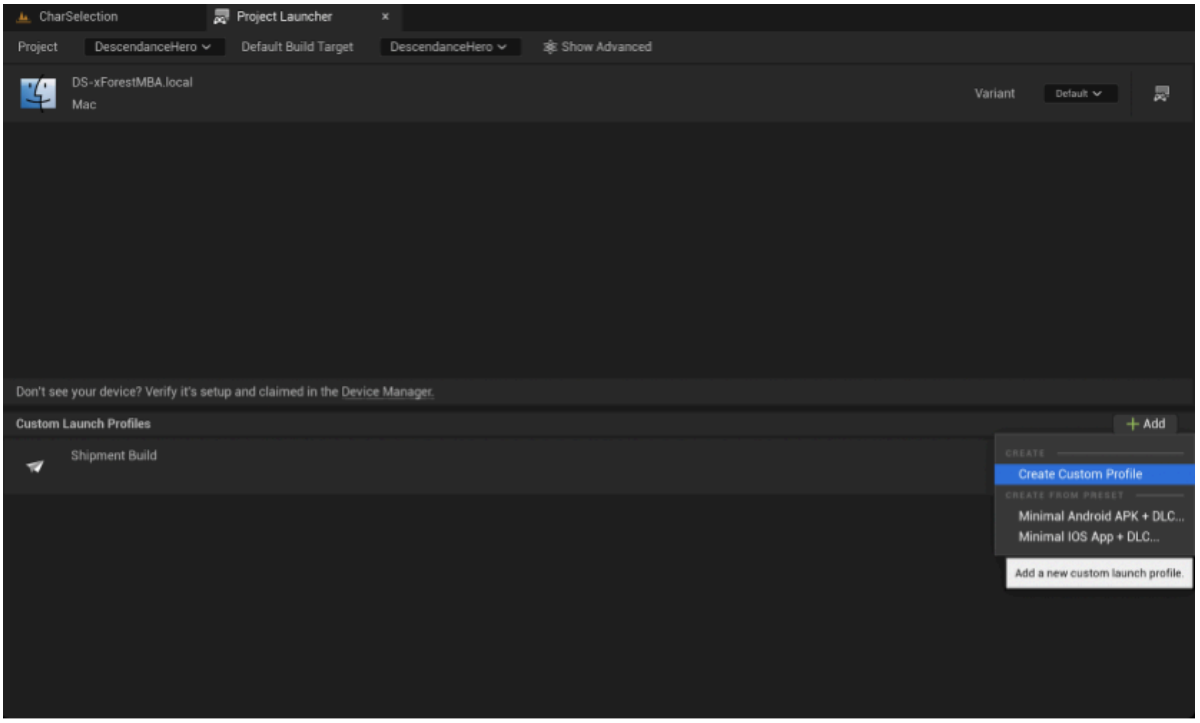


Рисунок 5.2 — Шлях до Create Custom Profile

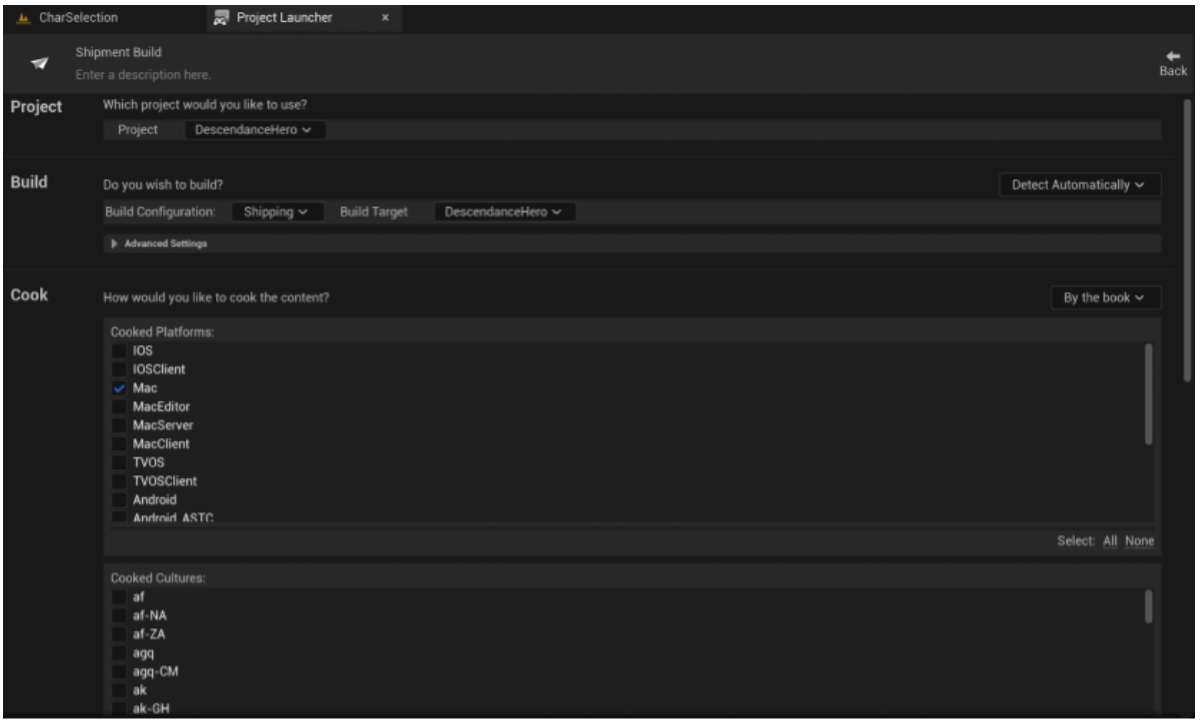


Рисунок 5.3 — Меню вибору параметрів для розгротання

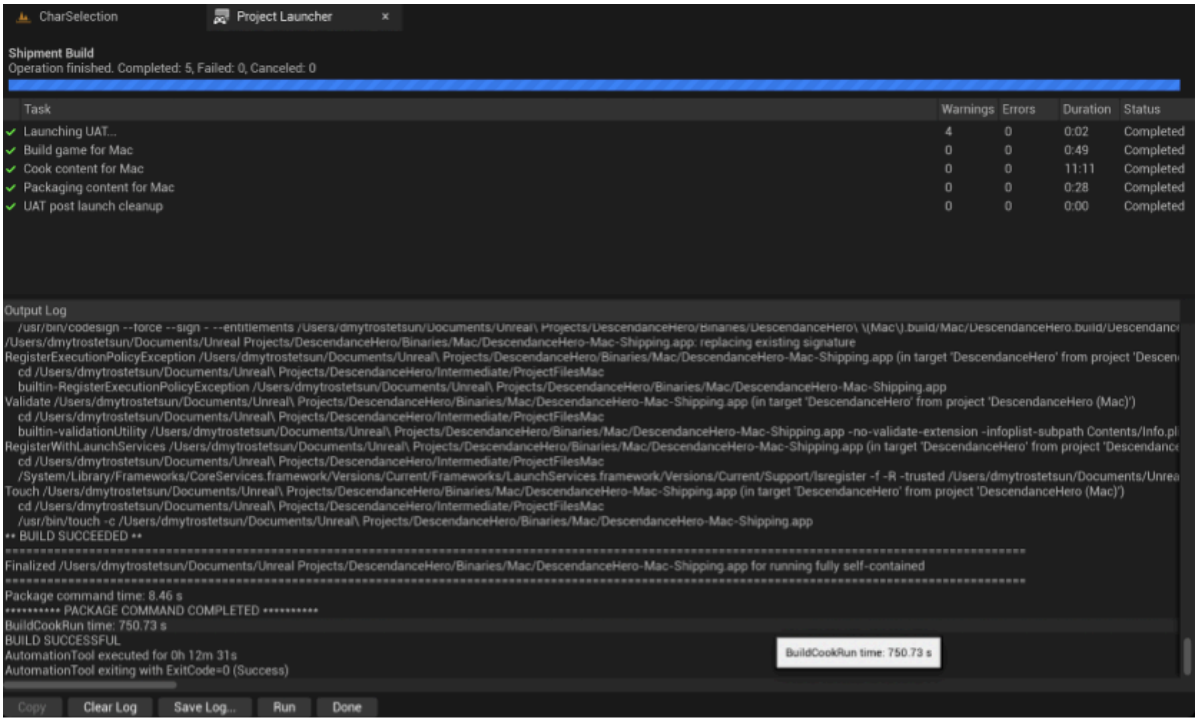


Рисунок 5.4 — Результат розгортання

5.2. Супровід програмного забезпечення

Керівництво користувача наведене в окремому документі.

Супровід розробленого ігрового застосунку відбувається за допомогою редактора Unreal Engine, оскільки базові класи для додавання нових персонажів, навичок чи елементів для GOAP, написані програмним кодом, містять весь необхідний функціонал. Нові елементи можуть бути легко створені за домогою візуального скриптингу всередині редактора Unreal Engine без використання текстових редакторів. При будь-яких внесених змінах достатньо запустити створений в розділі 5.1 профіль та завантажити створені файли до сервісів, де ігровий застосунок поширюється.

Висновок до розділу

У даному розділі було покроково описано процес розгортання розробленого програмного забезпечення. Оскільки розробка велась за допомогою редактора Unreal Engine, то розгортання було вирішено зробити також в ньому. Було вказано як створити профіль для розгортання застосунку та які налаштування потрібно обрати.

Також було описано, як оновити застосунок та провести розгортання знову. Алгоритм дій аналогічний до першого розгортання.

ВИСНОВКИ

У результаті виконання дипломного проекту було спроектовано та реалізовано однокористувацький ігровий застосунок у жанрі Action Role Play, було виконано усі поставлені задачі:

– був розроблений інтерфейс користувача, який охоплює ключові елементи взаємодії гравця з грою, а саме: меню початку гри з можливістю запуску або виходу; меню вибору персонажа, де користувач може переглянути характеристики доступних класів і обрати відповідного героя; інтерфейс взаємодії з продавцем, який дозволяє купувати або продавати ігрові предмети; меню вибору навичок, що дає змогу вивчати й активувати нові вміння залежно від стилю гри; меню інвентаря, де відображається екіпірування, предмети та їхні властивості; а також екранний інтерфейс (HUD), який під час гри показує важливу інформацію — рівень здоров’я та ресурсу, накопичений досвід, поточний рівень та активні навички персонажа;

– було спроєктовано та розроблено основні системи для ігрових персонажів, серед яких: система атрибутів, що включає первинні та вторинні характеристики, що залежать від первинних; система інвентаря, яка забезпечує взаємодію з предметами, включаючи передачу модифікаторів атрибутам; а також система навичок, яка реалізує різні типи умінь персонажа;

– було реалізовано повноцінний магазин, в якому гравець може придбати нові предмети, як-от зброя, броня або зілля, а також продати непотрібні речі, отримавши за них ігрову валюту;

– було створено систему штучного інтелекту на основі підходу GOAP (Goal-Oriented Action Planning) для неігрових персонажів, яка передбачає планування дій залежно від цілей і навколишніх умов. Для забезпечення її роботи були реалізовані цілі (наприклад, «атакувати гравця», «вижити») та набір дій, які NPC можуть виконувати відповідно до поточної ситуації;

– було створено декілька класів персонажів, кожен з яких має відмінні основні атрибути та стартові навички, що дозволяє користувачам вибрати бажаний стиль гри.

В якості середовища розробки було обрано редактор Unreal Engine та IDE Rider. Оскільки обраним ігровим рушієм для розробки даного дипломного проекту є Unreal Engine, то використання його редактора є обов’язковим для редагування мап, тривімірних моделей персонажів та використання візуального скриптингу (Blueprints). Для написання програмного коду використовувалось середовище Rider, оскільки воно має чудові аналізатори коду та індексатор для об’ємного коду рушія. Це призводить до продуктивної роботи з кодом, що значно пришвидшує процес розробки.

Розроблений ігровий застосунок це повноцінна гра, яка реалізовує основні жанрові особливості Action Role Play та має шанси бути популярним продуктом для гравців. На даний момент розробка знаходиться в досить далекому від публікації стані, оскільки потрібно продумати значно більшу частину навичок та баланс між значенням атрибутів.

Щодо графічної складової, тривимірні моделі персонажів, що були використані, є безкоштовними та завантажені з офіційного порталу Epic Games. Вони містять моделі, анімації та текстури, однак решта графічних елементів були створені власноруч та є досить простими, однак задачу щодо демонстрації їх ролі в середовищі виконують цілком повністю.

Незважаючи на те, що розробка не є в стані публікації та потрібно провести згадані покращення, поточний результат є завершеним прототипом основного функціоналу. Так як це Action Role Play, то для того, щоб створити хороший фінальний продукт, до даної розробки необхідно додати функціонал, що дасть змогу внести елементи сюжету. Це призведе до кращого досвіду гри у зв’язку з розумінням світу та персонажів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Власний Game Engine: як і навіщо створювати ігровий рушій. DOU GameDev. URL: https://gamedev.dou.ua/blogs/how-and-why-to-create-a-game-engine/ (дата звернення: 08.05.2025).

2. Tech Breakdown: AI with Finite State Machines. Little Polygon. URL: https://littlepolygon.com/game-ai-with-finite-state-machines/ (дата звернення: 08.05.2025).

3. Designing an AI Decision Tree. Game Dev Keys. URL: https://gamedevkeys.com/designing-an-ai-decision-tree/(дата звернення: 08.05.2025).

4. Goal Oriented Action Planning. Medium. Vedant Chaudhari. URL: https://medium.com/@vedantchaudhari/goal-oriented-action-planning-34035ed40d0b (дата звернення: 08.05.2025)

5. Diablo IV. Blizzard Entertainment. URL: https://diablo4.blizzard.com/ (дата звернення: 11.05.2025)

6. Dark Souls. Bandai Namco Entertainment. URL: https://en.bandainamcoent.eu/dark-souls (дата звернення: 11.05.2025)

7. The Witcher Universe. CD PROJEKT S.A. URL: https://www.thewitcher.com/ (дата звернення: 11.05.2025)

8. Unity User Manual. Unity Technologies. URL: https://docs.unity3d.com/Manual/index.html (дата звернення: 11.05.2025).

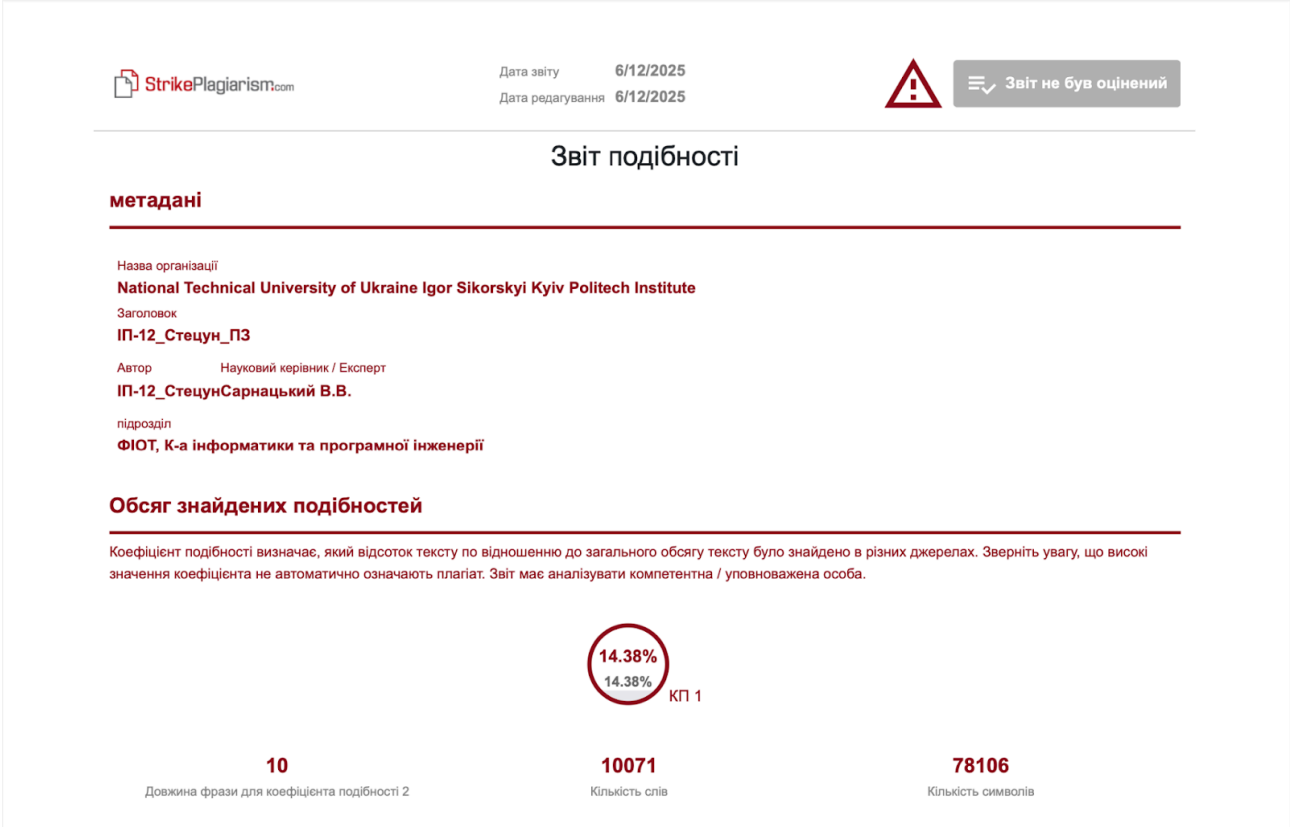
9. Unreal Engine Documentation. Epic Games. URL: https://docs.unrealengine.com/ (дата звернення: 11.05.2025).

10. Godot Docs – 4.4 branch. Godot Engine. URL: https://docs.godotengine.org/ (дата звернення: 11.05.2025)

11. JetBrains Rider. JetBrains. URL: https://www.jetbrains.com/rider/ (датазвернення: 11.05.2025)

ДОДАТКИ

ДОДАТОК А ЗВІТ ПОДІБНОСТІ



StrikePlagiarism.com

Дата звіту 6/12/2025
Дата редагування 6/12/2025

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute
Заголовок
ІП-12_Стецун_ПЗ
Автор Науковий керівник / Експерт
ІП-12_СтецунСарнацький В.В.
підрозділ
ФІОТ, К-а інформатики та програмної інженерії

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

14.38%
14.38%
КП 1

10	10071	78106
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ОДНОКОРИСТУВАЦЬКИЙ ІГРОВИЙ ЗАСТОСУНОК У ЖАНРІ ACTION RPG З ВИКОРИСТАННЯМ ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ НА РУШІЇ UNREAL ENGINE

Текст програми

КПІ.ІТ-0222.045480.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Владислав САРНАЦЬКИЙ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Дмитро СТЕЦУН

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду
https://github.com/DizzzM/DescendanceHero

Файл Ability.cpp
Реалізація функціональної вимоги Система навичок

```
// Descendance Hero by Dmytro Stetsun
#include "Ability.h"
#include "TimerManager.h"
void UAbility::Commit(const UWorld* World)
{
	UE_LOG(LogTemp, Warning, TEXT("Ability %s committed, on cooldown!"), *AbilityTag.ToString());
	bOnCooldown = true;
	World->GetTimerManager().SetTimer(
		CooldownTimerHandle,
		this,
		&UAbility::ResetCooldown,
		Cooldown,
		false
	);
}
void UAbility::ResetCooldown()
{
	UE_LOG(LogTemp, Warning, TEXT("Ability %s out off cooldown!"), *AbilityTag.ToString());
	bOnCooldown = false;
}
```

Файл Ability.h
Реалізація функціональної вимоги Система навичок

```
// Descendance Hero by Dmytro Stetsun
#pragma once
#include "CoreMinimal.h"
#include "AbilityInputData.h"
#include "GameplayTagContainer.h"
#include "Ability.generated.h"
UENUM(BlueprintType)
```

```
enum class EAbilityType : uint8 {
	Melee,
	Range,
	None  };
UCLASS(Blueprintable, BlueprintType)
class DESCENDANCEHERO_API UAbility : public UObject
{
	GENERATED_BODY()
public:
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Ability")
	FGameplayTag AbilityTag;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Ability")
	EAbilityType AbilityType = EAbilityType::None;

	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Ability")
	float Cost = 0.f;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Ability")
	bool bOnCooldown = true;

	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "Ability")
	float Cooldown = 5.f;

	UFUNCTION(BlueprintImplementableEvent)
	bool UseAbility(FAbilityInputData InputData);

	UFUNCTION(BlueprintCallable)
	void Commit(const UWorld* World);

protected:
	FTimerHandle CooldownTimerHandle;
	void ResetCooldown();
};
```

Файл AbilityComponent.cpp
Реалізація функціональної вимоги Система навичок

```
// Descendance Hero by Dmytro Stetsun
#include "AbilityComponent.h"
#include "BaseCharacter.h"
// Sets default values for this component's properties
UAbilityComponent::UAbilityComponent()
{
	PrimaryComponentTick.bCanEverTick = false;
}
// Called when the game starts
void UAbilityComponent::BeginPlay()
{
	Super::BeginPlay();
	const AActor* Owner = GetOwner();

	if
(Owner->GetClass()->ImplementsInterface(UIAttributeOwner::StaticClass()))
	{
		AttributeComponent =
IIAttributeOwner::Execute_GetAttributeComponent(Owner);
	}
}
// Called every frame
void UAbilityComponent::TickComponent(float DeltaTime, ELevelTick TickType,
FActorComponentTickFunction* ThisTickFunction)
{
	Super::TickComponent(DeltaTime, TickType, ThisTickFunction);
}
bool UAbilityComponent::GrantAbility(const TSubclassOf<UAbility>
AbilityClass)
{
	UAbility* NewAbility = NewObject<UAbility>(this, AbilityClass);
	AbilityMap.Add(NewAbility->AbilityTag, NewAbility);
	UE_LOG(LogTemp, Warning, TEXT("[AbilityComponent] <%s> Ability
granted: %s"), *GetOwner()->GetName(), *NewAbility->AbilityTag.ToString());
	return true;
}
```

```
bool UAbilityComponent::UseAbility(const FGameplayTag AbilityTag, FAbilityInputData InputData)
{
	if (!AttributeComponent)
	{
		UE_LOG(LogTemp, Warning, TEXT("[AbilityComponent] <%s> No AttributeComponent found on Owner!"), *GetOwner()->GetName());
		return false;
	}
	if (!AbilityMap.Contains(AbilityTag))
	{
		UE_LOG(LogTemp, Warning, TEXT("[AbilityComponent] <%s> Ability %s not found in AbilityMap!"), *GetOwner()->GetName(), *AbilityTag.ToString());
		return false;
	}
	UE_LOG(LogTemp, Warning, TEXT("[AbilityComponent] <%s> Ability %s available"), *GetOwner()->GetName(), *AbilityTag.ToString());
	if (AbilityMap[AbilityTag]->bOnCooldown){
		UE_LOG(LogTemp, Warning, TEXT("[AbilityComponent] <%s> Ability %s in cooldown!"), *GetOwner()->GetName(), *AbilityTag.ToString());
		return false;
	}
	if (!AttributeComponent->HasEnoughResource(AbilityMap[AbilityTag]->Cost)){
		UE_LOG(LogTemp, Warning, TEXT("[AbilityComponent] <%s> Not enough resource to use ability %s!"), *GetOwner()->GetName(), *AbilityTag.ToString());
		return false;
	}
	bool bUsed = AbilityMap[AbilityTag]->UseAbility(InputData);
	if (bUsed)
	{
		UE_LOG(LogTemp, Warning, TEXT("[AbilityComponent] <%s> Ability %s successfully activated!"), *GetOwner()->GetName(), *AbilityTag.ToString());

		AttributeComponent->UseResource(AbilityMap[AbilityTag]->Cost);
		AbilityMap[AbilityTag]->Commit(GetWorld());
```

```
	}
	return bUsed;

}
void UAbilityComponent::RevokeAbility(const FGameplayTag AbilityTag)
{
	AbilityMap.Remove(AbilityTag);
	UE_LOG(LogTemp, Warning, TEXT("[AbilityComponent] <%s> Ability %s revoked!"), *GetOwner()->GetName(), *AbilityTag.ToString());
}
TMap<FGameplayTag, UAbility*> UAbilityComponent::GetAbilities() const
{
	return AbilityMap;
}
```

Файл AbilityComponent.h

Реалізація функціональної вимоги Система навичок

```
// Descendance Hero by Dmytro Stetsun
#pragma once
#include "CoreMinimal.h"
#include "Ability.h"
#include "AttributeComponent.h"
#include "GameplayTagContainer.h"
#include "Components/ActorComponent.h"
#include "AbilityComponent.generated.h"
UCLASS( ClassGroup=(Custom), meta=(BlueprintSpawnableComponent) )
class DESCENDANCEHERO_API UAbilityComponent : public UActorComponent
{
	GENERATED_BODY()
public:
	// Sets default values for this component's properties
	UAbilityComponent();
protected:
	// Called when the game starts
	virtual void BeginPlay() override;
public:
```

```
	// Called every frame
	virtual void TickComponent(float DeltaTime, ELevelTick TickType, FActorComponentTickFunction* ThisTickFunction) override;

	UFUNCTION(BlueprintCallable)
	bool GrantAbility(TSubclassOf<UAbility> Ability);
	UFUNCTION(BlueprintCallable)
	bool UseAbility(FGameplayTag AbilityTag, FAbilityInputData InputData = FAbilityInputData() );
	UFUNCTION(BlueprintCallable)
	void RevokeAbility(FGameplayTag Ability);
	UFUNCTION(BlueprintCallable)
	TMap<FGameplayTag, UAbility*> GetAbilities() const;
protected:
	UPROPERTY(VisibleAnywhere, Instanced, BlueprintReadOnly, Category = "Abilities")
	TMap<FGameplayTag, UAbility*> AbilityMap;
	UPROPERTY(BlueprintReadOnly, Category = "Abilities|Attributes")
	UAttributeComponent* AttributeComponent;

};
```

Файл AbilityInputData.h

Реалізація функціональної вимоги Система навичок

```
#pragma once
#include "CoreMinimal.h"
#include "AbilityInputData.generated.h"
USTRUCT(BlueprintType)
struct DESCENDANCEHERO_API FAbilityInputData
{
	GENERATED_BODY()
	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	AActor* Instigator = nullptr;
	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	AActor* TargetActor = nullptr;
};
```

Файл AttributeComponent.cpp

Реалізація функціональних вимог: Система атрибутів, Можливість померти, Можливість вбити ворога

```
// Descendance Hero by Dmytro Stetsun
#include "AttributeComponent.h"
#include "BaseCharacter.h"
#include "IAttributeOwner.h"
#include "GameFramework/CharacterMovementComponent.h"
UAttributeComponent::UAttributeComponent()
{
	PrimaryComponentTick.bCanEverTick = true;
}
void UAttributeComponent::BeginPlay()
{
	Super::BeginPlay();
	LevelUp();
	Health = MaxHealth;
	Resource = MaxResource;
	OnHealthChanged.Broadcast(Health, MaxHealth);
	OnResourceChanged.Broadcast(Resource, MaxResource);
	OnLvlUp.Broadcast(Level);
	OnExpChanged.Broadcast(Experience, 10 +
ExpToLvlUpCurve->GetFloatValue(Level));
}
void UAttributeComponent::TickComponent(float DeltaTime, ELevelTick
TickType, FActorComponentTickFunction* ThisTickFunction)
{
	Super::TickComponent(DeltaTime, TickType, ThisTickFunction);
	PassiveRegenHealth(DeltaTime);
	PassiveRegenResource(DeltaTime);
}
float UAttributeComponent::TakeDamage(const FDamageInfo DamageInfo)
{
	const float Amount = DamageInfo.PhysicalDamage * PhysicalResistance +
DamageInfo.MagicDamage * MagicResistance;
	UE_LOG(LogTemp, Warning, TEXT("Damage taken: %f"), Amount);
	Health = Health - Amount;
	OnHealthChanged.Broadcast(Health, MaxHealth);
```

```
	if (Health <= 0)
	{
		bAlive = false;
		PrimaryComponentTick.bCanEverTick = false;
		AActor* Owner = GetOwner();
		if (Owner &&
Owner->GetClass()->ImplementsInterface(UIAttributeOwner::StaticClass()))
		{
			IIAttributeOwner::Execute_OnCharacterDied(Owner);
		}
		return GetExpOnKill();
	}
	return 0;
}
void UAttributeComponent::RegenHealth(const float Amount)
{
	Health = FMath::Clamp(Health + Amount, 0, MaxHealth);
	OnHealthChanged.Broadcast(Health, MaxHealth);
}
void UAttributeComponent::PassiveRegenHealth(const float DeltaTime)
{
	if (bAlive && HealthRegen > 0 && Health < MaxHealth){
		const float RegenAmount = HealthRegen * DeltaTime;
		RegenHealth(RegenAmount);
	}
	OnHealthChanged.Broadcast(Health, MaxHealth);
}
void UAttributeComponent::RegenResource(const float Amount)
{
	Resource = FMath::Clamp(Resource + Amount, 0, MaxResource);
	OnResourceChanged.Broadcast(Resource, MaxResource);
}
void UAttributeComponent::PassiveRegenResource(const float DeltaTime)
{
	if (bAlive && ResourceRegen > 0 && Resource < MaxResource){
		const float RegenAmount = HealthRegen * DeltaTime;
		RegenResource(RegenAmount);
	}
	OnResourceChanged.Broadcast(Resource, MaxResource);
```

```
}
bool UAttributeComponent::HasEnoughResource(const float Amount) const
{
	return Resource >= Amount;
}
void UAttributeComponent::UseResource(const float Amount)
{
	Resource = FMath::Clamp(Resource - Amount, 0, MaxResource);
	OnResourceChanged.Broadcast(Resource, MaxResource);
}
void UAttributeComponent::GainExperience(const float Amount)
{
	Experience += Amount;
	if (Experience > ExperienceToLvlUp)
	{
		Experience -= ExperienceToLvlUp;
		Level++;
		OnLvlUp.Broadcast(Level);
		LevelUp();
	}
	OnExpChanged.Broadcast(Experience, ExperienceToLvlUp);
}
void UAttributeComponent::LevelUp()
{
	ExperienceToLvlUp = ExpToLvlUpCurve->GetFloatValue(Level);
	switch (MainAttribute)
	{
		case EMainAttribute::Strength:
			Strength = MainAttributePerLevel * Level;
			Dexterity = SecondaryAttributePerLevel * Level;
			Intelligence = SecondaryAttributePerLevel * Level;
			break;
		case EMainAttribute::Dexterity:
			Strength = SecondaryAttributePerLevel * Level;
			Dexterity = MainAttributePerLevel * Level;
			Intelligence = SecondaryAttributePerLevel * Level;
			break;
		case EMainAttribute::Intelligence:
			Strength = SecondaryAttributePerLevel * Level;
```

```
                Dexterity = SecondaryAttributePerLevel * Level;
                Intelligence = MainAttributePerLevel * Level;
                break;
        }
        RecalculateStats();
}
void UAttributeComponent::AddItemModifier(FAttributeModifier* Item)
{
        Modifiers.Add(Item);
        RecalculateStats();
}
void UAttributeComponent::RemoveItemModifier(FAttributeModifier* Item)
{
        Modifiers.Remove(Item);
        RecalculateStats();
}
float UAttributeComponent::GetExpOnKill() const
{
        return Level;
}
void UAttributeComponent::RecalculateStats()
{
        float TotalStrength = Strength;
        float TotalDexterity = Dexterity;
        float TotalIntelligence = Intelligence;
        float ModifierDamage = 0.f;

        float ModifierMaxHealth = 0.f;
        float ModifierHealthRegen = 0.f;
        float ModifierMagicResistance = 0.f;
        float ModifierPhysicalResistance = 0.f;
        float ModifierAccuracy = 0.f;
        float ModifierAttackSpeed = 0.f;
        float ModifierMaxResource = 0.f;
        float ModifierResourceRegen = 0.f;
        float ModifierMagicAmplification = 0.f;
        float ModifierMovementSpeed = 0.f;
        if (Modifiers.Num() != 0)
        {
```

```
		for (const auto Modifier : Modifiers)
		{
			TotalStrength += Modifier->Strength;
			TotalDexterity += Modifier->Dexterity;
			TotalIntelligence += Modifier->Intelligence;
			ModifierDamage += Modifier->Damage;
			ModifierMaxHealth += Modifier->MaxHealth;
			ModifierHealthRegen += Modifier->HealthRegen;
			ModifierMagicResistance += Modifier->MagicResistance;
			ModifierPhysicalResistance += Modifier->PhysicalResistance;
			ModifierAccuracy += Modifier->Accuracy;
			ModifierAttackSpeed += Modifier->AttackSpeed;
			ModifierMaxResource += Modifier->MaxResource;
			ModifierResourceRegen += Modifier->ResourceRegen;
			ModifierMagicAmplification +=
Modifier->MagicAmplification;
			ModifierMovementSpeed += Modifier->MovementSpeed;
		}
	}

	switch (MainAttribute)
	{
		case EMainAttribute::Strength:
			Damage = TotalStrength + ModifierDamage;
			break;
		case EMainAttribute::Dexterity:
			Damage = TotalDexterity + ModifierDamage;
			break;
		case EMainAttribute::Intelligence:
			Damage = TotalIntelligence + ModifierDamage;
			break;
	}
	MaxHealth = TotalStrength * MaxHealthPerStrength + ModifierMaxHealth;
	HealthRegen = TotalStrength * HealthRegenPerStrength +
ModifierHealthRegen;
	MagicResistance = TotalStrength * MagicResistancePerStrength +
ModifierMagicResistance;
```

```
	PhysicalResistance = TotalDexterity * PhysicalResistancePerDexterity +
ModifierPhysicalResistance;
	Accuracy = BaseAccuracy + TotalDexterity * AccuracyPerDexterity +
ModifierAccuracy;
	AttackSpeed = TotalDexterity * AccuracyPerDexterity +
ModifierAttackSpeed;
	MaxResource = TotalIntelligence * ResourcePerIntelligence +
ModifierMaxResource;
	ResourceRegen = TotalIntelligence * ResourceRegenPerIntelligence +
ModifierResourceRegen;
	MagicAmplification = TotalIntelligence *
MagicAmplificationPerIntelligence + ModifierMagicAmplification;
	MovementSpeed = BaseMovementSpeed +
MovementSpeedCurve->GetFloatValue(Dexterity) + ModifierMovementSpeed;
	AActor* Owner = GetOwner();


Cast<ABaseCharacter>(Owner)->GetCharacterMovement()->MaxWalkSpeed =
MovementSpeed;
}
```

Файл AttributeComponent.h

Реалізація функціональних вимог: Система атрибутів, Можливість померти, Можливість вбити ворога

```
// Descendance Hero by Dmytro Stetsun
#pragma once
#include "CoreMinimal.h"
#include "AttributeModifier.h"
#include "Curves/CurveFloat.h"
#include "Components/ActorComponent.h"
#include "AttributeComponent.generated.h"
USTRUCT(BlueprintType)
struct FDamageInfo
{
	GENERATED_BODY()
	UPROPERTY(BlueprintReadWrite, EditAnywhere)
	float PhysicalDamage = 0.f;
	UPROPERTY(BlueprintReadWrite, EditAnywhere)
```

```
        float MagicDamage = 0.f;
        FDamageInfo() {}
        FDamageInfo(float InPhysical, float InMagic)
                : PhysicalDamage(InPhysical), MagicDamage(InMagic)
        {}
};
UENUM(BlueprintType)
enum class EMainAttribute : uint8
{
        Strength UMETA(DisplayName = "Strength"),
        Dexterity UMETA(DisplayName = "Dexterity"),
        Intelligence UMETA(DisplayName = "Intelligence")
};
UCLASS( ClassGroup=(Custom), meta=(BlueprintSpawnableComponent) )
class DESCENDANCEHERO_API UAttributeComponent : public
UActorComponent
{
        GENERATED_BODY()
public:
        // Sets default values for this component's properties
        UAttributeComponent();

DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(FOnHealthCha
nged, float, NewHealth, float, MaxHealth);
        UPROPERTY(BlueprintAssignable, Category = "Attributes")
        FOnHealthChanged OnHealthChanged;

DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(FOnResourceC
hanged, float, NewResource, float, MaxResource);
        UPROPERTY(BlueprintAssignable, Category = "Attributes")
        FOnResourceChanged OnResourceChanged;

DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FOnLevelUp,
int, NewLevel);

DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(FOnExpChang
ed, float, NewExp, float, MaxExp);
        UPROPERTY(BlueprintAssignable, Category = "Attributes")
        FOnExpChanged OnExpChanged;
```

```
DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FOnLvlUp,
float, NewLvl);
	UPROPERTY(BlueprintAssignable, Category = "Attributes")
	FOnLvlUp OnLvlUp;

	UPROPERTY(BlueprintReadOnly, Category="Attribute
Component|BaseAttribute")
	bool bAlive = true;

	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|BaseAttribute")
	EMainAttribute MainAttribute = EMainAttribute::Strength;

	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Level")
	int Level = 1;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Level")
	float Experience = 0.f;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Level")
	float ExperienceToLvlUp = 100.f;

	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|BaseAttribute")
	float Strength = 5.f ;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|BaseAttribute")
	float Dexterity = 5.f;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|BaseAttribute")
	float Intelligence = 5.f;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Strength")
	float Health = 100.f;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Strength")
	float MaxHealth = 100.f;
```

```
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute Component|Strength")
	float HealthRegen = 1.f;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute Component|Strength")
	float MagicResistance = 0.f;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute Component|Dexterity")
	float PhysicalResistance = 0.f; // HP points that will be blocked
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute Component|Dexterity")
	float AttackSpeed = 0.5; // Attacks Per Second
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute Component|Dexterity")
	float BaseAccuracy = 0.5; // Percentage of Attacks that will land a hit by default
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute Component|Dexterity")
	float Accuracy = BaseAccuracy; // Percentage of Attacks that will land a hit

	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute Component|Intelligence")
	float Resource = 100.f;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute Component|Intelligence")
	float MaxResource = 100.f;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute Component|Intelligence")
	float ResourceRegen = 1.f;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute Component|Intelligence")
	float MagicAmplification = 0.f;

	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute Component|MainAttribute")
	float Damage = 10.f;

	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute Component|Utility")
```

```
        float Range = 300.f;
        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Utility")
        float BaseMovementSpeed = 300.f;
        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Utility")
        float MovementSpeed = BaseMovementSpeed;
        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Scaling|BaseAttributes")
        float MainAttributePerLevel = 10.f;
        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Scaling|BaseAttributes")
        float SecondaryAttributePerLevel = 5.f;

        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Scaling|Strength")
        float MaxHealthPerStrength = 22.f;
        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Scaling|Strength")
        float HealthRegenPerStrength = 0.09f;
        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Scaling|Strength")
        float MagicResistancePerStrength = 0.1f;

        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Scaling|Dexterity")
        float PhysicalResistancePerDexterity = 1.f/3.f;
        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Scaling|Dexterity")
        float AccuracyPerDexterity = 0.1f;
        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Scaling|Dexterity")
        float AttackSpeedPerDexterity = 0.04f;
        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Scaling|Intelligence")
        float ResourcePerIntelligence = 12.f;
        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Scaling|Intelligence")
        float ResourceRegenPerIntelligence = 0.05f;
```

```
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category="Attribute
Component|Scaling|Intelligence")
	float MagicAmplificationPerIntelligence = 0.1f;

	UPROPERTY(EditDefaultsOnly, Category="Attribute
Component|Scaling|Curves")
	UCurveFloat* ExpToLvlUpCurve;

	UPROPERTY(EditDefaultsOnly, Category="Attribute
Component|Scaling|Curves")
	UCurveFloat* MovementSpeedCurve;

	TArray<FAttributeModifier*> Modifiers = {};
protected:
	virtual void BeginPlay() override;
public:
	virtual void TickComponent(float DeltaTime, ELevelTick TickType,
FActorComponentTickFunction* ThisTickFunction) override;
	UFUNCTION(BlueprintCallable)
	float TakeDamage(const FDamageInfo Amount);

	UFUNCTION(BlueprintCallable)
	void RegenHealth(const float Amount);
	UFUNCTION(BlueprintCallable)
	void PassiveRegenHealth(const float DeltaTime);
	UFUNCTION(BlueprintCallable)
	void RegenResource(const float Amount);
	UFUNCTION(BlueprintCallable)
	void PassiveRegenResource(const float DeltaTime);
	UFUNCTION(BlueprintCallable)
	bool HasEnoughResource(const float Amount) const;
	UFUNCTION(BlueprintCallable)
	void UseResource(const float Amount);

	UFUNCTION(BlueprintCallable)
	void GainExperience(const float Amount);
	UFUNCTION(BlueprintCallable)
	void LevelUp();
	void AddItemModifier(FAttributeModifier* Item);
```

```
        void RemoveItemModifier(FAttributeModifier* Item);
        float GetExpOnKill() const;

        UFUNCTION(BlueprintCallable)
        void RecalculateStats();
};
```

Файл AttributeModifier.h

Реалізація функціональних вимог: Система атрибутів, Можливість померти, Можливість вбити ворога

```
#pragma once
#include "CoreMinimal.h"
#include "AttributeModifier.generated.h"
USTRUCT(BlueprintType)
struct DESCENDANCEHERO_API FAttributeModifier :public FTableRowBase
{
        GENERATED_BODY()

        UPROPERTY(EditAnywhere, BlueprintReadWrite)
        float Strength = 0.f;
        UPROPERTY(EditAnywhere, BlueprintReadWrite)
        float Dexterity = 0.f;
        UPROPERTY(EditAnywhere, BlueprintReadWrite)
        float Intelligence = 0.f;
        UPROPERTY(EditAnywhere, BlueprintReadWrite)
        float Damage = 0.f;
        UPROPERTY(EditAnywhere, BlueprintReadWrite)
        float MaxHealth = 0.f;
        UPROPERTY(EditAnywhere, BlueprintReadWrite)
        float HealthRegen = 0.f;
        UPROPERTY(EditAnywhere, BlueprintReadWrite)
        float MagicResistance = 0.f;
        UPROPERTY(EditAnywhere, BlueprintReadWrite)
        float PhysicalResistance = 0.f;
        UPROPERTY(EditAnywhere, BlueprintReadWrite)
        float Accuracy = 0.f;
        UPROPERTY(EditAnywhere, BlueprintReadWrite)
```

```
	float AttackSpeed = 0.f;
	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	float MaxResource = 0.f;
	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	float ResourceRegen = 0.f;
	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	float MagicAmplification = 0.f;
	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	float MovementSpeed = 0.f;
	bool operator==(const FAttributeModifier& Other) const
	{
		return Strength == Other.Strength &&
			Dexterity == Other.Dexterity &&
			Intelligence == Other.Intelligence &&
			Damage == Other.Damage &&
			MaxHealth == Other.MaxHealth &&
			HealthRegen == Other.MaxHealth &&
			MagicResistance == Other.MagicResistance &&
			PhysicalResistance == Other.PhysicalResistance &&
			Accuracy == Other.Accuracy &&
			AttackSpeed == Other.Accuracy &&
			MaxResource == Other.MaxResource &&
			ResourceRegen == Other.ResourceRegen &&
			MagicAmplification == Other.MagicAmplification &&
			MovementSpeed == Other.MovementSpeed;
	}
};
```

Файл BaseCharacter.cpp

Реалізація функціональних вимог: Система атрибутів, Система інвентара, Система навичок

```
// Descendance Hero by Dmytro Stetsun
#include "BaseCharacter.h"
// Sets default values
ABaseCharacter::ABaseCharacter()
{
	PrimaryActorTick.bCanEverTick = false;
```

```
	AttributeComponent =
CreateDefaultSubobject<UAttributeComponent>(TEXT("AttributeComponent"));
	AbilityComponent =
CreateDefaultSubobject<UAbilityComponent>(TEXT("AbilityComponent"));
	InventoryComponent =
CreateDefaultSubobject<UInventoryComponent>(TEXT("InventoryComponent"))
;
}
// Called when the game starts or when spawned
void ABaseCharacter::BeginPlay()
{
	Super::BeginPlay();

}
// Called every frame
void ABaseCharacter::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);
}
// Called to bind functionality to input
void ABaseCharacter::SetupPlayerInputComponent(UInputComponent*
PlayerInputComponent)
{
	Super::SetupPlayerInputComponent(PlayerInputComponent);
}
UAttributeComponent*
ABaseCharacter::GetAttributeComponent_Implementation() const
{
	return AttributeComponent;
}
UAbilityComponent* ABaseCharacter::GetAbilityComponent_Implementation()
const
{
	return AbilityComponent;
}
bool ABaseCharacter::IsAlive_Implementation() const
{
	return AttributeComponent->bAlive;
}
```

```
void ABaseCharacter::OnCharacterDied_Implementation()
{
	GetMesh()->SetSimulatePhysics(true);
	GetMesh()->SetCollisionProfileName(FName("Ragdoll"));
	if (AController* Controller = GetController())
	{
		Controller->UnPossess();
	}
}
float ABaseCharacter::TakeDamage_Implementation(const FDamageInfo
DamageInfo) const
{
	return AttributeComponent->TakeDamage(DamageInfo);
}
void ABaseCharacter::Heal_Implementation(const float Amount) const
{
	AttributeComponent->RegenHealth(Amount);
}
```

Файл BaseCharacter.h

Реалізація функціональних вимог: Система атрибутів, Система інвентара, Система навичок

```
// Descendance Hero by Dmytro Stetsun
#pragma once
#include "CoreMinimal.h"
#include "AbilityComponent.h"
#include "AttributeComponent.h"
#include "InventoryComponent.h"
#include "IAttributeOwner.h"
#include "GameFramework/Character.h"
#include "BaseCharacter.generated.h"
UCLASS()
class DESCENDANCEHERO_API ABaseCharacter : public ACharacter, public
IIAttributeOwner
{
	GENERATED_BODY()
public:
	ABaseCharacter();
```

```
protected:
	virtual void BeginPlay() override;
	UPROPERTY(VisibleAnywhere, BlueprintReadWrite,
Category="Systems|Attributes")
	UAttributeComponent* AttributeComponent;

	UPROPERTY(VisibleAnywhere, BlueprintReadWrite,
Category="Systems|Abilities")
	UAbilityComponent* AbilityComponent;
	UPROPERTY(VisibleAnywhere, BlueprintReadWrite,
Category="Systems|Abilities")
	UInventoryComponent* InventoryComponent;

public:
	virtual void Tick(float DeltaTime) override;
	virtual void SetupPlayerInputComponent(class UInputComponent*
PlayerInputComponent) override;

	UFUNCTION(BlueprintCallable)
	virtual UAttributeComponent* GetAttributeComponent_Implementation()
const override;

	UFUNCTION(BlueprintCallable)
	virtual UAbilityComponent* GetAbilityComponent_Implementation() const
override;

	UFUNCTION(BlueprintCallable)
	virtual bool IsAlive_Implementation() const override;

	UFUNCTION(BlueprintCallable)
	virtual void OnCharacterDied_Implementation() override;

	UFUNCTION(BlueprintCallable)
	virtual float TakeDamage_Implementation(const FDamageInfo
DamageInfo) const override;
	UFUNCTION(BlueprintCallable)
	virtual void Heal_Implementation(const float Amount) const override;
};
```

Файл GOAPAct.cpp

Реалізація функціональних вимог: Робота штучного інтелекту

```
// Descendance Hero by Dmytro Stetsun
#include "GOAPAct.h"
#include "BehaviorTree/BlackboardComponent.h"
#include "BaseCharacter.h"
#include "GOAPAIController.h"
EBTNodeResult::Type UGOAPAct::ExecuteTask(UBehaviorTreeComponent&
OwnerComp, uint8* NodeMemory)
{
	AGOAPAIController* AIController =
Cast<AGOAPAIController>(OwnerComp.GetAIOwner());
	if (!AIController)
	{
		return EBTNodeResult::Failed;
	}
	UBlackboardComponent* BlackboardComponent =
OwnerComp.GetBlackboardComponent();
	if (AIController->CurrentPlan.Num() == 0) return EBTNodeResult::Failed;
	UGOAPAction* Action =
AIController->CurrentPlan[AIController->CurrentPlanIndex];
	if (!Action) return EBTNodeResult::Failed;
	ABaseCharacter* SelfActor =
Cast<ABaseCharacter>(AIController->GetPawn());
	ABaseCharacter* TargetActor =
Cast<ABaseCharacter>(BlackboardComponent->GetValueAsObject(TargetActorK
ey.SelectedKeyName));
	if (!SelfActor || !TargetActor)
	{

BlackboardComponent->SetValueAsBool(bReplanKey.SelectedKeyName, true);
		return EBTNodeResult::Failed;
	}
	if
(IIAttributeOwner::Execute_GetAbilityComponent(SelfActor)->UseAbility(Actio
n->AbilityTag, FAbilityInputData(SelfActor, TargetActor)))
	{
```

```
            AIController->CurrentPlanIndex++;
            return EBTNodeResult::Succeeded;
        }
        return EBTNodeResult::Failed;
}
```

Файл GOAPAct.h

Реалізація функціональних вимог: Робота штучного інтелекту

```
// Descendance Hero by Dmytro Stetsun
#pragma once
#include "CoreMinimal.h"
#include "BehaviorTree/BTTaskNode.h"
#include "GOAPAct.generated.h"
UCLASS()
class DESCENDANCEHERO_API UGOAPAct : public UBTTaskNode
{
        GENERATED_BODY()


public:
        UPROPERTY(EditAnywhere, Category = "Blackboard")
        FBlackboardKeySelector TargetActorKey;
        UPROPERTY(EditAnywhere, Category = "Blackboard")
        FBlackboardKeySelector bReplanKey;

        virtual EBTNodeResult::Type ExecuteTask(UBehaviorTreeComponent&
OwnerComp, uint8* NodeMemory) override;
};
```

Файл GOAPAction.cpp

Реалізація функціональних вимог: Робота штучного інтелекту

```
#include "GOAPAction.h"
int32 UGOAPAction::GetPostWorldState(int32& WorldState)
{
   int32 NewWorldState = WorldState | SetResultWorldState;
        return NewWorldState & (~ResetResultWorldState);
```

```
}
bool UGOAPAction::CheckPreconditions(const int32 WorldState)
{
	bool SetConditionsMet = (WorldState & SetPreconditionWorldState) ==
SetPreconditionWorldState;
	// Check all required RESET bits are absent
	bool ResetConditionsMet = (~WorldState & ResetPreconditionWorldState)
== ResetPreconditionWorldState;
	return SetConditionsMet && ResetConditionsMet;
}
```

Файл GOAPAction.h

Реалізація функціональних вимог: Робота штучного інтелекту

```
#pragma once
#include "CoreMinimal.h"
#include "AbilityInputData.h"
#include "GameplayTagContainer.h"
#include "GOAPAction.generated.h"
UCLASS(Blueprintable, BlueprintType)
class DESCENDANCEHERO_API UGOAPAction : public UObject
{
	GENERATED_BODY()
public:
	// The unique name or tag identifying this goal
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GOAP
Action")
	FName ActionName;
	// Priority for resolving conflicting goals (higher number = higher priority)
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GOAP
Action")
	int32 Cost;
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GOAP
Action")
	FGameplayTag AbilityTag;
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GOAP
Action")
	FAbilityInputData InputData;
	// Precondition bitmask representing the required world state
```

```
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GOAP
Action")
	int32 SetPreconditionWorldState;
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GOAP
Action")
	int32 ResetPreconditionWorldState;
	// Desired bitmask world state
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GOAP
Action")
	int32 SetResultWorldState;
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GOAP
Action")
	int32 ResetResultWorldState;
	// Custom function to compare two goals
	UFUNCTION(BlueprintCallable, Category = "GOAP Action")
	bool IsEqualTo(UGOAPAction* OtherAction) const
	{
		if (!OtherAction)
		{
			return false;
		}
		return SetPreconditionWorldState ==
OtherAction->SetPreconditionWorldState &&
			ResetPreconditionWorldState ==
OtherAction->ResetPreconditionWorldState &&
			SetResultWorldState == OtherAction->SetResultWorldState
&&
			ResetResultWorldState ==
OtherAction->ResetResultWorldState;
	}

	UFUNCTION(BlueprintCallable, Category = "GOAP Action")
	int32 GetPostWorldState(int32& WorldState);
	UFUNCTION(BlueprintCallable, Category = "GOAP Action")
	bool CheckPreconditions(int32 WorldState);
};
```

Файл GOAPAIController.cpp

Реалізація функціональних вимог: Робота штучного інтелекту

```
#include "GOAPAIController.h"
#include "BehaviorTree/BlackboardComponent.h"
#include "Perception/AIPerceptionComponent.h"
#include "Perception/AISenseConfig_Sight.h"
void AGOAPAIController::OnPossess(APawn* InPawn)
{
	Super::OnPossess(InPawn);
	if (BlackboardAsset)
	{
		if (UseBlackboard(BlackboardAsset, BlackboardComponent))
		{
			if (BehaviorTreeAsset)
			{
				RunBehaviorTree(BehaviorTreeAsset);
			}
		}
	}
	if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Controller]
<%s> AI Controller possessed"), *GetPawn()->GetName());
}
void AGOAPAIController::OnPerceptionUpdated(AActor* Actor, FAIStimulus
Stimulus)
{
	if (!BlackboardComponent) return;
	if (Stimulus.WasSuccessfullySensed())
	{
		if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP
Controller] <%s> Actor %s seen"), *GetPawn()->GetName(),
*Actor->GetName());
		// Set the actor seen in the blackboard

BlackboardComponent->SetValueAsObject(TargetActorKey.SelectedKeyName,
Actor);
		// Calculate distance to actor
		APawn* ControlledPawn = GetPawn();
		if (ControlledPawn)
```

```
                {
                        float Distance =
FVector::Dist(ControlledPawn->GetActorLocation(), Actor->GetActorLocation());

BlackboardComponent->SetValueAsFloat(DistanceToTargetActorKey.SelectedKe
yName, Distance);
                        if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP
Controller] <%s> Distance to actor: %f"), *GetPawn()->GetName(), Distance);
                }

BlackboardComponent->SetValueAsBool(bCanSeeTargetActorKey.SelectedKeyN
ame, true);
        }
        else
        {
                // Lost sight of this actor — clear if this is the current target
                AActor* CurrentTarget =
Cast<AActor>(BlackboardComponent->GetValueAsObject(TEXT("TargetActor"))
);
                if (CurrentTarget == Actor)
                {

BlackboardComponent->SetValueAsObject(TargetActorKey.SelectedKeyName,
nullptr);

BlackboardComponent->SetValueAsFloat(DistanceToTargetActorKey.SelectedKe
yName, 0.f);

BlackboardComponent->SetValueAsBool(bCanSeeTargetActorKey.SelectedKeyN
ame, false);
                }
        }
}
bool AGOAPAIController::GrantGoal(
        const FName& GoalName,
        const int32 Priority,
        const int32 SetPreconditionWorldState,
        const int32 ResetPreconditionWorldState,
        const int32 SetDesiredWorldState,
```

```
        const int32 ResetDesiredWorldState
)
{
        UGOAPGoal* NewGoal = NewObject<UGOAPGoal>(this);
        if (!NewGoal)
        {
                return false;
        }
        // Assign passed-in parameters
        NewGoal->GoalName = GoalName;
        NewGoal->Priority = Priority;
        NewGoal->SetPreconditionWorldState = SetPreconditionWorldState;
        NewGoal->ResetPreconditionWorldState = ResetPreconditionWorldState;
        NewGoal->SetDesiredWorldState = SetDesiredWorldState;
        NewGoal->ResetDesiredWorldState = ResetDesiredWorldState;
        // Add to Goals array
        Goals.Add(NewGoal);
        if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Controller]
<%s> Goal granted: %s"), *GetPawn()->GetName(),
*NewGoal->GoalName.ToString());
        return true;
}
bool AGOAPAIController::GrantAction(
                const FName& ActionName,
                const int32 Cost,
                const FGameplayTag AbilityTag,
                const FAbilityInputData InputData,
                const int32 SetPreconditionWorldState,
                const int32 ResetPreconditionWorldState,
                const int32 SetResultWorldState,
                const int32 ResetResultWorldState
        )
{
        UGOAPAction* NewAction = NewObject<UGOAPAction>(this);
        if (!NewAction)
        {
                return false;
        }
        NewAction->ActionName = ActionName;
```

```
        NewAction->Cost = Cost;
        NewAction->AbilityTag = AbilityTag;
        NewAction->InputData = InputData;
        NewAction->SetPreconditionWorldState = SetPreconditionWorldState;
        NewAction->ResetPreconditionWorldState = ResetPreconditionWorldState;
        NewAction->SetResultWorldState = SetResultWorldState;
        NewAction->ResetResultWorldState = ResetResultWorldState;
        Actions.Add(NewAction);
        if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Controller]
<%s> Action granted: %s"), *GetPawn()->GetName(),
*NewAction->ActionName.ToString());
        return true;
}
AGOAPAIController::AGOAPAIController()
{
        BlackboardComponent =
CreateDefaultSubobject<UBlackboardComponent>(TEXT("BlackboardComponen
t"));
        AIPerceptionComponent =
CreateDefaultSubobject<UAIPerceptionComponent>(TEXT("AIPerceptionCompo
nent"));
        // Create and configure sight sense
        SightConfig =
CreateDefaultSubobject<UAISenseConfig_Sight>(TEXT("SightConfig"));
        SightConfig->SightRadius = 2000.0f;
        SightConfig->LoseSightRadius = 2500.0f;
        SightConfig->PeripheralVisionAngleDegrees = 90.0f;
        SightConfig->SetMaxAge(5.0f);
        // Detect only enemies, or all if you prefer
        SightConfig->DetectionByAffiliation.bDetectEnemies = true;
        SightConfig->DetectionByAffiliation.bDetectNeutrals = true;
        SightConfig->DetectionByAffiliation.bDetectFriendlies = false;
        // Add the sight config to perception component
        AIPerceptionComponent->ConfigureSense(*SightConfig);

AIPerceptionComponent->SetDominantSense(SightConfig->GetSenseImplementa
tion());
        // Bind event for perception updates
```

```
	AIPerceptionComponent->OnTargetPerceptionUpdated.AddDynamic(this,
&AGOAPAIController::OnPerceptionUpdated);
}
UGOAPGoal* AGOAPAIController::GetDesiredGoal(const int32 WorldState)
{
	UGOAPGoal* DesiredGoal = nullptr;
	int HighestPriority = -999;
	for (const auto Goal : Goals)
	{
		if (Goal->CheckPreconditions(WorldState))
		{
			if (Goal->Priority > HighestPriority)
			{
				HighestPriority = Goal->Priority;
				DesiredGoal = Goal;
			}
		}
	}
	return DesiredGoal;
}
```

Файл GOAPAIController.h

Реалізація функціональних вимог: Робота штучного інтелекту

```
#pragma once
#include "CoreMinimal.h"
#include "GOAPAction.h"
#include "GOAPGoal.h"
#include "AIController.h"
#include "BehaviorTree/BehaviorTreeTypes.h"
#include "Perception/AISenseConfig_Sight.h"
#include "GOAPAIController.generated.h"
UCLASS()
class DESCENDANCEHERO_API AGOAPAIController : public AAIController
{
	GENERATED_BODY()
public:
	AGOAPAIController();
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "GOAP")
```

```
        bool bDebug = false;

        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "GOAP")
        UBlackboardData* BlackboardAsset;
        UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "GOAP")
        UBehaviorTree* BehaviorTreeAsset;
        UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "GOAP")
        UBlackboardComponent* BlackboardComponent;
        UPROPERTY(EditAnywhere, Category = "Blackboard")
        FBlackboardKeySelector TargetActorKey;
        UPROPERTY(EditAnywhere, Category = "Blackboard")
        FBlackboardKeySelector DistanceToTargetActorKey;
        UPROPERTY(EditAnywhere, Category = "Blackboard")
        FBlackboardKeySelector bCanSeeTargetActorKey;
        UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "GOAP")
        TArray<UGOAPGoal*> Goals;
        UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "GOAP")
        TArray<UGOAPAction*> Actions;

        UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "GOAP")
        TArray<UGOAPAction*> CurrentPlan;
        int CurrentPlanIndex = 0;

        UFUNCTION(BlueprintCallable)
        UGOAPGoal* GetDesiredGoal(const int32 WorldState);

protected:
        virtual void OnPossess(APawn* InPawn) override;

        UFUNCTION()
        void OnPerceptionUpdated(AActor* Actor, FAIStimulus Stimulus);
        UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "AI")
        UAIPerceptionComponent* AIPerceptionComponent;
        UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "AI")
        UAISenseConfig_Sight* SightConfig;
        UFUNCTION(BlueprintCallable)
        bool GrantGoal(
                const FName& GoalName,
                const int32 Priority,
```

```
            const int32 SetPreconditionWorldState,
            const int32 ResetPreconditionWorldState,
            const int32 SetDesiredWorldState,
            const int32 ResetDesiredWorldState
        );
        UFUNCTION(BlueprintCallable)
        bool GrantAction(
            const FName& ActionName,
            const int32 Cost,
            const FGameplayTag AbilityTag,
            const FAbilityInputData InputData,
            const int32 SetPreconditionWorldState,
            const int32 ResetPreconditionWorldState,
            const int32 SetResultWorldState,
            const int32 ResetResultWorldState
        );

};
```

Файл GOAPGoal.cpp

Реалізація функціональних вимог: Робота штучного інтелекту

```
#include "GOAPGoal.h"
bool UGOAPGoal::CheckPreconditions(int32 WorldState)
{
        bool SetConditionsMet = (WorldState & SetPreconditionWorldState) ==
SetPreconditionWorldState;
        // Check all required RESET bits are absent
        bool ResetConditionsMet = (~WorldState & ResetPreconditionWorldState)
== ResetPreconditionWorldState;
        return SetConditionsMet && ResetConditionsMet;
}
```

Файл GOAPGoal.h

Реалізація функціональних вимог: Робота штучного інтелекту

```
#pragma once
#include "CoreMinimal.h"
#include "GOAPGoal.generated.h"
```

```
UCLASS(Blueprintable, BlueprintType)
class DESCENDANCEHERO_API UGOAPGoal : public UObject
{
	GENERATED_BODY()
public:
	// The unique name or tag identifying this goal
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GOAP
Goal")
	FName GoalName;
	// Priority for resolving conflicting goals (higher number = higher priority)
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GOAP
Goal")
	int32 Priority;
	// Precondition bitmask representing the required world state
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GOAP
Goal")
	int32 SetPreconditionWorldState;
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GOAP
Goal")
	int32 ResetPreconditionWorldState;
	// Desired bitmask world state
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GOAP
Goal")
	int32 SetDesiredWorldState;
	int32 ResetDesiredWorldState;
	UFUNCTION(BlueprintCallable, Category = "GOAP Goal")
	bool IsEqualTo(UGOAPGoal* OtherGoal) const
	{
		if (!OtherGoal)
		{
			return false;
		}
		return SetPreconditionWorldState ==
OtherGoal->SetPreconditionWorldState &&
			ResetPreconditionWorldState ==
OtherGoal->ResetPreconditionWorldState &&
			SetDesiredWorldState == OtherGoal->SetDesiredWorldState
&&
```

```
                ResetDesiredWorldState ==
OtherGoal->ResetDesiredWorldState;
        }
        UFUNCTION(BlueprintCallable, Category = "GOAP Goal")
        bool CheckPreconditions(int32 WorldState);
};
```

Файл GOAPGoalChecker.cpp
Реалізація функціональних вимог: Робота штучного інтелекту

```
// Descendance Hero by Dmytro Stetsun
#include "GOAPGoalChecker.h"
#include "GOAPGoal.h"
#include "AIController.h"
#include "BehaviorTree/BlackboardComponent.h"
#include "GOAPAIController.h"
#include "WorldStateUtility.h"
UGOAPGoalChecker::UGOAPGoalChecker()
{
}
void UGOAPGoalChecker::TickNode(UBehaviorTreeComponent& OwnerComp,
uint8* NodeMemory, float DeltaSeconds)
{
        Super::TickNode(OwnerComp, NodeMemory, DeltaSeconds);
        AAIController* AIOwner = OwnerComp.GetAIOwner();
        if (!AIOwner)
                return;
        AIController = Cast<AGOAPAIController>(AIOwner);
        if (!AIController)
                return;
        if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Service] <%s>
AI Controller found!"),
                        *AIController->GetPawn()->GetName());
        BlackboardComp = OwnerComp.GetBlackboardComponent();
        if (!BlackboardComp)
                return;
        if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Service] <%s>
BlackboardComponent found!"),
                        *AIController->GetPawn()->GetName());
```

```
	if (!CheckWorldState())
		return;
	CheckGoal();
	UE_LOG(LogTemp, Warning, TEXT("[GOAP Service] <%s> AI Service
executed finished execution"),
		*AIController->GetPawn()->GetName());
}
void UGOAPGoalChecker::CheckGoal()
{
	UGOAPGoal* DesiredGoal = AIController->GetDesiredGoal(WorldState);
	if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Service] <%s>
DesiredGoal: %s"),
			*AIController->GetPawn()->GetName(),
*DesiredGoal->GoalName.ToString());
	UGOAPGoal* CurrentGoal =
Cast<UGOAPGoal>(BlackboardComp->GetValueAsObject(CurrentGoalKey.Sele
ctedKeyName));
	if (!CurrentGoal)
	{
		if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Service]
<%s> CurrentGoal not found, setting to desired!"),
				*AIController->GetPawn()->GetName());

BlackboardComp->SetValueAsObject(CurrentGoalKey.SelectedKeyName,
DesiredGoal);
		return;
	}
	if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Service] <%s>
CurrentGoal found!"),
			*AIController->GetPawn()->GetName());
	if (!CurrentGoal->IsEqualTo(DesiredGoal))
	{
		if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Service]
<%s> Desired goal is not the same as current!"),
				*AIController->GetPawn()->GetName());

BlackboardComp->SetValueAsObject(CurrentGoalKey.SelectedKeyName,
DesiredGoal);
```

```
            BlackboardComp->SetValueAsBool(bReplanKey.SelectedKeyName,
true);
            return;
        }
        if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Service] <%s>
Desired goal is the same as current!"),
                *AIController->GetPawn()->GetName());
        BlackboardComp->SetValueAsBool(bReplanKey.SelectedKeyName, false);
}
bool UGOAPGoalChecker::CheckWorldState()
{
        SelfActor =
Cast<ABaseCharacter>(BlackboardComp->GetValueAsObject(SelfActorKey.Sele
ctedKeyName));
        if (!SelfActor) SelfActor =
Cast<ABaseCharacter>(AIController->GetPawn());
        if (!SelfActor) return false;
        if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Service] <%s>
SelfCharacter found!"),
                *AIController->GetPawn()->GetName());
        // Get Target Character
        TargetActor =
Cast<ABaseCharacter>(BlackboardComp->GetValueAsObject(TargetActorKey.Se
lectedKeyName));
        if (bDebug)
        {
            if (TargetActor) UE_LOG(LogTemp, Warning, TEXT("[GOAP]
TargetCharacter found!"))
            else UE_LOG(LogTemp, Warning, TEXT("[GOAP] TargetCharacter
not found!"));
        }
        DistanceToTargetActor =
BlackboardComp->GetValueAsFloat(DistanceToTargetActorKey.SelectedKeyNam
e);
        if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Service] <%s>
DistanceToTarget found!"),
                *AIController->GetPawn()->GetName());
        bCanSeeTargetActor =
BlackboardComp->GetValueAsBool(bCanSeeTargetActorKey.SelectedKeyName);
```

```
        if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Service] <%s>
bCanSeeTarget found!"),
                                        *AIController->GetPawn()->GetName());
        WorldState = CreateWorldState();
        BlackboardComp->SetValueAsInt(WorldStateKey.SelectedKeyName,
WorldState);
        if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Service] <%s>
WorldState created: %d"),
                        *AIController->GetPawn()->GetName(), WorldState);
        return true;
}
int32 UGOAPGoalChecker::CreateWorldState() const
{
        int32 WorldState = 0;
        const UAttributeComponent* SelfAttributeComponent =
IIAttributeOwner::Execute_GetAttributeComponent(SelfActor);
        const UAbilityComponent* SelfAbilityComponent =
IIAttributeOwner::Execute_GetAbilityComponent(SelfActor);
        // Self HP & Alive
        UE_LOG(LogTemp, Warning, TEXT("[GOAP Service] <%s> Self HP: %f,
MaxHP: %f"), *AIController->GetPawn()->GetName(),
                SelfAttributeComponent->Health,
SelfAttributeComponent->MaxHealth);
        if (SelfAttributeComponent->Health > 2.f *
SelfAttributeComponent->MaxHealth / 3.f)
        {
                UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::HighHP);
        }
        else if (SelfAttributeComponent->Health >
SelfAttributeComponent->MaxHealth / 3.f)
        {
                UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::MidHP);
        }
        else
        {
                UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::LowHP);
```

```
        }
        if (SelfAttributeComponent->bAlive)
        {
                UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::Alive);
        }
        else
        {
                UWorldStateUtility::ResetStateFlag(WorldState,
EWorldStateFact::LowHP);
                UWorldStateUtility::ResetStateFlag(WorldState,
EWorldStateFact::MidHP);
                UWorldStateUtility::ResetStateFlag(WorldState,
EWorldStateFact::HighHP);
        }
        // Self RS
        UE_LOG(LogTemp, Warning, TEXT("[GOAP Service] <%s> Self RS: %f,
MaxRS: %f"), *AIController->GetPawn()->GetName(),
                SelfAttributeComponent->Resource,
SelfAttributeComponent->MaxResource);
        if (SelfAttributeComponent->Resource > 2.f *
SelfAttributeComponent->MaxResource / 3.f)
        {
                UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::HighRS);
        }
        else if (SelfAttributeComponent->Resource >
SelfAttributeComponent->MaxResource / 3.f)
        {
                UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::MidRS);
        }
        else
        {
                UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::LowRS);
        }
        // Self Abilities
        bool bHasMeleeAbility = false;
```

```
	bool bHasRangeAbility = false;
	for (const TPair Ability : SelfAbilityComponent->GetAbilities())
	{
		if (Ability.Value->AbilityType == EAbilityType::Melee &&
!Ability.Value->bOnCooldown)
		{
			bHasMeleeAbility = true;
		}
		else if (Ability.Value->AbilityType == EAbilityType::Range &&
!Ability.Value->bOnCooldown)
		{
			bHasRangeAbility = true;
		}
	}
	if (bHasMeleeAbility)
	{
		UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::HasMelee);
		UE_LOG(LogTemp, Warning, TEXT("[GOAP Service] Has Melee
Ability"));
	}
	if (bHasRangeAbility)
	{
		UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::HasRange);
		UE_LOG(LogTemp, Warning, TEXT("[GOAP Service] Has Range
Ability"));
	}
	if (TargetActor)
	{
		const UAttributeComponent* TargetAttributeComponent =
IIAttributeOwner::Execute_GetAttributeComponent(
			TargetActor);
		// Target HP
		UE_LOG(LogTemp, Warning, TEXT("[GOAP Service] <%s> Target
HP: %f, MaxHP: %f"),
			*AIController->GetPawn()->GetName(),
			TargetAttributeComponent->Health,
			TargetAttributeComponent->MaxHealth);
```

```
            if (TargetAttributeComponent->Health > 2.f *
TargetAttributeComponent->MaxHealth / 3.f)
            {
                UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::EnemyHighHP);
            }
            else if (TargetAttributeComponent->Health >
TargetAttributeComponent->MaxHealth / 3.f)
            {
                UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::EnemyMidHP);
            }
            else
            {
                UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::EnemyLowHP);
            }
            if (TargetAttributeComponent->bAlive)
            {
                UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::EnemyAlive);
            }
            else
            {
                UWorldStateUtility::ResetStateFlag(WorldState,
EWorldStateFact::EnemyLowHP);
                UWorldStateUtility::ResetStateFlag(WorldState,
EWorldStateFact::EnemyMidHP);
                UWorldStateUtility::ResetStateFlag(WorldState,
EWorldStateFact::EnemyHighHP);
            }
        }
        else
        {
            UWorldStateUtility::ResetStateFlag(WorldState,
EWorldStateFact::EnemyAlive);
            UWorldStateUtility::ResetStateFlag(WorldState,
EWorldStateFact::EnemyLowHP);
```

```
            UWorldStateUtility::ResetStateFlag(WorldState,
EWorldStateFact::EnemyMidHP);
            UWorldStateUtility::ResetStateFlag(WorldState,
EWorldStateFact::EnemyHighHP);
        }
        // Range
        if (DistanceToTargetActor > SelfAttributeComponent->Range)
        {
            UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::EnemyInFarRange);
        }
        else
        {
            UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::EnemyInCloseRange);
        }
        // In Sight
        if (bCanSeeTargetActor)
        {
            UWorldStateUtility::SetStateFlag(WorldState,
EWorldStateFact::EnemyInSight);
        }
        else
        {
            UWorldStateUtility::ResetStateFlag(WorldState,
EWorldStateFact::EnemyInSight);
            UWorldStateUtility::ResetStateFlag(WorldState,
EWorldStateFact::EnemyInFarRange);
            UWorldStateUtility::ResetStateFlag(WorldState,
EWorldStateFact::EnemyInCloseRange);
        }
        return WorldState;
}
```

Файл GOAPGoalChecker.h

Реалізація функціональних вимог: Робота штучного інтелекту

```
// Descendance Hero by Dmytro Stetsun
#pragma once
```

```
#include "CoreMinimal.h"
#include "BaseCharacter.h"
#include "BehaviorTree/BTService.h"
#include "GOAPAIController.h"
#include "GOAPGoalChecker.generated.h"
UCLASS()
class DESCENDANCEHERO_API UGOAPGoalChecker: public UBTService
{
	GENERATED_BODY()
public:
	UGOAPGoalChecker();
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GOAP")
	bool bDebug = false;
	UPROPERTY(EditAnywhere, Category = "Blackboard")
	FBlackboardKeySelector SelfActorKey;
	UPROPERTY(EditAnywhere, Category = "Blackboard")
	FBlackboardKeySelector TargetActorKey;
	UPROPERTY(EditAnywhere, Category = "Blackboard")
	FBlackboardKeySelector CurrentGoalKey;
	UPROPERTY(EditAnywhere, Category = "Blackboard")
	FBlackboardKeySelector bReplanKey;

	UPROPERTY(EditAnywhere, Category = "Blackboard")
	FBlackboardKeySelector DistanceToTargetActorKey;
	UPROPERTY(EditAnywhere, Category = "Blackboard")
	FBlackboardKeySelector WorldStateKey;

	UPROPERTY(EditAnywhere, Category = "Blackboard")
	FBlackboardKeySelector bCanSeeTargetActorKey;

	virtual void TickNode(UBehaviorTreeComponent& OwnerComp, uint8*
NodeMemory, float DeltaSeconds) override;
	UFUNCTION(BlueprintCallable)
	void CheckGoal();
	UFUNCTION(BlueprintCallable)
	bool CheckWorldState();
private:
	UPROPERTY()
	AGOAPAIController* AIController;
```

```
	UPROPERTY()
	UBlackboardComponent* BlackboardComp;
	UPROPERTY()
	ABaseCharacter* SelfActor;
	UPROPERTY()
	ABaseCharacter* TargetActor;
	UPROPERTY()
	float DistanceToTargetActor;
	UPROPERTY()
	bool bCanSeeTargetActor;
	UPROPERTY()
	int32 WorldState = 0;
	UFUNCTION()
	int32 CreateWorldState() const;
};
```

Файл GOALPlanActions.cpp

Реалізація функціональних вимог: Робота штучного інтелекту

```
#include "GOAPPlanActions.h"
#include "GOAPAIController.h"
#include "BehaviorTree/BehaviorTreeTypes.h"
#include "BehaviorTree/BlackboardComponent.h"
#include "GameFramework/Character.h"
UGOAPPlanActions::UGOAPPlanActions()
{
}
TArray<UGOAPAction*> UGOAPPlanActions::PlanActions(uint32
CurrentWorldState, TArray<UGOAPAction*>& Actions, UGOAPGoal* Goal)
const
{
	TArray<UGOAPAction*> BestActionPlan = {};
	int32 BestHeuristic = MAX_int32;
	TArray<FPlanNode> OpenSet;
	OpenSet.Add(FPlanNode(CurrentWorldState, {}, 0,
CalculateHeuristic(CurrentWorldState, Goal)));
	int Iter = 0;
	while (OpenSet.Num() > 0 && Iter < MaxIter)
```

```
	{
		int32 BestIndex = 0;
		for (int32 i = 1; i < OpenSet.Num(); ++i)
		{
			if (OpenSet[i].EstimatedTotalCost < 
OpenSet[BestIndex].EstimatedTotalCost) BestIndex = i;
		}
		FPlanNode CurrentNode = OpenSet[BestIndex];
		OpenSet.RemoveAt(BestIndex);
		int32 CurrentHeuristic = CalculateHeuristic(CurrentNode.WorldState, 
Goal);
		if (CurrentHeuristic < BestHeuristic)
		{
			BestHeuristic = CurrentHeuristic;
			BestActionPlan = CurrentNode.ActionSequence;
			if (BestHeuristic == 0) return BestActionPlan;
		}
		for (UGOAPAction* Action : Actions)
		{
			if (Action && 
Action->CheckPreconditions(CurrentNode.WorldState))
			{
				int32 NewWorldState = 
Action->GetPostWorldState(CurrentNode.WorldState);
				int32 NewCost = CurrentNode.CostSoFar + 
Action->Cost;
				int32 Heuristic = CalculateHeuristic(NewWorldState, 
Goal);
				TArray<UGOAPAction*> NewSequence = 
CurrentNode.ActionSequence;
				NewSequence.Add(Action);
				OpenSet.Add(FPlanNode(NewWorldState, 
NewSequence, NewCost, NewCost + Heuristic));
			}
		}
		Iter++;
	}
	return BestActionPlan;
}
```

```
int UGOAPPlanActions::CalculateHeuristic(const uint32 CurrentWorldState, const
UGOAPGoal* Goal)
{
	int Mismatch = 0;
	// Bits that should be set but aren't
	int32 MissingSetBits = Goal->SetDesiredWorldState & ~CurrentWorldState;
	for (int32 i = 0; i < 32; ++i)
	{
		if (MissingSetBits & (1 << i))
			++Mismatch;
	}
	// Bits that should be unset but are set
	int32 ExtraBits = CurrentWorldState & Goal->ResetDesiredWorldState;
	for (int32 i = 0; i < 32; ++i)
	{
		if (ExtraBits & (1 << i))
			++Mismatch;
	}
	return Mismatch;
}
EBTNodeResult::Type
UGOAPPlanActions::ExecuteTask(UBehaviorTreeComponent& OwnerComp,
uint8* NodeMemory)
{
	AGOAPAIController* AIController =
Cast<AGOAPAIController>(OwnerComp.GetAIOwner());
	if (!AIController)
	{
		return EBTNodeResult::Failed;
	}
	// Example: Get current world state and desired goal from AIController
	int32 CurrentWorldState =
OwnerComp.GetBlackboardComponent()->GetValueAsInt(WorldStateKey.Selecte
dKeyName);
	UGOAPGoal* DesiredGoal =
Cast<UGOAPGoal>(OwnerComp.GetBlackboardComponent()->GetValueAsObje
ct(CurrentGoalKey.SelectedKeyName));;
	if (!DesiredGoal)
	{
```

```
            if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Planner]
<%s> No desired goal found during planning"),
*AIController->GetPawn()->GetName() );
            return EBTNodeResult::Failed;
        }
        TArray<UGOAPAction*> Plan = PlanActions(CurrentWorldState,
AIController->Actions, DesiredGoal);
        AIController->CurrentPlan = Plan;
        AIController->CurrentPlanIndex = 0;
        if (Plan.Num())
        {
            if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Planner]
<%s> Planning failed for goal %s"), *AIController->GetPawn()->GetName(),
*DesiredGoal->GoalName.ToString());
            return EBTNodeResult::Failed;
        }
        if (bDebug) UE_LOG(LogTemp, Warning, TEXT("[GOAP Planner] <%s>
Action plan successfully generated for goal %s"),
*AIController->GetPawn()->GetName(), *DesiredGoal->GoalName.ToString());

        // Success — next task in tree will execute
        return EBTNodeResult::Succeeded;
}
```

Файл GOAPPlanActions.h

Реалізація функціональних вимог: Робота штучного інтелекту

```
#pragma once
#include "CoreMinimal.h"
#include "GOAPAction.h"
#include "GOAPGoal.h"
#include "BehaviorTree/BTTaskNode.h"
#include "GOAPPlanActions.generated.h"
USTRUCT(BlueprintType)
struct FPlanNode
{
        GENERATED_BODY()
        // The world state that this node represents.
```

```
	UPROPERTY()
	int32 WorldState;
	UPROPERTY()
	TArray<UGOAPAction*> ActionSequence;
	UPROPERTY()
	int32 CostSoFar;
	UPROPERTY()
	int32 EstimatedTotalCost;
	FPlanNode()
		: WorldState(0), CostSoFar(0), EstimatedTotalCost(0) {}
	FPlanNode(int32 InWorldState, const TArray<UGOAPAction*>&
InActions, int32 InCost, int32 InEstimate)
		: WorldState(InWorldState), ActionSequence(InActions),
CostSoFar(InCost), EstimatedTotalCost(InEstimate) {}
};
UCLASS()
class DESCENDANCEHERO_API UGOAPPlanActions : public UBTTaskNode
{
	GENERATED_BODY()
public:
	UGOAPPlanActions();

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "GOAP")
	bool bDebug = false;
	UPROPERTY(EditAnywhere, Category = "GOAP")
	int MaxIter = 100;
	UPROPERTY(EditAnywhere, Category = "Blackboard")
	FBlackboardKeySelector WorldStateKey;
	UPROPERTY(EditAnywhere, Category = "Blackboard")
	FBlackboardKeySelector CurrentGoalKey;
	UFUNCTION()
	TArray<UGOAPAction*> PlanActions(uint32 CurrentWorldState,
TArray<UGOAPAction*>& Actions, UGOAPGoal* Goal) const;
	UFUNCTION()
	static int CalculateHeuristic(const uint32 CurrentWorldState, const
UGOAPGoal* Goal);
	virtual EBTNodeResult::Type ExecuteTask(UBehaviorTreeComponent&
OwnerComp, uint8* NodeMemory) override;
};
```

Файл IAttributeOwner.h
Реалізація функціональних вимог: Система атрибутів

```
// Descendance Hero by Dmytro Stetsun
#pragma once
#include "CoreMinimal.h"
#include "AbilityComponent.h"
#include "AttributeComponent.h"
#include "UObject/Interface.h"
#include "IAttributeOwner.generated.h"
// This class does not need to be modified.
UINTERFACE(MinimalAPI)
class UIAttributeOwner : public UInterface
{
	GENERATED_BODY()
};
/**
 *
 */
class DESCENDANCEHERO_API IIAttributeOwner
{
	GENERATED_BODY()
	// Add interface functions to this class. This is the class that will be inherited
to implement this interface.
public:
	UFUNCTION(BlueprintCallable, BlueprintNativeEvent)
	UAttributeComponent* GetAttributeComponent() const;
	UFUNCTION(BlueprintCallable, BlueprintNativeEvent)
	UAbilityComponent* GetAbilityComponent() const;
	UFUNCTION(BlueprintCallable, BlueprintNativeEvent)
	void OnCharacterDied();
	UFUNCTION(BlueprintCallable, BlueprintNativeEvent)
	bool IsAlive() const;
	UFUNCTION(BlueprintCallable, BlueprintNativeEvent)
	float TakeDamage(FDamageInfo DamageInfo) const;
	UFUNCTION(BlueprintCallable, BlueprintNativeEvent)
	void Heal(float Amount) const;
};
```

Файл InventoryComponent.cpp
Реалізація функціональних вимог: Система інвентара

```
#include "InventoryComponent.h"
#include "IAttributeOwner.h"
bool UInventoryComponent::AddItem(const FInventoryItem& Item)
{
	if (InventoryItems.Num() == MaxInventorySize)
		return false;
	const int32 Hash = GetItemHash(Item);
	InventoryItems.Add(Hash, Item);
	return true;
}
void UInventoryComponent::EquipItem(const int32 Index, const EItemClass
Class)
{
	switch (Class)
	{
		case EItemClass::Weapon:
			if (EquippedWeapon == -1)
			{
				EquippedWeapon = Index;
				for (FAttributeModifier& Modifier :
InventoryItems[Index].AttributeModifiers)
				{

AttributeComponent->AddItemModifier(&Modifier);
				}
			}
			else
			{
				SwapEquippedItems(EquippedWeapon, Index, Class);
			}
			break;
		case EItemClass::Armor:
			if (EquippedArmor == -1)
			{
				EquippedArmor = Index;
```

```
                        for (FAttributeModifier& Modifier :
InventoryItems[Index].AttributeModifiers)
                        {

AttributeComponent->AddItemModifier(&Modifier);
                        }
                }
                else
                {
                        SwapEquippedItems(EquippedArmor, Index, Class);
                }
                break;
        case EItemClass::Accessory:
                if (EquippedAccessory == -1)
                {
                        EquippedAccessory = Index;
                }
                else
                {
                        SwapEquippedItems(EquippedAccessory, Index, Class);
                }
                break;
    }
    UE_LOG(LogTemp, Warning, TEXT("Equipped item:"));
}
void UInventoryComponent::UnequipItem(const int32 Index, const EItemClass
Class)
{
    for (FAttributeModifier& Modifier :
InventoryItems[Index].AttributeModifiers)
    {
        AttributeComponent->RemoveItemModifier(&Modifier);
    }
    switch (Class)
    {
        case EItemClass::Weapon:
            EquippedWeapon = -1;
            break;
        case EItemClass::Armor:
```

```
                EquippedArmor = -1;
                break;
            case EItemClass::Accessory:
                EquippedAccessory = -1;
                break;
        }
        UE_LOG(LogTemp, Warning, TEXT("Unequipped item:"));
}
void UInventoryComponent::RemoveItem(const int32 Index)
{
        InventoryItems.Remove(Index);
}
void UInventoryComponent::SwapEquippedItems(const int32 OldIndex, const
int32 NewIndex, const EItemClass Class)
{
        for (FAttributeModifier& Modifier :
InventoryItems[OldIndex].AttributeModifiers)
        {
            AttributeComponent->RemoveItemModifier(&Modifier);
        }
        switch (Class)
        {
            case EItemClass::Weapon:
                EquippedWeapon = NewIndex;
                break;
            case EItemClass::Armor:
                EquippedArmor = NewIndex;
                break;
            case EItemClass::Accessory:
                EquippedWeapon = NewIndex;
                break;
        }
        for (FAttributeModifier& Modifier :
InventoryItems[NewIndex].AttributeModifiers)
        {
            AttributeComponent->AddItemModifier(&Modifier);
        }
}
void UInventoryComponent::BeginPlay()
```

```
{
	Super::BeginPlay();
	const AActor* Owner = GetOwner();
	if
(Owner->GetClass()->ImplementsInterface(UIAttributeOwner::StaticClass()))
	{
		AttributeComponent =
IIAttributeOwner::Execute_GetAttributeComponent(Owner);
	}
}
int32 UInventoryComponent::GetItemHash(const FInventoryItem& Item)
{
	uint32 CombinedHash = FCrc::StrCrc32(*Item.Name);
	CombinedHash = FCrc::MemCrc32(&Item.Rarity, sizeof(EItemRarity),
CombinedHash);
	CombinedHash = FCrc::MemCrc32(&Item.Class, sizeof(EItemClass),
CombinedHash);

	for (const FAttributeModifier& Modifier : Item.AttributeModifiers)
	{
		CombinedHash = FCrc::MemCrc32(&Modifier,
sizeof(FAttributeModifier), CombinedHash);
	}
	return static_cast<int32>(CombinedHash);
}
```

Файл InventoryComponent.h

Реалізація функціональних вимог: Система інвентара

```
#pragma once
#include "CoreMinimal.h"
#include "AttributeComponent.h"
#include "InventoryItem.h"
#include "InventoryComponent.generated.h"
UCLASS(ClassGroup=(Custom), meta=(BlueprintSpawnableComponent))
class DESCENDANCEHERO_API UInventoryComponent : public
UActorComponent
{
```

```
	GENERATED_BODY()
public:
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly,
Category="Inventory")
	TMap<int32, FInventoryItem> InventoryItems;
	UPROPERTY(BlueprintReadOnly, Category="Inventory")
	UAttributeComponent* AttributeComponent;
	UPROPERTY(BlueprintReadOnly, Category="Inventory")
	int MaxInventorySize = 10;
	UPROPERTY(BlueprintReadOnly, Category="Inventory")
	int Money = 0;
	UPROPERTY(BlueprintReadOnly, Category="Inventory")
	int32 EquippedWeapon = -1;
	UPROPERTY(BlueprintReadOnly, Category="Inventory")
	int32 EquippedArmor = -1;
	UPROPERTY(BlueprintReadOnly, Category="Inventory")
	int32 EquippedAccessory = -1;

	UFUNCTION(BlueprintCallable)
	bool AddItem(const FInventoryItem& Item);
	UFUNCTION(BlueprintCallable)
	void EquipItem(const int32 Index, const EItemClass Class);
	UFUNCTION(BlueprintCallable)
	void UnequipItem(const int32 Index, const EItemClass Class);
	UFUNCTION(BlueprintCallable)
	void RemoveItem(const int32 Index);
protected:
	UFUNCTION(BlueprintCallable)
	void SwapEquippedItems(const int32 OldIndex, const int32 NewIndex,
const EItemClass Class);
	virtual void BeginPlay() override;
	static int32 GetItemHash(const FInventoryItem& Item);
};
```

Файл InventoryItem.h

Реалізація функціональних вимог: Система інвентара

```
#pragma once
#include "CoreMinimal.h"
```

```
#include "AttributeModifier.h"
#include "InventoryItem.generated.h"
UENUM(BlueprintType)
enum class EItemRarity : uint8
{
	Common,
	Uncommon,
	Rare,
	Epic,
	Legendary
};
UENUM(BlueprintType)
enum class EItemClass : uint8
{
	Weapon,
	Armor,
	Accessory
};
USTRUCT(BlueprintType)
struct FInventoryItem : public FTableRowBase
{
	GENERATED_BODY()
	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	FString Name;
	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	EItemRarity Rarity;
	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	EItemClass Class;
	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	float ModifierChance = 0.f;
	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	int Value = 0;

	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	UTexture2D* Icon;
	UPROPERTY(EditAnywhere, BlueprintReadWrite)
	TArray<FAttributeModifier> AttributeModifiers = {};
};
```

Файл WorldStateUtility.cpp
Реалізація функціональних вимог: Робота штучного інтелекту

```
#include "WorldStateUtility.h"
void UWorldStateUtility::SetStateFlag(int32& State, EWorldStateFact Fact)
{
	State |= (1 << static_cast<uint8>(Fact));
}
bool UWorldStateUtility::HasStateFlag(int32 State, EWorldStateFact Fact)
{
	return (State & (1 << static_cast<uint8>(Fact))) != 0;
}
void UWorldStateUtility::ResetStateFlag(int32& State, EWorldStateFact Fact)
{
	State &= ~(1 << static_cast<uint8>(Fact));
}
int32 UWorldStateUtility::CreateBitmask(TArray<EWorldStateFact> Facts)
{
	int32 State = 0;
	if (Facts.Num() != 0)
	{
		for (const auto Fact : Facts)
		{
			SetStateFlag(State, Fact);
		}
	}
	return State;
}
```

Файл WorldStateUtility.h
Реалізація функціональних вимог: Робота штучного інтелекту

```
#pragma once
#include "CoreMinimal.h"
#include "WorldStateUtility.generated.h"
UENUM(BlueprintType)
enum class EWorldStateFact : uint8
{
	Alive = 0,
```

```
        LowHP = 1,
        MidHP = 2,
        HighHP = 3,
        LowRS = 4,
        MidRS = 5,
        HighRS = 6,
        HasMelee=7,
        HasRange=8,
        EnemyAlive=9,
        EnemyLowHP=10,
        EnemyMidHP=11,
        EnemyHighHP=12,
        EnemyInCloseRange=13,
        EnemyInFarRange=14,
        EnemyInSight=15
};
UCLASS()
class DESCENDANCEHERO_API UWorldStateUtility : public UObject
{
        GENERATED_BODY()
public:
        UFUNCTION(BlueprintPure, Category="GOAP|Bitmask")
        static void SetStateFlag(UPARAM(ref) int32& State, EWorldStateFact
Fact);
        UFUNCTION(BlueprintPure, Category="GOAP|Bitmask")
        static bool HasStateFlag(int32 State, EWorldStateFact Fact);
        UFUNCTION(BlueprintPure, Category="GOAP|Bitmask")
        static void ResetStateFlag(UPARAM(ref) int32& State, EWorldStateFact
Fact);
        UFUNCTION(BlueprintPure, Category="GOAP|Bitmask")
        static int32 CreateBitmask(TArray<EWorldStateFact> Facts);
};
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ОДНОКОРИСТУВАЦЬКИЙ ІГРОВИЙ ЗАСТОСУНОК У ЖАНРІ ACTION RPG З ВИКОРИСТАННЯМ ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ НА РУШІЇ UNREAL ENGINE

Програма та методика тестування

КПІ.ІТ-0222.045480.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Владислав САРНАЦЬКИЙ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Дмитро СТЕЦУН

Київ – 2025

ЗМІСТ

1	ОБ’ЄКТ ВИПРОБУВАНЬ	3
2	МЕТА ТЕСТУВАННЯ	4
3	МЕТОДИ ТЕСТУВАННЯ	5
4	ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ	7

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є однокористувацький ігровий застосунок у жанрі Action RPG з використанням ігрового штучного інтелекту. Програмний продукт створено на рушії Unreal Engine мовою програмування C++. Програмний продукт може бути запущений на персональних комп’ютерах під управлінням операційних систем Window, Linux та macOS.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка сумісності застосунку з різними операційними системами (Windows, Linux та macOS);
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу;

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;
- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Під час проведення тестування будуть використовуватись наступні допоміжні засоби:

- статичний аналізатор коду вбудований в середовище розробки Rider;
- логування даних під час роботи програми;

Порядок проведення тестування буде наступним:

- проведення статистичного аналізу коду на дотримання норм програмування C++ та норм, описаних у документації Unreal Engine;
- проведення функціонального аналізу за допомогою логування даних під час їх обробки в процесі гри;
- написання тест-кейсів згідно з функціональними вимогами;
- мануальна перевірка відповідності роботи програми до написаних раніше тест-кейсів.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ОДНОКОРИСТУВАЦЬКИЙ ІГРОВИЙ ЗАСТОСУНОК У ЖАНРІ ACTION RPG З ВИКОРИСТАННЯМ ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ НА РУШІЇ UNREAL ENGINE

Керівництво користувача

КПІ.ІТ-0222.045480.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Владислав САРНАЦЬКИЙ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Дмитро СТЕЦУН

Київ – 2025

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ 3
2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ 4
2.1 Системні вимоги для коректної роботи 4
2.2 Завантаження застосунку 4
2.3 Перевірка коректної роботи 4
3 ВИКОНАННЯ ПРОГРАМИ 6

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Descendance Hero — однокористувацький ігровий застосунок у жанрі Action RPG з використанням ігрового штучного інтелекту на рушії Unreal Engine, призначений для розваг та зняття стресу серед великого кола користувачів, що грають в комп’ютерні ігри.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку висуваються певні вимоги.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i3-6100 або аналог за продуктивністю;

– об‘єм ОЗП: 6 Гб;

– Графічний процесор: Nvidia GeForce GTX980 або аналогічний за продуктивністю;

– Місце на диску: 10 ГБ HDD.

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i5-8400 або аналог за продуктивністю;

– об‘єм ОЗП: 8 Гб;

– Графічний процесор: Nvidia GeForce GTX980 або аналогічний за продуктивністю;

– Місце на диску: 10 ГБ SSD.

2.2 Завантаження застосунку

На даний момент застосунок можна встановити власноруч, використовуючи відповідний exe-файл на платформах Windows. Його спершу потрібно завантажити на пристрій та запустити для виконання. Для Linux та macOS встановлення не потрібне, достатньо Завантажити файл, який є самостійним застосунком, встановлювати його не потрібно.

Також, можна завантажити вихідний код застосунку з github та збудувати застосунок самостійно всередині редактора Unreal Engine.

2.3 Перевірка коректної роботи

На платформі Windows go завершенню встановлення додатка на робочому столі пристрою повинна відобразитись іконка даного застосунку. У разі, якщо дана іконка не з’явилась, то встановлення відбулось не успішно.

Після встановлення застосунку на пристрої з Windows або завантаження застосунку на Linux та macOS користувач має змогу запустити додаток, клацнувши на його іконку. Після натискання повинна відобразитись початкова сторінка застосунку.

3 ВИКОНАННЯ ПРОГРАМИ

На початку гри користувача зустрічає головне меню з кнопками початку грита виходу (рис 4.2).

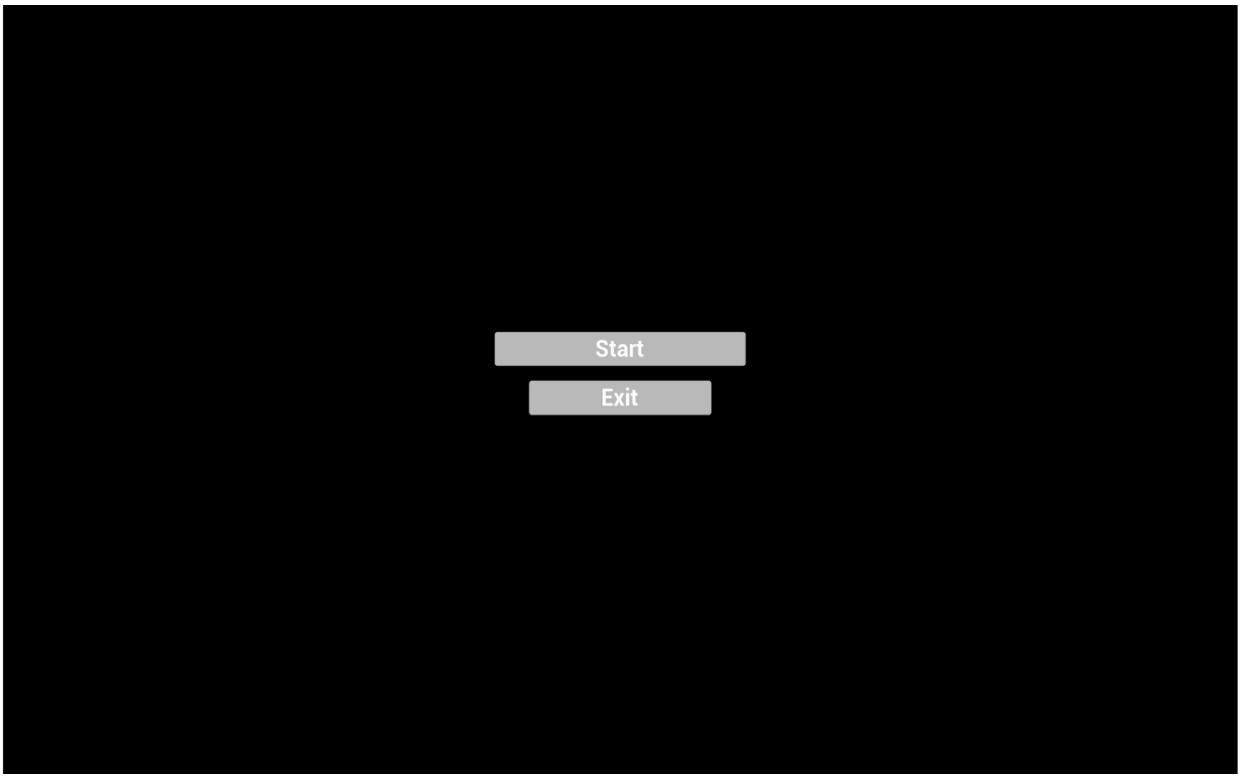


Рисунок 4.2 — Головне меню на початку гри

Після натиснення на кнопку початку відбувається перехід на меню вибору персонажа (рис 4.3). Після натискання кнопки вибору обраного персонажу відбувається перехід на стартову локацію (рис 4.4).

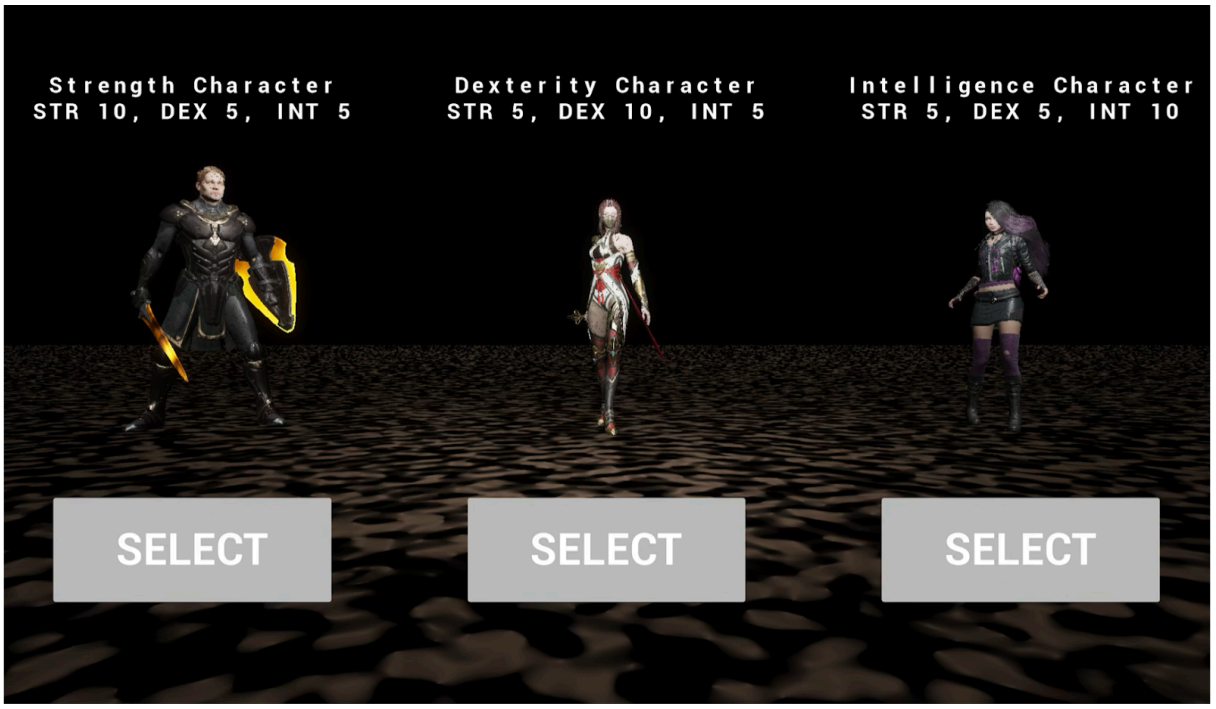


Рисунок 4.3 — Меню вибору персонажа

Рисунок 4.4 — Стартова локація

Тут можна придбати нові речі або продати старі, однак на даних момент у персонажу немає грошей та зайвих предметів для цього (рис 4.5). Тому, зайдемо в портал і потрапимо в перше підземелля (4.5).

У цьому лабіринті є вороги, яких потрібно знищити, однак головне завдання це знайти та вбити боса підземелля. При зустрічі ворога, щоб

завдати йому шкоди потрібно використати одну з доступних навичок (рис 4.7).

Рисунок 4.5 — Меню продавця

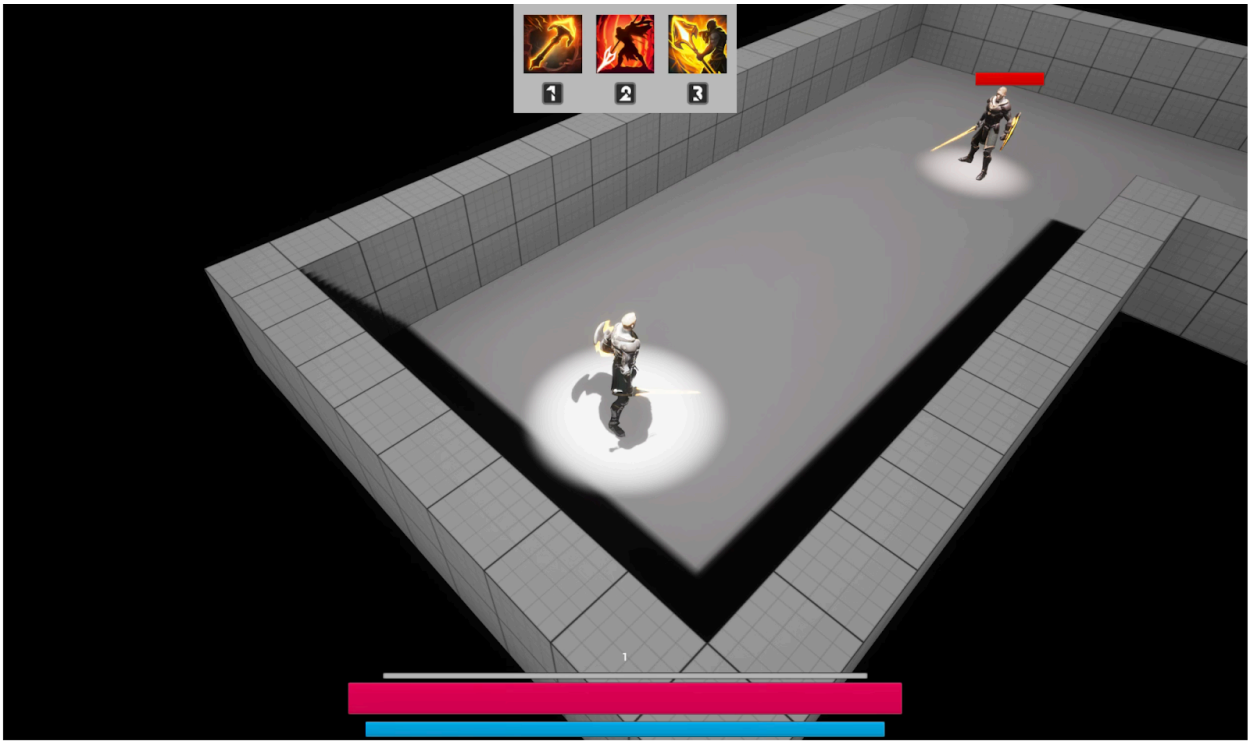


Рис 4.6 — Підземелля

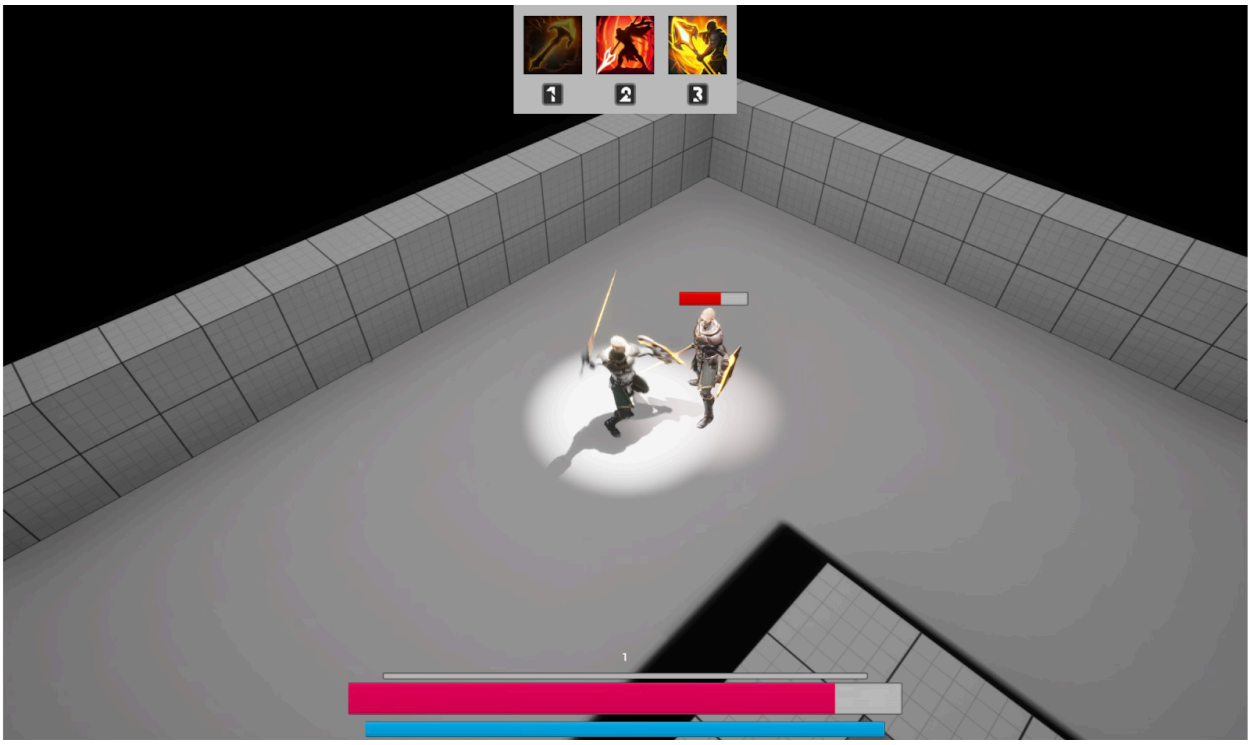


Рисунок 4.7 — Процес битви з ворогом

При смерті ворога з певною вірогідністю випадає предмет, який можна одягнути та покращити власні характеристики (рис. 4.8 - 4.10).

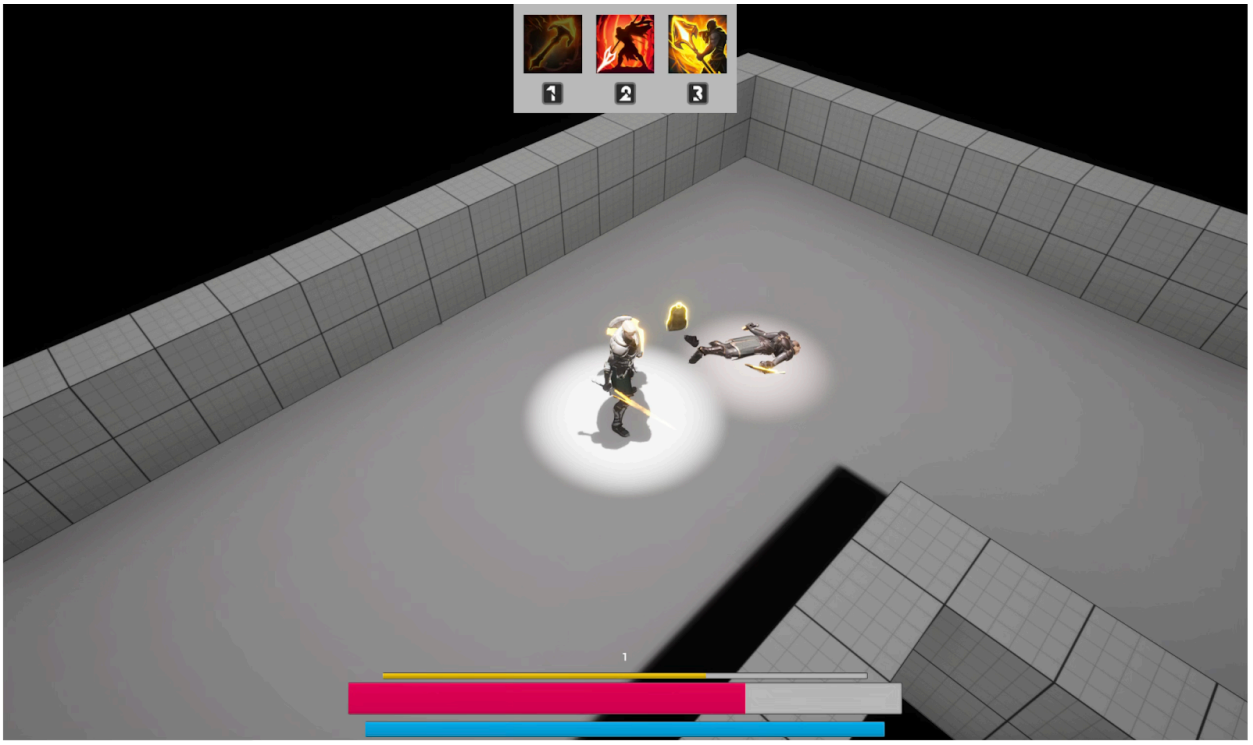


Рисунок. 4.8 — Предмет, що випав з ворога

Рисунок 4.9 — Предмет з ворога в інвентарі

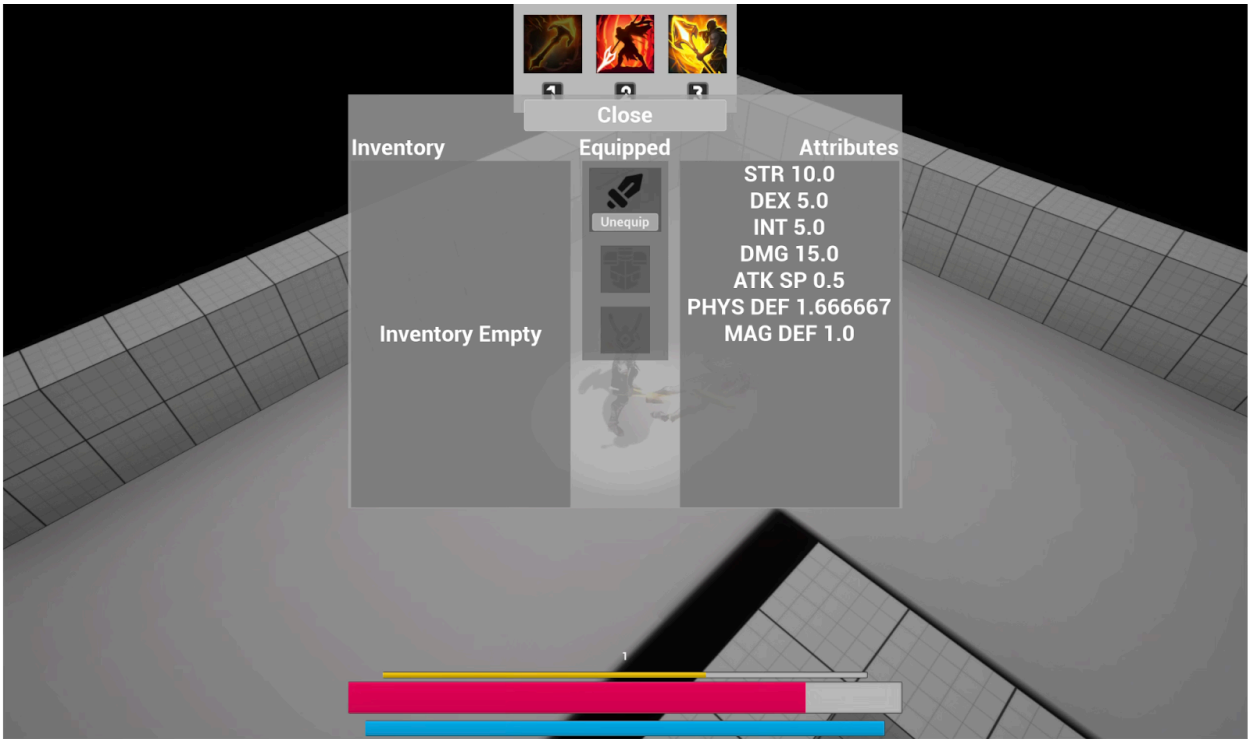


Рисунок 4.10 — Споряджений предмет

Після проходження певної частини лабіринту знайшли боса підземелля, що є дещо сильнішим за звичайних ворогів (рис 4.11).

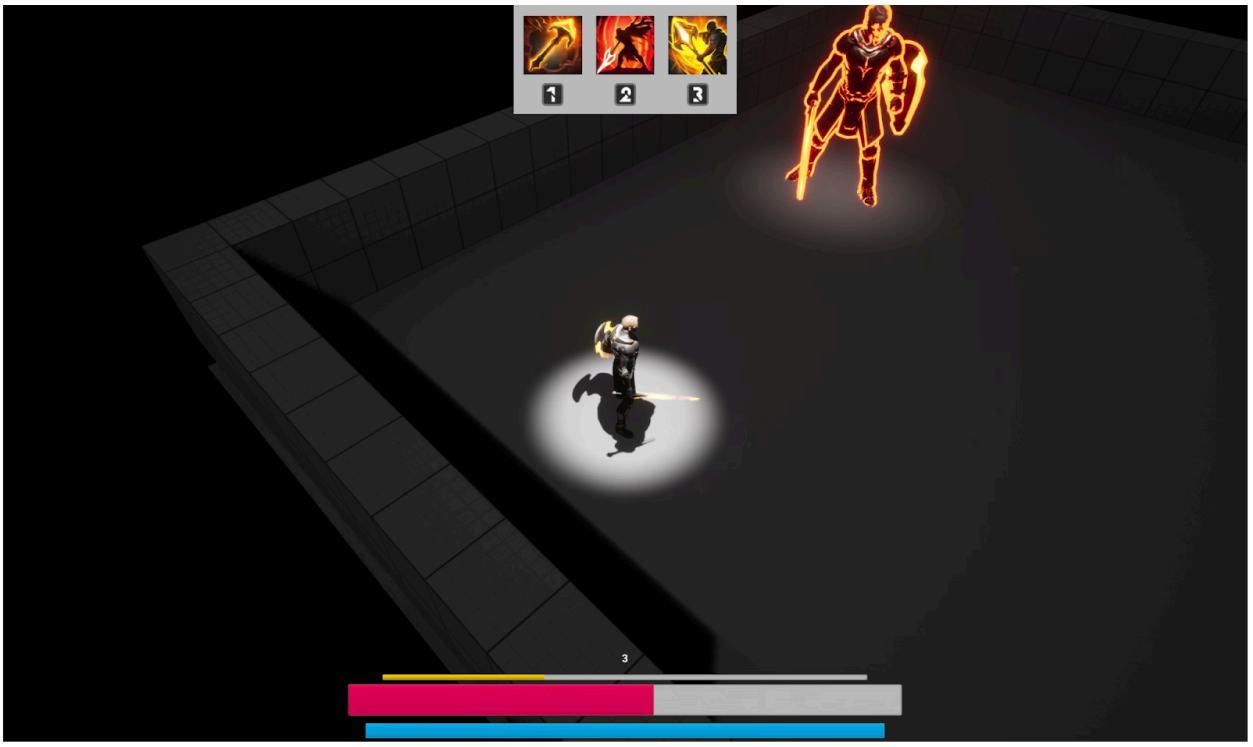


Рисунок 4.11 — Кімната боса підземелля

При вбивстві ворога персонаж отримує досвід, який використовується для отримання нових навичок, особливо багато досвіду приносить вбивство боса підземелля. Коли персонаж покращує свій рівень, він отримує одиниці прокачки навичок, що дають можливість використовувати покращені версії навичок (рис 4.12).

Вбивство боса призводить до появи порталу в стартову локацію, з якої потім потрапляємо в наступне підземелля (рис 4.13-4.14). Гра продовжується допоки персонаж гравця не помре. Після цього гравець починає гру з початку.

Рисунок 4.12 — Меню прокачки навичок

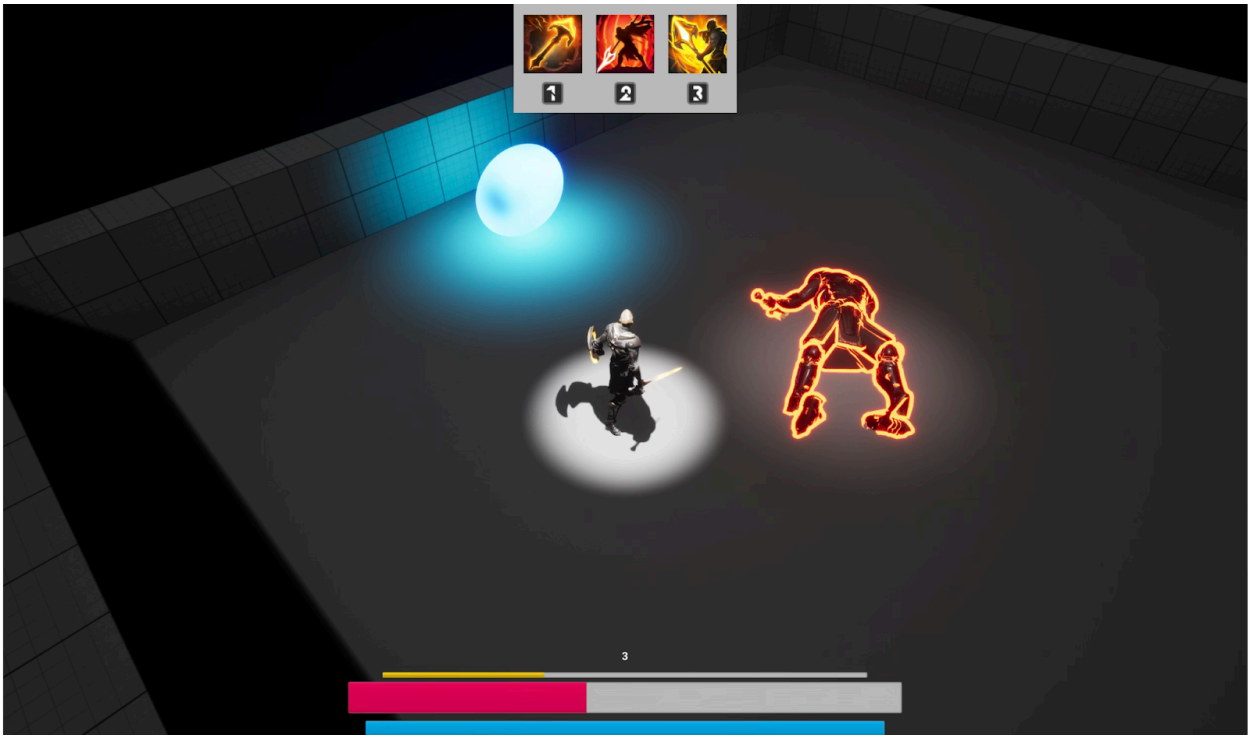


Рисунок 4.13 — Портал до стартової локації

Рисунок 4.14 — Перехід до стартової локації після підземелля

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ОДНОКОРИСТУВАЦЬКИЙ ІГРОВИЙ ЗАСТОСУНОК У ЖАНРІ ACTION RPG З ВИКОРИСТАННЯМ ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ НА РУШІЇ UNREAL ENGINE

Графічний матеріал

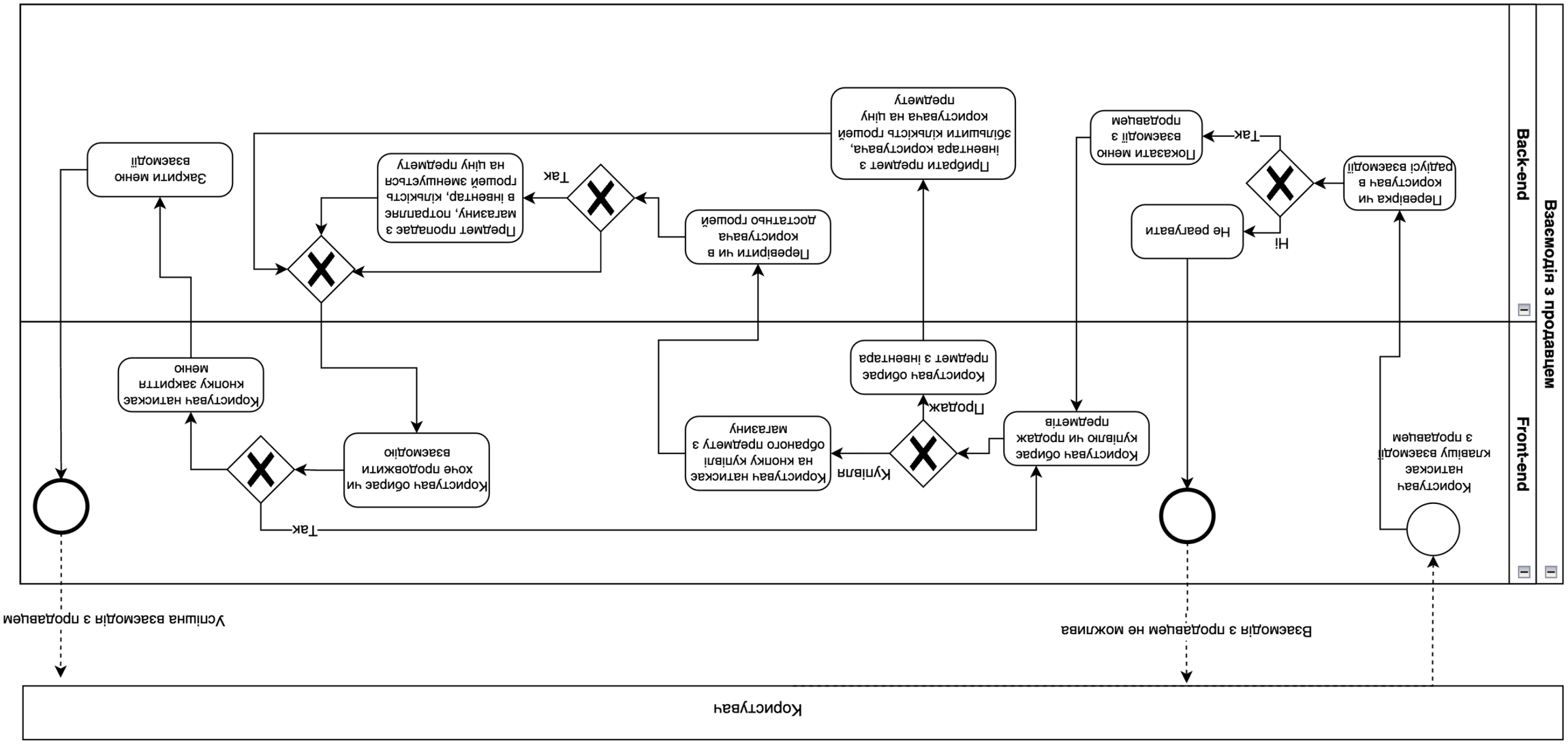
КПІ.ІТ-0222.045480.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Владислав САРНАЦЬКИЙ

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Дмитро СТЕЦУН

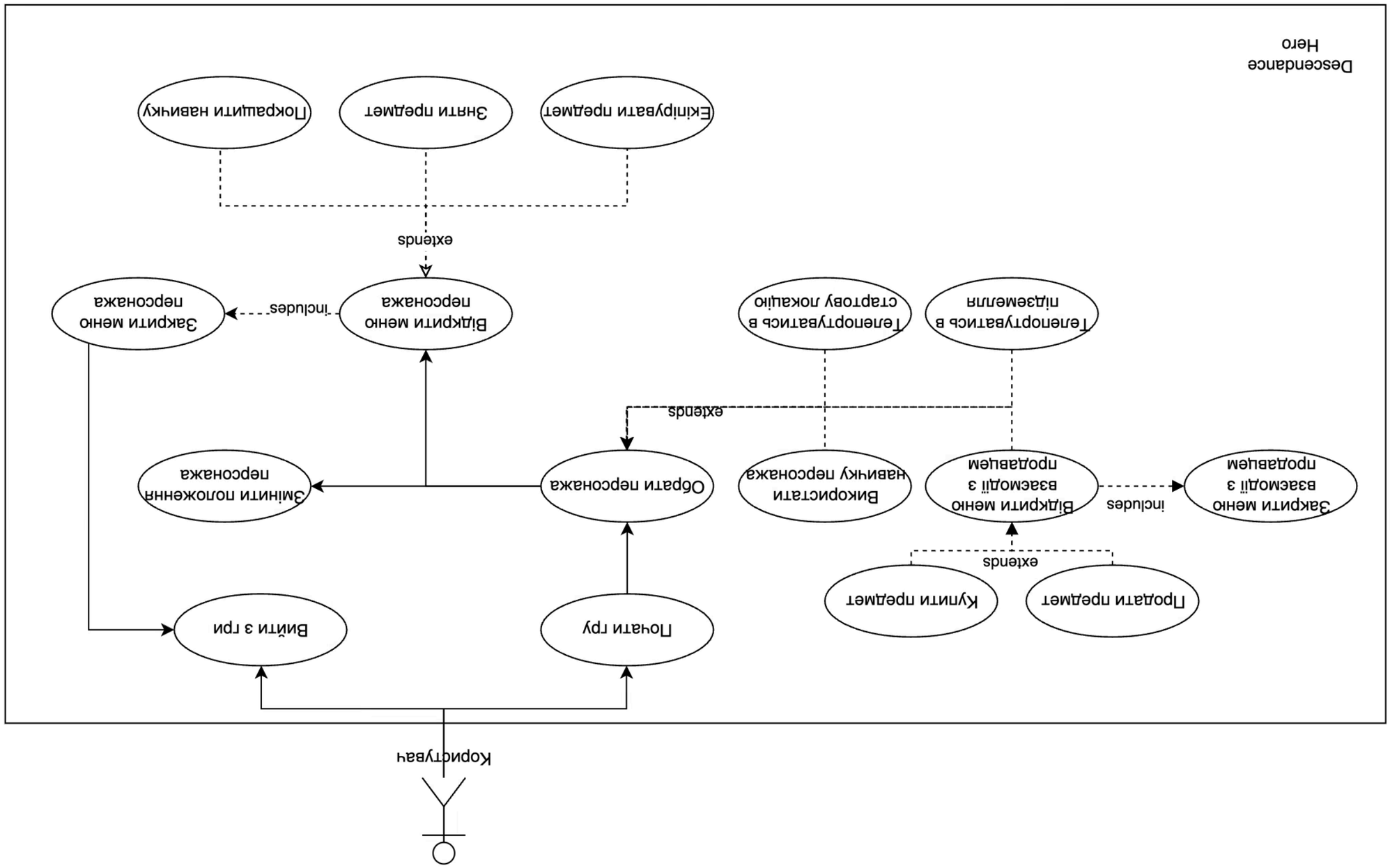
Київ – 2025



КПІ.ІТ-0222.045480.06.99.ССВ

							КПІ.ІТ-0222.045480.06.99.ССВ		
3м.	Арк.	№ документа	Підпис	Дата					
Розробив		Стецун Д. О.			Діаграма варіантів використання		Літера	Маса	Масштаб
Перевірив		Сарнацький В. В							
Т. контр.							Аркуш	Аркушів	
Н. контр.		Голованенко М. М.			ОДНОКОРИСТУВАЛЬКИЙ ІГРОВИЙ ЗАСТОСУНОК У ЖАНРІ ACTION RPG З ВИКОРИСТАННЯМ ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ НА РУШІЙ UNREAL ENGINE		КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12		
Затвердив		Жарікова Е.В.							

					КПІ.ІТ-0222.045480.06.99.ССА		
Зм.	Арк.	№ докум.	Підп.	Дата			
Розробив		Стецун Д. О.			Блок-схема алгоритму планування дій	Лит.	Маса / Масштаб
Перевірив		Сарнацький В. В					
Т. контр.						Аркуш	Аркушів
					ОДНОКОРИСТУВАЦЬКИЙ ІГРОВИЙ ЗАСТОСУНОК У ЖАНРІ ACTION RPG З ВИКОРИСТАННЯМ ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ НА РУШІЇ UNREAL ENGINE	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12	
Н. контр.		Головченко М. М.					
Затвердив		Жаріков Е.В.					



							КПІ.ІТ-0222.045480.06.99.ССВ			
Зм.	Арк.	№ документа	Підпис	Дата	Діаграма варіантів використання			Літера	Маса	Масштаб
Розробив		Стецун Д. О.								
Перевірив		Сарнацький В. В								
Т. контр.								Аркуш	Аркушів	
					ОДНОКОРИСТУВАЦЬКИЙ ІГРОВИЙ ЗАСТОСУНОК У ЖАНРІ ACTION RPG З ВИКОРИСТАННЯМ ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ НА РУШІЙ UNREAL ENGINE			КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12		
Н. контр.		Голованенко М. М.								
Затвердив		Жарікова Е.В.								

КПІ.ІТ-0222.045480.06.99.ССД