

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КІЇВСЬКИЙ 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 ЖАРИКОВ

(підпис)

(ім'я прізвище)

“ ”

2024 р.

**Дипломний проект**

**на здобуття ступеня бакалавра**

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

**спеціальності «121 Інженерія програмного забезпечення»**

на тему: Ігровий застосунок моделювання поведінки інтелектуальних  
агентів у 3D RPG з використанням ігрового рушія Unity.

Виконав студент IV курсу, групи IT-02

(шифр групи)

Терешкович Максим Олександрович

(прізвище, ім'я, по батькові)

(підпис)

Керівник доцент, к.т.н., доц., Фіногенов О.Д.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Рецензент

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Засвідчую, що у цій дипломній роботі  
немає запозичень з праць інших  
авторів без відповідних посилань.

Студент \_\_\_\_\_  
(підпис)

Київ –2024

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 ЖАРИКОВ  
(підпис) (ім'я прізвище)

“ \_\_\_\_ ” 2024 р.

**ЗАВДАННЯ  
на дипломний проект студенту**

Терешкович Максим Олександрович

(прізвище, ім'я, по батькові)

1. Тема проекту Ігровий застосунок моделювання поведінки інтелектуальних агентів у 3D RPG з використанням ігрового рушія Unity.

керівник проекту Фіногенов Олексій Дмитрович, к.т.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджено наказом по університету від «27» травня 2024 р. №2112-с

2. Термін подання студентом проекту « 17 » червня 2024 року

3. Вихідні дані до проекту: технічне завдання

---

4. Зміст пояснівальної записки

1)Передпроєктне обстеження предметної області: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі.

2) Розроблення вимог програмного забезпечення: вхідні дані, вихідні дані, опис структури бази даних.

3)Конструювання та розроблення програмного забезпечення: змістовна та математична постановка задачі, обґрутування та опис методу розв'язання.

4)Аналіз якості та тестування програмного забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів.

5)Розгортання та супровід програмного забезпечення: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу

- 1) Схема структурна варіантів використань
- 2) Схема структурна класів програмного забезпечення
- 3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «11» березня 2024 року

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	20.02.2024	
2	Аналіз існуючих методів розв'язання задачі	21.02.2024	
3	Постановка та формалізація задачі	03.03.2024	
4	Розробка інформаційного забезпечення	19.04.2024	
5	Алгоритмізація задачі	30.04.2024	
6	Обґрунтування вибору використаних технічних засобів	05.05.2024	
7	Розробка програмного забезпечення	10.05.2024	
8	Налагодження програми	14.05.2024	
9	Виконання графічних документів	20.05.2024	
10	Оформлення пояснівальної записки	29.05.2024	
11	Подання ДП на попередній захист	02.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студент

\_\_\_\_\_

(підпис)

Максим ТЕРЕШКОВИЧ

\_\_\_\_\_

(ініціали, прізвище)

Керівник

\_\_\_\_\_

(підпис)

Олексій ФІНОГЕНОВ

\_\_\_\_\_

(ініціали, прізвище)

## **АНОТАЦІЯ**

Пояснювальна записка дипломного проекту складається з чотирьох розділів, містить 40 таблиць, 95 рисунків та 15 джерел – загалом 123 сторінки.

Дипломний проект присвячений розробці ігрового застосунку та впровадженню покращеного інтелекту ворогів.

Метою розробки є розширення спектру моделей поведінки інтелектуальних агентів в ігровому застосунку 3D RPG

Предмет дослідження: Ігровий застосунок на рушії Unity та поведінка інтелектуальних агентів.

У розділі 1 розглянуто перед проєктне обстеження предметної області, що включає аналіз потреб користувачів та визначення основних функціональних можливостей майбутнього програмного забезпечення. Розділ 2 присвячений розробленню вимог до програмного забезпечення, де встановлюються технічні специфікації та критерії прийняття продукту. Розділ 3 описує конструювання та розроблення програмного забезпечення, включаючи деталі проектування архітектури, вибір технологій та методики реалізації функціоналу. Розділ 4 присвячений аналізу якості та тестуванню програмного забезпечення, де викладено процедури перевірки відповідності продукту встановленим вимогам та методи виявлення та усунення помилок. Програмне забезпечення впроваджено у розділі 5, який описує процес розгортання та методи супроводу, включаючи оновлення та виправлення помилок.

**КЛЮЧОВІ СЛОВА:** КОМП'ЮТЕРНИЙ ДОДАТОК, WINDOWS, UNITY, РПГ, ВОРОГ, МОДЕЛЬ ПОВЕДІНКА, ШТУЧНИЙ ІНТЕЛЕКТ, НІП, ІНТЕЛЕКТУАЛЬНИЙ АГЕНТ.

## **ABSTRACT**

The explanatory note of the diploma project consists of four sections, containing 40 tables, 95 figures and 15 sources – in total 119 pages.

The diploma project is dedicated to developing a gaming application and implementing improved enemy intelligence.

The development aims to expand the range of behavioural models of intelligent agents in a 3D RPG game application.

The subject of research: A game application based on the Unity engine and the behaviour of intelligent agents.

Section 1 discusses the pre-design survey of the subject area, including the analysis of user needs and the definition of the main functionalities of the future software. Section 2 is devoted to developing software requirements, where technical specifications and product acceptance criteria are established. Section 3 describes software design and development, including architecture design details, choice of technologies and methods of functionality implementation. Section 4 is devoted to quality analysis and software testing, which outlines the procedures for checking the product's compliance with the established requirements and methods for detecting and eliminating errors. The software is implemented in Chapter 5, which describes the deployment process and maintenance methods, including updates and bug fixes.

**KEYWORDS:** COMPUTER APP, WINDOWS, UNITY, RPG, ENEMY, BEHAVIOURAL MODEL, ARTIFICIAL INTELLIGENCE, NPC, INTELLIGENT AGENT.

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

\_\_\_\_\_ Едуард Жаріков

“\_\_\_” \_\_\_\_\_ 2024 р.

**ІГРОВИЙ ЗАСТОСУНОК МОДЕЛЮВАННЯ ПОВЕДІНКИ  
ІНТЕЛЕКТУАЛЬНИХ АГЕНТІВ У 3D RPG З ВИКОРИСТАННЯМ ІГРОВОГО  
РУШІЯ UNITY.**

**Технічне завдання**

КПІ.ІТ-0223.045440.01.91

“ПОГОДЖЕНО”

Керівник роботи:

\_\_\_\_\_ Олексій ФІНОГЕНОВ

Консультант:

\_\_\_\_\_ Максим ГОЛОВЧЕНКО

Виконавець:

\_\_\_\_\_ Максим ТЕРЕШКОВИЧ

Київ – 2024

## 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: .....	8
4.1.3	Вимоги до системи: .....	10
4.1.4	Додаткові вимоги: .....	10
4.2	Вимоги до надійності .....	11
4.3	Умови експлуатації .....	11
4.3.1	Вид обслуговування .....	12
4.3.2	Обслуговуючий персонал .....	12
4.4	Вимоги до складу і параметрів технічних засобів .....	12
4.5	Вимоги до інформаційної та програмної сумісності .....	13
4.5.1	Вимоги до входних даних .....	13
4.5.2	Вимоги до вихідних даних .....	13
4.5.3	Вимоги до мови розробки .....	13
4.5.4	Вимоги до середовища розробки .....	14
4.5.5	Вимоги до представленню вихідних кодів .....	14
4.6	Вимоги до маркування та пакування .....	14
4.7	Вимоги до транспортування та зберігання .....	14
4.8	Спеціальні вимоги .....	14
5	ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ .....	15
5.1	Попередній склад програмної документації .....	15
5.2	Спеціальні вимоги до програмної документації .....	15
6	СТАДІЇ І ЕТАПИ РОЗРОБКИ .....	16
7	ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ .....	17

## 1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Ігровий застосунок моделювання поведінки інтелектуальних агентів у 3D RPG з використанням ігровому рушії Unity.

Галузь застосування: Розваги та відпочинок.

Наведене технічне завдання поширюється на розробку програмного забезпечення, яке використовується для ігрового застосунку моделювання поведінки інтелектуальних агентів у 3D RPG з використанням ігрового рушія Unity та призначене для використання користувачами в повсякденному житті для поглиблення навичок, розваг та покращення морального стану широкого кола користувачів різних вікових категорій.

## **2 ПІДСТАВА ДЛЯ РОЗРОБКИ**

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

### 3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Призначення розробки даного ігрового застосунку полягає у створенні інтерактивного світу RPG (рольової гри), яке забезпечує динамічне моделювання поведінки інтелектуальних агентів. Це дозволить гравцям зануритися у віртуальний світ, де кожен персонаж володіє своєю моделлю поведінки, що реагують на дії гравця та зміни у середовищі гри.

Мета розробки: розширення спектру моделей поведінки інтелектуальних агентів в ігровому застосунку 3D RPG для урізноманітнення ігрового досвіду користувача.

## 4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### 4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

#### 4.1.1 Користувацького інтерфейсу

- Відображення стартової сторінки з основним меню та можливістю почати нову гру або продовжити стару.



Рис 4.1 – Стартова сторінка.

- Відображення сторінки створення персонажу (рис. 4.2).



Рисунок 4.2 – Сторінка вибору персонажа

- Меню для інвентарю, статистики, налаштувань, карт і тд (рис. 4.3 - 4.4).



Рисунок 4.3 – Меню інвентару



Рисунок 4.4 – Меню статистики персонажа

#### 4.1.2 Для користувача:

- 4.1.2.1 Функція 1 – Вибір персонажа;
- 4.1.2.2 Функція 2 – Збереження
- 4.1.2.3 Функція 3 – Продовження гри;
- 4.1.2.4 Функція 4 – Видалення збереженої гри;
- 4.1.2.5 Функція 5 – Початок нової гри

Після входу в ігровий світ користувач отримує такий функціонал:

##### 4.1.2.5. Керування персонажем:

- 4.1.2.5.1. Пересування персонажа.
- 4.1.2.5.2. Керування камерою.
- 4.1.2.5.3. Атака зброєю та магією.
- 4.1.2.5.4. Взаємодія з об'єктами ігрового всесвіту.

##### 4.1.2.6. Підтримка міні-карти та карти всесвіту:

- 4.1.2.6.1. Відображення приблизного рельєфу карти.
- 4.1.2.6.2. Відображення поточного місцезнаходження гравця.
- 4.1.2.6.3. Відображення ворогів.
- 4.1.2.6.4. Відображення магазинів.

4.1.2.6.5. Відображення сторонніх активностей.

4.1.2.7. Керування інвентарем:

4.1.2.7.1. Огляд власних предметів та інгредієнтів.

4.1.2.7.2. Відкриття та створення спелів та магії.

4.1.2.7.3. Переміщення спелів між інвентарем та основними слотами персонажа (1-8 слот на основному екрані).

4.1.2.7.4. Відображення назви, кількості та опису предмету або інгредієнту.

4.1.2.7.5. Вибір одного з 6 видів зброї.

4.1.2.7.6. Покращення характеристик.

4.1.2.7.7. Відображення кількості вбитих ворогів.

4.1.2.7.8. Відображення рівня гравця.

4.1.2.7.9. Відображення кількості доступних балів прокачки характеристики.

4.1.2.7.10. Перегляд поточного власного балансу ігрової валюти.

4.1.2.8. Користування магазинами:

4.1.2.8.1. Купівля предметів.

4.1.2.9. Керування екіпіровкою:

4.1.2.9.1. Вплив екіпіровкою на характеристики героя (зменшення отримуваної шкоди від ворогів).

4.1.2.10. Спілкування з NPC:

4.1.2.10.1. Початок діалогу.

4.1.2.10.2. Вибір варіанту відповіді із запропонованих.

4.1.2.10.3. Вплив обраної відповіді на подальший хід діалогу.

4.1.2.13. Перегляд власних поточних характеристик:

4.1.2.13.1. Атака.

4.1.2.13.2. Запас здоров'я.

4.1.2.13.3. Шанс критичного удару.

4.1.2.13.4. Множник критичного удару.

4.1.2.13.5. Рівень герою.

#### 4.1.3 Вимоги до системи:

- Функція 1 Забезпечити безпеку зберігання даних гравця.
- Функція 2 Забезпечити максимальну швидкодію застосунку

#### 4.1.4 Додаткові вимоги:

##### 4.1.4.1 Поведінка інтелектуальних агентів:

- 4.1.4.1.1 Вороги повинні атакувати гравця, якщо він в їх полі зору і не атакувати якщо він використовує магію невидимості.
- 4.1.4.1.2 Види ворогів зі своєю поведінкою (2-3 види).
- 4.1.4.1.3 Вороги повинні переслідувати гравця, якщо він втікає.
- 4.1.4.1.4 Вороги повинні охороняти певну територію, якщо їм це притаманно .
- 4.1.4.1.5 Якщо гравець пропав з поля зору, ворог повинен піти і оглядіти останнє місце де був помічений гравець.
- 4.1.4.1.6 Різні типи ворогів можуть сприймати гравця на різній дистанції.
- 4.1.4.1.7 Якщо вороги в групі і один з них бачить гравця, а інші ні – то і всі інші знають де гравець знаходиться.
- 4.1.4.1.8 У ворогів єшкала, яка відповідає за «страх» та «піднесення». Кожен пропущений удар від гравця збільшує шкалу страху, та зменшує «піднесення» та навпаки.
- 4.1.4.1.9 Якщо шкала «страху» перевищує «піднесення», то при втечі гравця його не переслідуватимуть.
- 4.1.4.1.10 Якщо шкала «піднесення» переважає то гравця переслідуватимуть.
- 4.1.4.1.11 Якщо шкала страху заповнена на максимум, ворог починає втечу чи переховується.

4.1.4.1.12 В групі ворогів, шкала «піднесення» має значну перевагу, так як є група ворогів, а гравець один.

4.1.4.1.13 Вороги можуть використовувати зброю та/або магію (залежить від типу ворогів) .

#### 4.1.4.2 Локалізація:

4.1.4.2.1 Гра має мати як мінімум це Англійський інтерфейс, як максимум Український.

### 4.2 Вимоги до надійності

Надійність ігрового застосунку є критично важливою, оскільки вона впливає на досвід користувача та загальне сприйняття продукту. Вимоги до надійності включають:

- Стабільність роботи: Застосунок має працювати без збоїв та випадкових зависань, навіть при тривалому використанні.
- Відновлення після збоїв: У разі виникнення помилок або збоїв, застосунок має забезпечувати швидке відновлення роботи без втрати даних користувача або зі збереженням часткової кількості даних користувача.
- Резервне копіювання: Автоматичне створення резервних копій даних гри для запобігання втраті прогресу користувача.
- Оптимізація ресурсів: Ефективне використання системних ресурсів для забезпечення плавної роботи на різноманітному обладнанні.
- Тестування: Регулярне проведення комплексного тестування для виявлення та усунення потенційних проблем перед випуском оновлень.

### 4.3 Умови експлуатації

Ігровий застосунок повинен бути розроблений з урахуванням наступних умов експлуатації:

- Температурний режим: Застосунок має працювати стабільно в температурному діапазоні від 10°C до 35°C.

#### 4.3.1 Вид обслуговування

Обслуговування ігрового застосунку включає:

- Оновлення контенту. Виправлення помилок для покращення ігрового досвіду.
- Моніторинг системи. Слідкування за стабільністю роботи застосунку та вчасне реагування на можливі проблеми.

#### 4.3.2 Обслуговуючий персонал

Для якісного обслуговування ігрового застосунку не потрібен технічний персонал.

### 4.4 Вимоги до складу і параметрів технічних засобів

Для забезпечення оптимальної роботи ігрового застосунку, технічні засоби повинні відповідати **наступним мінімальним вимогам**:

- Процесор: Intel Core i5 або AMD Ryzen 5 з частотою не менше 3.0 GHz.
- Оперативна пам'ять: Мінімум 8 GB RAM, рекомендовано 16 GB для кращої продуктивності.
- Відеокарта: NVIDIA GeForce GTX 1050 або AMD Radeon RX 560 з мінімум 4 GB відео-пам'яті.
- Місце на жорсткому дискі: Не менше 20 GB вільного місця.
- Операційна система: Windows 10/11.
- DirectX: Версія 11 або вище.

#### **Рекомендовані вимоги:**

- Процесор: Intel Core i9 14900-KS.
- Оперативна пам'ять: 128 GB RAM.
- Відеокарта: NVIDIA GeForce RTX 4090.
- Місце на жорсткому дискі: Не менше 100 GB вільного місця.
- Операційна система: Windows 11.
- DirectX: Версія 11 або вище.

**Також рекомендується наявність:**

SSD (твердотільний накопичувач) для швидшого завантаження ігор та зменшення часу відгуку системи.

Якісні аудіосистема та мікрофон для високоякісного звуку.

#### 4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем Windows.

##### 4.5.1 Вимоги до вхідних даних

Для забезпечення коректної роботи ігрового застосунку, вхідні дані повинні відповідати наступним критеріям:

- Дані користувача: Введення інформації про гравця, такої як ім'я персонажа, налаштування управління.
- Інтерактивність: Реагування на дії гравця в реальному часі, включаючи команди управління персонажем та інтерфейсом.

##### 4.5.2 Вимоги до вихідних даних

Вихідні дані ігрового застосунку повинні бути чітко структуровані та легко інтерпретовані:

- Збереження стану гри: Автоматичне збереження прогресу гравця та налаштувань.
- Результати дій гравця: Відображення змін у світі гри відповідно до виборів та дій гравця.

##### 4.5.3 Вимоги до мови розробки

Ігровий застосунок буде розроблено з використанням наступних мов програмування та технологій:

- C#: Основна мова розробки для Unity, що забезпечує гнучкість та потужні можливості для створення ігрової логіки.
- Shader Language: Для розробки власних шейдерів, що дозволяє створювати унікальні візуальні ефекти.
- UnityScript: Використання для написання скриптів, що взаємодіють з ігровим двигуном Unity.

#### 4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі Unity .

#### 4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді репозиторію з готовим прототипом або застосунку для використання користувачем.

#### 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

Зроблена інсталяційна програма встановлення гри.

## 5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

### 5.1 Попередній склад програмної документації

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

- технічне завдання;
- пояснювальна записка;
- текст програми;
- програма та методика тестування;
- керівництво користувача;

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

- схема структурна варіантів використання;
- схема структурна класів програмного забезпечення;
- креслення вигляду екранних форм;

### 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 роботи	21.02.2024	
2.	Розробка технічного завдання	03.03.2024	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04.2024	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.04.2024	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05.2024	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05.2024	Тести, результати тестування
7.	Розробка матеріалів текстової частини роботи	14.05.2024	Пояснювальна записка
8.	Розробка матеріалів графічної частини роботи	20.05.2024	Графічний матеріал проекту
9.	Оформлення технічної документації роботи	29.05.2024	Технічна документація

## **7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ**

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

## **Пояснювальна записка до дипломного проекту**

на тему: Ігровий застосунок моделювання поведінки інтелектуальних агентів  
у 3D RPG з використанням ігрового рушія Unity.

КПІ.ІТ-0223.045440.02.81

Київ – 2024

## ЗМІСТ

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ.....	6
1.1 Аналіз предметної області .....	6
1.2 Аналіз існуючих рішень.....	17
1.2.1 Аналіз відомих програмних продуктів.....	18
1.2.2 Аналіз відомих алгоритмічних та технічних рішень .....	24
1.3 Опис бізнес-процесів .....	28
1.4 Постановка задачі .....	31
Висновки до розділу .....	32
2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	33
2.1 Варіанти використання програмного забезпечення .....	33
2.2 Аналіз системних вимог.....	43
2.3 Розроблення функціональних вимог.....	44
2.4 Розроблення нефункціональних вимог.....	51
Висновки до розділу .....	51
3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ .....	53
3.1 Архітектура програмного забезпечення.....	53
3.2 Обґрунтування вибору засобів розробки .....	61
3.3 Конструювання програмного забезпечення.....	64
3.3.1 Графічне оформлення.....	64
3.3.2 Світ .....	66
3.3.3 Граве́ць.....	67
3.3.4 Вороги .....	95

3.3.5 Кінець гри .....	101
3.4 Аналіз безпеки даних.....	102
Висновки до розділу .....	102
<b>4 АНАЛІЗ ЯКОСТИ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ .....</b>	<b>103</b>
4.1 Аналіз якості ПЗ.....	103
4.2 Опис процесів тестування.....	104
4.3 Опис контрольного прикладу .....	111
Висновки до розділу .....	114
<b>5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ</b>	
116	
5.1 Розгортання програмного забезпечення.....	116
5.2 Супровід програмного забезпечення .....	118
Висновки до розділу .....	118
<b>ВИСНОВКИ.....</b>	<b>119</b>
<b>СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....</b>	<b>121</b>
<b>ДОДАТКИ.....</b>	<b>123</b>

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	– Integrated Development Environment – інтегроване середовище розробки.
API	– Application programming interface, прикладний програмний Інтерфейс
SDK	– Software development kit
IT	– Інформаційні технології
ER	– Entity-Relation diagram
Рушій (Engine)	– Спеціальний вид програм для полегшеної розробки ігор
Сиквел	– Продовження проекту, його наступна частина
NPC(НПП)	– Non-Player Character, не ігровий персонаж

## ВСТУП

У ігровому середовищі, що швидко розвивається, розробка ігор будь яких жанрів, стала процвітаючою галуззю, яка захоплює мільйони гравців у всьому світі. Існує безліч різновидів ігор, серед яких особливо виділяється жанр RPG, який пропонують захоплюючий віртуальний світи на будь який смак, де гравці можуть вирушити в епічні пригоди, перемогти легендарних ворогів, створити неймовірну зброю та бути керівником власної долі. Цей проект має на меті заглибитися в сферу розробки RPG та всіх її аспектів, дослідити актуальність, цілі, об'єкти та суб'єкти, залучені до створення цих неймовірних світів.

Актуальність розробки інтелектуальних агентів завжди висока, оскільки вона забезпечує більш реалістичну та занурюючу гру. Агенти можуть адаптуватися до дій гравців, роблячи гру більш динамічною або різноманітною. Вони також дозволяють створювати унікальні ігрові сценарії, що підвищують індивідуальний досвід кожного гравця. Інтелектуальні агенти сприяють поліпшенню стратегічної складової гри, вимагаючи від гравців більш глибокого аналізу ситуації та змушують адаптуватися до умов світу. Врешті-решт, вони вносять вклад у загальний розвиток інтелекту ворогів та його застосування. Останніми роками ми стали свідком безпрецедентного сплеску індустрії, особливо на фоні COVID-19, який підживлювався тим що, люди сидячи вдома потребували розваг і нових вражень. Таким чином попит на створення інтерактивних розваг стрімко зрос в порівнянні з минулими роками. RPG, зокрема, набули величезної популярності тим, що дають тобі можливість поринути у віртуальний світ на будь який смак, інколи навіть з друзями, стираючи межі між реальністю і вигадкою. З появою нового обладнання та покращенням старого, розробка ігор набула більших та значущих обертів – даючи змогу користувачам побувати, як у стилізованому світі «Pillars of Eternity»[1] так і у реалістичному The Witcher 3: Wild Hunt [2].

# 1 ПЕРЕДПРОЕКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Аналіз предметної області

Комп'ютерні ігри (відеоігри) – це форма інтерактивних цифрових розваг, що занурюють гравця у віртуальний світі та ставлять перед ним різні завдання[12]. Ці ігри використовують можливості комп'ютерних технологій для отримання неперевершеного ігрового досвіду.

Вони поєднують у собі інноваційні ігрові механіки, стилізовану або реалістичну графіку, якісний та чіткий звук, що створює середовище, в якому гравці можуть і хочуть досліджувати світ, виконувати квестові місії, отримувати нагороди, створювати та проходити перешкоди, вирішувати головоломки, відігравати роль «героя» або «лиходія» чи просто проживати звичайне життя персонажа другого плану.

Ігри дають велику степінь інтерактивності, що дозволяє гравцям приймати активну участь і впливати на результат, тим самим створюючи відчуття сили, свободи та індивідуальності. Від набитих екшеном пригод до стратегічних і вдумливих боїв, від задумливих наративів, що спонукають до роздумів, і симуляцій віртуальних бойових та повсякденних дій, комп'ютерні ігри не просто пропонують, а й надають різноманітний вибір ігрового досвіду, зможе задовольнити різні потреби всіх вікових категорій. Починаючи від «З в ряд» для зовсім малих дітей, закінчуєчи «драмою» або «сторітелерамом» для зовсім дорослих поціновувачів.

Розвиток ігор був каталізатором того, що процес їх створення потребував все більше і більше потужностей та кращого обладнання. Що в свою чергу рухало вперед самі технології та техніку. Наприклад якихось 20 років назад гру можна було написати в звичайній консолі одній людині без знань 3Д графіки, наративного дизайну, музики і тд. Зараз же, написати одному гру, яка зможе перевершити очікування рядового гравця майже не можливо, бо це сталий ринок зі своїми правилами та нюансами. Для цього компанії не цураються залучати купу програмістів для створення власних рушії, які б

полегшили реалізацію ігор. Також витрачають шалені кошти на покращення вже існуючих застосунків та введення нового функціоналу. Наприклад Nvidia створили графічні карти серії RTX[3], що дозволило зробити ігровий досвід набагато кращим, шляхом покращення освітлення, трасування променів і відображення картинки. Також вони додали в свої відеокарти технологію DLSS[4] – яка за допомогою штучного інтелекту домальовую в реальному часі проміжні кадри під час гри. Таким чином продуктивність вашого ПК зростає майже на 30-50 відсотків від чого ігровий досвід стає приємнішим, плавнішим і цікавішим. Ці технологічні досягнення продовжують відкривати нові горизонти, створюючи ще більш візуально приголомшливи та захоплюючі ігрові світи.

Крім того, доступність різних ігрових платформ - від консолей PS5 до ПК, смартфонів і навіть VR-пристроїв - зробило комп'ютерні ігри більш доступними і поширеними, ніж коли-небудь. Граючи в ігри сам або зі своїми товаришами, це стало основною сучасною і важливою формою розваг, яка приваблює, за статистикою, майже 3 млрд гравців у всьому світі, з яких більше половини, а саме 1.5 млрд надають перевагу іграм на ПК[5]. Така велика популярність породила компанії, профспілки, групи, сервери, спільноти і світові турніри – перетворивши це не просто на «ігри для дітей», а світову індустрію, з трильйонами доларів річного обороту, для якої не існує кордонів, меж та рамок. Будь хто може стати її частиною та спробувати привнести в неї свій вклад.

Кожна гра – це унікальний продукт своїх творців. Це сотні або тисячі годин кропіткої роботи цілої команди розробників, артистів, менеджері, режисерів і тд. Адже комп'ютерні ігри охоплюють безліч жанрів - від класичних шутерів від першого обличчя до стратегій де ти керуєш цілими націями в боротьбі за виживання. Всі вони пропонують кожному спробувати зануритися у неймовірний світ пригод та самостійно вирішити чи ти герой своєї історії, чи ти лише незначна ланка в цілій історії цього світу. Всі знайдуть щось на свій смак. Для прикладу, на думку спадає досить популярна гра за останні пів року

– Helldivers 2 [6], яка хоч і виглядає як звичайна PVE (Player versus Environment – гра при якій гравець взаємодіє більше з ігровим світом ніж з живими гравцями, яких може навіть і не бути в грі) шутер від 3 обличчя, але її особливість в тому, що вона пропонує саркастично викривлене представлення демократії та можливість взаємодіяти зі своїми товаришами на максимальному рівні, майже як в реальному житті (починаючи від того, що ти можеш банально випадково роздавити свого опонента при приземленні на місію, закінчуєчи скидуванням на нього 500 кг «демократії» (аналог місцевої ядерної бомби)). Що в наш час, нажаль, досить популярне питання в повсякденному військовому стані.

Як вже згадували до цього, розробка ігрових застосунків — це дуже динамічна галузь, що розвивається не по днях, а по годинах, роблячи все, щоб користувач отримав найкращий ігровий досвід у своєму житті і, само собою, підтримав розробників у фінансовому плані для подальшого розвитку проекту. У цьому параграфі ми заглибимося в загальні положення розробки ігор, надаючи вам вичерпний огляд предметної області.

Для початку давайте дамо чітке визначення – що таке «розробка ігор». Це динамічна сфера, яка об’єднує багатогранний підхід, креативне мислення, досвід та навички технічних спеціалістів для створення унікального візуального досвіду та інтерактивного контенту[6]. Цей процес включає в себе:

- ігровий дизайн (*сценаристи та дизайнери*), які відповідають за розробку концепції гри, правил, цілей і механік, які разом формують основу ігрового процесу, а вирішальне рішення завжди залишається за креативним директором;
- програмування (*геймплей програмісти, аудіо програмісти, технічні програмісти i тд*). Вони використовують мови програмування, такі як C++, C#, Python, або JavaScript, для реалізації ігрової механіки та інтерфейсу користувача, звуку, анімацій, та технічних аспектів рушія;
- графічний дизайн (*3Д художники по персонажам, пропсам, зброї, 2D артисти, концептери, аніматори i тд*). На них лежить створення

візуального контенту, включаючи персонажів, середовище, текстури та анімацію;

- звуковий дизайн (*саунд дизайнери*). Їх задача це розробка звукових ефектів, музики та озвучування персонажів та елементів гри, які покращують ігровий досвід. Що насправді відіграє досить значну роль, адже як тільки в грі з'являються звуку – вона зразу починає грати новими фарбами, що перевірено під час розробки та виконання даного дипломного проектування;
- тестування та гарантія якості (*QA та альфа і бета тестери від ком'юніті*). Вони виявляють та повідомляють програмістів про баги та помилки, які потребують вирішення для подальшого забезпечення стабільності та якості гри;
- маркетинг та видавництво (*самі компанії розробники або компанії видавці*). Їх задача - це розробка стратегій для просування гри та залучення аудиторії, закупівля реклами, дистрибуція, створення дисків та носіїв для збуту товару, підрахунок статистики та активна комунікація з клієнтами;
- керування проектом (*ПМ – проджект менеджер*): Координація команди, ресурсів та часу для ефективної реалізації проекту. Це стосується процесу проектування, розробки та розгортання ігор, які в першу чергу призначені для гри та використання через онлайн-платформи (в сучасному світі навіть одно-користувацькі ігри продаються та відкриваються в більшості на онлайн платформах таких, як Steam [7] або Epic Game Store [8]. Ці платформи так можуть включати веб-браузери, спеціальні ігрові клієнти, мобільні пристрої або навіть середовища віртуальної реальності);

Звісно ще є безліч ролей та спеціальностей, без яких ваша улюблена гра ніколи б не побачила цей світ, яких вистачає на 5-10 хв титри вкінці кожного великого AAA тайтлу (Alot of Money, Alot of Time, Alot of Resources – багато грошей, багато часу, багато ресурсів). Кожна з цих областей відіграє вирішальну роль у створенні успішної гри.

Ігрові механіки мають один з вирішальних впливів на розробку ігор та створення цікавого досвіду взаємодії з ними. Якщо в мобільних маленьких

іграх зазвичай 1-2 механіки (приклад: біг, підняття монет), то в ПК іграх лічильник йде на сотні. Ці механізми визначають, як гравці виконує ті чи інші дії, взаємодіє з персонажами та NPC, правила, яких вони повинні дотримуватися, і цілі, які поставлені перед ними в грі. Вам потрібно мати гарний досвід в психології, для того щоб створити таку гру, в яку людина захоче повернутися та витратити ще одну годину свого життя на ці розваги. Також історія, нагороди та вороги також потребують балансування аби не зробити ігровий досвід значно складним або досить легким. Хоча зараз все ще дуже популярні ігри жанру Souls-like – які є апогеєм найбільшої складності в іграх, де кожен рядовий ворог може тебе легко вбити а буде відчуватися, як босс рівня. Проте зараз не про них.

Наступний аспект розробки гри – це створення особливого, унікального та впізнаваного візуального стилю. Це передбачає використання передових графічних технологій, шейдерів, VFX, анімацій і тд, щоб світ відчувався динамічним та живим. Звуковий дизайн, включаючи музику та звукові ефекти, покращує загальний досвід та дає змогу відчути, що твої дії мають відгук та фідбек.

Тепер основне – те для чого створюють ігри, гроші[13]. В першу чергу це завжди про бізнес, а не про благодійність. Якщо гра не окуповує свою розробку – не буде сиквела, або навіть звичайної підтримки та фіксу багів. Тому стратегії, як монетизувати свою роботу безліч. Все залежить від типу гри та аудиторії, на яку вона спрямована. Наприклад розробники однокористувальських ігор роблять ставку на початкові продажи гри, які зазвичай мають ціну вищу, за онлайн проєкти, DLS[9] та краудфандинг. Це коли будь хто може задонатити пожертву на розробку гри ще до того як вона вийде в світ, за що на релізі отримає певні подарунки від розробника. Інші роблять ставку на внутрішні покупки, платні скіни, варіації зброї та гравців або платний Battlepass (тип монетизації проєкту) з додатковим ігровим контентом. Або ж є звичайна підписка за, яку ти стабільно отримуєш нову гру або декілька ігор раз в місяць. Тому економічний аспект відіграє, напевно, ключову роль у

створенні ігор – адже це не благодійність, а робота, яка має оплачуватися відповідно.

В останні роки, з розвитком доповненої реальності (AR) та віртуальної реальності (VR) розробка ігор поширилася і на них. Шляхом впливу безпосередньо на візуальну складову нашого мозку – вони надають нам новий неперевершений ігровий досвід. Під час цього гравець поринає у віртуальні світи повністю з головою, інколи навіть не відрізняючи реальний та ігровий світ, проте поки, лише візуально.

Кожен розробник та геймер повинен приймати активну роль та бути в курсі останніх новин, тенденцій в ігровій галузі. Адже те що технології намагаються покращити для створення ігор, потім впливає на те, що ваш ПК та система працює швидше в інших проєктах та системах, що робить ваш досвід користування технікою та технологіями кращим від роки в рік. Новинки в технологічній сфері дають нові можливості для само-покращення, навчання та досягнення нових вершин. Звідси постійне навчання, дослідження та експерименти та колаборація з іншими програмістами є життєво необхідною для створення передових ігрових світів.

Взагалі, ринок створення Ігор, як вже було зазначено дуже великий і тому стан справ у розробників не завжди знаходиться на стабільно хорошому рівні. В галузі, як і в інших сферах, присутньо багато мінусів і проблем, що перешкоджають створенню якісного контенту або його продажу.

Для початку ці основні проблеми треба кваліфікувати та зробити дослідження того, які потенційні шляхи вирішення проблем існують:

**Велика конкуренція.** Ринок налічує в собі більше 3 млрд користувачів, і кожен з них може спробувати себе в створенні ігор – в результаті ми отримуємо перенасичену індустрію, з безліччю якісного та «зробленого на колінці в підвалі» контенту, що ускладнює виділення вашого проєкту з поміж інших. Особливо коли реліз вашої гри співпав з релізом AAA проєкту від світової студії – тоді фінансові втрати вам гарантовані. Проджект Менеджери

та голови компаній повинні чітко проаналізувати ринок та конкурентів перед тим, як обирати яку гру вони будуть робити і дату її релізу.

Тривалість розробки дуже довга. Зазвичай термін створення ПК гри середньої якості займає від 1 року до 2, інколи 3 років. В свою сергу це вимагає значних фінансових вкладень та кваліфікованих спеціалістів. Також якщо ваша гра хоче претендувати на світові винагороди та визнання серед гравців – в грі мусить бути не просто гарна картинка чи цікаві механіки, а сам світ і його історія повинні бити детально розписані.

Залежність від трендів. Ця проблема виникає напряму з тривалості розробки. Компанія, банально, може не встигнути попасти в тренд і гра, механіки якої були популярні останні 2 роки, в цей рік може бути не актуальна. Тому дотримування дедлайнів дуже важливо в цій сфері.

Підвищена вразливість до критики та незадоволення очікування гравців. Медійність команди розробки, сьогодні це один з ключових етапів, адже треба показати гравця, що їх улюблений проект живий і успішно проходить всі етапи розробки. Таким чином, обіцяючи що на релізі гри в ній будуть певні механіки і персонажі, які потім не з'являються – то це підбиває гравців до знецінення виконаної роботи. В результаті будуть жахливі відгуки на ваш проект там безліч замовлень на повернення грошей. Не обіцяйте те – чого не зможете виконати.

Обмежений доступ до технології. Амбіції розробників ростуть з кожним днем. Вони все більше і більше хочуть показати гравця, в надії на те – що саме їх гру вони виберуть для купівлі. Проте апаратні ліміти, обмеженість технологій та повільна робота заліза зазвичай не дозволяє реалізувати все, що було придумано геймдизайнером. Також банально не всі гравці мають топовий ПК для отримування найкращого ігрового досвіду. Як результат – оцінюйте власні можливості раціонально та з досвідом.

Не кваліфіковані кадри та їх зміна на проекті. Швидкий розвиток технологій та досить складний процес – не дозволяє компаніям наймати Junior розробників на вакантні позиції, а для Middle рівня потрібен деякий час для

опанування документації та всіх аспектів проєкту. Тим самим, витягти людину з одного проєкту і вставити в інший не являє собою щось можливе в рамках стиснутих дедлайнів та обмеженого бюджету. Для вирішення проблеми потрібно робити комплексні навчальні плани на базі самої компанії, для створення та покращення робочих місць і спеціалістів.

Варіативність та інклузивність. Не дивлячись на те що, індустрія налічує жанри та ігри на будь який смак, цікавих, великих проєктів, які варті уваги буде не більше 10-20 для кожного жанру. Це насправді досить мала цифра при таких показниках користувачів. Тим самим, ця проблема, може відштовхнути користувачів від глибшого пізнання індустрії, бо банально вони або все вже знають, або бояться, що витраченого часу буде шкода на таку малу кількість проєктів. Компанії і розробники не повинні боятися експериментувати та привносити щось нове у їх улюблений жанр.

Відгуки користувачів та контроль якості програмного забезпечення. Ігри завжди виходять з проблемами, бо не можливо за такий короткий період часу позбутися всіх багів та протестувати гру на всі можливі сценарії. Особливо з поширенням тенденції на зменшення часу розробки, що в результаті тільки збільшує кількість вильотів гри та багів. Тому важливо безпосередньо взаємодіяти з користувачем після релізу для швидкого вирішення проблем та випуску патчів оновлення. Перед релізом розробники повинні надати ключову роль перевірці системи на помилки та поломки, щоб їх продукт був належної якості.

Для недопущення вище прописаних недоліків потрібно зрозуміти, що співпраця та командні зусилля вкрай важливі на даних проєктах. Для цього існує багато спільнот, профспілок та асоціацій, які допоможуть врегулювати будь яку проблему, нададуть свій релевантний досвід та поділяться своїми спеціалістами для досягнення поставленої мети. Не забуваймо й про фінансовий аспект – курси, гранти, додаткове фінансування завжди сприяє поліпшенню розробки ігрових проєктів та розширенню функціональних можливостей команди творців.

На додачу, середовище, яке заохочує розробників до креативності та експериментів, творчих та фінансових ризиків – в результаті може привести до створення проривного контенту в технологічній галузі або візуальній. Особливо використанні новітніх технологій, таких як AI генератори та штучний інтелект можуть пришвидшити та покращити створення ігор.

Підсумовуючи, намагаючись вирішити цілий кластер проблем пов'язаних зі створення ігрових застосунків, таких як якість, дедлайні, фінансування, інклузивність, кваліфіковані кадри, конкуренція та тривалість розробки може, сфера може розвиватися, покращуватися та надавати новий незабутній досвід гравцям.

Далі наведено існуючі жанри відеоігор.

Таблиця 1.1 - Жанрова класифікація комп'ютерних ігор

<b>Рейтинг популярності</b>	<b>Жанр</b>	<b>Опис</b>
1	Action (Екшн)	Ігри з акцентом на динамічному і швидкому геймплеї
2	Strategy	Ігри з акцентом на створенні продуманої стратегії, плануванні та прийнятті рішення
3	RPG (Role-Playing-Game)	Ігри з акцентом на відіграні, якоїсь ключової ролі для створеного світу

Продовження таблиці 1.1

<b>Рейтинг популярності</b>	<b>Жанр</b>	<b>Опис</b>
4	Simulation	Ігри, задача яких симулювати певні події, професії або завдання
5	Adventure	Ігри, де гравець повинен не просто дослідити світ, а й вирішити певні задачі і головоломки
6	Racing (Гонки)	Ігри, де фокус йде на змагання в певних перегонах між транспортними засобами
7	Sport (Спортивний симулятор)	Ігри, що відтворюють будь який вид спорту
8	Shooter (Шутер)	Ігри, де гравець стріляє з різної стрілецької зброї
9	Horror (Хорор)	Ігри, чия задача налякати гравця
10	MMO (Масова мультиплеєрна онлайн гра)	Ігри для великої кількості одночасних гравців
11	Puzzle	Ігри, які кидають виклик логічним здібностям гравця
12	Music (Музична гра)	Ігри, де гравець відіграє певний ритм або гру на інструментах
13	Інді (Indie)	Ігри, створені аматорами з малим бюджетом
14	AR/VR	Ігри з використанням доповненої або віртуальної реальності

За візуальною складовою ігри поділяються на 3 варіації:

2D – ігри, в яких є лише двовимірний простір, лише X та Y та всі елементи виконані у двомірній графіці.(рис. 1.1)



Рисунок 1.1 – WARSPEAR ONLINE

3D – ігри з трохвимірною графікою та осями координат X Y Z (рис. 1.2)

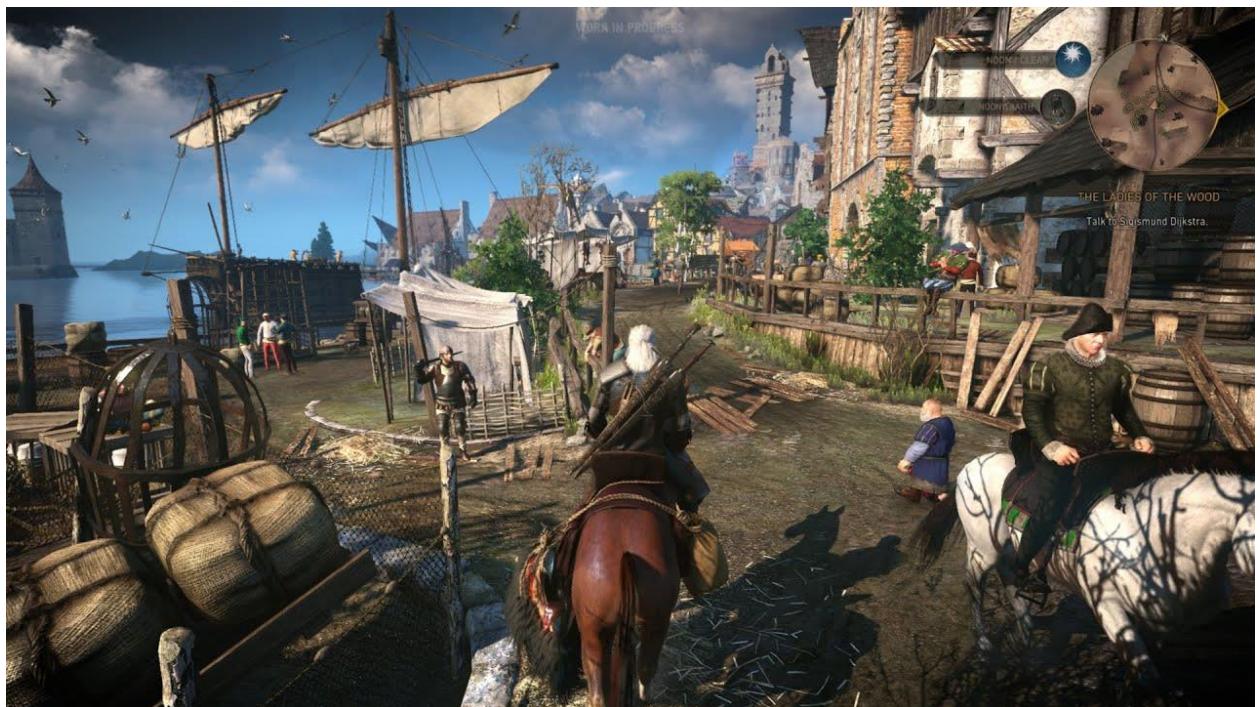


Рисунок 1.2 – The Witcher 3[2]

І останній – Текстові. Основний елемент в яких це взаємодія гравця зі світом шляхом спілкування у текстовій манері.(рис. 1.3)



Рисунок 1.3 – LIFESTREAM

В результаті ретельного аналізу самих популярних жанрів ігор, їх проблематики та візуальної складової – було прийнято рішення зупинитися на жанрі RPG з візуальною складовою 3D. Що само собою потребує високих навичок роботи не тільки з кодом, а й з цілим пакетом візуальних ефектів, персонажів, анімацій і тд. В результаті отримаємо комплексну роботу високого рівня, як для одиночного розробника.

## 1.2 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації Ігровий застосунок моделювання поведінки інтелектуальних агентів у 3D RPG з використанням ігровому рушії Unity. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

### 1.2.1 Аналіз відомих програмних продуктів

Далі будуть розглянуті аналоги 3D RPG, що існують на ринку вже, хто їх творець, на якому двигуні вони були зроблені та за який час.

The Witcher 3: Wild Hunt[2] – це комп’ютерна гра світового масштабу у стилі Action RPG, розроблена польською компанією CD Projekt RED. Гра є третьою частиною серії “Відьмак” і завершує трилогію про пригоди відьмака Геральта з Рівії. Вона поєднує нелінійний сюжет з відкритим світом, який у 30 разів більший за світ попередньої гри. Гравцям пропонується близько 50 годин основного сюжету та стільки ж для побічних квестів. (рис. 1.4)

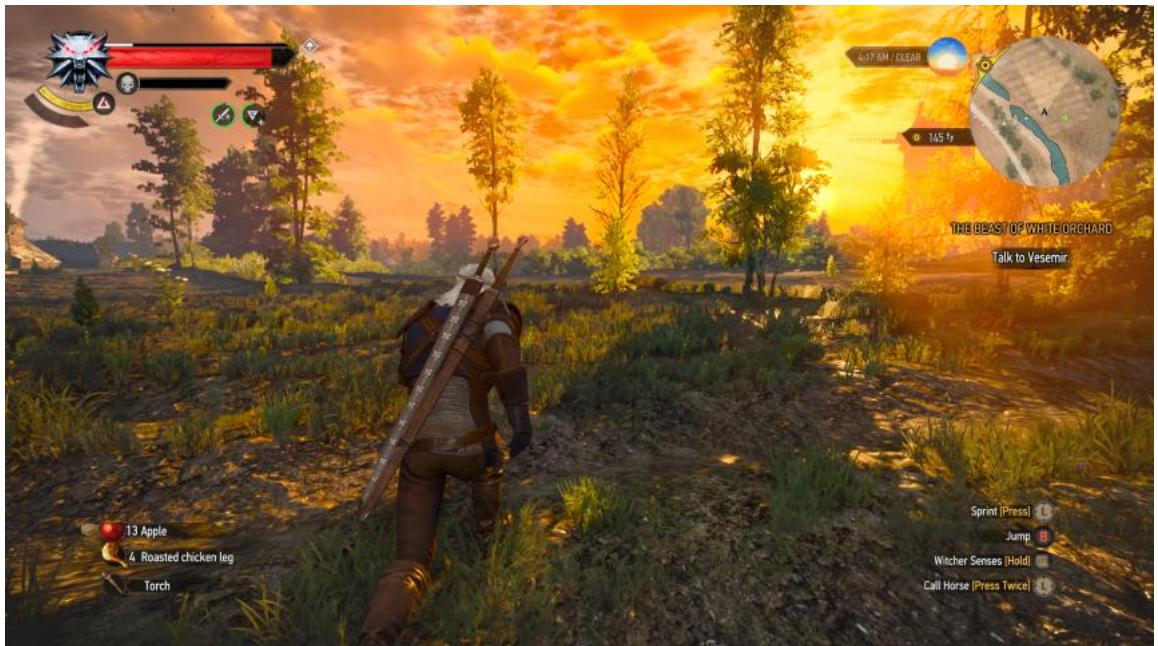


Рисунок 1.4 – The Witcher 3: Wild Hunt

Adventure War: Battlefield – це інді гра у жанрі дійової пригоди RPG, розроблена Gold Turtle Games. Вона відбувається у фентезійному світі Нур, де гравці виступають у ролі авантюристів, які допомагають королівству Ротоніс відбити напад ворожого королівства Болас. Гра має відкритий світ для дослідження та виконання квестів, а також дозволяє вибирати між різними персонажами з різними стилями гри. (рис. 1.5)



Рисунок 1.5 – Adventure War: Battlefield

RPG World - Action RPG – це гра, яка має елементи інструментарію для створення пригодницьких ігор у жанрі дійового RPG, який дозволяє користувачам без знань програмування розробляти, ділитися та грати у 3D ігри. Він пропонує готові світи для завоювання та інструменти для створення власних пригод, підтримує багатокористувацький режим, що дозволяє грати разом або проти друзів. (рис. 1.6)



Рисунок 1.6 – RPG World – Action RPG

Crimson Keep – це гра від першої особи, action RPG. Гравці занурюються в постійно змінюваний лабіринт, повний монстрів, пасток та скарбів. Вони повинні знайти затоплені руїни стародавнього замку та очистити його. Велика

сила, яку можна знайти всередині, може стати єдиною надією на втечу. (рис. 1.7)



Рисунок 1.7 – Crimson Keep

Орієнтуючись на інформацію вище – для прикладу було обрано 3 різні RPG гри. Перша – це The Witcher 3[2], на даний момент найкраща в своєму жанрі гра з культовим статусом «Легендарна», проте це не інді проєкт і над його розробкою працювало понад 300 розробників, менеджерів, артистів з бюджетом понад 80 млн \$. Друга гра, третя гра та четверта – це вже кращий варіант, бо це такий же інді проєкт над яким працювало від 2 до 10 людей. На таблиці 1.2 можна побачити порівняння кожної з кожним.

Таблиця 1.2 – Порівняння різних RPG між собою

<b>Назва</b>	<u>The Witcher 3</u>	<u>Adventure War:</u> <u>Battlefield</u>	<u>RPG World – Action RPG</u>	Crimson Keep
<b>Розробник</b>	CD Projekt Red	Gold Turtle Games	RaveyLarge	Ian Atherton
<b>Рушій</b>	REDengine 3	Unity	Unity	Unity
<b>Бюджет</b>	81 млн \$	5-10 тис \$	10-20 тис \$	5-8 тис \$
<b>Тривалість розробки</b>	3,5 років	10-12 місяців	1 рік	1 рік
<b>Кількість розробників</b>	300+	5	10	2

Всі ці ігри наповнені світами, предметами та створіннями на повну та мають безліч можливостей. Проте одне з основних аспектів що приносить гравцям новий досвід в грі – це вороги та їх поведінка.

З Відьмаком складно конкурувати, він має понад 30 різних ворогів зі своєю унікальною поведінкою. Тому краще оцінити інших ворогів з трьох наступних ігор.

У Adventure War – присутні 3 типи ворогів, кожен з них візуально відрізняється один від одного, проте їх поведінка взагалі ніяка. Вони не помічають гравця до моменту, як він з'явиться перед їх очима на відстані долоні. Навіть після цього, вони атакують але якщо гравець відійде – ворог не поженеться за тобою, він буде просто стояти і все. Дуже рідко один з них поворухнеться на метр і зупиниться. (рис. 1.8) В результаті можемо сказати, що в грі відсутні патерні поведінки персонажів.



Рисунок 1.8 – Приклад ворогів у грі Adventure War

У RPG World – є більше 10 видів ворогів проте вони також не володіють унікальними чи хоча б цікавими патернами поведінки. Вони часто не знаходять шлях до гравця, не намагаються уникнути шкоди чи викликати допомогу. Їх пошук шляху займає досить довгий час. (рис. 1.9)



Рисунок 1.9 – Усі вороги RPG World

У Crimson Keep у ворогів деяких є якась своя особливість, проте зустрічається вона досить рідко. С вороги які можуть використовувати магію в бою, або викликати допомогу. Проте всі ці вороги настільки дурні, що коли їх багато – то починають або бити один одного, або бігти не в тому напрямку і тд. (рис. 1.10)



Рисунок 1.10 – Приклад ворогів у Crimson Keep

Нижче в таблиці 1.3 можна прослідкувати наявність тих чи інших механік поведінки ворогів у перелічених проектах та зробити висновок, що інтелекту та поведінці ворогів розробники уділяли дуже мало часу і все чим вони відрізняються, це текстурою і не більше. Тому в нашій грі, було приділено трішки більше часу ворогам ніж дані проекти і привнесені певні особливості поведінки кожному з ворогів.

Таблиця 1.3 – Порівняння механік поведінки ворогів у заданих проектах

<b>Механіки поведінки ворогів</b>	<b>Echoes of Eternity</b>	<b>Adventure War: Battlefield</b>	<b>RPG World - Action RPG</b>	<b>Crimson Keep</b>
Типи ворогів	4	3	60	10
Переслідування	+	-	+	+
Система зору	+	-	-	+
Ухилення	+	-	-	-
Збереження власного життя (втеча)	+	-	-	-
Патрулювання	+	+	+	-

### 1.2.2 Аналіз відомих алгоритмічних та технічних рішень

Почнемо з того, що при створенні RPG гри, всі вороги та і сам гравець повинні якось керуватися. Для цього в Unity є чудова система під назвою NavMeshSystem. Давайте розглянемо як він працює.

Перш за все він створює навігаційну сітку NavMesh з геометрії гри під час процесу запікання. Ця NavMesh представляє собою поверхню, якою може ходити гравець або NPC до якого застосували NavMeshAgent. Проте який же алгоритм працює в цій системі. Коли ви призначаєте NavMeshAgent'у пункт призначення, він внутрішньо застосовує алгоритм пошуку найкоротшого шляху від поточної позиції агента до пункту призначення під назвою A\*, уникуючи перешкод та слідуючи за NavMesh. Ось так виглядає сітка після побудови, по якій йде рух гравця.(рис 1.11).

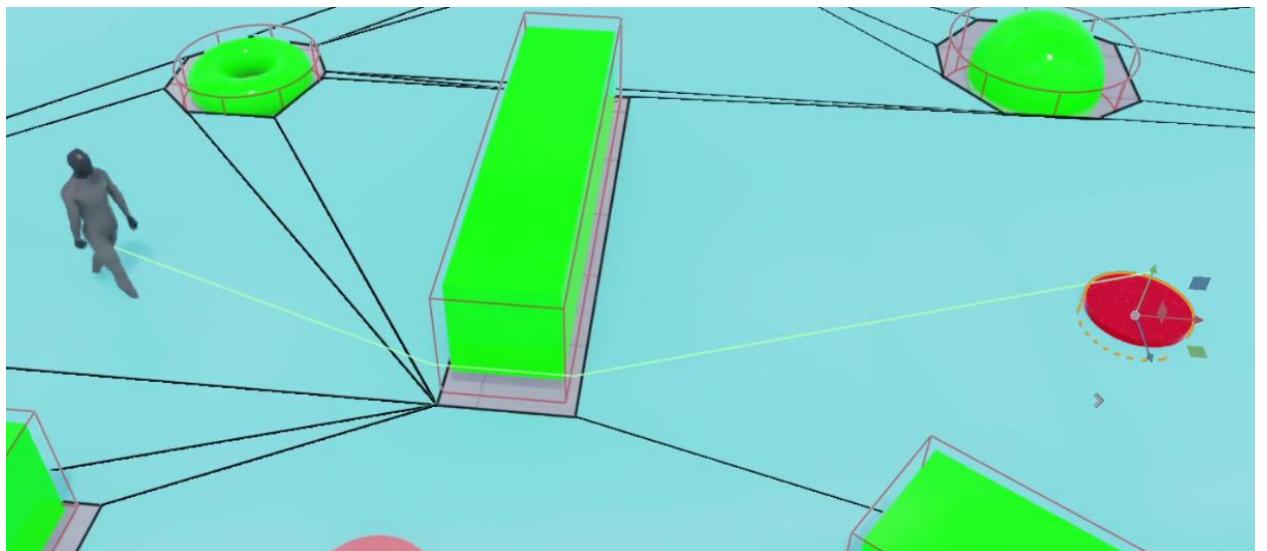


Рисунок 1.11 – Сітка Nav Mesh

Сам же алгоритм оцінює усі можливі маршрути до точки, шляхом евристики (наприклад, відстані до цілі) в поєднанні з ціною переміщення по маршруту, що робить його ефективним для пошуку оптимальних шляхів в навігаційних сценаріях.

Проте існує ще декілька варіантів алгоритмів, які використовуються в сценаріях пошуку найкоротшого шляху. Крім того, алгоритми, можуть виявляти проблемні колізії анімації чи зіткнень предметів і персонажів, що негативно може впливати на ігровий процес. Тому підхід до вибору алгоритма має важливе значення.

Отже давайте порівняємо A\* з іншими алгоритмами, такими як BFS, DFS та алгоритмом Декстри.

**BFS (Пошук в ширину).** Він досліджує всі сусідні вершини на поточній глибині, перш ніж перейти до вершин на наступному рівні дерева. Це стає гарантією того, що BFS знайде найкоротший шлях з точки зору кількості ребер проте він не враховує фактичну ціну (вартість) досягнення вершини, що може бути проблематичним у сценаріях, де різні шляхи мають різну вартість проходження.

**DFS (Пошук в глибину).** Він досліджує якомога далі вздовж кожної гілки перед тим, повернутися назад. Основні реалізації – за допомогою стеку або

рекурсивно. Однак DFS не гарантує найкоротший шлях. Він може швидко знайти рішення, якщо ціль знаходиться далеко від початку, але воно може бути не найкоротшим і не найоптимальнішим. Також для більшості ігрових сценаріїв він не актуальний, оскільки ймовірність застрягання на глибоких шляхах досить висока, перш ніж він зможе дослідити менші, можливо потенційно кращі шляхи.

**Алгоритм Дейкстри.** Працює так, що знаходить найкоротший шлях від стартової вершини до усіх інших вершин зваженого графа. Дослідження йде в порядку відстані від початкової вершини. Основна проблема в тому, що він, на відміну від A\*, не використовує евристичну при спробі пошуку найкоротшого шляху, що означає меншу ефективність в сценаріях пошуку. Також працює лише коли ребра ваги не від'ємні.

**Алгоритм A\*** – це алгоритм інформованого пошуку, який об'єднує всі переваги як BFS, так і алгоритму Дейкстри, використовуючи евристичну для керування пошуком. Його задача оцінити вузли, комбінуючи вартість досягнення вузла з початкового вузла (відома як "g-вартість") з евристичною оцінкою вартості досягнення мети з вузла (відома як "h-вартість"). Спочатку A\* досліджує шляхи з меншою сумарною вартістю, що робить його ефективним і здатним знайти найкоротший шлях, якщо евристика є прийнятною і несуперечливою. Тому A\* широко використовується для пошуку шляхів в іграх, оскільки він ефективно обробляє сценарії зі змінною вартістю шляху і забезпечує оптимальні рішення, коли використовуються відповідні евристики.

У висновку A\* це найкращий варіант в нашому виборі, бо він поєднує все найкраще з Дейкстри та Алгоритму пошуку в ширину, додаючи до цього ще й порівняльну евристичну. Деталі наявні у таблиці 1.4

Таблиця 1.4 – Порівняння алгоритмів

Алгоритм	Продуктивність	Точність	Ресурсоємність
A*	Найвищий	Найвища	Низька
Дейкстра	Середній-Високий(залежить від кількості вузлів)	Найвища	Висока
BFS	Середній	Висока	Висока
ДФС	Низький	Низька	Низька

Вище наведена таблиця порівняння основних аспектів кожного алгоритму, таких як – Продуктивність, Точність, Ресурсоємність. З результатів можна побачити та зробити висновок, що A\* був обраний правильно для подальшої і коректної роботи проекту.

Підсумовуючи, при моделюванні поведінки гравців у грі, алгоритми пошуку найкоротшого шляху відіграють життєво важливу роль у досягненні мети поставленої гравцем. Тому важливо, щоб розробники виконували порівняльний аналіз всіх існуючих варіантів вирішення проблеми, для того що реалізувати основні механіки коректно. Проте не завжди готовий варіант алгоритму може бути найоптимальнішим – інколи треба розробити свою

варіацію системи, яка врахує всі існуючи параметри та недоліки для знаходження оптимального рішення.

### 1.3 Опис бізнес-процесів

Процеси, які відбуваються в грі, детально описані за допомогою BPMN моделей.

## Процес запуску гри:

- Гравець в головному меню обирає команту «» або «» для початку або продовження вже наявної гри.
  - Відкривається меню вибора персонажа або меню вибору збережень.
  - Гравець обирає персонажа, який йому подобається, вводить ім'я або обирає збереження з обраного меню.
  - Якщо все виконано правильно – завантажується світ та починається пригода.

На рисунку 1.12 зображене процес запуску гри.

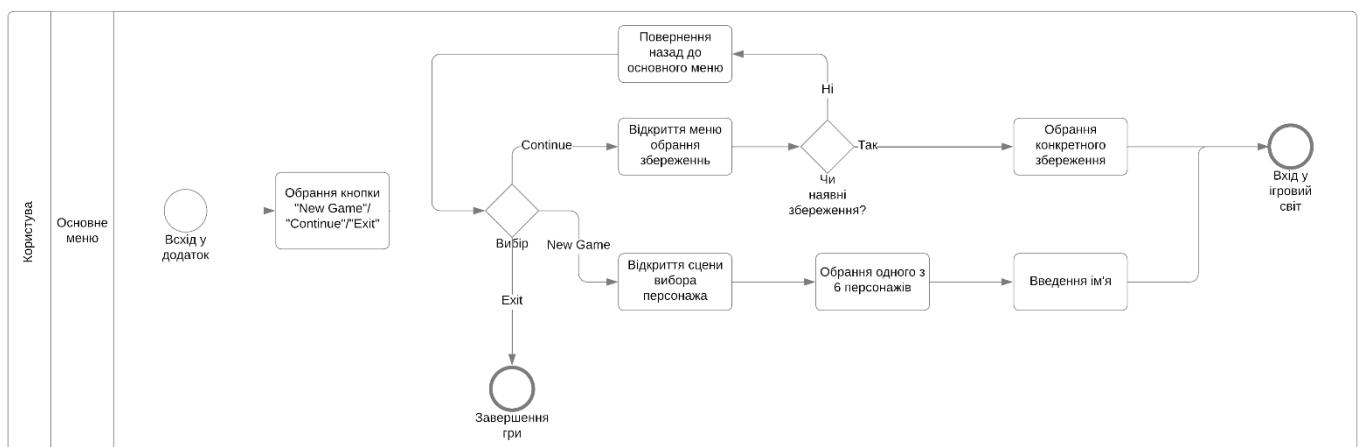


Рисунок 1.12 – Процес запуску гри

## *Процес збереження гри:*

- Гравець повинен знаходитися в ігровому світі.

- Якщо гравець намагається вийти в головне меню і гра не зберігалася останні 5 хв, гра зберігається автоматично.
- Якщо ж гравець обирає самостійно зберегти гру, він відкриває меню зупинки, обирає кнопку «Save progress» та доступну ячейку збереження куди запишеться дата та час збереження.
- Через невеличкий проміжок часу - гра зберіглась.

На рисунку 1.13 зображене процес збереження гри

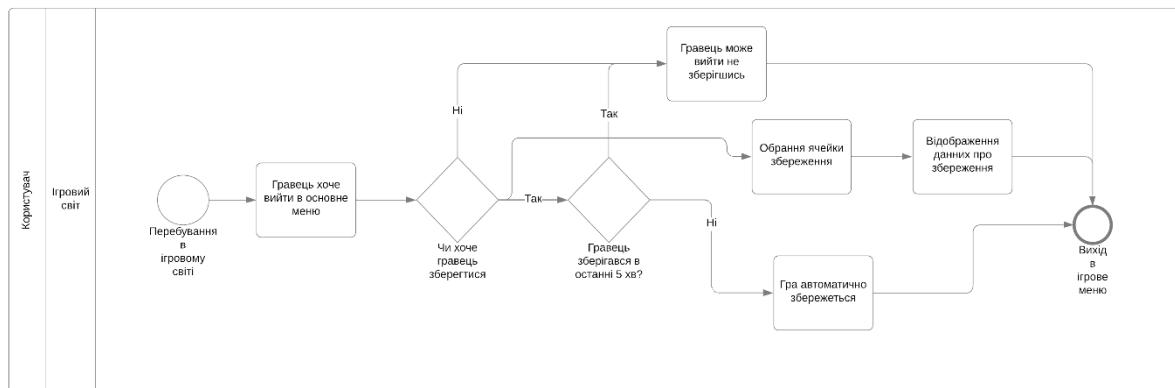


Рисунок 1.13 – Процес збереження гри

#### Процес взаємодії з ігровими об'єктами:

- Гравець хоче про взаємодіяти з певним об'єктом або отримати певний об'єкт.
- Гравець знайшов його та починає підходити до об'єкту.
- Йде класифікація який це предмет.
- Якщо це ящик – то при наближенні нічого не відбудеться. Потрібно вдарити його зброєю для його знищення. Без зброї – це неможливо.

- Якщо це ворог – гравець повинен його атакувати, що без зброї/магії неможливо, або він помре. Після знищення ворога – є 10с. для підняття монет, які випали з ворога.
- Якщо це скриня – то гравець потребує ключ для відкриття. Після чого отримає певну нагороду.
- Якщо це предмет, який лежить на землі, то він просто піднімається, тоді коли по ньому пройдеться головний герой.

- Якщо це НП, то до нього треба підійти на достатню відстань, після чого активується процес діалогу.

На рисунку 1.14 зображене процес взаємодії з ігровими об'єктами.

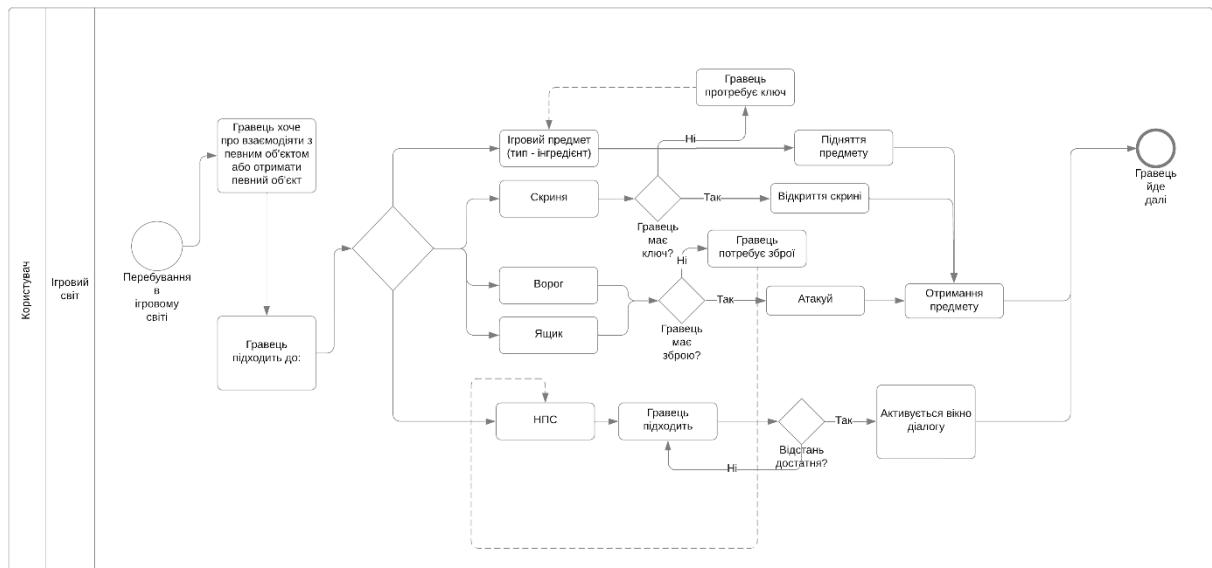


Рисунок 1.14 – Процес взаємодії з об'єктами

#### Процес створення заклинань та купівлі:

- Гравець потребує використання магії. Треба перевірити чи взагалі наявно в нас те закляття. Якщо так – обираємо ціль і використовуємо, або використовуємо на себе якщо це допоміжні закляття.
- Якщо його нема, треба його створити за допомогою книги, яку ми знайдемо в Пабі.
- Для створення заклять гравець потребує інгрідієнтів, які може знайти у світі.
- Якщо інгрідієнтів достатня кількість – створюємо необхідну магію, в іншому випадку йдемо до НП та купляємо або шукаємо у світі.

На рисунку 1.15 зображене процес створення магії та купівля інгрідієнтів.

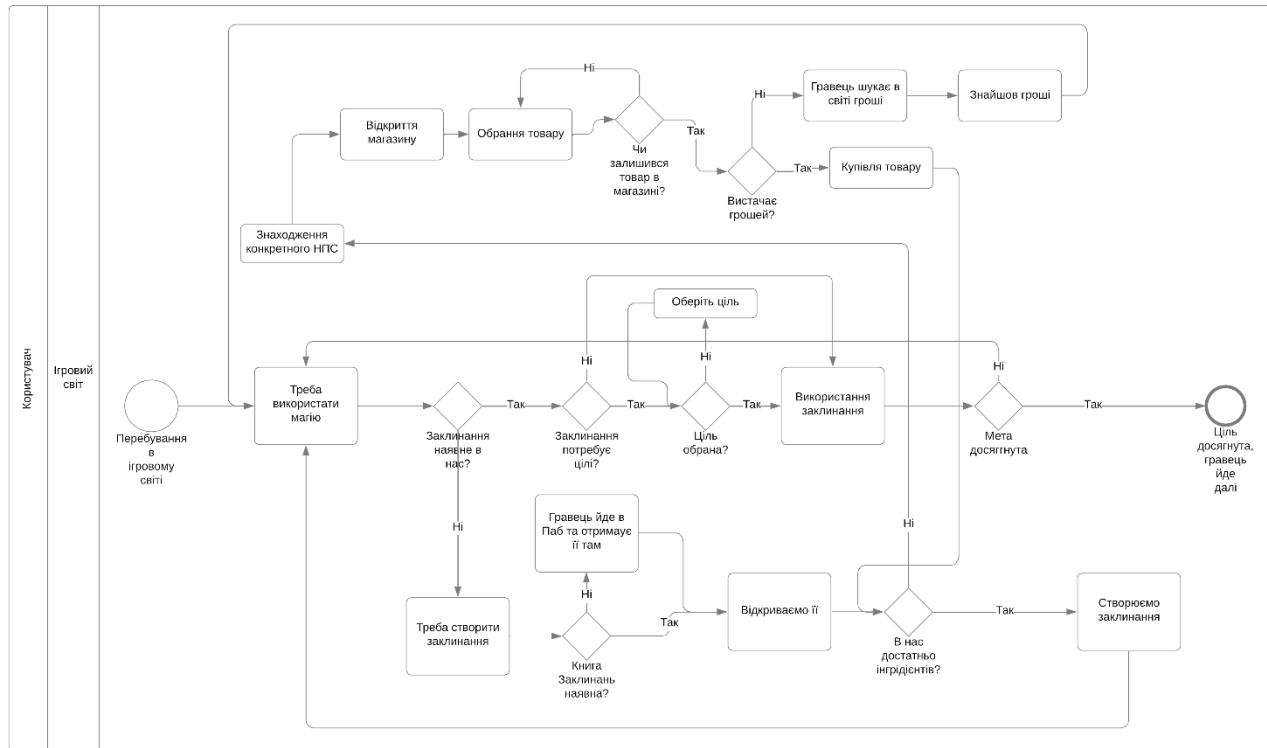


Рисунок 1.15 – Процес створення магії та купівлі

#### 1.4 Постановка задачі

Метою розробки є розширення спектру моделей поведінки інтелектуальних агентів в ігровому застосунку 3D RPG, що надасть користувачам можливість урізноманітнити ігровий досвід та проводити час за улюбленою справою, сформує нове ком’юніті навколо гри, продемонструє технології та покаже те, що навіть один розробник може створити досить цікаву гру. Таке програмне забезпечення дозволить людям, які ще не були знайомі з цією сферою – дізнатися щось нове та сформувати свою думкою про інді розробку. Виходячи з написаного вище, можна сформулювати новий список функціональних задач розробки:

- Реалізація інтелекту ворогів зі своїми особливими спектрами поведінки.
- Реалізація ігрового інтерфейсу.
- Реалізація механік бою.

- Реалізація функціоналу інвентарю, магазинів та ігрової економіки.
- Реалізація функціонали створення та використання магії.
- Реалізація функціоналу взаємодії з ігровими об'єктами.

## **Висновки до розділу**

В першому розділі ми розглянули та описали всі теоретичні аспекти розробки комп'ютерної гри, подивилися на вже існуючи варіанти, рушій та мови програмування, дізналися які алгоритми і коли краще використовувати. Підсумовуючи, розробка ігор — це завжди складна та кропітка робота, яка потребує багато зусиль, навичок кваліфікованих спеціалістів, грошей та часу, проте у фіналі ваші труди не будуть не помічені і ви завжди знайдете визнання серед однодумців і ком'юніті. Просто треба не забувати та оцінювати те, які механіки, сетінг та світи актуальні на даний момент та підтримувати зворотній зв'язок з потенційний клієнтом.

Було розглянуто можливості та функціонал Unity. Визначено основні переваги та недоліки. Це один з ключових рушій на даний момент, який складає конкуренцію навіть UE5.

## 2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### 2.1 Варіанти використання програмного забезпечення

Головною функцією програмного забезпечення є інтерактивність, іммерсивність та можливість розвивати свого персонажа, діаграма варіантів використання знаходиться в графічному метріалі, креслення 1. Також в грі гравець здатен обирати одного з 6 персонажей для гри, керувати ним, використовувати закляття та створювати їх, користуватися екіпіруванням, що покращить шанси на виживання, взаємодіяти з NPC та ворогами та багато іншого. Більше функцій можна побачити на рисунку 2.1

В таблицях 2.1 - 2.16 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-01

Use case name	Початок гри
Use case ID	UC-01
Goals	Розпочати гру
Actors	Користу-вач
Trigger	Користувач бажає розпочати гру
Pre-conditions	-
Flow of Events	Користувач відкриває додаток і бачить основне меню, натискає Нова гра
Extension	Якщо у користувача проблеми з пк, програма може не запуститися
Post-Condition	Користувач може обрати чи обрати новго персонажа, чи вийти з гри, чи продовжити вже наявну гру.

Таблиця 2.2 – Варіант використання UC-02

Use case name	Вихід з гри
Use case ID	UC-02
Goals	Закінчити гру
Actors	Користувач
Trigger	Користувач бажає завершити ігровий процес
Pre-conditions	Користувач почав (увійшов) в гру, Збереження гри,
Flow of Events	В меню початку гри є кнопка завершення гри
Extension	-
Post-Condition	Гра закривається

Таблиця 2.3 – Варіант використання UC-03

Use case name	Вибір персонажа
Use case ID	UC-03
Goals	Обрати персонажа для входження в світ
Actors	Користувач
Trigger	Користувач бажає обрати ігрового персонажа
Pre-conditions	Користувач почав (увійшов) в гру
Flow of Events	Після натискання клавіші (Нова Гра) відкривається сцена для обирання одного з 6 персонажів
Extension	Якщо користувач не натисне «Нова гра» нічого не станеться
Post-Condition	Вхід до ігрового світу

Таблиця 2.4 – Варіант використання UC-04

Use case name	Продовження гри
Use case ID	UC-04
Goals	Продовжити попередню збережену гру
Actors	Користувач
Trigger	Користувач бажає продовжити попередню гру
Pre-conditions	Користувач почав (увійшов) в гру і наявне збереження
Flow of Events	В меню початку гри є кнопка «Продовження» гри
Extension	Кнопка не активна, якщо немає збережень
Post-Condition	Гра починається

Таблиця 2.5 – Варіант використання UC-05

Use case name	Вхід до ігрового світу
Use case ID	UC-05
Goals	Вхід і розпочаток пригод
Actors	Користувач
Trigger	Користувач бажає розпочати свої пригоди
Pre-conditions	Користувач обрав персонажа або продовжив гру з місця збереження
Flow of Events	При виборі персонажа є кнопка «Accept» яка приймає вибір гравця і впускає його у ігровий світ
Extension	Поки гравець не обере персонажа – його не пустять у світ
Post-Condition	Зберегти гру, використання екіпірування, керування камерою та її обертанням, пересування персонажем;

Таблиця 2.6 – Варіант використання UC-06

Use case name	Збереження гри
Use case ID	UC-06
Goals	Зберегти прогрес гри
Actors	Користувач
Trigger	Користувач бажає зберегти свій прогрес
Pre-conditions	Вхід у ігровий світ
Flow of Events	При завершенні гри вона автоматично збережеться або можна буде зберегти шляхом натискання «Save»
Extension	Поки гравець не потрапить у світ, збереження неможливе
Post-Condition	Вихід з гри, Продовження гри, Початок нової гри

Таблиця 2.7 – Варіант використання UC-07

Use case name	Пересування персонажем
Use case ID	UC-07
Goals	Дослідження ігрового всесвіту
Actors	Користувач
Trigger	Користувач бажає підійти в конкретне місце
Pre-conditions	Користувач увійшов до ігрового світу
Flow of Events	При натисканні на територію під гравцем, він буде рухатися в місце натисканні

### Продовження таблиці 2.8

Extension	Якщо користувач не увійшов у ігровий світ, дана дія не буде доступна
Post-Condition	Взаємодія з НІПЙ

Таблиця 2.8 – Варіант використання UC-08

Use case name	Підтримка візуалізації міні карти
Use case ID	UC-08
Goals	Знати про розташування ворогі
Actors	Користувач
Trigger	Користувач бажає дізнатися де вороги і де він сам
Pre-conditions	Користувач увійшов до ігрового світу
Flow of Events	Праворуч у кутку буде доступна міні карта з відображенням поточного місця гравця та ворогів, те саме але в більшому маштабі буде доступно в меню «Мап» інвентару
Extension	Якщо користувач не увійшов у ігровий світ, дана дія не буде доступна
Post-Condition	-

Таблиця 2.9 – Варіант використання UC-09

Use case name	Керування камерою та її обертанням
Use case ID	UC-09
Goals	Краще розгледіти навколоішнє середовище
Actors	Користувач

### Продовження таблиці 2.9

Trigger	Користувач бажає краще розгледіти об'єкт
Pre-conditions	Користувач увійшов до ігрового світу
Flow of Events	В гравця є дві камери, одна Вільна інша Статична. При натисканні клавіші С – камери змінюються. При активній Вільній камері, можна обертати її навколо гравця правою кнопкою мишки
Extension	Якщо користувач не увійшов у ігровий світ, дана дія не буде доступна
Post-Condition	-

Таблиця 2.10 – Варіант використання UC-10

Use case name	Використання спелів та заклять
Use case ID	UC-10
Goals	Нанесення додаткової шкоди супротивнику
Actors	Користувач
Trigger	Користувач бажає нанести шкоду ворогам
Pre-conditions	Створення спелів та заклять
Flow of Events	Треба обрати створени спел чи закляття і помістити його в активну ячейку основного меню
Extension	Якщо користувач не створив закляття, він нічого не використає
Post-Condition	-

Таблиця 2.11 – Варіант використання UC-11

Use case name	Взаємодія з НП
Use case ID	UC-11
Goals	Пізнання світу та відкриття магазину
Actors	Користувач
Trigger	Користувач бажає взаємодіяти з НП або щось купити
Pre-conditions	Користувач підійшов до НП
Flow of Events	При достатній відстані до НП з'явиться автоматично діалогове вікно з запропонованими варіантами відповіді.
Extension	Якщо користувач не підійде на потрібну відстань, діалог не почнеться
Post-Condition	Користування різними магазинами

Таблиця 2.12 – Варіант використання UC-12

Use case name	Користування різними магазинами
Use case ID	UC-12
Goals	Придбання нового екіпірування, поповнення здоров'я та інгредієнтів
Actors	Користувач
Trigger	Користувач бажає отримати нові обладунки, інгредієнти та зброю
Pre-conditions	Користувач про взаємодіяв з НП
Flow of Events	При взаємодії з НП є варіант відкрити їх унікальний магазин

Продовження таблиці 2.12

Extension	Якщо користувач не взаємодіє з НІП – магазин не відкрити
Post-Condition	Створення спелів та заклять; Використання Екіпіруванням

Таблиця 2.13 – Варіант використання UC-13

Use case name	Створення спелів та заклять
Use case ID	UC-13
Goals	Нанесення додаткової шкоди ворогам та полегшення життя гравцю
Actors	Користувач
Trigger	Користувач бажає створити нові закляття
Pre-conditions	Користувач увійшов до ігрового світу
Flow of Events	Для створення спелів і заклять треба спочатку відкрити книгу заклинань в таверні, і перемогти всіх суперників в міні данжі для відкриття піктограми
Extension	Якщо користувач не закупився або не знайшов інгрідієнти, він нічого не створить
Post-Condition	Використання спелів

Таблиця 2.14 – Варіант використання UC-14

Use case name	Використання Екіпіруванням
Use case ID	UC-14
Goals	Зміна зброї або покращення можливостей виживання
Actors	Користувач

### Продовження таблиці 2.14

Trigger	Користувач бажає зменшити отримувану шкоду та екіпірувати кращу зброю
Pre-conditions	Користувач увійшов до ігрового світу
Flow of Events	При купівлі броні вона автоматично одягнеться, а зброю завжди можна змінити у вкладці «Stats»
Extension	Якщо користувач не увійшов у ігровий світ та не купив потрібні предмети, дана дія не буде доступна
Post-Condition	-

Таблиця 2.15 – Варіант використання UC-15

Use case name	Вбивство ворогів
Use case ID	UC-15
Goals	Отримання винагороди та нового рівня персонажа
Actors	Користувач
Trigger	Користувач бажає отримати додаткове золото
Pre-conditions	Користувач увійшов до ігрового світу
Flow of Events	Підходиш до ворога і натискаєш кнопку атаки (Z)
Extension	Якщо користувач не увійшов у ігровий світ та не має зброї або заклять – ця дія не можлива
Post-Condition	Прокачування характеристик

Таблиця 2.16 – Варіант використання UC-16

Use case name	Прокачування характеристик
Use case ID	UC-16
Goals	Покращення характеристик персонажа з метою легшого проходження гри
Actors	Користувач
Trigger	Користувач бажає покращити характеристики персонажа
Pre-conditions	Користувач вбив достатню кількість ворогів та отримав новий рівень
Flow of Events	В меню інвентаря у вкладці «Stats» біля характеристик з'явиться «+» що означає, що гравець може покращити характеристику на вибір.
Extension	Якщо у користувача немає вільних «S/P» - пойнти для прокачки, які даються за отримання нового рівня.
Post-Condition	-

## 2.2 Аналіз системних вимог

Crimson Keep	
СИСТЕМНІ ВИМОГИ	
<b>МІНІМАЛЬНІ:</b>	<b>РЕКОМЕНДОВАНІ:</b>
Потребує 64-бітних процесора та операційної системи	Потребує 64-бітних процесора та операційної системи
ОС *: Windows 7 or later	
Операцівна пам'ять: 2 GB ОП	
Місце на диску: 3 GB доступного місця	
* З 1 січня 2024 року клієнт Steam буде підтримувати лише Windows 10 чи новіші версії цієї ОС.	
Adventure	
СИСТЕМНІ ВИМОГИ	
<b>МІНІМАЛЬНІ:</b>	<b>РЕКОМЕНДОВАНІ:</b>
Потребує 64-бітних процесора та операційної системи	Потребує 64-бітних процесора та операційної системи
ОС *: Windows 7 (64 bits)	ОС: Windows 10 (64 bits)
Процесор: Intel Core i5-3450 / AMD FX-6300 X6	Процесор: Intel Core i7-7700K / Ryzen 5 1600
Операцівна пам'ять: 8 GB ОП	Операцівна пам'ять: 16 GB ОП
Відеокарта: GeForce GTX 660 / Radeon HD 7870	Відеокарта: GeForce GTX 1060 / Radeon RX 470
DirectX: версії 10	DirectX: версії 11
Місце на диску: 2 GB доступного місця	Місце на диску: 2 GB доступного місця
RPG	
СИСТЕМНІ ВИМОГИ	
<b>МІНІМАЛЬНІ:</b>	<b>РЕКОМЕНДОВАНІ:</b>
ОС: 7, 8, 10 - (64 Bit)	ОС: 7, 8, 10 - (64 Bit)
Процесор: Intel Dual-Core 2GHz or AMD Dual-Core 2GHz	Процесор: Intel Quad-Core (i5 2300) or AMD Octo-Core (FX 8150)
Операцівна пам'ять: 2 GB ОП	Операцівна пам'ять: 2 GB ОП
Відеокарта: DirectX11 Shader Model 5 capable GPU	Відеокарта: DirectX11 Shader Model 5 capable GPU
DirectX: версії 11	DirectX: версії 11
Місце на диску: 2 GB доступного місця	Місце на диску: 2 GB доступного місця
Звукова карта: DirectX11 Compatible Sound Card with latest drivers	Звукова карта: DirectX11 Compatible Sound Card with latest drivers

Рисунок 2.1 – Мінімальні та рекомендовані параметри для аналогічних ігор

На рисунку 2.1 можна побачити, які системні вимоги зазвичай висувають видавці та розробники до своїх ігрових проектів. Вся інформація була зібрана з платформ, де ігри були випущені – наприклад Steam. Тому в таблиці 2.17 можна побачити мінімальні системні вимоги, базуючись на інформації зібраний до цього.

Таблиця 2.17 – Опис мінімальних системних вимог системних вимог

Назва вимоги	Мінімальні характеристики
Процесор	Intel Core i3
Об’єм ОЗП	2 GB RAM.
Швидкість підключення до мережі Інтернет	Не потрібна
Відеокарта	NVIDIA GeForce GTX 1060.
Жорсткий диск	SSD 1 GB
Операційна система	Windows

У таблиці 2.18 описані рекомендовані системні вимоги для стабільної роботи застосунку, базуючись на тому, які характеристики мало апаратне забезпечення, на якому виконувалася розробка проекту.

Таблиця 2.18– Опис рекомендованих системних вимог системних вимог

Назва вимоги	Рекомендовані характеристики
Процесор	Intel Core i9-13900K
Об’єм ОЗП	96 GB RAM.
Швидкість підключення до мережі Інтернет	Не потрібна
Відеокарта	NVIDIA GeForce RTX 4090 Suprim Liquid X.
Жорсткий диск	SSD 2 TB
Операційна система	Windows

### 2.3 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.19 наведено загальну модель вимог, а в

таблиці 2.20 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.2.

Таблиця 2.19 – Загальна модель вимог

№	Назва	ID вимоги	Пріоритети
1	Користувацький інтерфейс	FR-1	Високий
2	Початок гри	FR-2	Високий
2.1	Вибір персонажа	FR-3	Середній
2.2	Продовження гри	FR-4	Високий
2.3	Збереження гри	FR-5	Високий
2.4	Видалення збереженої гри	FR-6	Середній
3	Пересування персонажа	FR-7	Високий
3.1	Керування камерою	FR-8	Середній
3.2	Атака зброєю та магією	FR-9	Високий
3.3	Взаємодія з об'єктами ігрового всесвіту.	FR-10	Високий
4	Підтримка міні-карти та карти всесвіту	FR-11	Середній
4.1	Відображення приблизного рельєфу карти	FR-12	Середній
4.2	Відображення поточного місцезнаходження гравця	FR-13	Середній
4.3	Відображення ворогів	FR-14	Середній
4.4	Відображення магазинів.	FR-15	Середній
4.5	Відображення сторонніх активностей	FR-16	Середній
5	Огляд власних предметів та інгредієнтів	FR-17	Високий
5.1	Відкриття та створення спелів та магії	FR-18	Високий

Продовження таблиці 2.19

5.2	Переміщення спелів між інвентарем та основними слотами персонажа (1-8 слот на основному екрані)	FR-19	Високий
5.3	Відображення назви, кількості та опису предмету або інгредієнту.	FR-20	Високий
5.4	Вибір одного з 6 видів зброї.	FR-21	Високий
5.5	Покращення характеристик	FR-22	Високий
5.6	Відображення кількості вбитих ворогів	FR-23	
5.7	Відображення рівня гравця	FR-24	Високий
5.8	Відображення кількості доступних поинтів прокачки характеристики.	FR-25	Високий
5.9	Перегляд поточного власного балансу ігрової валюти.	FR-26	Високий
6	Купівля предметів	FR-27	Високий
7	Вплив екіпіровкою на характеристики героя	FR-28	Високий
8	Спілкування з NPC	FR-29	Середній
8.1	Початок діалогу	FR-30	Середній
8.2	Вибір варіанту відповіді із запропонованих	FR-31	Середній
8.3	Вплив обраної відповіді на подальший хід діалогу	FR-32	Середній
9	Перегляд власних поточних характеристик	FR-33	Високий
10	Різна поведінка інтелектуальних агентів	FR-34	Високий

Таблиця 2.20 – Перелік функціональних вимог

ID ВИМОГИ	Назва та опис
FR-1	Користувачький інтерфейс. В грі повинен бути інтерфейс для взаємодії користувача з грою, такий як вибір персонажа. Початок гри, інвентар і тд.
FR-2	Початок гри При запуску гри повинна бути можливість розпочати гру шляхом натискання кнопки «New Game» або «Continue»
FR-3	Вибір персонажа Користувачі повинні мати можливість обрати для гри одного з 6 персонажів
FR-4	Продовження гри Користувачі повинні мати можливість продовжити грати в збережену стару гру
FR-5	Збереження гри Користувач повинен мати можливість зберегти гру, в той момент коли йому захочеться
FR-6	Видалення збереженої гри Користувач повинен мати можливість видалити не потрібне йому збереження гри
FR-7	Пересування персонажа Ігровий персонаж повинен вільно керуватися у будь яку точку карти, якщо вона в межах досяжності.
FR-8	Керування камерою Користувачам повинна бути надана можливість керувати камерою та змінювати її кут при бажанні.

### Продовження таблиці 2.20

FR-9	Атака зброєю та магією В грі повинна бути присутня можливість наносити шкоду супротивникам за допомогою магії та зброї.
FR-10	Взаємодія з об'єктами ігрового всесвіту Користувачі повинні мати змогу збирати об'єкти ігрового всесвіту або отримувати їх у вигляді винагороди за певні дії.
FR-11	Підтримка міні-карти та карти всесвіту В грі повинна бути присутня карта світу, та міні карта.
FR-12	Відображення приблизного рельєфу карти На карті повинно відображатися приблизний вигляд світу зверху.
FR-13	Відображення поточного місцезнаходження гравця На карті повинно позначатися сам гравець.
FR-14	Відображення ворогів На карті повинні відображатися вороги.
FR-15	Відображення магазинів На карті повинні відображатися доступні магазини.
FR-16	Відображення сторонніх активностей Якщо присутні сторонні активності, вони повинні позначатися також на карті
FR-17	Огляд власних предметів та інгредієнтів Користувач повинен оглянути власний інвентар
FR-18	Відкриття та створення спелів та магії Користувач повинен мати можливість створити заклинання та магію.

### Продовження таблиці 2.20

FR-19	Переміщення спелів між інвентарем та основними слотами персонажа  Створені закляття повинні мати можливість бути переміщеними на основний UI слот персонажа
FR-20	Відображення назви, кількості та опису предмету або інгредієнту  Користувач має мати можливість дізнатися інформацію про предмети в інвентарі, їх кількість та найменування.
FR-21	Вибір одного з 6 видів зброї  Користувач повинен мати можливість використовувати один з 6 видів зброї на вибір та легко його змінювати у інвентарі.
FR-22	Покращення характеристик  Користувач повинен мати можливість покращувати характеристики персонажа з кожним новим рівнем.
FR-23	Відображення кількості вбитих ворогів  В меню інвентару повинно показувати скільки ворогів було знищено гравцем
FR-24	Відображення рівня гравця  В меню інвентару повинно показувати рівень персонажа.
FR-25	Відображення кількості доступних поинтів прокачки характеристики  В меню інвентару повинно показувати кількість доступних бали прокачки
FR-26	Перегляд поточного власного балансу ігрової валюти  В меню інвентару повинно показувати кількість доступної валюти

Кінець таблиці 2.20

FR-27	<p>Купівля предметів</p> <p>Користувач повинен мати можливість купити товар в магазині НП за ігрову валюту</p>
FR-28	<p>Вплив екіпіровкою на характеристики героя</p> <p>Броня повинна зменшувати отримувану шкоду (Легка – на 20%, Важка – на 40% менше)</p>
FR-29	<p>Спілкування з NPC</p> <p>Користувач повинен мати можливість спілкуватися з НП</p>
FR-30	<p>Початок діалогу</p> <p>Коли користувач підходить на достатню відстань – повинно відкриватися діалогове вікно з НП</p>
FR-31	<p>Вибір варіанту відповіді із запропонованих</p> <p>У варіантах діалогу повинно бути хоча б 2 варіанти відповіді чи питань</p>
FR-32	<p>Вплив обраної відповіді на подальший хід діалогу</p> <p>Обравши другий варіант, повинен відкриватися магазин, а при першому просто йде коротке оповідання навколошньої ситуації.</p>
FR-33	<p>Перегляд власних поточних характеристик</p> <p>Користувач повинен мати можливість оцінити наявну кількість ігрових характеристик його персонажа</p>
FR-34	<p>Різна поведінка інтелектуальних агентів</p> <p>Вороги повинні вести себе по різному відносно один одного, мати різні характеристики, рівень шкоди, патерни поведінки та цілі існування.</p>

	FR-01	FR-02	FR-03	FR-04	FR-05	FR-06	FR-07	FR-08	FR-09	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17	FR-18	FR-19	FR-20	FR-21	FR-22	FR-23	FR-24	FR-25	FR-26	FR-27	FR-28	FR-29	FR-30	FR-31	FR-32	FR-33	FR-34
UC-01	+	+			+																													
UC-02		+																																
UC-03	+		+																+															
UC-04	+	+	+	+	+													+																
UC-05				+	+	+												+																
UC-06	+																																	
UC-07	+	+	+																															
UC-08	+																																	
UC-09																																		
UC-10																																		
UC-11	+																																	
UC-12																																		
UC-13																																		
UC-14	+																																	
UC-15																																		
UC-16	+																																	

Рисунок 2.2 – Матриця трасування вимог

## 2.4 Розроблення нефункціональних вимог

Нефункціональні вимоги для RPG гри визначають якість і спосіб реалізації гри, а не конкретні її функції. Ось деякі з основних нефункціональних вимог які є ключовими для стабільної роботи програми:

Пріоритетними нефункціональними вимогами є:

- Надійність гри має бути стабільною та здатною відновлюватися після помилок, збоїв та багів.
- Зручність використання, інтерфейс має бути інтуїтивно зрозумілим будь якій категорії користувачів та не викликати дискомфорт під час користування.
- Локалізація гра повинна мати мінімум Англійську мову інтерфейсу, як максимум Українську.
- Продуктивність гра та її відповіді на дії користувача повинні бути без затримок та пінгу, забезпечуючи плавність ігрового процесу та насолоду грою.

## Висновки до розділу

У другому розділі дипломної роботи було здійснено детальне визначення вимог до програмного продукту, що включало аналіз можливих сценаріїв використання, системних характеристик, а також розробку функціональних і нефункціональних вимог. Вимоги, представлені в цьому розділі, закладають надійний фундамент для наступних етапів розробки гри, надаючи точне бачення її можливостей та очікуваних особливостей. Глибокий

аналіз сценаріїв використання допоміг виявити ключові моменти взаємодії користувачів з системою та її елементами, виділивши важливі функції, які повинен підтримувати та реалізовувати розроблюваний додаток. Огляд системних вимог дозволив уточнити технічні параметри та умови використання системи, що є критичним для забезпечення її безперебійної роботи. Опис функціональних вимог надав докладний перелік завдань, які система повинна виконувати, що дало змогу точно визначити обсяг роботи та можливості системи. Нефункціональні вимоги, у свою чергу, визначили основні якісні атрибути програмного забезпечення. На основі аналізу в цьому розділі було сформульовано технічне завдання для розробки 3Д RPG гри.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

### 3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

#### 3.1 Архітектура програмного забезпечення

Гра це не звичайний додаток, тому кількість компонентів та значущість сильно відрізняється від архітектури, яку демонструють при роботі в іншими ПЗ або сайтами. Звісно розробник повинен поглиблювати свої навички завжди і всюди, що може сприяти його кар'єрному росту та продуктивності. В розробці ігрового застосунку буде також використовуватися компонентно-орієнтоване програмування, яке дозволить використовувати один й той самий код безліч раз для різних речей – це певна особливість Unity. Також в рамках даної розробки було прийнято рішення використати монолітну архітектуру в силу невеликої кількості функцій і відсутність потреби в розбитті компонентів програми на окремі слої (рівні), а також відсутність необхідності в взаємодії між рівнями. Загальна діаграма класів знаходиться у графічному матеріалі, креслення 2. Детальний демонстрація класів представлена на діаграмах класів (рис. 3.2-3.7).

Монолітна архітектура [14] для RPG-проекту на Unity – це коли весь код гри зосереджений у одному проекті. Це спрощує розробку, оскільки всі компоненти тісно інтегровані та легко доступні. Зміни та оновлення можна швидко впроваджувати, не координуючи їх між різними сервісами. Менеджмент проекту стає простішим, адже все знаходиться в одному місці. Для невеликих або середніх проектів, де не потрібне швидке масштабування, монолітна архітектура може бути більш ефективною. Unity підтримує цей підхід, надаючи необхідні інструменти для розробки. Таким чином, монолітна архітектура є вибором для проектів, які не вимагають великої гнучкості чи масштабування, як в нашему випадку.(рис. 3.1)

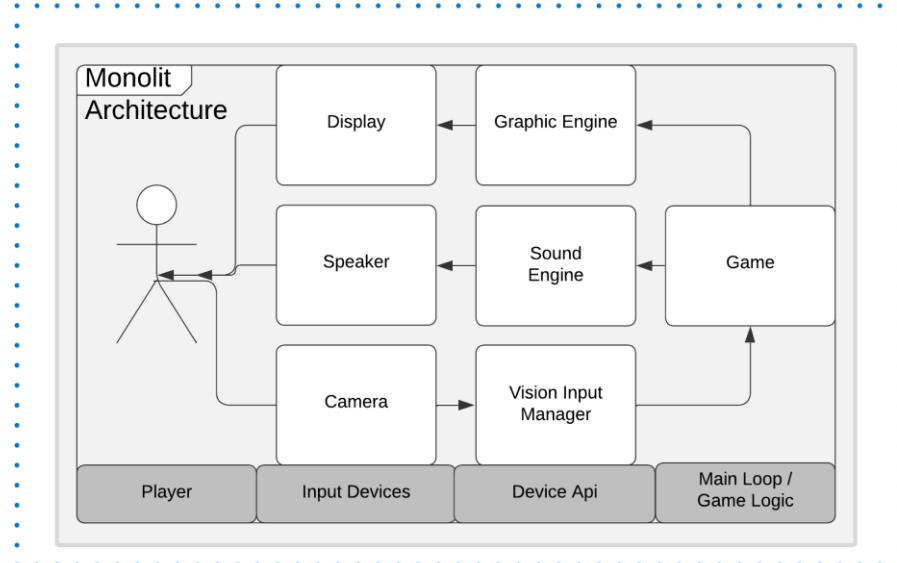


Рисунок 3.1 – Монолітна архітектура гри

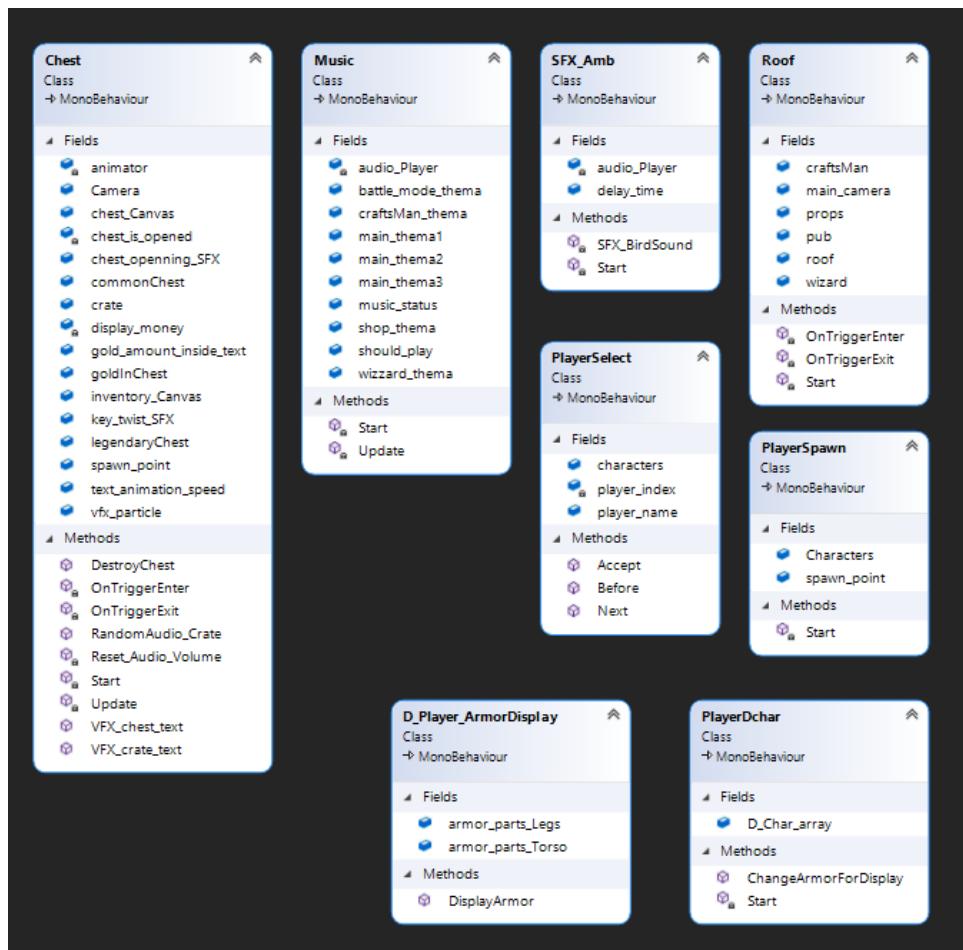


Рисунок 3.2 – Діаграма класів відповідальних за музику, вибір персонажа, взаємодію зі скрине. і тд.

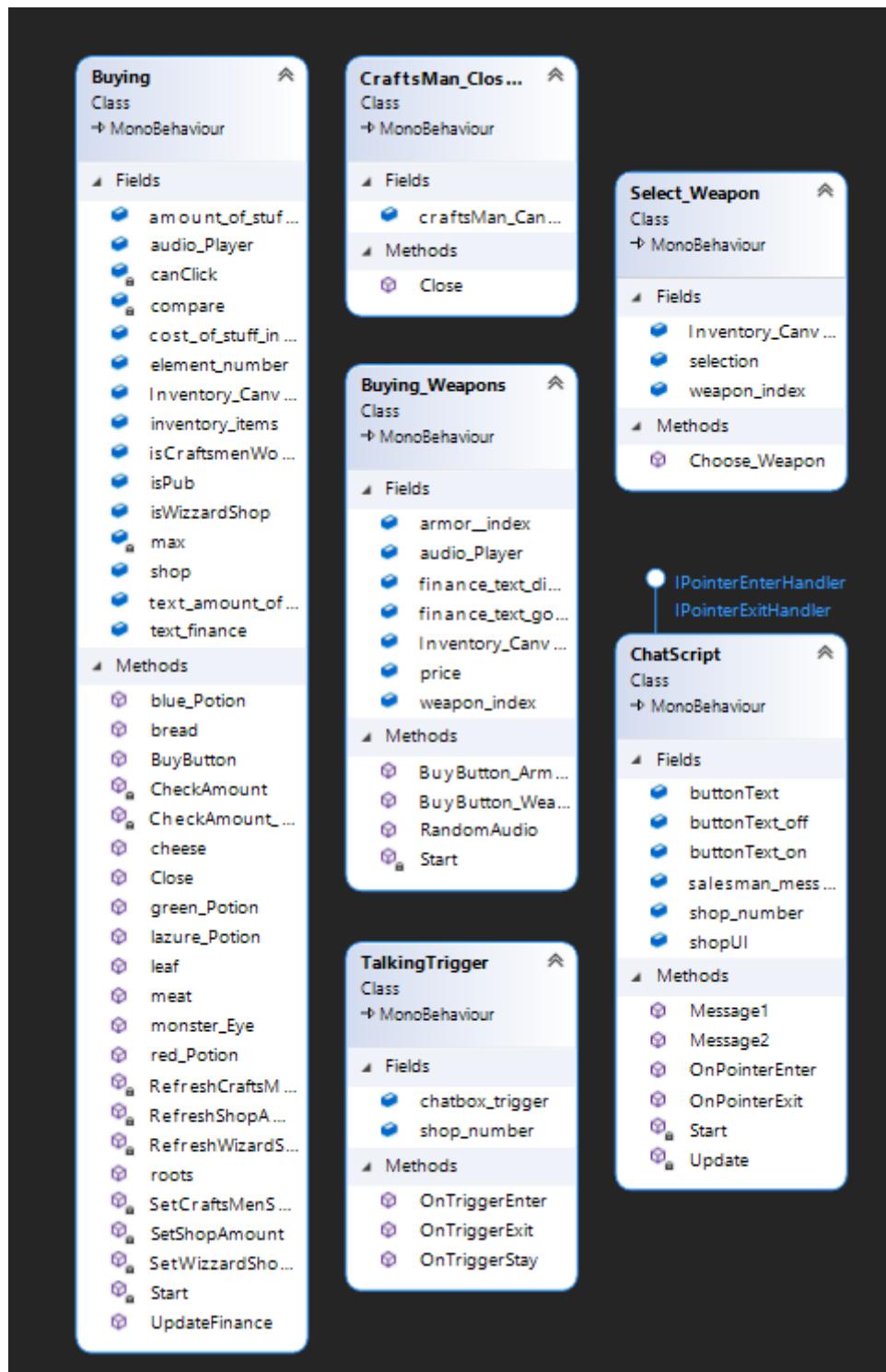


Рисунок 3.3 – Діаграма класів системи купівлі товарів, зброї, магазинів і тд.

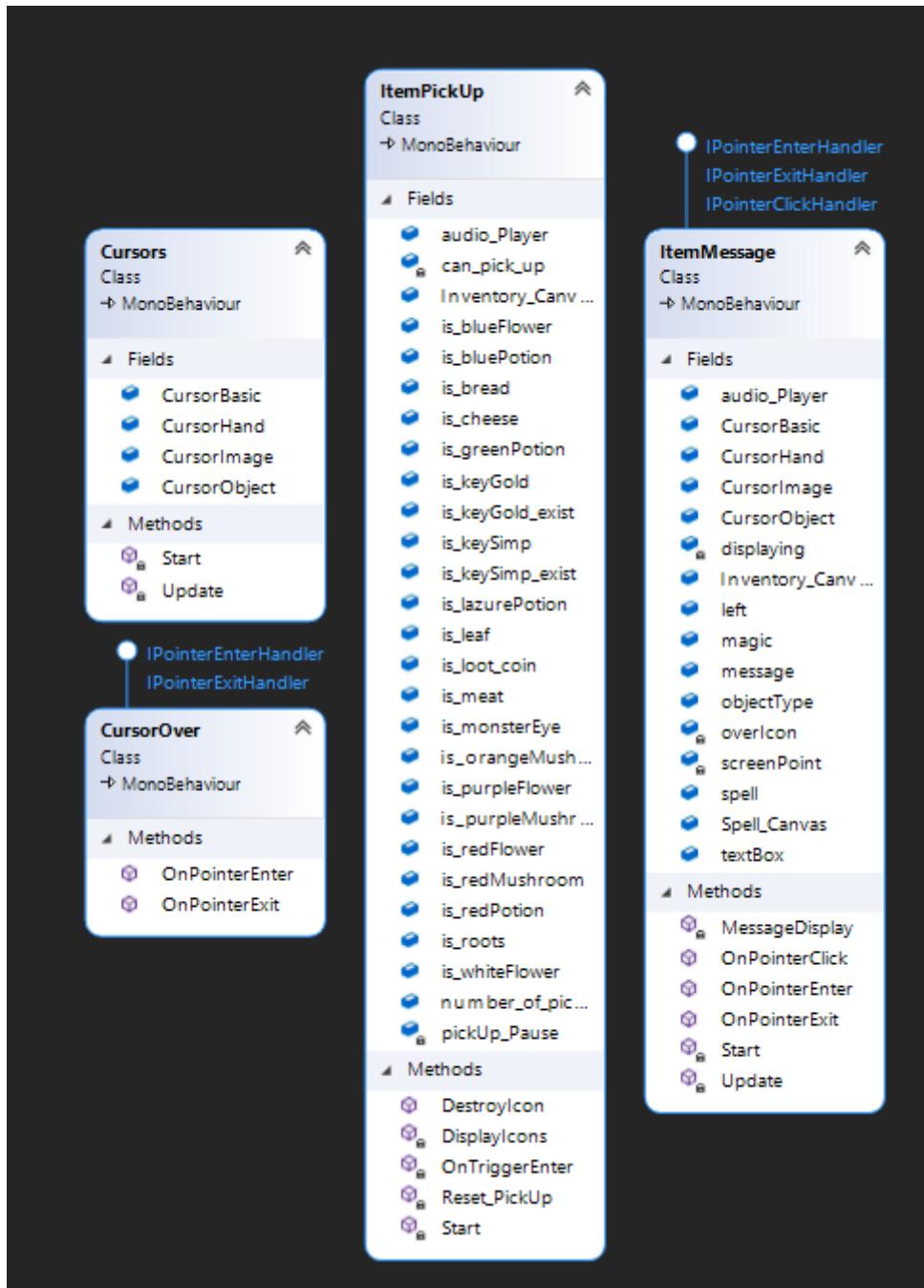


Рисунок 3.4 – Діаграма класів системи взаємодії з предметами на землі та інформації про них.

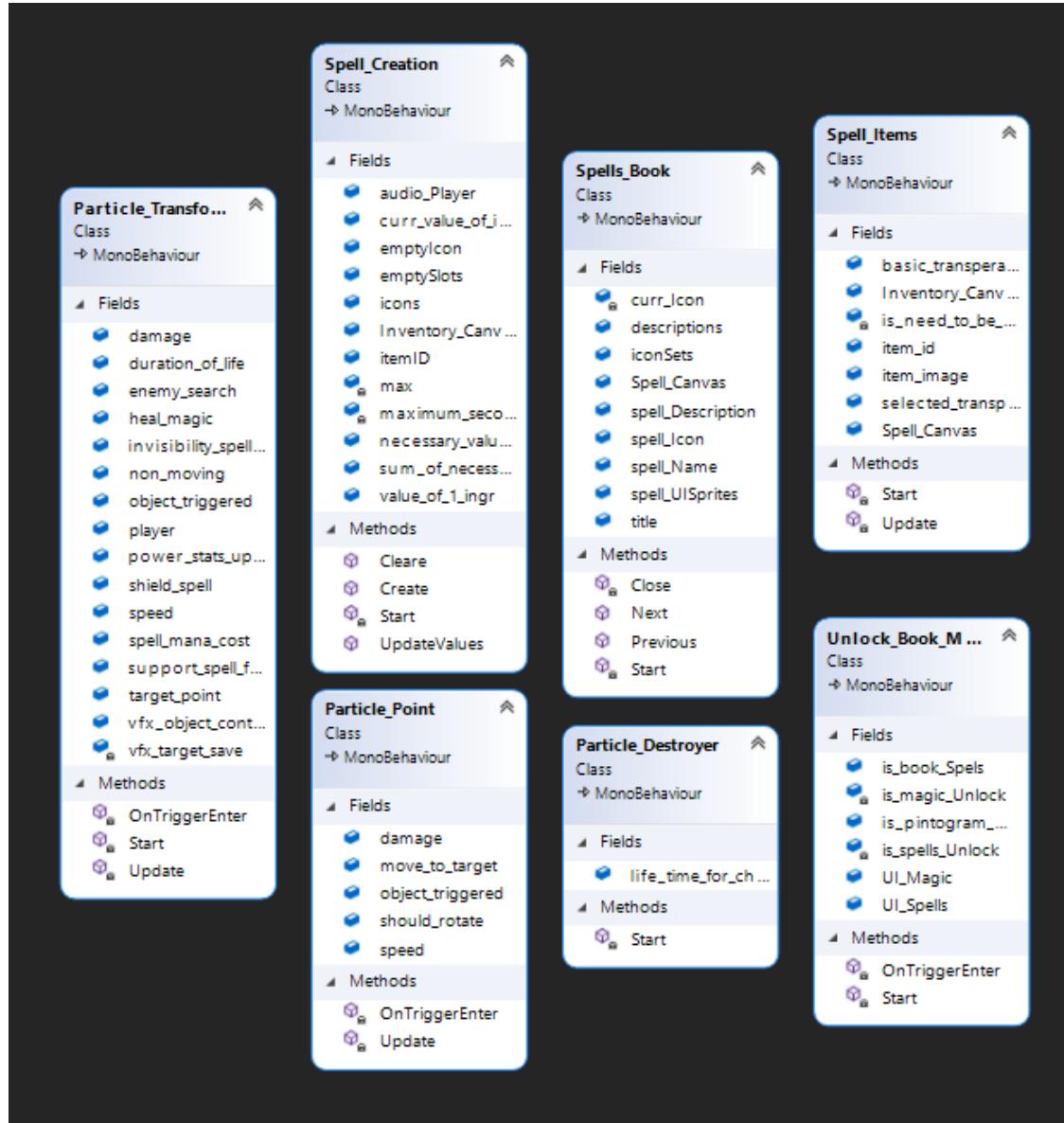


Рисунок 3.5 – Діаграма класів системи часток, магії, заклять, їх створення та управління.

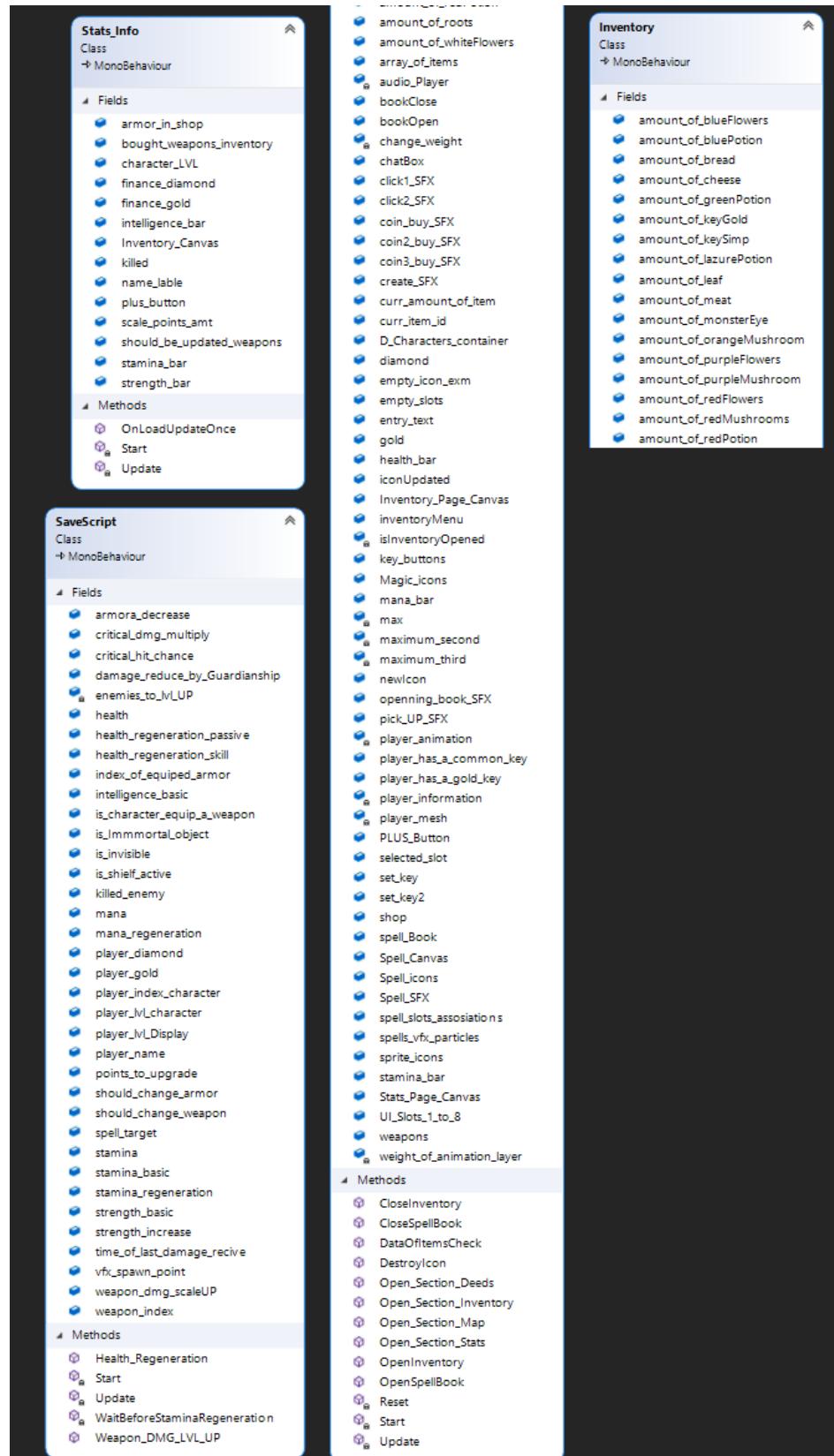


Рисунок 3.6 – Діаграма класів основної системи інвентару, характеристик та збереження прогресу.

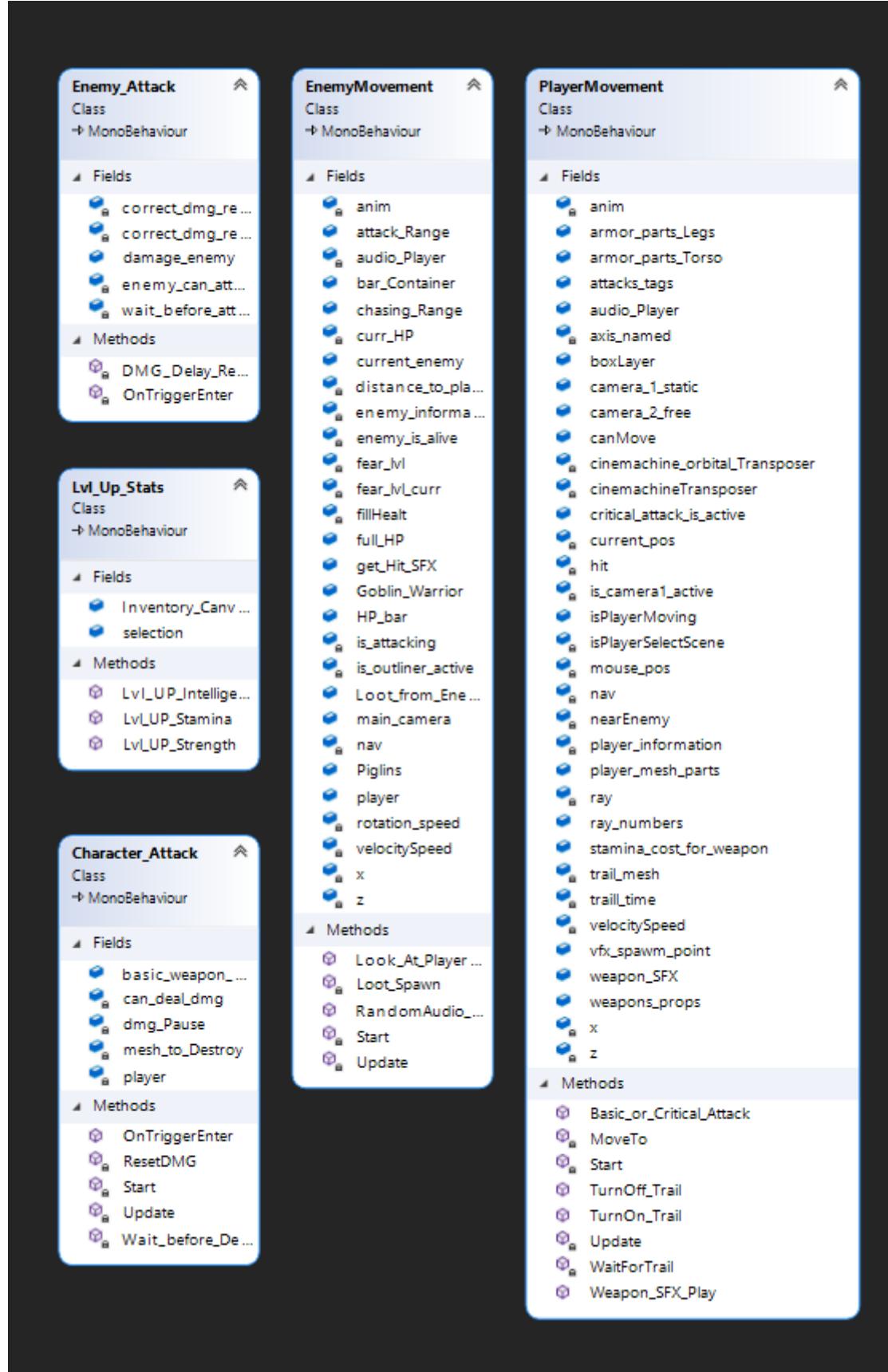


Рисунок 3.7 – Діаграма класів поведінки гравця, ворогів, характеристик та бойової системи персонажів.

Тепер перейдемо до термінів, які потрібно знати для подальшого розуміння проєкту:

- *Ігровий рушій (двигун)*. Це Unity Engine, основа, яка надає інструменти, об'єкти та функції, які полегшують життя розробника та зменшують час та потребу в додаткових ресурсах. Вона може включати в себе такі компоненти як: Рендеринг, анімація об'єктів, фізичні властивості, звукові компоненти, системи різноманітних скриптів та їх застосування, штучний інтелект (як для ворогів і НП, так і для звичайних розробників) і тд.

- *Сценарії поведінки об'єктів (Scripts) та Компонентний підхід*. Unity дозволяє працювати з кожним кодом, як з окремим компонентом та застосовувати його лише до тих об'єктів, які потрібні розробнику. Все це можливо за рахунок компонентно-орієнтованого підходу, де різні аспекти гри, керуються через різні модульні частини. Наприклад курсор на екрані гри абошкала здоров'я. Все це працює не залежно від інших компонентів і має свої задачі при запуску коду та при оновленні кадрів кожний раз.

- *Менеджери (Managers)*. Спеціальні скрипти, задача яких полягає в керуванні глобальними процесами, такими як перехід на інший рівень, управління станами гри, ігровий процес і тд.

- *Інтерфейси (UI)*. Присутня система, яка допомагає та полегшує роботу з 2д спрайтами та інтерфейсами. З її допомогою візуальна частина таких компонентів, як «Інвентар», «Магазин», «Статуси персонажів», «Меню вибору і тд. Створюються та адаптуються в два кліки. А на додачу до компонентної архітектури – робота зі створення візуалу приносить одне задоволення.

- *Асети (Assets)*. В рушії наявний дуже великий об'єм предметів та об'єктів, які будь хто може вставити в свою гру всього лише за два кліки. Не витрачаючи години на їх пошук, розуміння та імплементацію.

- *Фізичний двигун (Physics Engine)*. При створенні гри було використано компоненти, які давали можливість симулювати та

прораховувати реалістичну фізику та кінематику об'єктів, їх взаємодію та пересічення.

- *Система освітлення (Lighting System)*: Ця система відповідає за освітлення сцени та об'єктів, що включає динамічне та статичне освітлення. Також для більшої продуктивності рекомендується запікати світло з об'єктів в окремі карти, для того щоб програма не прораховувала кожен кадр нове освітлення.
- *Система частинок (Particle System)*. Для кращого візуального сприйняття та естетичного вигляду треба використовувати систему часток, що допомагає продемонструвати таки захопливі ефекти, як вогонь, вибух, вода, удар, невидимість і тд.

В результаті в розробників є цілий стек компонентів архітектури, використання яких піде тільки на користь проєкту та допоможе зекономити час, зусилля і гроші.

### 3.2 Обґрунтування вибору засобів розробки

Почнемо з того, що для розробки існує безліч, так званих, програмних рушіїв, які створені для того, щоб полегшити розробку гри. Звісно ви можете спробувати зробити все в консолі, як робили старі ігри, але якість продукту, який ви видасте – не порівняти навіть з найдешевшими аналогами на ринку, тим паче швидкість розробки залишатиме бажати кращого. Першими по популярності звісно є UnrealEngine5 та Unity. Наступні аналоги це CryEngine, Godot та GameMaker Studio. Давайте порівняємо їх усіх.

Таблиця 3.1 Порівняння Рушій

Рушій	Плюси	Мінуси	Мова
<i>Unity</i>	Широке розповсюдження; Візуальне написання сценаріїв; Великий Asset Store; Гнучкість;	Оптимізація; Якість графіки; Вартість підписки;	C#
<i>UE5</i>	Приголомшлива графіка; Візуальні сценарії; Великий маркетплейс; Крос-платформна розробка; Спільнота та підтримка;	Складність для новачків; Продуктивність; Модель Підписки;	C++/Blueprints
<i>Godot</i>	Open Source; Легкість та інтуїтивність; Візуальні сценарії; Спільнота;	Мала документація; Мала кількість матеріалів для навчання; Обмеженість сховища;	C#
<i>CryEngine</i>	Безкоштовний; Продуктивний; Висока якість графіки;	Складність; Слабка підтримка; Мала документація	C++
<i>GameMaker Studio</i>	Зручний інтерфейс; Швидкість;	2D фокус; Обмеженість скриптів;	GML

Як бачимо, в цілому всі рушії заслуговують уваги, та хороші в тих чи інших випадках. Проте все ж нам треба обрати один, а не 5 для нашої роботи. Тому для початку відкинемо всі де мова програмування не C#, так як це моє власне побажання, бо я найкраще розбираюся саме в цій мові. Отже залишається на вибір Godot та Unity. З них двох, я схильний обрати на користь Unity, тому що він легкий в опануванні, має багато навчального матеріалу та високу якість графіки. В результаті ми виділили та обрали найкращий варіант для нашої розробки гри.

Під час розробки ігрового застосунку, особливо коли це відбувається лише за допомогою одного розробника, дуже важливо мати вільну, просту та багату на документацію платформу. Unity одна з таких, вона має найбільшу кількість пізнавального матеріалу, який може бути використаний програмістами для покращення своєї роботи. Також, як я вже згадував, система монетизації в Unity досить проста, дешева та дозволяє заробляти розробникам хороші гроші. Згідно з ліцензійною політикою Unity, якщо ваша компанія або ви, як індивідуальний розробник, отримуєте \$100,000 і більше кількість грубого доходу (це не чистий прибуток) за фінансовий рік (4 квартали по 3 місяці), вам потрібно перейти на оновлену платну підписку Unity Plus або Unity Pro. Кожна з яких має підвищений ліміт заробітку.

Наприклад Unity Plus має ліміт \$200,000 умовних одиниць, тоді як Unity Pro та Unity Enterprise взагалі не мають обмежень. Це означає, що маючи підписку останнього рівня, компанія не буде змушувати вас платити більше і більше з кожним разом, як ви проходите поріг заробітку.

Крім того, починаючи з 2024 року, Unity вводить нову систему Runtime Fee для ігор, створених або оновлених до Unity 6. Цей збір застосовується лише після того, як гра перетне два пороги: \$1,000,000 грубого доходу за останні 12 місяців та 1,000,000 і початкових інвестицій. Якщо ваша гра досягне обох цих порогів, вам буде нараховано збір, який буде меншим з двох: 2.5% від місячного грубого доходу гри або збір на основі місячних початкових залучень. Проте для нас, як для інді розробників, які не претендують навіть

на мінімальні цифри у 100 000 – нема чому переживати, бо наша гра не планується, як фінансовий проект, а лише являє собою завдання на дипломне проєктування, що означає ніяких фінансових зобов'язань перед компанією.

Також двигун Unity був обраний через те, що я володію мовою C# краще ніж тим же C++ чи GML(власна специфічна мова програмування для сторонніх рушіїв). На додачу середовищем написанні коду було обрано Microsoft Visual Studio – так як, це найкращий сумісний продукт з Unity, з безліччю сторонніх розширень, компонентів та відносно легкою навігацією по проєкту.

### **3.3 Конструювання програмного забезпечення**

#### **3.3.1 Графічне оформлення**

Візуальна частина є основною складовою будь якої гри. Якщо гра не гарна – не важливо, які класні там механіки, ніхто не захоче в неї пограти. Перед початком опису графічної складової проєкту – слід виокремити основні назви та терміни, які будуть використовуватися у подальшому. Таких умовних позначень є декілька:

**Спрайт** (з англ. Sprite) – це двовимірне растрове зображення, інтегроване у, найчастіше, 2D-відеогрі. Проте також використовується і у 3D іграх. (рис.3.8)



Рисунок 3.8 – Приклад спрайтів

**Меш** (з англ. Mesh) – або іншими словами полігональна сітка, це сукупність вершин, ребер, і граней, які утворюють площину між ними під назвою фейс, який в свою чергу складається з двох трикутників. Тим самим утворює трьох вимірну форму для будь якого об'єкта. Як я вже казав, гранями зазвичай є трикутники, чотирикутники або інші прості опуклі багатокутники (полігони), оскільки це спрощує підрахунок для відеокарти.(рис.3.9)



Рисунок 3.9 – Приклад моделей (мешів)

**Префаби** (з англ. Prefabs) – це такий особливий компонент об'єкту для Unity, який дозволяє зберігати повністю сконфігуровані ігровий об'єкт у проєкті для повторного використання. Ним може бути будь що, що використовується у сцені.

Отож, для реалізації графічної складової такого складного та великого проєкту було застосовано безліч асsetів з Unity Asset Store та створено власноруч у Photoshop. Також всі основні моделі НПП були завантажені з безкоштовного ресурсу Mixamo.

На початкових етапах – налаштовувати префаби моделей ніяк не потрібно, вони вже були налаштовані авторами пакетів, а от налаштування нашого з вам створеного UI потребує додаткової уваги. Якщо просто додати в проєкт наші спрайти, рушій не буде відображати їх коректно. Для цього існує вбудована функція для роботи з спрайтами під назвою Sprite Editor, яка допомагає зрозуміти системі, що цей об'єкт повинен мати характеристики спрайту. (рис.3.10)

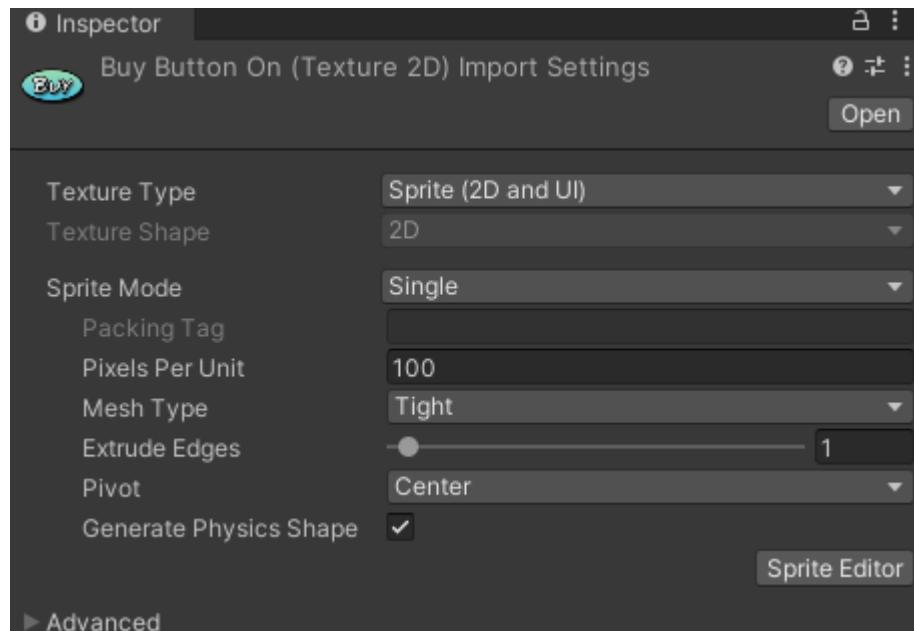


Рисунок 3.10 – Приклад налаштувань одного з спрайтів

### 3.3.2 Світ

Для створення світу (рис. 3.11) було використано моделі з таких паків як Polygone Dungeon [10] та Medieval house modular lite [11]. Вони дають доступ

до великої кількості готових моделей, які цілком і повністю підпадають під наші потреби. Звісно, я міг би і сам створити всі ці моделі, що я й роблю на роботі, проте мені гроші зараз за це не платять і на створення кожної моделі йде приблизно 120 год, що в умовах коротких дедлайнів по практиці – не реально виконати самому. Тому вважаю використання готових моделей, анімацій та спецефектів цілком допустимим.



Рисунок 3.11 – Створений світ основного рівня.

### 3.3.3 Гравець

#### 3.3.3.1 Вибір персонажа

На даний момент в грі присутня можливість обирати одного з 6 головних герої для проходження гри. Все це присутнє у меню вибору персонажу на початку гри. Характеристики персонажів на разі ніяк не відрізняються і слугують лише як форма естетичного задоволення для користувача. Кожен знайде щось на свій смак. (рис. 3.13 - 3.18). Присутня також можливість ввести ім'я свого героя з яким він буде грати всю гру. Все це працює за допомогою скрипта, який допомагає обрати персонажа.(рис. 3.12). Він досить простий і просто присвоює конкретний індекс персонажа в файл збереження.

```
0 references
public void Accept()
{
    SaveScript.player_index_character = player_index;
    SaveScript.player_name = player_name.text;

    Debug.Log(player_index + " INDEX");

    SceneManager.LoadScene(1); //Load Terrain1
}
```

Рисунок 3.12 – Код для выбору персонажа



Рисунок 3.13 – Персонаж №1



Рисунок 3.14 – Персонаж №2



Рисунок 3.15 – Персонаж №3



Рисунок 3.16 – Персонаж №4



Рисунок 3.17 – Персонаж №5



Рисунок 3.18 – Персонаж №6

До кожного з наших персонажів застосований такий компонент, який ми вже описували на початку 1 розділу – NavMesh Agent, що використовує алгоритм A\* для пошуку найкоротшого шляху. Ось так він виглядає с поміж великої кількості допоміжних колайдерів на персонажі (рис. 3.19). Проте без коду, жоден персонаж, колайдер чи меш не буде рухатися, тому пропоную подивитися на основний скрипт під назвою PlayerMovement.cs (рис. 3.23). Він досить великий, тому по всіх параметрах проходить зараз не буду. Нас

цікавить тільки основне місце де відбувається прорахунок того куди нам треба йти.(рис 3.14). Весь інший код можна буде побачити в Додатках до звіту.



Рисунок 3.19 – Демонстрація персонажа та NavMeshAgent

### 3.3.3.2 Пересування персонажа

Отже спочатку нам треба отримати інформацію про компонент NavMeshAgent, який закріплений на гравці. (рис. 3.20)

```
void Start()
{
    nav = GetComponent<UnityEngine.AI.NavMeshAgent>();
    anim = GetComponent<Animator>();
```

Рисунок 3.20 – Отримаємо інформацію про NavMeshAgent

Далі в основній частині програми ми використовуємо систему променів та швидкості по X та Z для розуміння чи рухається гравець у даний момент чи ні. Після вступає в роботу метод від основної/допоміжної камери ScreenPointToRay(Input.mousePosition), який створює певну кількість променів між місцем натискання мишко на колайдері землі та камерою. В результаті ми отримуємо значення позиції куди треба потрапити, проте перед відправленням нашого агента туди, треба підрахувати середнє значення між ними для більш чіткої позиції. Звісно в коді присутнє ще безліч моментів, які підраховують чи

це ворог, чи зараз активний екран блокування, вибору і тд але розглядати в даній секції ми їх не будемо.(рис. 3.21)

```

if (isPlayerSelectScene == false)
{
    x = nav.velocity.x;
    z = nav.velocity.z;
    velocitySpeed = new Vector2(x, z).magnitude;

    Ray[] rays = new Ray[ray_numbers];

    if (Input.GetMouseButtonUp(0) && player_information.IsTag("nonAttack") && !anim.IsInTransition(0))
    {
        if (canMove == true)
        {
            for (int i = 0; i < ray_numbers; i++)
            {
                rays[i] = Camera.main.ScreenPointToRay(Input.mousePosition);
            }

            Vector3 averageHitPoint = Vector3.zero;

            foreach (Ray ray in rays)
            {
                RaycastHit hit;

                if (Physics.Raycast(ray, out hit, 300, boxLayer))
                {
                    if (hit.transform.gameObject.CompareTag("enemy"))
                    {
                        nav.isStopped = false;
                        SaveScript.spell_target = hit.transform.gameObject;
                        averageHitPoint += hit.point;
                        transform.LookAt(SaveScript.spell_target.transform);
                        StartCoroutine(MoveTo()); //wait 3 sec and than isStopped == true
                    }
                    else
                    {
                        SaveScript.spell_target = null;
                        averageHitPoint += hit.point;
                        nav.isStopped = false;
                    }
                }
            }

            averageHitPoint /= rays.Length;
            nav.destination = averageHitPoint;
        }
    }
}

```

Рисунок 3.21 – Основний код пересування персонажа

Також гравець може зупинити пересування персонажа у будь який момент за допомогою клавіші «S» (рис. 3.22).

```

if (Input.GetKeyDown(KeyCode.S))
{
    anim.SetBool("sprinting", false);
    nav.destination = transform.position;
}

```

Рисунок 3.22 – Система зупинки персонажа

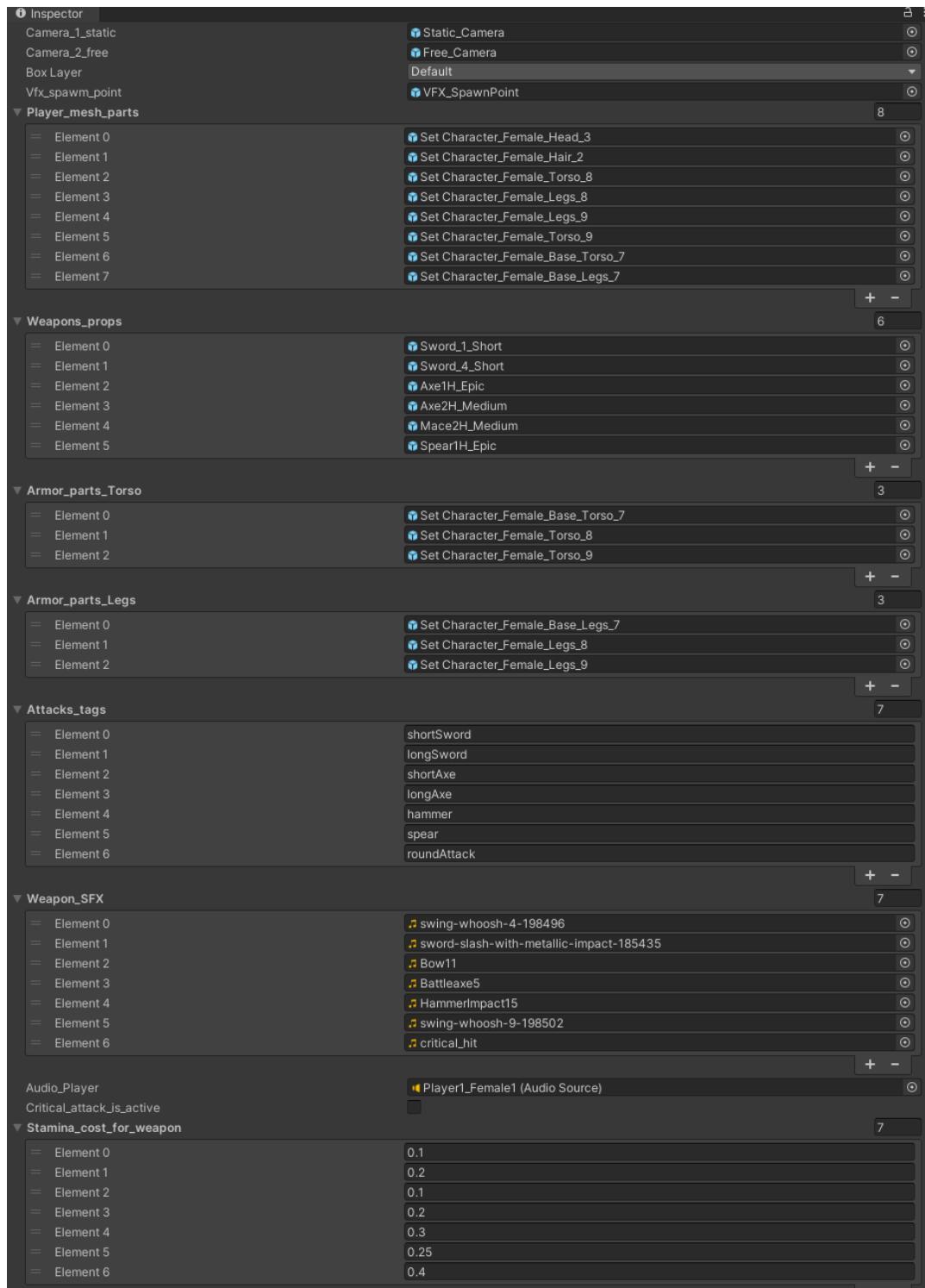


Рисунок 3.23 – Всі параметри PlayerMovement.cs

### 3.3.3.3 Атака персонажа та анімація

Всі анімації в проєкті були створені за допомогою компоненту State Machine, коду та Аватарів. Для прикладу система анімацій персонажа виглядає так. В ній два основних рівні: 1 – основний і відповідає за анімації базової

складності, такі як анімації пересування, смерті ,атаки персонажа різними видами зброї, критичні атаки і тд. та 2 – лише одна анімація, яка програвається тоді коли персонаж використовує закляття.(рис. 3.24)

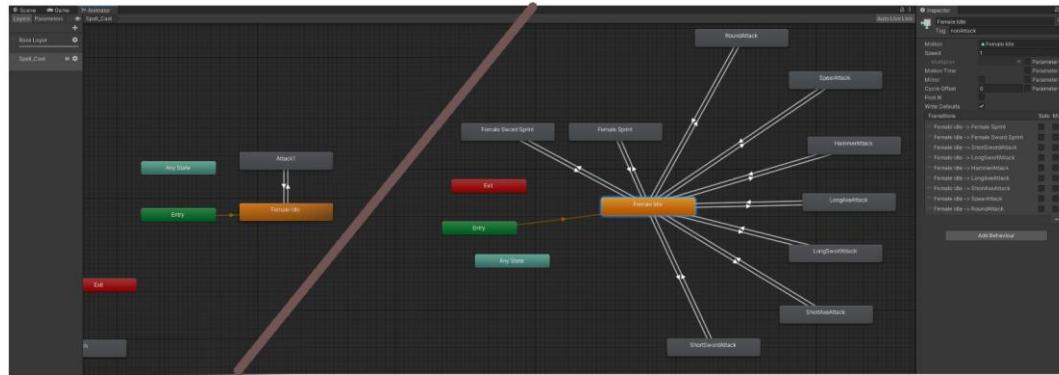


Рисунок 3.24 – (State Machine)Аніматор персонажа

Тепер подивимося на елемент коду який відповідає за програвання анімацій. Хочеться зазначити, що для кожного типу анімацій є свої умови виконання. Наприклад у кожного персонажа є два стилі бігу – зі зброєю та без. І для їх активації повинні виконуватися конкретні умови це булеве значення sprinting = true та equip\_a\_weapon = true/false (в залежності від типу анімації) (рис. 3.25). З атаками працює схожа система тригерів.

```
// Check if the character is moving (forward or backward)
anim.SetBool("sprinting", velocitySpeed > 0.1f);
if(velocitySpeed != 0)
{
    if(SaveScript.is_character_equip_a_weapon == false)
    {
        anim.SetBool("sprinting", true);
        anim.SetBool("equip_a_weapon", false);
    }
    if (SaveScript.is_character_equip_a_weapon == true)
    {
        anim.SetBool("sprinting", true);
        anim.SetBool("equip_a_weapon", true);
    }
    isPlayerMoving = true;
}
if (velocitySpeed == 0)
{
    anim.SetBool("sprinting", false);
    isPlayerMoving = false;
}
```

Рисунок 3.25 – (State Machine)Аніматор персонажа

До прикладу якщо швидкість персонажа більша за 0.1f ми кажемо аніматору, що треба активувати біг. Також якщо в нас є зброя ми повідомляємо його про наявність зброї, або її відсутність. Якщо ж гравець не рухається, то тригер дезактивований. Теж саме все працює і при виконанні атаки зброєю. Ми просто повідомляємо що треба активувати тригер, який відповідає за програвання саме тої анімації, зброя якої екіпірована на даний момент або якщо випав шанс, то активується критична атака, яка нанесе додаткову шкоду ворогу.(рис. 3.26)

```

if (Input.GetKeyDown(KeyCode.Z))
{
    if (SaveScript.is_character_equip_a_weapon == true && SaveScript.stamina > 0.2)
    {
        Basic_or_Critical_Attack();
    }
}

if(SaveScript.health <= 0.0f)
{
    SceneManager.LoadScene(0); // 0 - Player Select 1 - Terrain1 (More can check in File -> Build Settings)
    SaveScript.health = 1.0f;
}

}

1 reference
public void Basic_or_Critical_Attack()
{
    float randomNumber = Random.value;
    if (randomNumber <= SaveScript.critical_hit_chance)
    {

        critical_attack_is_active = true;
        anim.SetTrigger(attacks_tags[6]);
        audio_Player.volume = 0.4f;
        audio_Player.clip = weapon_SFX[6];
        audio_Player.Play();
        SaveScript.stamina -= stamina_cost_for_weapon[6];

    }
    else
    {
        critical_attack_is_active = false;
        anim.SetTrigger(attacks_tags[SaveScript.weapon_index]);
        audio_Player.volume = 0.3f;
        audio_Player.clip = weapon_SFX[SaveScript.weapon_index];
        //audio_Player.Play();
        SaveScript.stamina -= stamina_cost_for_weapon[SaveScript.weapon_index];
    }
}

```

Рисунок 3.26 – Код атаки

Тепер давайте глибше подивимося, яким само чином працює атака, як вона реєструє попадання по ворогу і за якими принципами. Основний принцип працює на так званих колайдерах. Які є на зброї і на ворогах, при їх пересіченні

відбувається запрограмована дія. Ось так виглядають колайдери на зброї.(рис. 3.27)

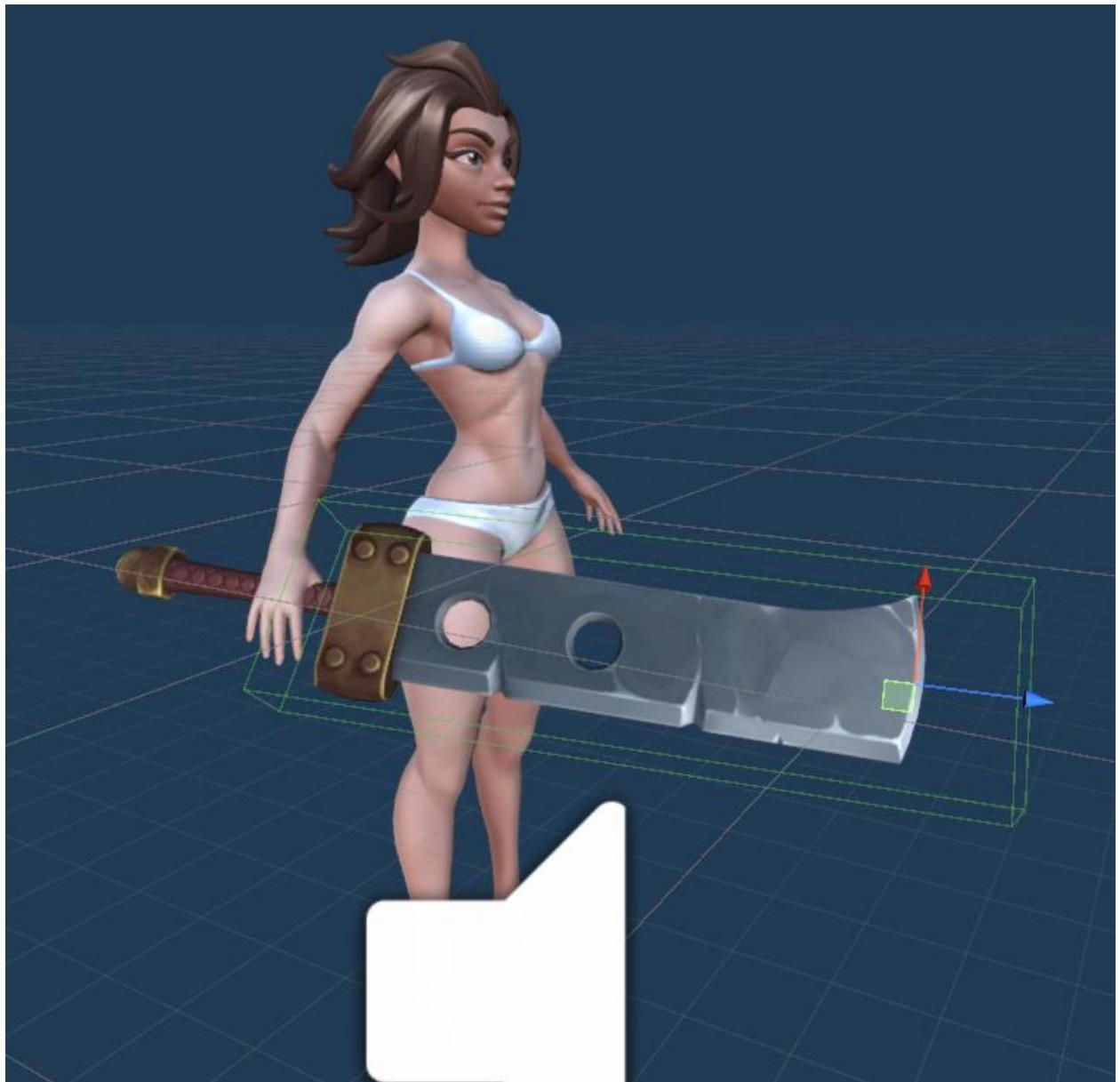


Рисунок 3.27 – Приклад колайдеру.

Проте якщо вони активні весь час, це потребує значних ресурсів програмних – тому вони запрограмовані так, щоб за умовою бути вимкнені і пересікатися з колайдерами ворогів просто так, поки зброя лежить в руці. Вони активуються лише в конкретний проміжок 2 - 3 кадрів виконання атаки. Коли ж відбувається пересічення – запускається подія OnTriggerEnter на скрипті зброї. Детальніше код можна побачити на рис. 3.28:

```

if (other.CompareTag("enemy") && can_deal_dmg == true )
{
    int dmg_check = 0;
    if(player.GetComponent<PlayerMovement>().critical_attack_is_active == true)
    {
        dmg_check = (basic_weapon_damage + SaveScript.weapon_dmg_scaleUP + SaveScript.strength_increase) * SaveScript.critical_dmg_multiply;
        other.transform.gameObject.GetComponent<EnemyMovement>().full_HP -= ((basic_weapon_damage + SaveScript.weapon_dmg_scaleUP + SaveScript.strength_increase) * SaveScript.critical_dmg_multiply);
        can_deal_dmg = false;
    }
    else
    {
        dmg_check = (basic_weapon_damage + SaveScript.weapon_dmg_scaleUP + SaveScript.strength_increase);
        other.transform.gameObject.GetComponent<EnemyMovement>().full_HP -= (basic_weapon_damage + SaveScript.weapon_dmg_scaleUP + SaveScript.strength_increase);
        can_deal_dmg = false;
    }
    Debug.Log(basic_weapon_damage + " " + SaveScript.weapon_dmg_scaleUP + " " + SaveScript.strength_increase);
    Debug.Log("Monster = " + other.name + " HP = " + other.transform.gameObject.GetComponent<EnemyMovement>().full_HP + " DMG = " + dmg_check);
    StartCoroutine(ResetDMG());
}

```

Рисунок 3.28 – Логіка роботи атаки персонажа.

З самого початку ми перевірімо чи річ, яку ми атакуємо має тег «enemy» та чи не вдаряли ми її до цього. Це зроблено для того, щоб уникнути безлімітного пересічення колайдерів і моментальної смерті ворога. Якщо все окей, ми витягаємо зі скрипта ворога його здоров'я та мінусуємо значення, яке еквівалентне нашій базовій шкоді від зброї плюс наш скейл від покращення рівня гравця плюс рівень силовох характеристики персонажа. Якщо ж це була критична атака, то тоді все це ще й множимо на показник критичної атаки, для тестів було обране значення «2».

При використанні заклинань це працює за схожим принципом.(рис. 3.29)

```

Unity Message | 0 references
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("enemy") && other.transform.gameObject != object_triggered)
    {
        other.transform.gameObject.GetComponent<EnemyMovement>().full_HP -= damage;
        object_triggered = other.transform.gameObject;
    }
}

```

Рисунок 3.29 – Логіка роботи атаки персонажа магією

### 3.3.3.4 Взаємодія з предметами

Одна з ключових функцій, які повинні бити в RPG грі – це можливість збирати предмети. Це одна з основ яка, допомагає краще пізнати світ. На разі в грі наявні 20 предметів, з якими можна взаємодіяти шляхом піднімання та додавання в інвентар їх.(рис. 3.30)

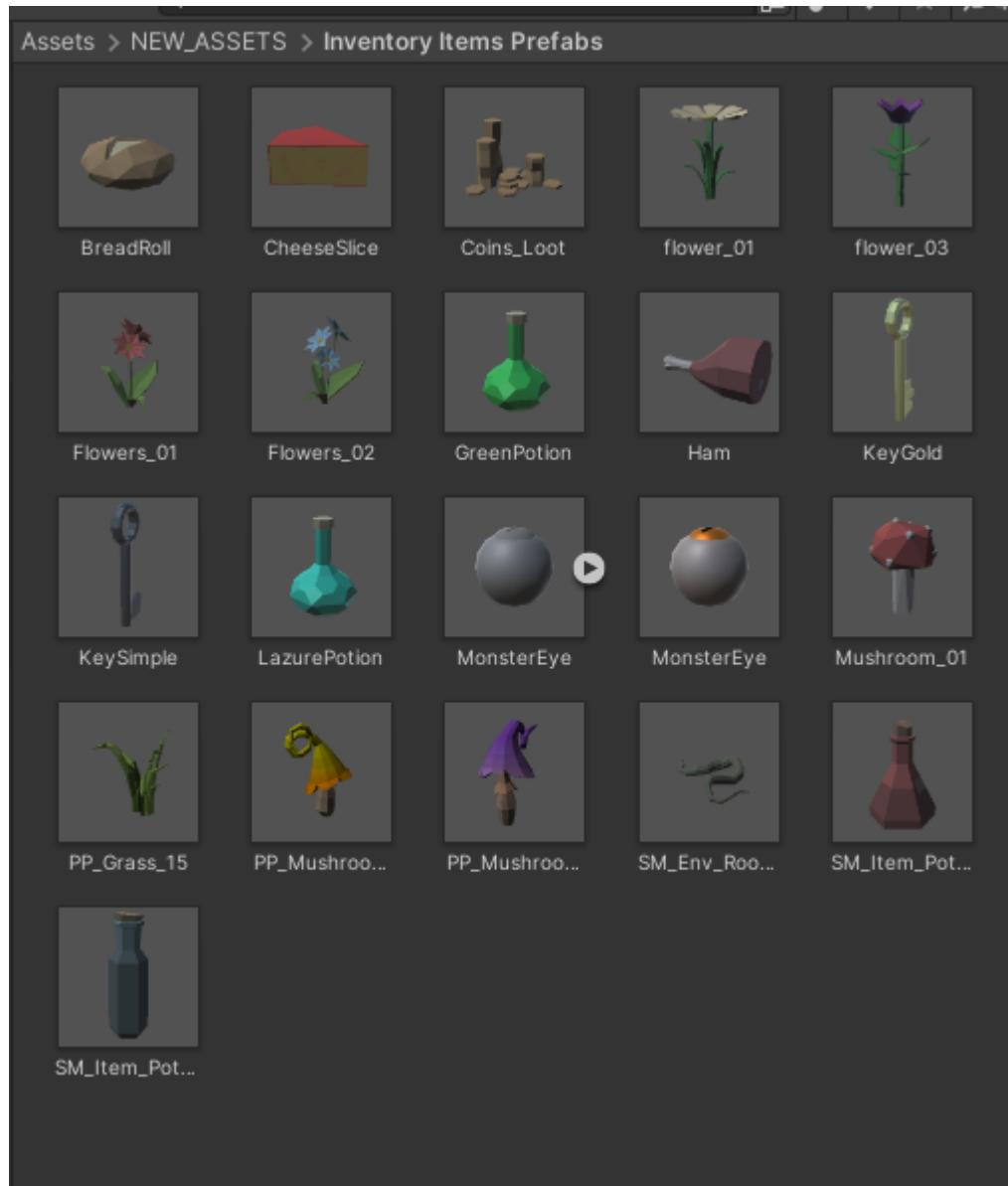


Рисунок 3.30 – Кількість предметів в грі, з якими може взаємодіяти гравець

З них Coins\_Loot – те що випадає з мертвих ворогів і зараховується як ігрова валюта, KeyGold та KeySimp використовуються для відкриття звичайних та легендарних сундуків. Всі інші слугують для поновлення ХП та створення спелів, у ролі інгредієнтів для них. До всіх предметів також застосований PickUp.cs скрипт. (рис. 3.31). В якому все працює за принципом колайдерів. В кожного об'єкта він є і при пересіченні з колайдером гравця відбувається програвання звуку підняття, додача в інвентар та знищення об'єкта з карти.

```

Unity Message | 0 references
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player") && can_pick_up == true)
    {
        can_pick_up = false;

        audio_Player.clip = Inventory_Canvas.GetComponent<Inventory>().pick_UP_SFX;
        audio_Player.Play();
        if (is_redMushroom == true)
        {
            if (Inventory.amount_of_redMushrooms == 0)
            {
                DisplayIcons();
            }
            Inventory.amount_of_redMushrooms++;
            Destroy(gameObject);
        }
        else if (is_blueFlower == true)
        {
            if (Inventory.amount_of_blueFlowers == 0)
            {
                DisplayIcons();
            }
            Inventory.amount_of_blueFlowers++;
            Destroy(gameObject);
        }
        else if (is_whiteFlower == true)
        {
            if (Inventory.amount_of_whiteFlowers == 0)
            {
                DisplayIcons();
            }
            Inventory.amount_of_whiteFlowers++;
            Destroy(gameObject);
        }
    }
}

```

Рисунок 3.31 – Логіка підняття предметів

В грі передбачена також взаємодія гравця з такими предметами як Скриня (рис.3.32 – 3.33) та Ящик (рис. 3.34 – 3.35). Перший, при умові наявності ключа, можна відкрити, другий – тільки знищити і отримати винагороду у вигляді золота.

```
//if we are attacking crate
if (other.CompareTag("Crate"))
{
    other.transform.gameObject.GetComponentInParent<Chest>().VFX_crate_text();
    mesh_to_Destroy = other.transform.parent.gameObject;

    Destroy(other.transform.gameObject);
    StartCoroutine(Wait_before_Destroy());
}
```

Рисунок 3.32 – Логіка знищення ящика



Рисунок 3.33 – Приклад знищення Ящика

В результаті гравець отримав 100 золота.



Рисунок 3.34 – Приклад відкриття скрині

Логіка роботи скрипту з відкриттям скрині також цілком і повністю активна на коллайдерах, проте крім методу, який викликається при вході в коллайдер (рис. 3.35) – є метод який викликається на виході. І його задача це

запустити трігер програвання анімації для закриття скрині.(рис. 3.36) А сама функція знищення викликається з аніматора.

```

if(crate == false)
{
    if (other.CompareTag("Player"))
    {

        //For Common Chest
        if (Inventory.player_has_a_common_key == true && commonChest == true && chest_is_opened == false)
        {
            animator.SetTrigger("openChest");
            Inventory.gold = Inventory.gold + goldInChest;
            goldInChest = 0;

            //audio SFX for chest
            inventory_Canvas.GetComponent<AudioSource>().clip = key_twist_SFX;      //too low
            inventory_Canvas.GetComponent<AudioSource>().Play();                      // too low

            inventory_Canvas.GetComponent<AudioSource>().clip = chest_openning_SFX;
            inventory_Canvas.GetComponent<AudioSource>().PlayDelayed(1 * Time.deltaTime);

            chest_is_opened = true;
            Inventory.amount_of_keySimp--;
            if (Inventory.amount_of_keySimp == 0)
            {
                ItemPickUp.is_keySimp_exist = false;
                // ItemPickUp.DestroyIcon();

            }
            Debug.Log("Gold = " + Inventory.gold);

        }
        else
        {
            Debug.Log("You Don't have a common Key");
        }
    }
}

```

Рисунок 3.35 – Логіка відкриття скрині

```

Unity Message | 0 references
private void OnTriggerEnter(Collider other)
{
    if (crate == false)
    {
        if (other.CompareTag("Player"))
        {
            if (Inventory.player_has_a_common_key == true && commonChest == true && chest_is_opened == true)
            {
                animator.SetTrigger("closeChest");
            }
            else if (Inventory.player_has_a_gold_key == true && legendaryChest == true && chest_is_opened == true)
            {
                animator.SetTrigger("closeChest");
            }

        }
    }
}

```

Рисунок 3.36 – Логіка закриття скрині

### 3.3.3.5 Взаємодія з НП

В грі, для наповнення світу існує 3 види НП, всі вони знаходяться в своїх магазинах, або по світу. При наближенні до них з'являється діалогове вікно для комунікації з ними.(рис. 3.37)

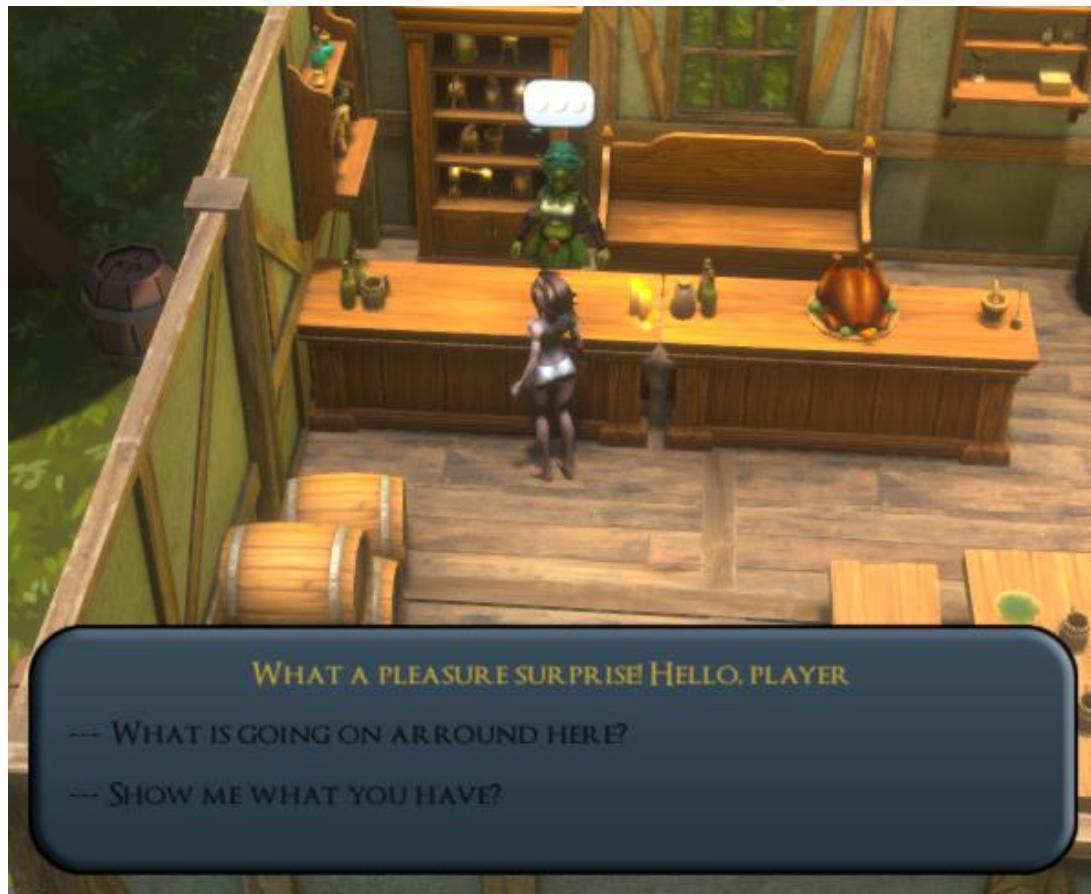


Рисунок 3.37 – Діалогове вікно з НП

При виборі першого варіанту відповіді НП розкаже що відбувається навколо, проте поки що нічого особливого не відбувається (рис. 3.38).

```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    if (PlayerMovement.canMove == true && PlayerMovement.isPlayerMoving == true)
    {
        if (shopUI != null)
        {
            shopUI[shop_number].SetActive(false);
        }
    }
}
```

Рисунок 3.38 – Логіка відкриття магазину

Тому перейдемо до другого варіанту. Він дає нам можливість відвідати магазин в Пабі.(рис. 3.39)



Рисунок 3.39 – Магазин в Пабі

В таверні є можливість купити інгредієнти які будуть відновлювати тобі життя. Наступний тип магазину – це чаклунський. Він активується так само але вже поряд з НПП «Чаклун». Ось так це виглядає. (рис. 3.40)



Рисунок 3.40 – Чаклун

Його магазин вже виглядає більш обширно, так як він продає основні інгредієнти для створення спелів та магії. Без них ви не зможете їх створювати (рис. 3.41).



Рисунок 3.41 – Магазин Чаклуна

І останній магазин, який відповідає за купівлю озброєння та екіпування – це магазин у НПР Ремесленик. Ось так він виглядає. (рис.3.42)



Рисунок 3.42 – Ремесленик

Він нам дає доступ до свого магазину де ми можемо придбати 6 видів зброї, кожна з яких має свій стиль атаки, колір та числові параметри шкоди. Також є можливість купити два види броні – легку та тяжку. Легка зменшує

отриману шкоду на 20 відсотків, в той час як тяжка дозволяє отримувати шкоди на 40 відсотків менше (рис. 3.43).



Рисунок 3.43 – Магазин зброї та броні

Сама логіка купівлі теж досить проста, бо ми просто записуємо індекс броні в основний скрипт збереження. А от зброя не просто записується одним значенням, а цілим масивом.(рис. 3.44)

```

0 references
public void BuyButton_Weapon()
{
    if(Inventory.gold >= price)
    {
        Inventory.gold -= price;
        Inventory_Canvas.GetComponent<Inventory>().weapons[weapon_index] = true;

        //RANDOM SFX COIN
        RandomAudio();
        //

        finance_text_diamond.text = Inventory.diamond.ToString();
        finance_text_gold.text = Inventory.gold.ToString();

    }
}

0 references
public void BuyButton_Armor()
{
    if (Inventory.gold >= price)
    {
        Inventory.gold -= price;
        SaveScript.index_of_equiped_armor = armor_index;
        SaveScript.should_change_armor = true;
        //RANDOM SFX COIN
        RandomAudio();
        //

        finance_text_diamond.text = Inventory.diamond.ToString();
        finance_text_gold.text = Inventory.gold.ToString();

    }
}

```

Рисунок 3.44 – Логіка купівлі зброї та броні

Насправді скрипт купівлі зброї досить простий, бо не потрібно відстежувати оновлення кількості доступних елементів у магазині, як в інших. Наприклад основний скрипт купівлі для інших магазинів з лімітованою кількістю інгредієнтів виглядає ось так.(рис.3.45 – 3.47)

```

public void BuyButton()
{
    if (canClick == true)
    {
        for (int i = 0; i < max; i++)
        {
            if (text_amount_of_stuff_in_shop[i] == compare)
            {
                max = i;
                if (amount_of_stuff_in_shop[i] > 0)
                {

                    if (isPub == true)
                    {
                        RefreshShopAmount();
                    } else if (isWizzardShop == true)
                    {
                        RefreshWizardShopAmount();
                    } else if (isCraftsmenWorkshop == true)
                    {
                        RefreshCraftsMenShopAmount();
                    }

                    if (Inventory.gold >= cost_of_stuff_in_shop[i])
                    {
                        if (inventory_items[i] == 0)
                        {
                            Inventory.newIcon = element_number[i];
                            Inventory.iconUpdated = true;
                        }
                        Inventory.gold -= cost_of_stuff_in_shop[i];

                        //RANDOM SFX COIN
                        int randomNumber = UnityEngine.Random.Range(1, 101);
                        if (randomNumber > 0 && randomNumber < 33) ... else if (randomNumber >= 33 && randomNumber < 66)
                        {
                            audio_Player.clip = Inventory_Canvas.GetComponent<Inventory>().coin2_buy_SFX;
                        } else if (randomNumber >= 66 && randomNumber < 101) ...
                            audio_Player.Play();
                        //RANDOM SFX COIN

                        if (isPub == true)
                        {
                            SetShopAmount(i);
                        } else if (isWizzardShop == true)
                        {
                            SetWizzardShopAmount(i);
                        } else if (isCraftsmenWorkshop == true)
                        {
                            SetCraftsMenShopAmount(i);
                        }
                    }
                }
            }
        }
    }
}

```

Рисунок 3.45 – Логіка купівлі інгредієнтів

```

1 reference
void SetShopAmount(int item)
{
    switch (item)
    {
        case 0:
            Inventory.amount_of_bread++;
            break;
        case 1:
            Inventory.amount_of_cheese++;
            break;
        case 2:
            Inventory.amount_of_meat++;
            break;

        default:
            break;
    }

    amount_of_stuff_in_shop[item]--;
    text_amount_of_stuff_in_shop[item].text = amount_of_stuff_in_shop[item].ToString();
    UpdateFinance();
    max = amount_of_stuff_in_shop.Length;
}

```

Рисунок 3.46 – Логіка задання кількості в магазині

```

3 references
void CheckAmount(int items_number_general)
{
    if (amount_of_stuff_in_shop[items_number_general] > 0)
    {
        canClick = true;
    }
    else
    {
        canClick = false;
    }
}

```

Рисунок 3.47 – Логіка перевірки кількості в магазині

Логіка полягає в тому щоб порівняти кількість інгредієнтів з масивом в якому ми задали їх кількість і поточним значенням, після чого якщо все ок ми купуємо і оновлюємо кількість елементу, до поки він не буде дорівнювати 0. Після чого можливість купити буде закрита. Взагалі скрипт наче і виглядає легко, проте він тісно пов'язаний з інвентарем, що робить його написання досить складним з точки зору порівняння всіх масивів, компонентів і тд. Також в Visual Studio неможливо запустити відладку коду, не знаю чому, напевно бо це Unity зі своєю компонентною обробкою – і знайти помилку інколи дуже складно. Тому компонентна обробка це і плюс і мінус.

### 3.3.3.6 Інвентар

Напевно це була найскладніша частина всього процесу, так як основна суть полягає в тому, що ми порівнюємо масиви. В одному в нас є всі картинки

елементів, їх назви в іншому, індекси в третьому і самі клітинки для заповнення в четвертому. Все це має бути в конкретному порядку і я не мав права допуститися помилки в такому стратегічно - важливому місці. Також сам інвентар є другою відправною точкою, яка пов'язує майже всі елементи гри разом. Якісь об'єкти використовують її аудіо компоненти, хтось через неї отримує доступ до магазину і головного бару слотів і тд.

Спочатку ми перевіряємо чи потрібно нам оновити інформацію про інгредієнти в інвентарі. Зазвичай тригер на це йде або зі скрипту підняття предмету, або з магазину. (рис.3.48)

```
//Debug.Log("iconUpdated = " + iconUpdated);
if (iconUpdated == true)
{
    for (int i = 0; i < max; i++)
    {
        if (empty_slots[i].sprite == empty_icon_exm)
        {
            max = i;
            empty_slots[i].sprite = sprite_icons[newIcon];
            empty_slots[i].transform.gameObject.GetComponent<ItemMessage>().objectType = newIcon;
        }
    }
    StartCoroutine(Reset());
}
```

Рисунок 3.48 – Логіка роботи інвентарю

Після чого йде пошук найближчого слота, який відповідає спрайту «Порожня Іконка». Задаємо його як максимальне значення для виходу з циклу і передаємо йому значення нової іконки, яка відповідає нашому інгредієнту, та задаємо тип нового предмету. Після чого починає роботу метод Reset() визваний через StartCoroutine(). Це зроблено для того, щоб скрипт не намагався зробити це щофреймово, а ця функція дозволяє відтермувати це.(рис.3.49). В ньому ми приираємо тригер оновлення інвентарю і задаємо максимальну «відстань» у максимальній кількості доступних вільних слотах.

```
Unity Message | 1 reference
IEnumerator Reset()
{
    yield return new WaitForSeconds(0);
    iconUpdated = false;
    max = empty_slots.Length;
}
```

Рисунок 3.49 – Переназначаємо значення

Звісно там є ще безліч функцій та коду, проте на разі це самий основний момент логіки роботи інвентаря. Все інше можна буде побачити в Додатках. Сам же інвентар виглядає ось так. (рис. 3.50)



Рисунок 3.50 – Інвентар

### 3.3.3.7 Створення заклинань

Як і в кожній RPG грі, в нас присутня можливість використовувати заклинання. Проте як бачимо на рисунку (рис. 3.51) – вони не доступні на початку гри. Для цього нам потрібно відвідати першу споруду на нашому шляху – Паб, в якому знаходиться стенд з магічною книгою. При підході до неї – гравець забирає її в свій інвентар та відкриває можливість з її допомогою створювати заклинання.(рис. 3.52) Після цього, в гравця є можливість натиснувши на неї, отримати доступ до меню створення, яке покаже картинку заклинання, назву, опис та потрібні інгредієнти. (рис.3.53)



Рисунок 3.51 – Початковий стан інвентарю



Рисунок 3.52 – Стенд з магічною книгою



Рисунок 3.53 – Меню створення заклинань

Створення не відбувається автоматично, для цього необхідно обрати потрібні інгредієнти з вашого інвентаря, тоді компонент автоматично віднімається від вашої кількості і додається до книги заклять, підсвітивши доданий елемент. Інші частини, які ще потрібно додати – напів прозорі.(рис.3.54)



Рисунок 3.54 – Приклад додавання інгредієнтів

Після додавання всіх компонентів – відкривається можливість створити заклинання. Воно поміщається у місця пустих клітинок «Spells».(рис.3.55)



Рисунок 3.55 – Створене заклинання

Всі заклинання, а їх 6 – одноразову, після використання їх потрібно створити знову. (рис.3.56)

Заклинання:

- Allmighty (підвищує наносиму шкоду ворогам на 100 одиниць)
- Flame Nove (атака вогнем у 4 сторони)
- Guardianship (зменшує отримувану шкоду на 50%)
- Heal (відновлює здоров'я)
- Phantome Move (робить вас невидимим для ворогів)
- Variable Wind (потужна атака вітром по площі навколо гравця)



Рисунок 3.56 – Всі можливі заклинання

Тепер подивимося на логіку створення(рис.3.57). Для кожного заклинання – потрібна певна кількість інгредієнтів. Умовно для Невидимості треба інгредієнт А + Б + С, які в сумі дають Д. Для кожного з цих елементів є певний індекс, який ми створювали при розробці системи інвентарю. При натисканні і додаванні певного елемента в книгу – ми перевіряємо чи, по-

перше, його індекс дорівнює одному з індексів на сторінці створення, а по-друге, чи сума цих індексів дорівнює заданій кількості до цього в масиві. Якщо все добре, відбудеться створення. Якщо ж ви випадково натиснули на не той інгредієнт, він буде зарахований і зникне з вашого інвентарю, проте його індекс не буде дорівнювати потрібному, і ви просто втратите цінний ресурс. Магія – не вибачає помилок!

```
public void Create()
{
    if (necessary_value_for_creation_spells == curr_value_of_ingr)
    {
        for(int i=0; i < max; i++)
        {
            if(emptySlots[i].sprite == emptyIcon)
            {
                max = i;
                emptySlots[i].sprite = icons[itemID];
                emptySlots[i].transform.gameObject.GetComponent<ItemMessage>().objectType = itemID + 26;
                audio_Player.clip = Inventory_Canvas.GetComponent<Inventory>().create_SFX;
                audio_Player.Play();
                curr_value_of_ingr = 0;
                value_of_1_ingr = 0;
            }
        }
        max = emptySlots.Length;
    }
}
```

Рисунок 3.57 – Основна логіка створення

Другий тип магії, який відкривається – якщо відвідати міні дандж, «Magic». Його елементи не потрібно створювати, вони багаторазові і їх в грі 7.

Магія:

- *FlameTwist* (створює два шари магії, які поступово крутяться і розширяються навколо гравця, наносячи шкоду супротивникам)
- *Lightning Stun* (закляття, яке створює блискавку навколо супротивника)
  - *Realistic-Blast* (реалістичний вибух)
  - *RedBlast* (вибух червоного диму)
  - *Self-Directed Fire Attack* (атака трьома шарами по обраній цілі)
  - *Tornado* (торнадо, яке поступово з'являється на місці ворога)
  - *Ultimate Explosion* (дуже потужний вибух по площі)

### 3.3.3.8 Характеристики персонажа

Система прокачки рівня персонажа залежить від кількості вбитих ворогів. З кожним рівнем треба аніглювати все більше і більше. За прокачку вам даються додаткові знаки «S/P», які можна використовувати для покращення ось цих параметрів персонажа (рис.3.58) на сторінці «Stats» меню інвентарю (рис.3.59).

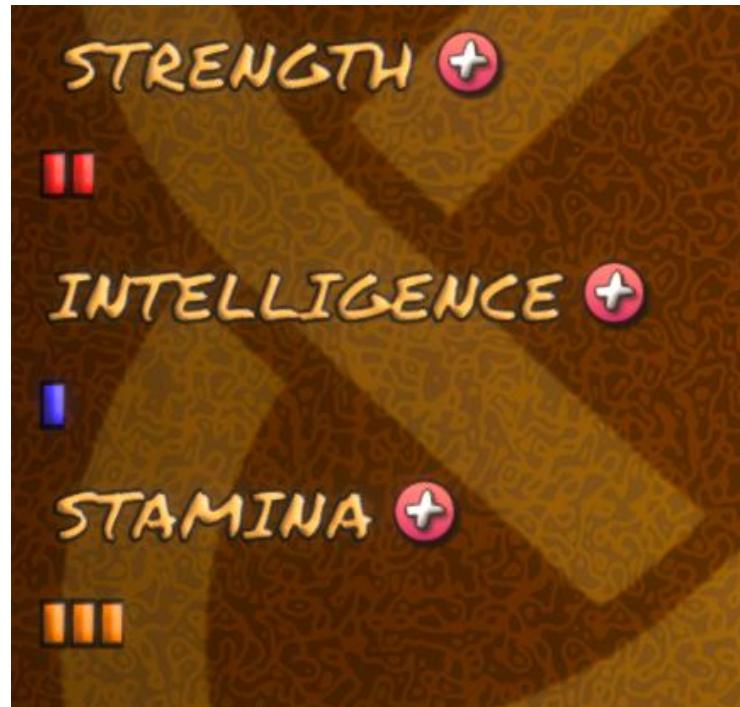


Рисунок 3.58 – Параметри при покращенні



Рисунок 3.59 – Сторінка покращення характеристик

Звісно покращення робиться не просто так. Наприклад з кожним новим рівнем гравець автоматично починає наносити більше шкоди ворогам. Але це значення пасивне і мале порівняно з тим, як воно збільшується при прокачуванні Strength.

При покращенні Stamina – підвищується швидкість регенерації витривалості персонажа, що означає більшу кількість атак (кожна атака потребує свою кількість витривалості).

При покращенні Intelligence – підвищується швидкість регенерації мани персонажа, що означає більшу кількість заклять та можливість використовувати їх довше.

При покращенні Strength – підвищується швидкість регенерації здоров'я та кількість шкоди, яку наносить персонажа.

### 3.3.4 Вороги

Вороги, це одна з ключових механік гри, притаманна RPG жанру. Без них світ був би порожнім та не цікавим. Так як в нас фентезі, то присутні популярні види ворогів: Гоблін Воїн, Піглін, Скелет та Голем. Всі вони мають власну поведінку, атаки, характеристики і тд. Для початку була розроблена система зору ворогів. Кожен з них має кут зору схожий з людським, який дорівнює, в нашему випадку 90 градусів. При потраплянні гравця в поле зору ворогів – вони почнуть атакувати його, якщо ж вони втратять його з поля зору, то будуть рухатися в те місце, де останній раз його бачили після чого спробують оглядітися навколо в пошуках гравця. Якщо він буде знайдений – продовжать атакувати. Також якщо в групі ворогів один бачить головного героя, а інші ні – тоді перший повідомляє всіх про його позицію і вороги, які до того моменту не знали де ви знаходитесь, почнуть вас переслідувати та атакувати. Звісно нижче (рис.3.60) ви побачите тільки невеличку частину системи, а саме ту, яка відповідає за відтворення променів.

```

1 reference
void Check_If_Player_is_InSight()
{
    Vector3 player_dir = player.transform.position - transform.position;
    float angle = Vector3.Angle(player_dir, transform.forward);

    if (angle < 90f && player_dir.magnitude < distance_of_ray)
    {
        RaycastHit hit;

        if (Physics.Raycast(transform.position + transform.up, player_dir.normalized, out hit, distance_of_ray))
        {
            Debug.DrawRay(transform.position, player_dir * 10f, Color.red);
            if (hit.transform == player.transform)
            {
                Debug.DrawRay(transform.position, player_dir * 10f, Color.green);

                Nearby_Enemy_Will_Know();
                look_for_player = false;
                player_is_inSight = true;
                last_seen_position = player.transform.position;
            }
        }
    }
    else if (player_is_inSight)
    {
        Debug.DrawRay(transform.position, player_dir * 10f, Color.red);
        player_is_inSight = false;
        nav.SetDestination(last_seen_position);
        look_for_player = true;
    }
}

```

Рисунок 3.60 – Raycast система (Система зору ворогів)

### 3.3.4.1 Гоблін Воїн



Рисунок 3.61 – Гоблін Воїн

Він швидкий та небезпечний. Наносять досить швидко шкоду та завжди помітить персонажа на великій відстані. Також даний тип ворогів має здібність уникати атаки гравця за допомогою ролів в різні сторони. В яку саме сторону – вирішує алгоритм за допомогою рівня агресії Гобліна. Якщо вона висока – то більше ймовірності, що перекат буде в сторону гравця для подальшого пресингу, якщо вона помірна – буде ліворуч, праворуч, при низькому рівні агресії перекат виконується назад (рис.3.62) . Взагалі, вороги завжди знають

де знаходиться гравець, проте в силу деяких перевірок, ми забороняємо їм враховувати ці параметри при пошуку гравця і виконанні певних дій. Також присутня така змінна яка відповідає за радіус, в якому вороги побачать персонажа та почнуть до нього бігти і атакувати (рис. 3.63).

```

    i reference
public void Dodge()
{
    Vector3 playerDirection = player.transform.position - transform.position;
    playerDirection.Normalize();

    Vector3[] roll_directions = {
        -transform.forward, // roll back
        transform.forward, // roll forward
        -transform.right, // roll left
        transform.right // roll right
    };

    string[] anim_Roll_triggers = {
        "roll_B",
        "roll_F",
        "roll_L",
        "roll_R"
    };
    float[] weights = new float[roll_directions.Length];
    for (int i = 0; i < roll_directions.Length; i++)
    {
        Vector3 dodgePosition = transform.position + roll_directions[i] * dodgeDistance;
        if (NavMesh.SamplePosition(dodgePosition, out NavMeshHit hit, 1.0f, NavMesh.AllAreas))
        {
            // Calculate weight based on direction, distance to player, and aggression level
            float weight_of_dirrection = Vector3.Dot(playerDirection, roll_directions[i]);
            weight_of_dirrection = (1 - Mathf.Abs(weight_of_dirrection)) * (1 - aggression_lvl);
            weights[i] = weight_of_dirrection;
        }
        else
        {
            weights[i] = -1; // Invalid direction
        }
    }
    // Choose the direction with the highest weight
    int the_best_dirrection = -1;
    float the_best_weight = -1;
    for (int i = 0; i < weights.Length; i++)
    {
        if (weights[i] > the_best_weight)
        {
            the_best_weight = weights[i];
            the_best_dirrection = i;
        }
    }

    if (the_best_dirrection != -1)
    {
        anim.SetTrigger(anim_Roll_triggers[the_best_dirrection]);
    }
}

```

Рисунок 3.62 – Логіка перекатів Гобліна

Система ухилення працює за таким принципом. Спочатку йде обчислення ваги для кожного напрямку ухилення. Цикл проходиться по напрямкам та обраховує вагу для кожної сторони, враховуючи відстань до гравця та рівень агресії. Якщо потенційний напрямок для ухилення є досяжним, то вага цього напрямку підраховується. Якщо недосяжна, вага буде дорівнювати -1. Після обчислення для всіх напрямків, код автоматично обирає напрямок з найвищою вагою для ухилення і активує відповідний тригер для

анімації. Сам перекат не може бути задіяний завжди, бо це банально було б читерство зі сторони комп'ютера. Тому є певна затримка між відновленням перекатів.

```

Code Snippet
public void Main_Attack_System()
{
    if (is_patrolling == false && Goblin_Warrior == true || Piglins == true && piglin_was_hit == true || Skeleton == true && change_position == false)
    {
        if (distance_to_player < attack_Range || distance_to_player > chasing_Range && destination_run != true) //if character is out of view range or attack range - than enemy stop
        {
            if (destination_run != true)
            {
                nav.isStopped = true;
            }

            //if(distance_to_player < chasing_Range)
            //{
            //Look_At_Player_Spherical_LERP();      //can be claimed as self-directed attack
            //}

            if (distance_to_player < attack_Range && enemy_information.IsTag("nonAttack") && SaveScript.is_invisible != true && destination_run != true)
            {
                if (is_attacking == false)
                {
                    is_attacking = true;
                    anim.SetTrigger("attack");
                    Look_At_Player_Spherical_LERP();  //little bit chunky
                }
            }

            if (distance_to_player < attack_Range && enemy_information.IsTag("attack"))
            {
                if (is_attacking == true)
                {
                    is_attacking = false;
                }
            }
        }
        else if (distance_to_player > attack_Range && enemy_information.IsTag("nonAttack") && !anim.IsInTransition(0))
        {
            if (SaveScript.is_invisible == false && destination_run == false)
            {
                Go_To_Player();
            }
        }
    }
}

```

Рисунок 3.63 – Невеличка частина коду відповідальна за виконання атаки по гравцю

Також однією з проблем при прописуванні логіки руху ворогів, було те що вони погано поверталися або зовсім не в ту сторону атакували гравця. Ця проблема була вирішена за допомогою математичної функції Spherical\_LERP. Функція Lerp в Unity означає лінійну інтерполяцію. Вона використовується для плавного переходу між двома значеннями в часі і приймає три параметри: початкове значення, кінцеве значення і число з плаваючою комою, яке представляє інтерполяцію між ними. Параметр інтерполяції зазвичай затискається між 0 і 1, де 0 повертає початкове значення, 1 повертає кінцеве значення, а будь-яке значення між ними повертає інтерпольоване значення.

### 3.3.4.2 Піглін



Рисунок 3.64 – Піглін

Повільний, сильний, розумний та небезпечний ворог. Ви можете від нього втекти проте він має більше значення шкоди та здоров'я, що робить його набагато небезпечнішим ніж Гобліни. Його унікальна спеціальність в тому, що він не атакує гравця, в якого є зброя або поки його першим не атакують. Також при отриманні значної шкоди буде рятуватися бігством намагаючись зберегти собі життя.

### 3.3.4.3 Скелет

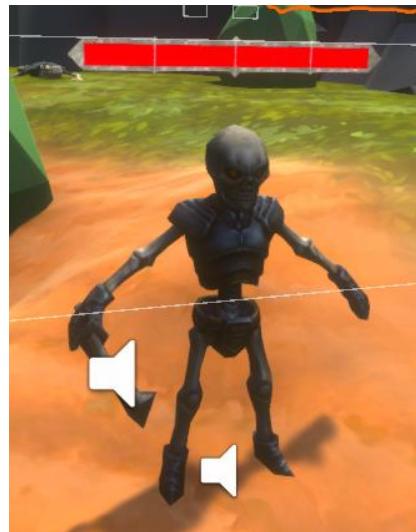


Рисунок 3.65 – Скелет

Наступний ворог – Скелет. Досить базовий для ігор даного жанру. Його основна особливість це – виклик допомоги, коли навколо немає інших супутніх ворогів і коли рівень агресії гравця перевищує певний поріг. Він також патрулює територію певну. Проте не кожен скелет може це зробити. Розпізнати його можна по габаритах – ворог, який буде більший за інших, може викликати допомогу. Коли скелет викликав допомогу – вони надалі будуть залишатися в групі і патрулювати місцевість разом.

### 3.3.4.4 Голем

Останній ворог в грі – це Голем. (рис. 3.66) Зустріти його можливо тільки в підземеллі, де ви маєте дістати піктограму для розблокування багаторазової магії.



Рисунок 3.66 – Голем

Голем має 20 відсотковий захист від магії та 15 відсоткову можливість уникнути будь якої шкоди. В нього є 3 типи атак та одна атака на бігу. Також гравець має можливість виконати «стан», запинити на пару секунд ворога. Основна задача даного ворога – це захист території, а отже він буде атакувати гравця доти, доки той не помре або не втече з даної території. Кожен з трьох типів атак залежить від того на який відстані знаходиться гравець, якщо ж гравець знаходився поруч, а потім раптово втік – то голем виконає розбіг з

ручними атаками для того щоб наздогнати супротивника. Нижче можна побачити, як виглядає анімаційний контролер для даного ворога та взаємозв'язки між ними. (рис. 3.67)

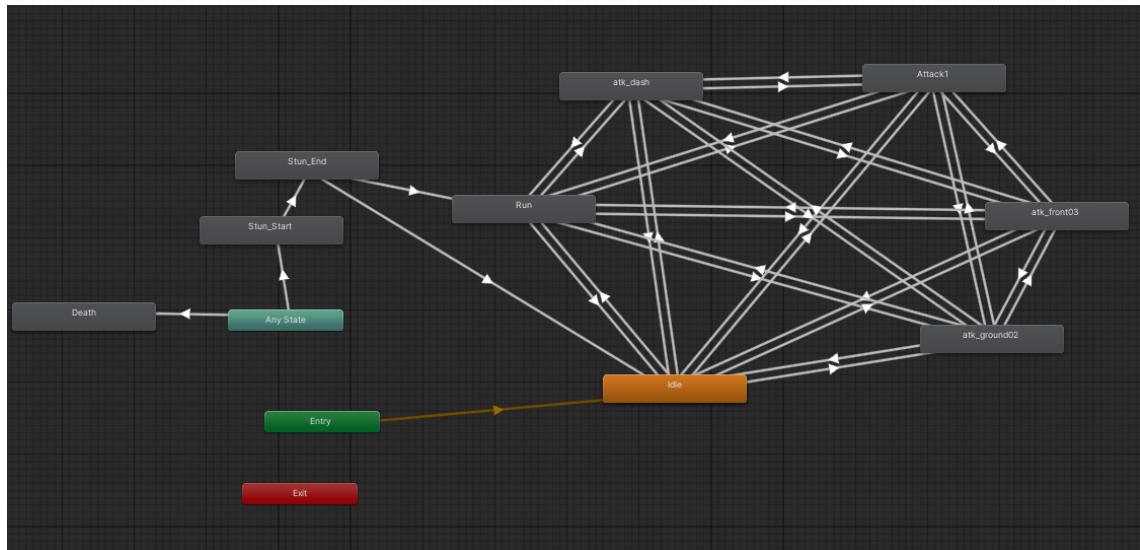


Рисунок 3.67 – Анімаційний контролер голема

### 3.3.5 Кінець гри

Так як це RPG, то в цілому в гри немає якоїсь конкретної фінальної точки. Гравець може робити все що йому забажається та стільки скільки забажається. Він може зберегти свою гру для того щоб продовжити її потім. Для цього потрібно підійти до одного з багатьох розташованих по світу і обрати в меню «Так» при питання «Чи бажаєте ви зберегти гру?». (рис. 3.68) Проте якщо він помре, все розпочнеться спочатку та перекине його на меню вибору персонажа.



Рисунок 3.68 – Меню збереження гри

### 3.4 Аналіз безпеки даних

Так як дана програма не буде виконуватися на безлічі пристройв одночасно, не взаємодіє із зовнішніми базами даних чи їх джерелами, не потребує підключення до мережі, не збирає особистих даних користувачів – то і загрози знищенню, викриттю чи крадіжки інформації немає. В результаті чого додаткові умови безпеки і аналізу не потребуються.

### Висновки до розділу

В поточному розділі бакалаврського проекту було створено пояснення для покращення розуміння того, як працює створений прототип гри. Було докладно описано принципи роботи інтерфейсу головного меню, можливості, якими володіє ігровий персонаж, вороги, світ та НІП, сторонні активності та предмети. Враховуючи поставлені вимоги до нашого прототипу, було описано мету гри, умови для перемоги або програшу, можливості для боротьби с ворогами. Описаний покращений інтелект ворогів та їх унікальні можливості, притаманні саме цьому типу ворогів.

Виходячи з матеріалів розділу можна зробити висновок, що було створено весь основний функціонал прототипу ігрового застосунку:

- Різні види ворогів зі штучним інтелектом та своїми патернами поведінки, які мають мету нанести шкоду персонажу, охороняти певні об'єкти, патрулювати, зберігати нейтралітет, зберігати своє життя або викликати допомогу;
- Стартове меню з можливістю продовження гри, початку нової, вибору персонажа;
- Інтуїтивно зрозумілий інтерфейс;
- Створення нової магії, використання зброї;
- Взаємодія з магазинами, предметами та НІП;
- Покращення та дослідження характеристик персонажа.
- Система збереження та відновлення прогресу

## 4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### 4.1 Аналіз якості ПЗ

Для забезпечення стабільної та якісної роботи ігрового застосунку було виконано декілька аналізів якості та тестування, проте так як це не веб додаток чи програмне забезпечення, яке контактує з сервером – то основний акцент було зроблено на покриття програми мануальними тестами для оцінки її коректної роботи. Також було розроблено сценарії тестування для забезпечення правильного функціоналу. Вони включають в себе тестування всіх функцій програмного забезпечення, щоб бути впевненим в стабільно правильній роботі застосунку за умов використання в реальних ситуаціях. Об'єктом випробування є мануальне тестування 3D RPG ігрового застосунку.

Метою тестування є перевірка наступних аспектів:

- функціональність: програмне забезпечення повинно належним чином виконувати всі функції, передбачені для його роботи, а також ефективно обробляти запити клієнтів і надавати коректну інформацію у відповідях;
- зручність використання: інтерфейс програми повинен бути простим і зрозумілим для користувачів, щоб користувачі могли легко скористатися всіма необхідними функціями без будь-якого надання додаткової інформації про них.

Наприклад, The Witcher 3 - пропонує інтуїтивно зрозуміле керування з багатим набором дій для бою та взаємодії зі світом. Гравці можуть використовувати магію, різноманітну зброю та предмети, а також вибирати діалоги або та ж сама DOTA 2 [15] – система керування дипломного проекту досить схожа з даною грою, де пересування гравця відбувається натисканням клавіші миші на землю, після чого персонаж пересувається в дане місце. З цього можна зробити висновок, що інтерфейс ігрового застосунку буде зручний у використанні звичайному користувачу.

## 4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення для забезпечення практичної перевірки функціональних вимог, опис відповідних тестів наведено у таблицях 4.1 – 4.15.

Таблиця 4.1 – Тест 1.1 Відкриття гри

Початковий стан системи	Користувач знаходиться на робочому столі
Вхідні дані	Немає
Опис проведення тесту	Відкрити .exe файл та перевірити чи гра запускається і чи наявне головне меню.
Очікуваний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку авторизації.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Тест 1.2 Нова гра

Початковий стан системи	Гра завантажена та головне меню активне
Вхідні дані	Кнопка «New Game» натиснута
Опис проведення тесту	Натиснути кнопку «New Game» для початку обрання персонажу.
Очікуваний результат	Відкриття меню обрання персонажу.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест 1.3 Початок пригод

Початковий стан системи	Активне меню обрання персонажу
Вхідні дані	Обраний персонаж
Опис проведення тесту	Обираючи персонажа, задайте йому клас, ім'я та натисніть кнопку «Accept»
Очікуваний результат	Завантаження ігрового світу.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест 1.4 Рух персонажа

Початковий стан системи	Гравець у початковій локації
Вхідні дані	Ліва кнопка миші
Опис проведення тесту	Натискання на землю під ногами персонажа змусить пересуватися персонажа у вказане місце.
Очікуваний результат	Персонаж рухається у відповідному напрямку.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест 1.5 Тест бою

Початковий стан системи	Гравець має зброю або магію та знаходиться поруч з ворогом.
Вхідні дані	Натискання клавіші «Z» (бій) / клавіші 1-8 (магія)
Опис проведення тесту	Натиснути клавішу атаки персонажа поруч з ворогом або використовуючи магію клавішами 1-8 направляючи на ворога.
Очікуваний результат	Ворог отримує певний рівень шкоди або гине.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 1.6 Тест інвентарю

Початковий стан системи	Гравець підняв предмет
Вхідні дані	Натиснута книга інвентарю в лівому нижньому кутку екрану.
Опис проведення тесту	Відкрити інвентар та перевірити наявність зібраного предмета
Очікуваний результат	Іконка предмету відображається в інвентарі.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 1.7 Тест діалогу з НП

Початковий стан системи	Гравець підійшов до НП на коротку відстань
Вхідні дані	Немає
Опис проведення тесту	Взаємодія з НП для початку діалогу при знаходженні на достатньо близькій відстані один від одного.
Очікуваний результат	Відображення діалогового вікна.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест 1.8 Тест збереження гри

Початковий стан системи	Гравець підійшов до багаття
Вхідні дані	Немає
Опис проведення тесту	Зберегти гру та всю інформацію про гравця.
Очікуваний результат	Гра зберігається та можливість завантажити інформацію про ігрову сесію з головного меню гри.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 1.9 Тест завантаження збереження гри

Початковий стан системи	Гравець в головному меню
Вхідні дані	Натиснута кнопка «Continue»
Опис проведення тесту	При наявності збережень кнопка продовження гри буде активна і гравець може завантажити свою останню сесію.
Очікуваний результат	Завантаження останньої ігрової сесії.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Тест 1.10 Тест створення магії

Початковий стан системи	Гравець має чарівну книгу та достатню кількість інгредієнтів.
Вхідні дані	Інгредієнти
Опис проведення тесту	Для створення магії треба відкрити книгу в меню інвентару, обрати потрібне закляття та шляхом натискання на інгредієнти в інвентарі по прикладу з книги – додати всі потрібні елементи та натиснути «Create»
Очікуваний результат	Створення заклинання.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.11 – Тест 1.11 Тест прокачування характеристик персонажу

Початковий стан системи	Гравець знищив достатню кількість ворогів та отримав новий рівень і бали прокачки.
Вхідні дані	Бали прокачки
Опис проведення тесту	Відкривши меню, обравши секцію «Stats» та натиснувши на плюси біля однієї з трьох індикаторів сили, інтелекту або стаміни.
Очікуваний результат	Підвищення характеристик та візуальна зміна індикатора
Фактичний результат	Збігається з очікуваним.

Таблиця 4.12 – Тест 1.12 Тест взаємодія з Пігліном

Початковий стан системи	Гравець потрапив у поле зору ворога або атакував його
Вхідні дані	Інформація про рівень агресії, позицію гравця, наявність зброї, активність заклять.
Опис проведення тесту	При відсутності зброї ворог повинен атакувати гравця та переслідувати, якщо ворог отримав значну кількість пошкоджень – він рятує своє життя бігством.
Очікуваний результат	Ворог атакує, очікує або втікає.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.13 – Тест 1.13 Тест взаємодія з Гобліном

Початковий стан системи	Гравець потрапив у поле зору ворога
Вхідні дані	Інформація про рівень агресії, позицію гравця, наявність зброї, активність заклять та агресія самого Гобліна.
Опис проведення тесту	Гоблін патрулює територію і при потраплянні гравця в поле зору – почне його атакувати. Якщо гравець спробує вдарити
Очікуваний результат	Ворог атакує, очікує або втікає.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.14 – Тест 1.14 Тест взаємодія з Скелетом

Початковий стан системи	Гравець потрапив у поле зору ворога
Вхідні дані	Інформація про рівень агресії, позицію гравця, наявність зброї, активність заклять та статус Скелету.
Опис проведення тесту	Скелет патрулює територію і при потраплянні гравця в поле зору – він почне його атакувати. Проте якщо навколо немає інших ворогів, а рівень агресії головного героя зашкалює – скелет викликає допомогу у вигляді ще 2 скелетів.
Очікуваний результат	З'являється підкріплення у вигляді 2 скелетів. Вони діють та пересуваються у групі.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.15 – Тест 1.15 Тест взаємодія з Големом

Початковий стан системи	Гравець потрапив у підземелля
Вхідні дані	Інформація про рівень агресії, позицію гравця, наявність зброї, активність заклять.
Опис проведення тесту	В залежності від позиції гравця, дальності атаки та агресії голем виконує 4 різні атаки та намагається знищити суперника.
Очікуваний результат	Голем виконує різні атаки та переслідує гравця.
Фактичний результат	Збігається з очікуваним.

#### 4.3 Опис контрольного прикладу

Для контрольного прикладу було обрано процес початку гри та взаємодії з ворогами.

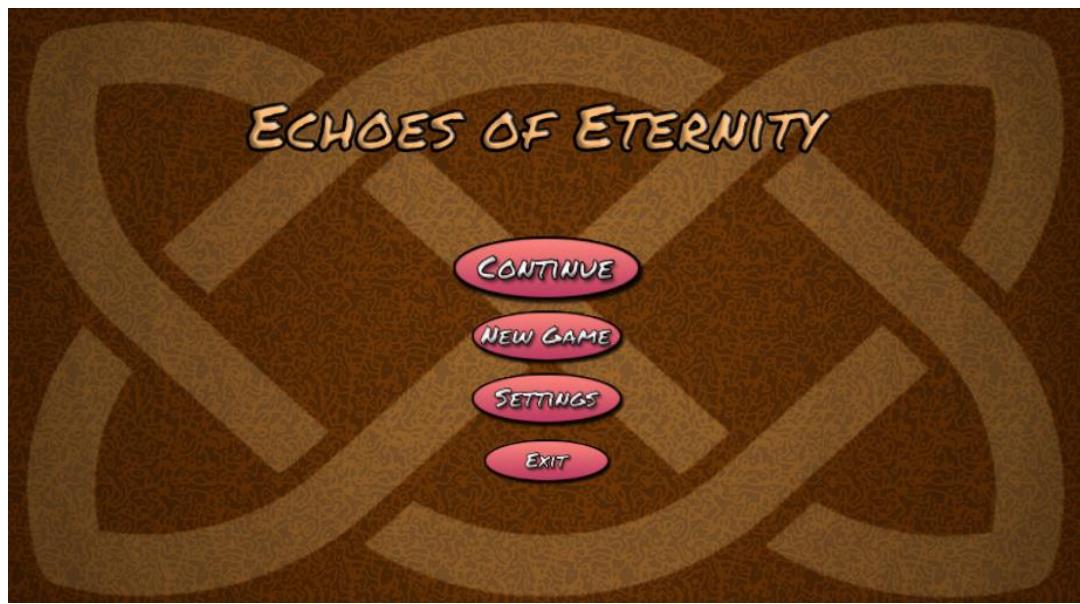


Рисунок 4.1 – Головне меню гри

При відкритті гри користувач бачить головне меню гри (рис. 4.1). Натискаючи на кнопку початку нової гри «New Game», Відкривається меню

обрання персонажу. (рис.4.2) Отже оберемо бажаний варіант, клас та задамо ім'я. Далі натиснемо кнопку «Accept» і розпочнемо гру.



Рисунок 4.2 – Меню обрання персонажа

Після цього гравець потрапляє у відкритий світ. Де він може пересуватися та змінювати кут камери. (рис. 4.3)



Рисунок 4.3 – Вигляд камери та пересування персонажу

Наступний крок – це взаємодія з ворогами. Перший ворог це Піглін. Як бачите на рисунку 4.4, без зброї він починає нас атакувати, як тільки ми екіпіруємо ніж – агресія припиняється. Якщо ми вдаримо його – він вдарить нас у відповідь, а при низькому рівні здоров'я спробує втекти рятуючи своє життя.

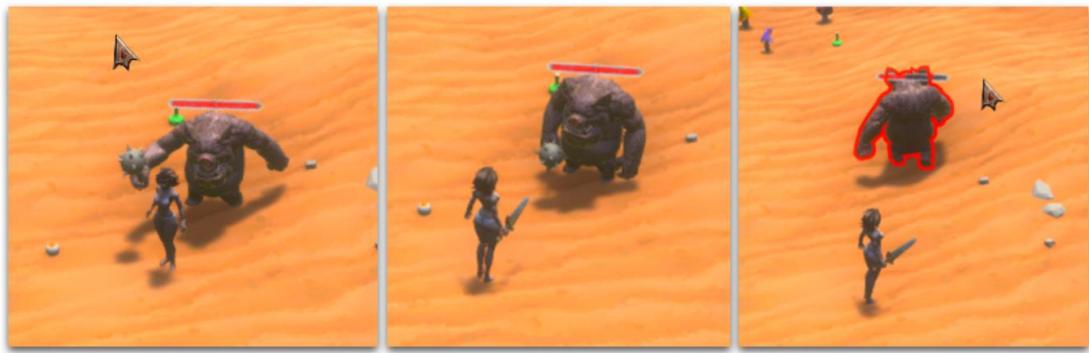


Рисунок 4.4 – Стадії поведінки Пігліна

Далі в світі ви зможете зустріти Скелета, який буде патрулювати певну територію. Як тільки ви потрапите в його поле зору – він почне вас переслідувати та атакувати, від цього ваш скритий показник агресії буде рости, що змусить його, в певний момент часу коли поблизу не буде інших ворогів – викликати підмогу собі. Після цього вони будуть пересуватися та патрулювати територію в групі (рис. 4.5), що робить їх небезпечними ворогами.



Рисунок 4.5 – Етапи поведінки Скелета

Третій ворог цього світу, як вже зазначалося – це Гоблін. (рис. 4.6) Він пересувається досить швидко та патрулює територію навколо певного об'єкту. Він також буде намагатися знищити супротивника, за умови перебування його

в полі зору, але в нього є дві унікальні особливості, швидкість та можливість уникати атак гравця. В гобліна також є скритий показник агресії, який поступово росте і при досягненні максимального значення – ухилення відбуватиметься в сторону гравця.



Рисунок 4.6 – Етапи поведінки Гобліну

Останній та найбільш небезпечний – Голем, має різноманітні атаки та наносить значну шкоду супротивнику (рис. 4.7) . Поведінка зав'язана на позиції гравця та його агресії. Якщо агресія значна, то голем з більшою ймовірністю захиститься від ворожої атаки.



Рисунок 4.7 – Різноманітність поведінки Голема

## **Висновки до розділу**

У цьому розділі було виконано детальний аналіз якості програмного забезпечення 3D RPG ігрового застосунку, створеного на платформі Unity. Основний акцент було зроблено на мануальному тестуванні, що дозволило оцінити коректність роботи застосунку в умовах, максимально наблизених до реального використання. Було розроблено та виконано сценарії тестування,

спрямовані на перевірку функціональності, зручності використання та надійності збереження даних. Процеси тестування були організовані відповідно до заздалегідь підготовленої програми та методики, що забезпечило систематичний підхід до виявлення та усунення потенційних проблем. Результати тестування були документовані у вигляді таблиць, що дозволило здійснити оцінку кожного аспекту роботи застосунку. Детально було описано процес початку гри та взаємодії з ворогами, включаючи різні стадії поведінки інтерактивних персонажів, таких як Піглін, Скелет, Гоблін та Голем.

Загалом, проведене тестування підтвердило правильність функціонування ігрового застосунку.

## 5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### 5.1 Розгортання програмного забезпечення

З самого початку потрібно виконати білд програми. Для цього в налаштуваннях Unity треба обрати Build Settings та виставити налаштування, які можна побачити на рисунку 5.1. Після цього потрібно перейти в налаштування гравця «Player Settings» та виставити більш детальні параметри. Їх можна побачити на рисунку 5.2. Під кінець, коли всі параметри задані правильно – натискається кнопка Build та обирається місце, куди відбудеться конфігурація застосунку. В результаті з'являється програму для користування, а саме .exe файл (рис. 5.3) – все що потрібно для розгортання застосунку. При подальшій розробці гри, можлива реалізація та випуск гри на окремих платформах таких як Steam або Epic Game Store, проте для цього потрібно заручитися підтримкою видавців, що в нашому випадку неможливо.

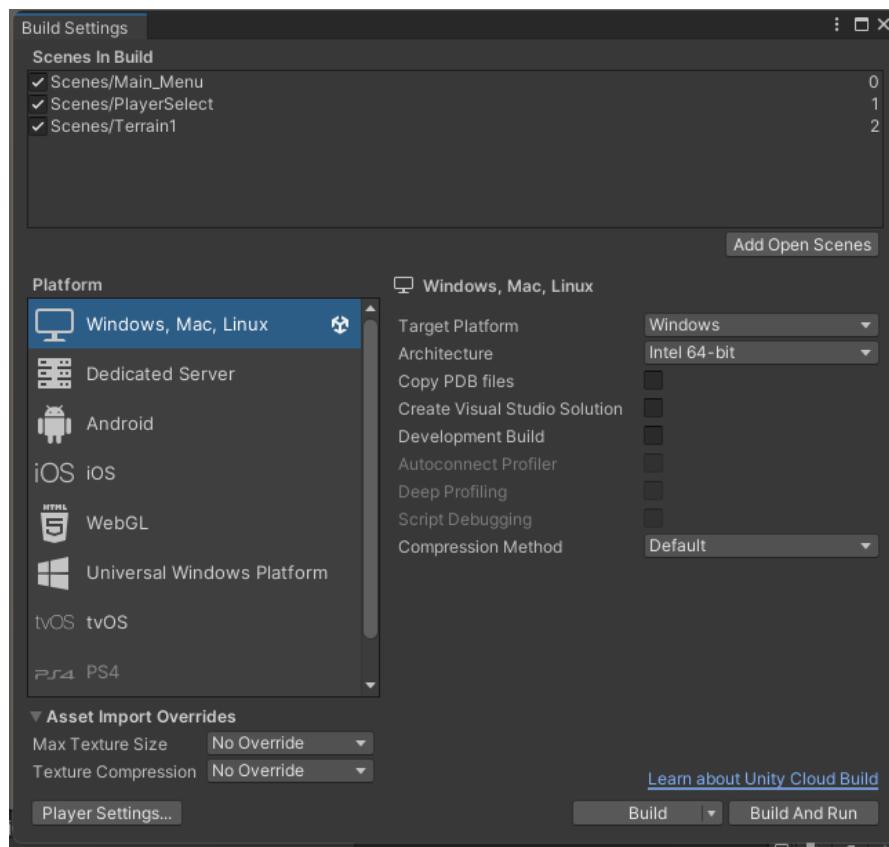


Рисунок 5.1 – Базові налаштування білда

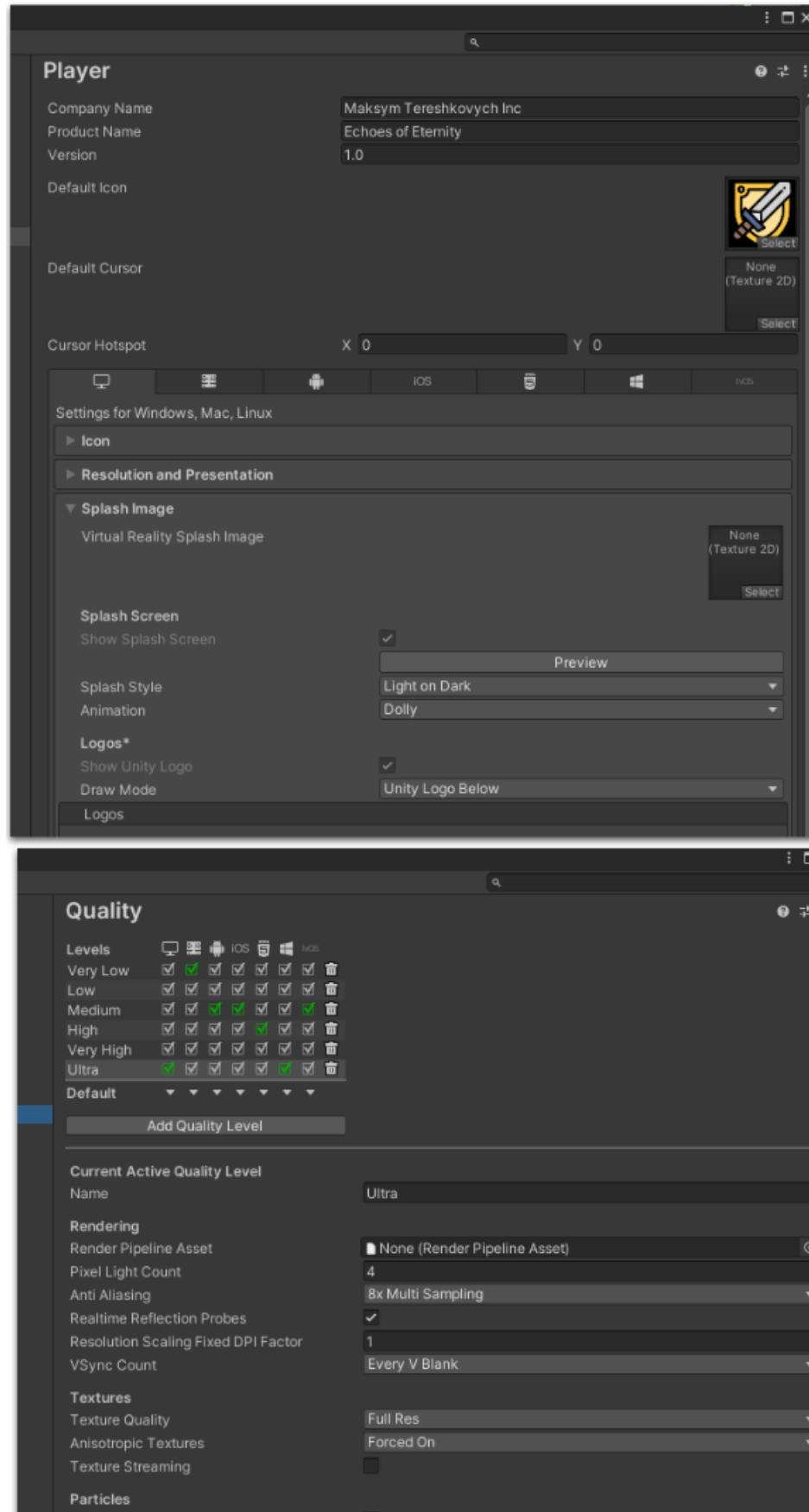


Рисунок 5.2 – Детальні налаштування білда

Ім'я	Дата змінення	Тип	Розмір
Echoes of Eternity_Data	02.06.2024 15:01	Папка файлів	
MonoBleedingEdge	02.06.2024 15:01	Папка файлів	
<b>Echoes of Eternity</b>	<b>02.06.2024 15:01</b>	<b>Застосунок</b>	<b>639 КБ</b>
UnityCrashHandler64	02.06.2024 15:01	Застосунок	1 098 КБ
UnityPlayer.dll	02.06.2024 15:01	Розширення заст...	28 358 КБ

Рисунок 5.3 – Вихідний файл білду

## 5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі.

Супровід програмного забезпечення та корекція багів відбувається за допомогою білда на платформі Unity. При внесенні нових правок до вже існуючого проекту потрібно знову проробити інструкцію з пункту 5.1 і всі внесені зміни застосуються до ігрового програмного забезпечення також будуть збережені всі ігрові сеанси, що дасть змогу продовжити гру з місця, де гравець зупинився.

### Висновки до розділу

В даному розділі наведено інформацію про розгортання ігрового застосунку для нашої гри, що є дуже простим процесом, який вимагає лише наявності виконуваного файлу (.exe), без потреби у сторонніх сервісах. Це забезпечує легкість установки та запуску гри кінцевим користувачем. У майбутньому, існує потенціал для розширення доступності гри шляхом її випуску на популярних ігрових платформах, таких як Steam або Epic Game Store. Однак, це вимагатиме підтримки видавців, що наразі є недосяжним для нас. Супровід, корекція помилок та оновлення ПЗ відбуваються через білди, створені на платформі Unity, що дозволяє нам швидко реагувати на звіти про баги та вдосконалювати гру, підтримуючи високий рівень задоволеності користувачів.

## ВИСНОВКИ

Розробка ігрового застосунку на дипломному проєкті дала можливість заглибитися у вивчення розробки відеоігор та покращити навички в розумінні та створенні всіх аспектів гри, особливо поведінки інтелектуальних агентів. Проаналізувавши предметну область та аналогічні ігрові застосунки дозволило правильно обрати ігровий рушій, мову програмування, жанр та дізнатися, які аспекти поведінки ворогів потребують покращення та доопрацювання.

Метою дипломного проєкту було розширення спектру моделей поведінки інтелектуальних агентів в ігровому застосунку 3D RPG для урізноманітнення ігрового досвіду користувача. При плануванні було детально проаналізовано концепцію гри, жанр, ключові механіки та формат, які присутні на ринку відеоігор. Виконано відбір потрібних графічних пакетів, асsetів, звуків, анімацій, створено власні об'єкти, скрипти та механіки, які необхідні для налаштування та відтворення основного інтерфейсу гри. Модель поведінки та пошуку ворогів базувалась на принципі A\* алгоритму та системі відтворення променів, що імітувало реальне середовище з реалістичною поведінкою ворогів.

Для досягнення мети було сформульовано ряд функціональних задач. Вони мають наступний стан вирішеності:

**Реалізація інтелекту ворогів зі своїми особливими спектрами поведінки** – система реалізована повністю в ПЗ. В ігровому застосунку наявні 4 види ворогів кожен зі своєю унікальною поведінкою та особливостями.

**Реалізація ігрового інтерфейсу** – система реалізована повністю в ПЗ. Гравець має можливість користуватися інтерфейсом застосунку таким як, меню, інвентар, магазини, об'єкти і тд.

**Реалізація механік бою** – система реалізована повністю в ПЗ. Гравець може взаємодіяти з усіма ворогами в грі, знищувати їх, наносити їм шкоду, використовувати для цього зброю, закляття та екіпірувати броню для захисту себе.

**Реалізація функціоналу інвентарю, магазинів та ігрової економіки –** система реалізована повністю в ПЗ. Гравець може купувати та використовувати всі предмети, які наявні в грі, для цього існують магазини та інвентар. Інвентар в свою чергу існує, як місце збереження та колекціонування об'єктів, отриманих гравцем. Також там можна знайти поточні задання, карту, меню зброї, характеристик, магії та можливість прокачувати свого персонажа.

**Реалізація функціонали створення та використання магії –** система реалізована повністю в ПЗ. Гравець має можливість створювати магію та заклинання базуючись на зібраних до цього інгредієнтах. Кожне закляття дає певні переваги гравцю при проходженні гри. В грі є 7 видів магії та 6 видів заклять.

**Реалізація функціоналу взаємодії з ігровими об'єктами –** система реалізована повністю в ПЗ. Гравець може взаємодіяти з великою кількістю об'єктів, які розкидані по всьому світу. Також є можливість взаємодіяти зі скринями, НПС, ящиками і тд.

Для персонажів та ворогів в проекті використовувалися об'єкти такі як, Animator для створення, налаштування та корекції анімацій, та ряд інших інструментів.

Фінальним етапом являлася розробка керівництва для користувача та проведення тестування, яке підтвердило відповідність розробленого програмного забезпечення функціональним вимогам.

Цей проект відкриває широкі перспективи для удосконалення ігрового середовища. Реалізація проекту дозволила глибше ознайомитися з принципами роботи передових ігрових платформ, а також вдосконалити уміння управління складними системами та використання алгоритмів. Набуті знання та досвід стануть міцним фундаментом для майбутньої кар'єри у сфері створення відеоігор.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- 1) Pillars of Eternity on Steam. *Welcome to Steam.* URL: [https://store.steampowered.com/app/291650/Pillars\\_of\\_Eternity/](https://store.steampowered.com/app/291650/Pillars_of_Eternity/) (date of access: 01.06.2024).
- 2) The Witcher Universe | Action-Adventure RPGs. *The Witcher Universe* / *Action-Adventure RPGs.* URL: <https://www.thewitcher.com/us/en/witcher3> (date of access: 01.06.2024).
- 3) Технологія трасування променів: що це, навіщо потрібно і чим відрізняється від звичайного RTX? | Каталог цін Е-Katalog. *e-katalog.pl* - порівняння, відгуки, ціни в інтернет-магазинах. URL: <https://e-katalog.pl/ua/post/5066/298-ray-tracing-technology-what-is-it-why-is-it-needed-and-how-is-it-different-from-regular-rtx/> (дата звернення: 01.06.2024).
- 4) NVIDIA анонсувала DLSS 3.5 – покращений ШІ-апскейлінг із реконструкцією трасування променів. ITC.ua. URL: <https://itc.ua/ua/novini/nvidia-anonsuvala-dlss-3-5-pokrashhenyj-shi-apskejling-iz-rekonstruktsiyeyu-trasuvannya-promeniv/> (дата звернення: 01.06.2024).
- 5) Граєте в ігри на смартфоні або комп’ютері? Це нормально, таким займається 40% населення Землі. hromadske.ua. URL: <https://hromadske.ua/posts/grayete-v-igri-na-smartfoni-abo-kompyuteri-ce-normalno-takim-zajmayetsya-40-naselenna-zemli> (дата звернення: 01.06.2024).
- 6) HELLDIVERS™ 2 on Steam. *Welcome to Steam.* URL: [https://store.steampowered.com/app/553850/HELLDIVERS\\_2/](https://store.steampowered.com/app/553850/HELLDIVERS_2/) (date of access: 01.06.2024).
- 7) Steam, The Ultimate Online Game Platform. *Welcome to Steam.* URL: <https://store.steampowered.com/about/> (date of access: 01.06.2024).
- 8) Contributors to Wikimedia projects. Epic Games - Wikipedia. Wikipedia, the free encyclopedia. URL: [https://en.wikipedia.org/wiki/Epic\\_Games](https://en.wikipedia.org/wiki/Epic_Games) (date of access: 01.06.2024).

9) III R. E. W. What Is DLC? Everything Players and Parents Need to Know. Lifewire. URL: <https://www.lifewire.com/what-is-dlc-in-gaming-how-does-it-work-4707377> (date of access: 01.06.2024).

10) POLYGON Dungeons - Low Poly 3D Art by Synty | 3D Dungeons | Unity Asset Store. Unity Asset Store - The Best Assets for Game Making. URL: <https://assetstore.unity.com/packages/3d/enviroments/dungeons/polygon-dungeons-low-poly-3d-art-by-synty-102677> (date of access: 01.06.2024).

11) Medieval house modular lite | 3D Fantasy | Unity Asset Store. Unity Asset Store - The Best Assets for Game Making. URL: <https://assetstore.unity.com/packages/3d/environments/fantasy/medieval-house-modular-lite-189718> (date of access: 01.06.2024).

12) Contributors to Wikimedia projects. Video game - Wikipedia. Wikipedia, the free encyclopedia. URL: [https://en.wikipedia.org/wiki/Video\\_game](https://en.wikipedia.org/wiki/Video_game) (date of access: 02.06.2024).

13) Contributors to Wikimedia projects. List of best-selling video games - Wikipedia. Wikipedia, the free encyclopedia. URL: [https://en.wikipedia.org/wiki/List\\_of\\_best-selling\\_video\\_games](https://en.wikipedia.org/wiki/List_of_best-selling_video_games) (date of access: 02.06.2024).

14) Що краще моноліт чи мікросервіси? | IAMPM. IAMPM. URL: <https://iampm.club/ua/blog/shho-krashhe-monolit-chi-mikroservisi-yak-obrati-arhitekturu-projektu/> (дата звернення: 02.06.2024).

15) Dota 2. Dota 2. URL: <https://www.dota2.com/home> (date of access: 02.06.2024).

**ДОДАТКИ**

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

\_\_\_\_\_ Едуард ЖАРИКОВ

“\_\_\_” \_\_\_\_\_ 2024 р.

**ІГРОВИЙ ЗАСТОСУНОК МОДЕЛЮВАННЯ ПОВЕДІНКИ  
ІНТЕЛЕКТУАЛЬНИХ АГЕНТІВ У 3D RPG З ВИКОРИСТАННЯМ  
ІГРОВОГО РУШІЯ UNITY.**

**Текст програми**

КПІ.ІТ-0223.045440.03.12

“ПОГОДЖЕНО”

Керівник проєкту:

\_\_\_\_\_ Олексій ФІНОГЕНОВ

Нормоконтроль:

Виконавець:

\_\_\_\_\_ Максим ГОЛОВЧЕНКО

\_\_\_\_\_ Максим ТЕРЕШКОВИЧ

Київ – 2024

**Посилання на репозиторій з повним текстом програмного коду:**  
<https://github.com/mcmcmax437/3D-RPG-Diploma-Thesis>

## 1.1 Реалізація інтелекту ворогів зі своїми особливими спектрами поведінки.

### 1.1.1 Файл *EnemyMovement.cs*:

```
public class EnemyMovement : MonoBehaviour
{
    public bool Goblin_Warrior = false;
    public bool Poglins = false;
    public bool Skeleton = false;
    public bool temp_Priority = false; // temp to check
    private bool can_call_support = false;
    private Vector3 buffed_Skeleton = new Vector3(15.0f, 15.0f, 15.0f);
    private int buffed_probability = 10;
    private bool sup_skill_used = false;
    private bool change_position = false;
    private float sup_skill_CD = 10f;
    private int amount_of_reinforcement = 2;
    public GameObject support_enemy;

    public GameObject Loot_from_Enemy;

    public GameObject current_enemy;
    private bool is_outliner_active = false;

    private AnimatorStateInfo enemy_information;
    private NavMeshAgent nav;
    private Animator anim;
    private float x;
    private float z;
    private float velocitySpeed;
    public GameObject player;
    private float distance_to_player;
    private bool is_attacking;
    public float attack_Range = 2.0f;
    public float chasing_Range = 12.0f; //range in which enemy will run after character
    public float rotation_speed = 500.0f; //perfect
    private float stop_distance = 2f;
    private float group_brain_radius = 10f;
```

```

public Transform patrol_main_obj;
public float patrol_radius = 15.0f;
public float wait_time_at_point = 2.0f;

private Vector3 targetPoint;
private bool is_waiting;
private float wait_timer;
private bool is_patroling = true;

private int maxHP;

public int full_HP = 100;
private int curr_HP;

private int fear_lvl = 100;
private int fear_lvl_curr;

private bool enemy_is_alive = true;

private AudioSource audio_Player;
public AudioClip[] get_Hit_SFX;

public GameObject bar_Container;
public Image HP_bar;
private float fillHealth;
public GameObject main_camera;

private bool destination_run = false;

private Vector3 escape_point;
public Transform[] escape_target_point;

private bool roll_out = false;
private bool roll_is_active = false;
public float dodgeDistance = 5f;
public float aggression_lvl = 0.5f; // 0 (passive) to 1 (aggressive)
private bool playerNearby = false;
private float aggression_increase = 0.05f;
private float aggression_decrease = 0.025f;
public float max_aggression = 1.0f;
public float min_aggression = 0.0f;

public bool piglin_was_hit = false;
private bool player_is_armorless = true;
private bool should_reset_armor_trigger = true;

public float distance_of_ray = 12f;

```

```

public float time_for_search = 3f;
private Vector3 last_seen_position;
private float search_Timer;
private bool player_is_inSight;
private bool look_for_player;
private bool reset_piglins_chase_range = false;

// Start is called before the first frame update
void Start()
{
    audio_Player = GetComponent<

```

```

        }

    }

// Update is called once per frame
void Update()
{
    if (main_camera == null)
    {
        main_camera = GameObject.Find("Main Camera");
    }
    if (patrol_main_obj == null)
    {
        new WaitForSeconds(1);
    }
    if(SaveScript.weapon_index != -1)
    {
        reset_piglins_chase_range = false;
    }
    else
    {
        reset_piglins_chase_range = true;
        piglin_was_hit = true;
    }
    if(reset_piglins_chase_range == true)
    {
        chasing_Range = 12.0f;

    }
    if (reset_piglins_chase_range == false && Piglins == true && chasing_Range != 60)
    {
        chasing_Range = 0.0f;
    }
}

bar.Container.transform.LookAt(main_camera.transform.position);

if (Input.GetKeyDown(KeyCode.Z) && distance_to_player < 5f && SaveScript.stamina > 0.2)
{
    roll_out = true;
}

if (enemy_is_alive == true)
{
    Enemy_Outline();
    if (player == null)

```

```

    {
        player = GameObject.FindGameObjectWithTag("Player");
    }
    Enemy_Running();

    enemy_information = anim.GetCurrentAnimatorStateInfo(0);
    distance_to_player = Vector3.Distance(transform.position, player.transform.position);

    if (destination_run == true && Pglins == true)
    {
        chasing_Range = 0;
    }

    if (distance_to_player <= chasing_Range && destination_run == false)
    {
        Check_If_Player_is_InSight();

        if (player_is_inSight == true)
        {
            //last_seen_position = player.transform.position;
            search_Timer = 0f;
            nav.destination = player.transform.position;
            Main_Attack_System();
        }
        else if (!player_is_inSight && last_seen_position != Vector3.zero)
        {
            NavMeshPath path = new NavMeshPath();
            nav.CalculatePath(last_seen_position, path);
            if (path.status != NavMeshPathStatus.PathComplete)
            {
                Look_Aroun_Yourself();
            }
            else if (look_for_player == true)
            {
                search_Timer += Time.deltaTime;
                if (search_Timer >= time_for_search)
                {
                    look_for_player = false;
                    search_Timer = 0f;
                }
                //Debug.Log(search_Timer);
                Look_Aroun_Yourself();
            }
        }
    }
}

```

```

if (Goblin_Warrior == true && look_for_player == false)
{
    if(patrol_main_obj != null)
    {
        Patrol();
    }
    Correct_Aggression();
}

if (distance_to_player <= chasing_Range)
{
    is_patrolling = false;
}

if (roll_out == true && roll_is_active == false)
{
    roll_is_active = true;
    Roll();
    StartCoroutine(Reset_Roll_Triger());
}

//Debug.Log(Skeleton + " " + can_call_support + " " + sup_skill_used);
if (Skeleton == true && can_call_support == true && sup_skill_used == false)
{
    bool enemy_is_near_skeleton = Search_Enemy_Near_Skeleton();
    if (enemy_is_near_skeleton == false && distance_to_player <= 9f && SaveScript.agression_lvl > 0.7f)
    {
        sup_skill_used = true;
        SaveScript.agression_lvl -= 0.5f;
        anim.SetTrigger("skill");
        Spawn_Reinforcement();
        StartCoroutine(Reset_Sup_Skill());
        change_position = false;
    }
}

//curr_HP = was
//full_hp - are
if (curr_HP > full_HP)
{
    anim.SetTrigger("hit");
    curr_HP = full_HP;
    RandomAudio_Hit();
    fillHealth = Convert.ToSingle(full_HP) / Convert.ToSingle(maxHP);
    Debug.Log(fillHealth);
    HP_bar.fillAmount = fillHealth;
}

```

```

        if (GetComponent<Enemy_Type>().enemyType == Enemy_Type.EnemyType.Piglin)
        {
            piglin_was_hit = true;
            chasing_Range = 60f;
            StartCoroutine(Reset_Piglin_Renge());
        }

    }

    if (full_HP < maxHP / 2 && Poglins == true && destination_run == false)
    {
        destination_run = true;
        chasing_Range = 0;
        //Debug.Log("RUN AWAY");
        Run_Away();
    }

}

Vector3 dest = nav.destination;
if (Vector3.Distance(current_enemy.transform.position, dest) <= 1.0f)
{
    StartCoroutine(Reset_RunAwayTrigger());
}

//Debug.Log(nav.isStopped);
//Debug.Log(Vector3.Distance(current_enemy.transform.position, dest));

if (full_HP <= 1 && enemy_is_alive == true)
{
    Enemy_is_Dead();
}

}

public void Main_Attack_System()
{
    if (is_patroling == false && Goblin_Warrior == true || Poglins == true && piglin_was_hit == true || Skeleton ==
true && change_position == false)
    {
        if (distance_to_player < attack_Range || distance_to_player > chasing_Range && destination_run != true) //if
character is out of view range or attack range - than enemy stop
        {
            if (destination_run != true)
            {
                nav.isStopped = true;
            }
        }
    }
}

```

```

        }

        //if(distance_to_player < chasing_Range)
        //{
        //Look_At_Player_Spherical_LERP();    //can be claimed as self-directed attack
        //}

        if (distance_to_player < attack_Range && enemy_information.IsTag("nonAttack") &&
SaveScript.is_invisible != true && destination_run != true)
{
    if (is_attacking == false)
    {

        is_attacking = true;
        anim.SetTrigger("attack");
        Look_At_Player_Spherical_LERP(); //little bit chunky
    }
}

if (distance_to_player < attack_Range && enemy_information.IsTag("attack"))
{

    if (is_attacking == true)
    {
        is_attacking = false;
    }
}

else if (distance_to_player > attack_Range && enemy_information.IsTag("nonAttack") &&
!anim.IsInTransition(0))
{
    if (SaveScript.is_invisible == false && destination_run == false)
    {
        Go_To_Player();
    }
}

public void Go_To_Player()
{
    NavMeshPath path = new NavMeshPath();
    if (NavMesh.CalculatePath(transform.position, player.transform.position, NavMesh.AllAreas, path))
}

```

```

{
    if (path.status == NavMeshPathStatus.PathComplete)
    {
        nav.destination = player.transform.position;
        nav.isStopped = false;
    }
    else if (path.status == NavMeshPathStatus.PathPartial)
    {
        nav.destination = path.corners[path.corners.Length - 1];
        nav.isStopped = false;
    }
    else
    {
        nav.isStopped = true;
    }
}
else
{
    nav.isStopped = true;
}

if (nav.isStopped && nav.velocity.sqrMagnitude < 0.1f)
{
    nav.speed = 0;
}
else
{
    nav.speed = 3.5f;
}

if(Piglins == true)
{
    nav.stoppingDistance = 2f;
}
else
{
    nav.stoppingDistance = stop_distance;
}

}

public void Look_At_Player_Spherical_LERP()
{
    Vector3 Pos = (player.transform.position - transform.position).normalized;
    Quaternion PosRotation = Quaternion.LookRotation(new Vector3(Pos.x, 0, Pos.z));
    transform.rotation = Quaternion.Slerp(transform.rotation, PosRotation, Time.deltaTime * rotation_speed);
}

```

```

public void Enemy_is_Dead()
{
    SaveScript.agression_lvl = SaveScript.agression_lvl + 0.2f;
    enemy_is_alive = false;
    nav.isStopped = true;
    anim.SetTrigger("death");
    SaveScript.amount_of_chasing_enemies--;
    current_enemy.GetComponent<Outline>().enabled = false;
    is_outliner_active = false;
    nav.avoidancePriority = 1;
    StartCoroutine(Loot_Spawn());
}

public void Enemy_Outline()
{
    //outline
    if (is_outliner_active == false)
    {
        is_outliner_active = true;
        if (SaveScript.spell_target == current_enemy)
        {
            current_enemy.GetComponent<Outline>().enabled = true;
        }
    }
    if (is_outliner_active == true)
    {
        if (SaveScript.spell_target != current_enemy)
        {
            current_enemy.GetComponent<Outline>().enabled = false;
            is_outliner_active = false;
        }
    }
    //
}

public void Enemy_Running()
{
    x = nav.velocity.x;
    z = nav.velocity.z;
    velocitySpeed = new Vector2(x, z).magnitude;
    // velocitySpeed = x+z;
    if (velocitySpeed == 0)
    {
        anim.SetBool("running", false);
        // Debug.Log("RUN = " + check);
    }
    else if (velocitySpeed != 0)
}

```

```

    {

        anim.SetBool("running", true);
        // check = anim.GetBool("running");
        is_attacking = false;
        //Debug.Log("running = " + check);

    }

}

public void RandomAudio_Hit()
{
    int randomNumber = UnityEngine.Random.Range(1, 101);
    if (randomNumber > 0 && randomNumber < 33)
    {
        audio_Player.clip = get_Hit_SFX[0];
    }
    else if (randomNumber >= 33 && randomNumber < 66)
    {
        audio_Player.clip = get_Hit_SFX[1];
    }
    else if (randomNumber >= 66 && randomNumber < 101)
    {
        audio_Player.clip = get_Hit_SFX[2];
    }
    audio_Player.Play();
}

IEnumerator Loot_Spawn()
{
    Enemy_Type enemy_type = GetComponent<Enemy_Type>();
    if (enemy_type.enemyType == Enemy_Type.EnemyType.Skelet)
    {
        yield return new WaitForSeconds(2);
    }
    else
    {
        yield return new WaitForSeconds(1);
    }
    Instantiate(Loot_from_Enemy, transform.position, transform.rotation);
    SaveScript.killed_enemy++;
    Destroy(gameObject, 0.2f);
}

public void Run_Away()
{
    anim.SetBool("running", true);
}

```

```

nav.isStopped = false;

//int pos = Random.Range(0, 3);
//nav.destination = escape_target_point[pos].transform.position;
Calculate_Escape_Point();
nav.speed = 1.8f;
nav.destination = escape_point;

}

IEnumerator Reset_RunAwayTrigger()
{
    yield return new WaitForSeconds(5);
    destination_run = false;
}

IEnumerator Reset_Roll_Triger()
{
    yield return new WaitForSeconds(3f);
    roll_out = false;
    roll_is_active = false;
}

IEnumerator Reset_Piglin_Renge()
{
    yield return new WaitForSeconds(7f);
    Look_At_Player_Spherical_LERP();
    piglin_was_hit = false;
    if(SaveScript.weapon_index != -1)
    {
        chasing_Range = 3f;
    }
    else
    {
        chasing_Range = 12f;
    }

}

IEnumerator Reset_Sup_Skill()
{
    yield return new WaitForSeconds(sup_skill_CD);
    sup_skill_used = false;
}

IEnumerator Wait_and_Attack()
{
    yield return new WaitForSeconds(10f);
}

```

```

        Main_Attack_System();
    }

    public void Calculate_Escape_Point()
    {

        Vector3 escape_dir = Vector3.zero;
        float max_escape_distance = 0f;
        Vector3 player_dir = (player.transform.position - transform.position).normalized;

        for (int i = 0; i < 360; i += 5)
        {
            Vector3 new_direction = Quaternion.Euler(0, i, 0) * transform.forward;
            if (Vector3.Dot(new_direction.normalized, player_dir) < 0)
            {
                NavMeshHit hit;
                if (NavMesh.Raycast(transform.position, transform.position + new_direction * 100f, out hit,
NavMesh.AllAreas))
                {
                    float distance = Vector3.Distance(transform.position, hit.position);
                    if (distance > max_escape_distance)
                    {
                        max_escape_distance = distance;
                        escape_dir = new_direction;
                    }
                }
            }
        }
        if (max_escape_distance > 0f && escape_dir != Vector3.zero)
        {
            NavMeshHit ray_hit_for_escape;
            if (NavMesh.SamplePosition(transform.position + escape_dir * max_escape_distance, out ray_hit_for_escape,
max_escape_distance, NavMesh.AllAreas))
            {
                escape_point = ray_hit_for_escape.position;
            }
            else
            {
                escape_point = transform.position;
            }
        }
    }

    public void Set_Petrol_Destination()
    {
        Vector3 rand_dirrection = UnityEngine.Random.insideUnitSphere * patrol_radius;
        rand_dirrection += patrol_main_obj.position;
    }
}

```

```

NavMeshHit navHit;
NavMesh.SamplePosition(rand_dirrection, out navHit, patrol_radius, -1);

anim.SetBool("running", true);
nav.isStopped = false;
nav.destination = navHit.position;
}

public void Patrol()
{
    is_patroling = true;
    if (!is_waiting && nav.remainingDistance <= 2.0f)
    {

        is_waiting = true;
        wait_timer = wait_time_at_point;
        is_patroling = false;
    }
    if (is_waiting)
    {
        wait_timer -= Time.deltaTime;

        if (wait_timer <= 0 || SaveScript.is_invisible == true)
        {

            is_waiting = false;
            Set_Petrol_Destination();
            nav.isStopped = false;
        }
    }
}

if (SaveScript.is_invisible == true)
{
    is_waiting = false;
    Set_Petrol_Destination();
    nav.isStopped = false;
}

public void Roll()
{
    Vector3 playerDirection = player.transform.position - transform.position;
    playerDirection.Normalize();

    Vector3[] roll_dirrections = {
        -transform.forward, // roll back

```

```

        transform.forward, // roll forward
        -transform.right, // roll left
        transform.right // roll right
    };

string[] anim_Roll_triggers = {
    "roll_F",
    "roll_B",
    "roll_L",
    "roll_R"
};

float[] weights = new float[roll_directions.Length];
for (int i = 0; i < roll_directions.Length; i++)
{
    Vector3 roll_pos = transform.position + roll_directions[i] * dodgeDistance;
    if (NavMesh.SamplePosition(roll_pos, out NavMeshHit hit, 1.0f, NavMesh.AllAreas))
    {
        // Calculate weight based on direction, distance to player, and aggression level
        float weight_of_direction = Vector3.Dot(playerDirection, roll_directions[i]);
        weight_of_direction = (1 - Mathf.Abs(weight_of_direction)) * (1 - aggression_lvl);
        weights[i] = weight_of_direction;
    }
    else
    {
        weights[i] = -1; // Invalid direction
    }
}

int the_best_direction = -1;
float the_best_weight = -1;
for (int i = 0; i < weights.Length; i++)
{
    if (weights[i] > the_best_weight)
    {
        the_best_weight = weights[i];
        the_best_direction = i;
    }
}

if (the_best_direction != -1)
{
    anim.SetTrigger(anim_Roll_triggers[the_best_direction]);
}
}

public void Correct_Aggression()
{
    if (curr_HP < 0.5f)
    {

```

```

        aggression_lvl -= aggression_increase * Time.deltaTime;
    }
else
{
    aggression_lvl += aggression_decrease * Time.deltaTime;
}

float distanceToPlayer = Vector3.Distance(transform.position, player.transform.position);
if (distanceToPlayer < 10f)
{
    aggression_lvl += aggression_increase * Time.deltaTime;
    playerNearby = true;
}
else
{
    playerNearby = false;
}

aggression_lvl = Mathf.Clamp(aggression_lvl, min_aggression, max_aggression);

if(aggression_lvl == 1)
{
    StartCoroutine(Reset_Aggression_Lvl());
}

Debug.Log("Aggression Level: " + aggression_lvl);
}

IEnumerator Reset_Aggression_Lvl()
{
    yield return new WaitForSeconds(3f);
    aggression_lvl = 0.2f;
}

public bool Search_Enemy_Near_Skeleton()
{
    Collider[] all_colliders = Physics.OverlapSphere(transform.position, 10f);
    foreach (Collider collider in all_colliders)
    {
        if (collider.CompareTag("enemy") && collider.gameObject != gameObject)
        {
            return true;
        }
    }
    return false;
}

public void Spawn_Reinforcement()

```

```

{
    for (int i = 0; i < amount_of_reinforcement; i++)
    {
        Instantiate(support_enemy, GetRandom_Point_Around(), Quaternion.identity);
        support_enemy.GetComponent<EnemyMovement>().Goblin_Warrior = true;
        support_enemy.GetComponent<EnemyMovement>().patrol_main_obj = current_enemy.transform;
        SaveScript.amount_of_chasing_enemies++;
    }
}

public Vector3 GetRandom_Point_Around()
{
    float angle = UnityEngine.Random.Range(0f, Mathf.PI * 2);
    float x = Mathf.Cos(angle) * 8f;
    float z = Mathf.Sin(angle) * 8f;
    Vector3 point_for_spawn = new Vector3(transform.position.x + x, transform.position.y, transform.position.z + z);
    return point_for_spawn;
}

void Check_If_Player_is_InSight()
{
    Vector3 player_dir = player.transform.position - transform.position;
    float angle = Vector3.Angle(player_dir, transform.forward);

    if (angle < 90f && player_dir.magnitude < distance_of_ray)
    {
        RaycastHit hit;

        if (Physics.Raycast(transform.position + transform.up, player_dir.normalized, out hit, distance_of_ray))
        {
            Debug.DrawRay(transform.position, player_dir * 10f, Color.red);
            if (hit.transform == player.transform)
            {
                Debug.DrawRay(transform.position, player_dir * 10f, Color.green);

                Nearby_Enemy_Will_Know();
                look_for_player = false;
                player_is_inSight = true;
                last_seen_position = player.transform.position;
            }
        }
    }
    else if (player_is_inSight)
    {
        Debug.DrawRay(transform.position, player_dir * 10f, Color.red);
    }
}

```

```

        player_is_inSight = false;
        nav.SetDestination(last_seen_position);
        look_for_player = true;
    }

}

public void Nearby_Enemy_Will_Know()
{
    try
    {
        Vector3 player_dir = player.transform.position - transform.position;
        Collider[] all_colliders = Physics.OverlapSphere(transform.position, group_brain_radius);
        foreach (var collider in all_colliders)
        {
            EnemyMovement raycast_system = collider.GetComponent<EnemyMovement>();
            if (raycast_system != null && collider.gameObject != gameObject)
            {
                Debug.Log(raycast_system + " KNOW");
                Debug.DrawRay(transform.position, player_dir * 10f, Color.green);
                raycast_system.player_is_inSight = true;
                raycast_system.look_for_player = false;
                raycast_system.last_seen_position = player.transform.position;
            }
        }
    }
    catch (Exception e)
    {
        Debug.Log(e);
    }
}

public void Look_Aroun_Yourself()
{
    transform.Rotate(0, 120 * Time.deltaTime, 0);
}
}

```

### 1.1.2 Файл *EnemyAttack.cs*

```

public class Enemy_Attack : MonoBehaviour
{
    private AudioSource audio_Player;
    private bool enemy_can_attack = true;
    public float damage_enemy = 0.1f;
    private WaitForSeconds wait_before_attack = new WaitForSeconds(1);

    private float correct_dmg_reduce_by_Skill;
    private float correct_dmg_reduce_by_armor;
}

```

```

void Start()
{
    audio_Player = GetComponent< AudioSource >();
}
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player"))
    {
        // Debug.Log("Attack = true");
        float dmg_check;
        if (enemy_can_attack == true && SaveScript.is_Immortal_object != true)
        {
            Deal_DMG_to_Character();
            SaveScript.time_of_last_damage_recive = Time.time;
            audio_Player.Play();
            StartCoroutine(DMG_Delay_Restart());
        }
    }
}

IEnumerator DMG_Delay_Restart()
{
    yield return wait_before_attack;
    enemy_can_attack = true;
}

public void Deal_DMG_to_Character()
{
    correct_dmg_reduce_by_armor = 1.0f - SaveScript.armora_decrease;
    enemy_can_attack = false;
    if(SaveScript.is_shielf_active == true)
    {
        SaveScript.agression_lvl = SaveScript.agression_lvl + 0.05f;
        correct_dmg_reduce_by_Skill = 1.0f - SaveScript.damage_reduce_by_Guardianship;
        SaveScript.health -= (damage_enemy * correct_dmg_reduce_by_armor * correct_dmg_reduce_by_Skill);
    }
    else
    {
        SaveScript.agression_lvl = SaveScript.agression_lvl + 0.1f;
        SaveScript.health -= damage_enemy * correct_dmg_reduce_by_armor;
    }
}

```

### 1.1.3 Файл Golem\_Movement.cs

```
public class Golem_Movement : MonoBehaviour
{
    public GameObject Loot_from_Enemy;
    public bool Golem = true;

    public GameObject current_enemy;
    private bool is_outliner_active = false;

    private AnimatorStateInfo enemy_information;
    private NavMeshAgent nav;
    private Animator anim;
    private float x;
    private float z;
    private float velocitySpeed;
    public GameObject player;
    private float distance_to_player;
    private bool is_attacking;
    public float attack_Range = 2.0f;
    public float chasing_Range; //range in which enemy will run after character
```

```

public float rotation_speed = 500.0f; //perfect
public float dmg_block_probability = 0.15f;

private bool is_reset = false;
private bool stun = false;
private int maxHP;

public float golem_stamina_MAX = 1.0f;
public float golem_stamina;
public float golem_stamina_regeneration = 0.05f;

public int full_HP = 100;
private int curr_HP;
private int fear_lvl = 100;
private int fear_lvl_curr;
public bool enemy_is_alive = true;

private bool skill_was_used = false;

public AudioSource audio_Player;
public AudioClip[] get_Hit_SFX;

public AudioClip block_SFX;

public GameObject bar_Container;
public Image HP_bar;
private float fillHealth;
public GameObject main_camera;

// Start is called before the first frame update
void Start()
{
    audio_Player = GetComponent<AudioSource>();
    current_enemy.GetComponent<Outline>().enabled = false;
    nav = GetComponent<NavMeshAgent>();
    anim = GetComponent<Animator>();
    nav.avoidancePriority = UnityEngine.Random.Range(1, 1);
    curr_HP = full_HP;
    maxHP = full_HP;
    golem_stamina = golem_stamina_MAX;
}

// Update is called once per frame
void Update()
{
    bar_Container.transform.LookAt(main_camera.transform.position);
    //HP_bar.transform.LookAt(main_camera.transform.position);

    if (enemy_is_alive == true)
    {
        //outline
        if (is_outliner_active == false)
        {
            is_outliner_active = true;
            if (SaveScript.spell_target == current_enemy)
            {
                current_enemy.GetComponent<Outline>().enabled = true;
            }
        }
        if (is_outliner_active == true)
        {
            if (SaveScript.spell_target != current_enemy)
            {
                current_enemy.GetComponent<Outline>().enabled = false;
                is_outliner_active = false;
            }
        }
    }
}

```

```

//



Golem_Stamina_Regeneration();

if (player == null)
{
    player = GameObject.FindGameObjectWithTag("Player");
}
x = nav.velocity.x;
z = nav.velocity.z;
velocitySpeed = new Vector2(x, z).magnitude;
if (velocitySpeed == 0)
{
    anim.SetBool("running", false);
}
else if (velocitySpeed != 0)
{
    anim.SetBool("running", true); ;
    is_attacking = false;
}

enemy_information = anim.GetCurrentAnimatorStateInfo(0);
distance_to_player = Vector3.Distance(transform.position, player.transform.position);

//Debug.Log(distance_to_player);

if (enemy_information.IsName("atk_dash") == true && skill_was_used == false)
{
    //golem_stamina -= 0.6f;
    skill_was_used = true;
}

if (skill_was_used == true)
{
    StartCoroutine(Reset_Dash());
}

if (distance_to_player >= 10.0f )//&& golem_stamina > 0.61f
{
    anim.SetBool("player_too_far", true);
}
else
{
    anim.SetBool("player_too_far", false);
}

if (golem_stamina > 0.01f)
{
    if (distance_to_player < attack_Range || distance_to_player > chasing_Range)
    {
        nav.isStopped = true;

        if (distance_to_player < attack_Range && enemy_information.IsTag("nonAttack") && SaveScript.is_invisible != true) //&& golem_stamina > 0.1f
        {

            if (is_attacking == false)

```

```

    {
        Look_At_Player_Spherical_LERP();

        int randomNumber = UnityEngine.Random.Range(1, 101);
        if (randomNumber > 0 && randomNumber < 51)
        {
            if (distance_to_player <= 2.0f)
            {
                is_attacking = true;
                //golem_stamina -= 0.1f;
                anim.SetTrigger("player_too_close");
            }
        }
        else
        {
            int randomNumber2 = UnityEngine.Random.Range(1, 101);
            is_attacking = true;
            //golem_stamina -= 0.1f;
            anim.SetInteger("random", randomNumber2);
            anim.SetTrigger("attack");
        }
    }

    if (distance_to_player < attack_Range && enemy_information.IsTag("attack"))
    {
        if (is_attacking == true)
        {
            is_attacking = false;
        }
    }
    else if (distance_to_player > attack_Range && enemy_information.IsTag("nonAttack") && !anim.IsInTransition(0))
    {

        if (SaveScript.is_invisible == false)
        {
            nav.isStopped = false;
            nav.destination = player.transform.position;
        }
    }
}

//curr_HP = was
//full_hp - are
if (curr_HP > full_HP)
{
    golem_stamina -= 0.05f;
    anim.SetTrigger("hit");
    curr_HP = full_HP;
    RandomAudio_Hit();
    fillHealth = Convert.ToSingle(full_HP) / Convert.ToSingle(maxHP);
    Debug.Log(fillHealth);
    HP_bar.fillAmount = fillHealth;
}

if (nav.isStopped == false || distance_to_player > 6.0f && enemy_information.IsTag("attack"))
{
    anim.ResetTrigger("player_near");
    anim.ResetTrigger("player_too_close");
    anim.ResetTrigger("attack");
    if (is_attacking == true)
    {
        is_attacking = false;
    }
}

```

```

        }

    }

    if (full_HP < maxHP / 2 && stun == false)
    {
        stun = true;
        StartCoroutine(Stun_Duration());
    }

    if (full_HP <= 1 && enemy_is_alive == true)
    {
        enemy_is_alive = false;
        nav.isStopped = true;
        anim.SetTrigger("death");
        current_enemy.GetComponent<Outline>().enabled = false;
        is_outliner_active = false;
        nav.avoidancePriority = 1;
        StartCoroutine(Loot_Spawn());
    }

}

public void Look_At_Player_Spherical_LERP()
{
    Vector3 Pos = (player.transform.position - transform.position).normalized;
    Quaternion PosRotation = Quaternion.LookRotation(new Vector3(Pos.x, 0, Pos.z));
    transform.rotation = Quaternion.Slerp(transform.rotation, PosRotation, Time.deltaTime * rotation_speed);
}

public void RandomAudio_Hit()
{
    int randomNumber = UnityEngine.Random.Range(1, 101);
    if (randomNumber > 0 && randomNumber < 33)
    {
        audio_Player.clip = get_Hit_SFX[0];
    }
    else if (randomNumber >= 33 && randomNumber < 66)
    {
        audio_Player.clip = get_Hit_SFX[1];
    }
    else if (randomNumber >= 66 && randomNumber < 101)
    {
        audio_Player.clip = get_Hit_SFX[2];
    }
    audio_Player.Play();
}

public void Golem_Stamina_Regeneration()
{
    golem_stamina += golem_stamina_regeneration * Time.deltaTime;
    golem_stamina = Mathf.Clamp(golem_stamina, 0, golem_stamina_MAX);
}

IEnumerator Loot_Spawn()
{
    yield return new WaitForSeconds(2.5f);
    Instantiate(Loot_from_Enemy, transform.position, transform.rotation);
    SaveScript.killed_enemy++;
    Destroy(gameObject, 0.2f);
}

IEnumerator Stun_Duration()
{
}

```

```

anim.SetTrigger("stun_start");
nav.isStopped = true;
yield return new WaitForSeconds(5);
anim.SetTrigger("stun_end");
nav.isStopped = false;
}

IEnumerator Reset_Dash()
{
    yield return new WaitForSeconds(5);
    skill_was_used = false;
}
}

```

## 1.2 Реалізація ігрового інтерфейсу.

### 1.2.1 Файл Lvl\_Up\_Stats.cs

```

public class Lvl_Up_Stats : MonoBehaviour
{
    public AudioClip selection;
    public AudioSource Inventory_Canvas;

    public void Lvl_UP_Strength()
    {
        if (SaveScript.points_to_upgrade > 0)
        {
            // SaveScript.strength_basic += SaveScript.player_lvl_character;
            SaveScript.strength_basic += 0.05f;
            SaveScript.points_to_upgrade--;
        }
    }

    public void Lvl_UP_Intelligence()
    {
        if (SaveScript.points_to_upgrade > 0)
        {
            // SaveScript.intelligence_basic += SaveScript.player_lvl_character;
            SaveScript.intelligence_basic += 0.05f;
            SaveScript.points_to_upgrade--;
        }
    }

    public void Lvl_UP_Stamina()
    {
        if (SaveScript.points_to_upgrade > 0)
        {
            // SaveScript.stamina_basic += SaveScript.player_lvl_character;
            SaveScript.stamina_basic += 0.05f;
        }
    }
}

```

```
        SaveScript.points_to_upgrade--;  
    }  
}  
}
```

## 1.2.2 Файл *Main\_Menu.cs*

```
public class Main_Menu : MonoBehaviour
{
    public GameObject continue_;
    public GameObject load_;
    public GameObject save_;
    void Start()
    {
        TurnOn_Continue_If_Exists();
        Cursor.visible = true;
    }

    public void Start_New_Game()
    {
        SceneManager.LoadScene(1);
    }

    public void Continue_Button()
    {
        load_.SetActive(true);
        save_.SetActive(true);
        SaveScript.take_data_to_load = true;
        StartCoroutine(LoadGame());
    }

    public void Exit()
    {
        Application.Quit();
    }

    public void Settings()
    {

    }

    IEnumerator LoadGame()
    {
        yield return new WaitForSeconds(1);
        SceneManager.LoadScene(2);
    }

    public void TurnOn_Continue_If_Exists()
    {
```

```

        if (Application.persistentDataPath + "/preservation.data" != null)
        {
            continue_.SetActive(true);
        }
        else
        {
            continue_.SetActive(false);
        }
    }

}

```

## 1.3 Реалізація механік бою.

### 1.3.1 PlayerMovement.cs

```

public class PlayerMovement : MonoBehaviour
{
    private UnityEngine.AI.NavMeshAgent nav;
    private Animator anim;
    private Ray ray;
    private RaycastHit hit;

    private float x;
    private float z;
    private float velocitySpeed;
    public static int ray_numbers = 6;

    //For Camera
    CinemachineTransposer cinemachineTransposer;
    //public CinemachineVirtualCamera playerCamera; //free
    CinemachineOrbitalTransposer cinemachine_orbital_Transposer;

    private Vector3 mouse_pos;
    private Vector3 current_pos;
    private string axis_named = "Mouse X";

    private bool isPlayerSelectScene;
    public static bool canMove = true;
    public static bool isPlayerMoving = false;

    public GameObject camera_1_static;
    public GameObject camera_2_free;
    private bool is_camera1_active = true;

    private float previous_health = 1.0f;
}

```

```

public GameObject get_hit_VFX_Place;
private WaitForSeconds life_time_hit_effect = new WaitForSeconds(0.1f);

//for roof box colider
public LayerMask boxLayer;

public GameObject vfx_spawm_point;
private WaitForSeconds nearEnemy = new WaitForSeconds(0.4f);

public GameObject[] player_mesh_parts;
public GameObject[] weapons_props;
public GameObject[] armor_parts_Torso;
public GameObject[] armor_parts_Legs;
public string[] attacks_tags;
public AudioClip[] weapon_SFX;
public AudioSource audio_Player;

private AnimatorStateInfo player_information;

private GameObject trail_mesh;
private WaitForSeconds traill_time = new WaitForSeconds(0.1f);
public bool critical_attack_is_active = false;

public float[] stamina_cost_for_weapon;
void Start()
{
    nav = GetComponent<UnityEngine.AI.NavMeshAgent>();
    anim = GetComponent<Animator>();

    camera_1_static.SetActive(false);
    camera_2_free.SetActive(true);
    SaveScript.vfx_spawn_point = vfx_spawm_point;
    //cinemachineTransposer = playerCamera.GetCinemachineComponent< CinemachineTransposer >();
    //current_pos = cinemachineTransposer.m_FollowOffset;
    cinemachineTransposer
    =
    camera_1_static.gameObject.GetComponent< CinemachineVirtualCamera >().GetCinemachineComponent< CinemachineTranspo
ser >();
    cinemachine_orbital_Transposer
    =
    camera_2_free.gameObject.GetComponent< CinemachineVirtualCamera >().GetCinemachineComponent< CinemachineOrbitalTr
ansposer >();

    for (int i = 0; i < weapons_props.Length; i++)
    {
        weapons_props[i].SetActive(false);
    }
}

```

```

if (SceneManager.GetActiveScene().name == "PlayerSelect")
{
    isPlayerSelectScene = true;

}

if(SceneManager.GetActiveScene().buildIndex == 2)
{
    Display_Correct_ArmorInShop();
}
Check_Class_Info();
get_hit_VFX_Place.SetActive(false);

}

void Update()
{
    if (SceneManager.GetActiveScene().buildIndex == 2)
    {
        Display_Correct_ArmorInShop();
    }
    //Debug.Log("can mpve " + canMove);
    player_information = anim.GetCurrentAnimatorStateInfo(0); //listen to Animator

    //change correct weapon
    if (SaveScript.should_change_weapon == true)
    {
        SaveScript.should_change_weapon = false;
        for (int i = 0; i < weapons_props.Length; i++)
        {
            weapons_props[i].SetActive(false);
        }
        weapons_props[SaveScript.weapon_index].SetActive(true);
        StartCoroutine(WaitForTrail());
    }

}

if (isPlayerSelectScene == false)
{
    x = nav.velocity.x;
    z = nav.velocity.z;
    velocitySpeed = new Vector2(x, z).magnitude;

    Ray[] rays = new Ray[ray_numbers];

    if (Input.GetMouseButtonDown(0) && player_information.IsTag("nonAttack") && !anim.IsInTransition(0))

```

```

    {
        if (canMove == true)
        {
            for (int i = 0; i < ray_numbers; i++)
            {
                rays[i] = Camera.main.ScreenPointToRay(Input.mousePosition);
            }

            Vector3 averageHitPoint = Vector3.zero;

            foreach (Ray ray in rays)
            {
                RaycastHit hit;

                if (Physics.Raycast(ray, out hit, 300, boxLayer))
                {
                    if (hit.transform.gameObject.CompareTag("enemy"))
                    {
                        nav.isStopped = false;
                        SaveScript.spell_target = hit.transform.gameObject;
                        averageHitPoint += hit.point;
                        transform.LookAt(SaveScript.spell_target.transform);
                        StartCoroutine(MoveTo()); //wait 3 sec and than isStopped == true
                    }
                    else
                    {
                        SaveScript.spell_target = null;
                        averageHitPoint += hit.point;
                        nav.isStopped = false;
                    }
                }
            }

            averageHitPoint /= rays.Length;
            nav.destination = averageHitPoint;
        }
    }

    if (Input.GetMouseButton(1))
    {
        cinemachine_orbital_Transposer.m_XAxis.m_InputAxisName = axis_named; //we put "Mouse X" into field
        of orbital camera to be able to rotate it
    }

    if (Input.GetMouseButtonUp(1))
    {

```

```

cinemachine_orbital_Transposer.m_XAxis.m_InputAxisName = null;
cinemachine_orbital_Transposer.m_XAxis.m_InputAxisValue = 0;
}

// Check if the character is moving (forward or backward)
anim.SetBool("sprinting", velocitySpeed > 0.1f);
if(velocitySpeed != 0)
{
    if(SaveScript.is_character_equip_a_weapon == false)
    {
        anim.SetBool("sprinting", true);
        anim.SetBool("equip_a_weapon", false);
    }
    if (SaveScript.is_character_equip_a_weapon == true)
    {
        anim.SetBool("sprinting", true);
        anim.SetBool("equip_a_weapon", true);
    }
    isPlayerMoving = true;
}
if (velocitySpeed == 0)
{
    anim.SetBool("sprinting", false);
    isPlayerMoving = false;
}

if (Input.GetKeyDown(KeyCode.S))
{
    anim.SetBool("sprinting", false);
    nav.destination = transform.position;
}

if (Input.GetKeyDown(KeyCode.C))
{
    if(is_camera1_active == true)
    {
        camera_1_static.SetActive(false);
        camera_2_free.SetActive(true);

        is_camera1_active = false;
    }
    else if (is_camera1_active == false)
    {
        camera_1_static.SetActive(true);
    }
}

```

```

        camera_2_free.SetActive(false);

        is_camera1_active = true;
    }
}

//make player invisible
if (player_mesh_parts[0].activeSelf == true)
{
    if(SaveScript.is_invisible == true)
    {
        SaveScript.agression_lvl = SaveScript.agression_lvl - 0.15f;
        for (int i = 0; i < player_mesh_parts.Length; i++)
        {
            player_mesh_parts[i].SetActive(false);
        }
    }
}

//make player visible
if (SaveScript.mana <= 0.05)
{
    if (SaveScript.is_invisible == false)
    {
        for (int i = 0; i < player_mesh_parts.Length; i++)
        {
            player_mesh_parts[i].SetActive(true);
        }
    }
    SaveScript.should_change_armor = true;
}
}

if(SaveScript.should_change_armor == true)
{
    for(int i = 0; i < armor_parts_Torso.Length; i++)
    {
        armor_parts_Torso[i].SetActive(false);
        armor_parts_Legs[i].SetActive(false);
    }
    armor_parts_Torso[SaveScript.index_of_equiped_armor].SetActive(true);
    armor_parts_Legs[SaveScript.index_of_equiped_armor].SetActive(true);
    SaveScript.should_change_armor = false;
}
}

```

```

if (Input.GetKeyDown(KeyCode.Z))
{
    if (SaveScript.is_character_equip_a_weapon == true && SaveScript.stamina > 0.2)
    {
        Basic_or_Critical_Attack();
    }
}

if(SaveScript.health <= 0.0f)
{
    if (SaveScript.unique_features_index == 3 && Time.time - SaveScript.time_of_unique_feature_activasion >
SaveScript.unique_features_index_CD)
    {
        SaveScript.time_of_unique_feature_activasion = Time.time;
        SaveScript.health = 0.5f;
    }
    else
    {
        SceneManager.LoadScene(0); // 0 - Player Select 1 - Terrain1 (More can check in File -> Build Settings)
        SaveScript.health = 1.0f;
    }
}

if(previous_health > SaveScript.health)
{
    CharacterGetHit();
}
}

public void Basic_or_Critical_Attack()
{
    float randomNumber = Random.value;
    if (randomNumber <= SaveScript.critical_hit_chance)
    {

        critical_attack_is_active = true;
        anim.SetTrigger(attacks_tags[6]);
        audio_Player.volume = 0.4f;
        audio_Player.clip = weapon_SFX[6];
        audio_Player.Play();
        SaveScript.stamina -= stamina_cost_for_weapon[6];
    }
}

```

```

        }

        else
        {
            critical_attack_is_active = false;
            anim.SetTrigger(attacks_tags[SaveScript.weapon_index]);
            audio_Player.volume = 0.3f;
            audio_Player.clip = weapon_SFX[SaveScript.weapon_index];
            //audio_Player.Play();
            SaveScript.stamina -= stamina_cost_for_weapon[SaveScript.weapon_index];
        }
    }

IEnumerator TurnOff_Hit_VFX()
{
    yield return life_time_hit_effect;
    get_hit_VFX_Place.SetActive(false);
}

public void CharacterGetHit()
{
    get_hit_VFX_Place.SetActive(true);
    previous_health = SaveScript.health;
    StartCoroutine(TurnOff_Hit_VFX());
}

public void Weapon_SFX_Play()
{
    audio_Player.Play();
}

public void TurnOn_Trail()
{
    trail_mesh.GetComponent<Renderer>().enabled = true;
}

public void TurnOff_Trail()
{
    trail_mesh.GetComponent<Renderer>().enabled = false;
}

IEnumerator MoveTo()
{
    yield return nearEnemy;
    nav.isStopped = true;
}

IEnumerator WaitForTrail()

```

```

    }

    yield return traill_time;

    trail_mesh = GameObject.Find("Trail");

    trail_mesh.GetComponent<Renderer>().enabled = false;

}

public void Check_Class_Info()
{
    if (SaveScript.unique_features_index == 0)
    {
        Debug.Log(SaveScript.class_Avarage + " " + SaveScript.class_Mage + " " + SaveScript.class_Seller + " " +
SaveScript.class_Warrior);
        Debug.Log("None Ability");
    }
    else if (SaveScript.unique_features_index == 1)
    {
        Debug.Log(SaveScript.class_Avarage + " " + SaveScript.class_Mage + " " + SaveScript.class_Seller + " " +
SaveScript.class_Warrior);
        Debug.Log("More Mana Regeneration and +20% spell/magic damage");
    }
    else if (SaveScript.unique_features_index == 2)
    {
        Debug.Log(SaveScript.class_Avarage + " " + SaveScript.class_Mage + " " + SaveScript.class_Seller + " " +
SaveScript.class_Warrior);
        Debug.Log("Price in shop is -20% lower");
    }
    else if (SaveScript.unique_features_index == 3)
    {
        Debug.Log(SaveScript.class_Avarage + " " + SaveScript.class_Mage + " " + SaveScript.class_Seller + " " +
SaveScript.class_Warrior);
        Debug.Log("You can survive lethal damage and regain 50% HP (500 sec CD)");
    }
}

public void Display_Correct_ArmorInShop()
{
    if(isPlayerSelectScene == true)
    {
        if (SaveScript.player_index_character == 1 || SaveScript.player_index_character == 2 ||
SaveScript.player_index_character == 0)
        {
            GetComponent<Stats_Info>().armor_in_shop[0].SetActive(true);
            GetComponent<Stats_Info>().armor_in_shop[1].SetActive(false);
        }
    }
}

```

```
        else
        {
            GetComponent<Stats_Info>().armor_in_shop[0].SetActive(false);
            GetComponent<Stats_Info>().armor_in_shop[1].SetActive(true);
        }
    }
}
```

### **1.3.2 Файл Character\_Attack.cs**

```
public class Character_Attack : MonoBehaviour
{
    private GameObject mesh_to_Destroy;
    public int basic_weapon_damage;

    private GameObject player;

    private bool can_deal_dmg = true;
    private WaitForSeconds dmg_Pause = new WaitForSeconds(0.1f);
    // Start is called before the first frame update
    void Start()
    {
        if (player == null)
        {
            player = GameObject.FindGameObjectWithTag("Player");
        }
    }

    // Update is called once per frame
    void Update()
    {

    }

    public void OnTriggerEnter(Collider other)
    {
        //if we are attacking crate
        if (other.CompareTag("Crate"))
        {
            other.transform.gameObject.GetComponentInParent<Chest>().VFX_crate_text();
            mesh_to_Destroy = other.transform.parent.gameObject;
            Destroy(other.transform.gameObject);
            StartCoroutine(Wait_before_Destroy());
        }
    }
}
```

```

        }

        if (other.CompareTag("enemy") && can_deal_dmg == true )
        {
            SaveScript.agression_lvl = SaveScript.agression_lvl + 0.2f;

            Enemy_Type enemy_type = other.GetComponent<Enemy_Type>();
            int dmg_check = 0;
            if(player.GetComponent<PlayerMovement>().critical_attack_is_active == true)
            {
                dmg_check = (basic_weapon_damage + SaveScript.weapon_dmg_scaleUP + SaveScript.strength_increase) *
SaveScript.critical_dmg_multiply;
                if (enemy_type.enemyType == Enemy_Type.EnemyType.Golem)
                {
                    if (Random.Range(0f, 1f) >= other.GetComponent<Golem_Movement>().dmg_block_probability) //15 per
cent to block dmg
                    {
                        other.transform.gameObject.GetComponent<Golem_Movement>().full_HP -= ((basic_weapon_damage +
SaveScript.weapon_dmg_scaleUP + SaveScript.strength_increase) * SaveScript.critical_dmg_multiply);
                    }
                    else
                    {
                        other.GetComponent<Golem_Movement>().audio_Player.clip =
other.GetComponent<Golem_Movement>().block_SFX;
                        other.GetComponent<Golem_Movement>().audio_Player.Play();
                    }
                }
                else
                {
                    other.transform.gameObject.GetComponent<EnemyMovement>().full_HP -= ((basic_weapon_damage +
SaveScript.weapon_dmg_scaleUP + SaveScript.strength_increase) * SaveScript.critical_dmg_multiply);

                }
                can_deal_dmg = false;
            }
            else
            {
                dmg_check = (basic_weapon_damage + SaveScript.weapon_dmg_scaleUP + SaveScript.strength_increase);

                if(enemy_type.enemyType == Enemy_Type.EnemyType.Golem)
                {
                    if (Random.Range(0f, 1f) >= other.GetComponent<Golem_Movement>().dmg_block_probability) //15 per
cent to block dmg
                    {
                        other.transform.gameObject.GetComponent<Golem_Movement>().full_HP -= (basic_weapon_damage +
SaveScript.weapon_dmg_scaleUP + SaveScript.strength_increase);
                    }
                }
            }
        }
    }
}

```

```

        else
        {
            other.GetComponent<Golem_Movement>().audio_Player.clip
            other.GetComponent<Golem_Movement>().block_SFX;
            other.GetComponent<Golem_Movement>().audio_Player.Play();
        }
    }
    else
    {
        other.transform.gameObject.GetComponent<EnemyMovement>().full_HP -= (basic_weapon_damage +
SaveScript.weapon_dmg_scaleUP + SaveScript.strength_increase);
    }

    can_deal_dmg = false;
}

Debug.Log(basic_weapon_damage + " " + SaveScript.weapon_dmg_scaleUP + " " +
SaveScript.strength_increase);
if (enemy_type.enemyType == Enemy_Type.EnemyType.Golem)
{
    Debug.Log("Monster = " + other.name + " HP = " +
other.transform.gameObject.GetComponent<Golem_Movement>().full_HP + " DMG = " + dmg_check);
}
else
{
    Debug.Log("Monster = " + other.name + " HP = " +
other.transform.gameObject.GetComponent<EnemyMovement>().full_HP + " DMG = " + dmg_check);
}

StartCoroutine(ResetDMG());
}

IEnumerator Wait_before_Destroy()
{
    yield return new WaitForSeconds(2);
    Destroy(mesh_to_Destroy);
}

IEnumerator ResetDMG()
{
    yield return dmg_Pause;
}

```

```
    can_deal_dmg = true;
}
```

```
}
```

## 1.4 Реалізація функціоналу інвентарю, магазинів та ігрової економіки.

### 1.4.1 Файл Buying.cs

```
public class Buying : MonoBehaviour
{
    public GameObject shop;
    public GameObject Inventory_Canvas;

    public AudioSource audio_Player;

    public int[] amount_of_stuff_in_shop;
    public int[] cost_of_stuff_in_shop;
    public int[] element_number;

    public int[] inventory_items;

    public Text[] text_amount_of_stuff_in_shop;
    public Text[] text_finance;

    private Text compare;

    public bool isPub;
    public bool isWizzardShop;
    public bool isCraftsmenWorkshop;

    private int max = 0;
    private bool canClick;

    public Text[] price_per_obj;

    void Start()
    {
        shop.SetActive(false);
        max = text_amount_of_stuff_in_shop.Length;
        text_finance[0].text = Inventory.gold.ToString();
        text_finance[1].text = Inventory.diamond.ToString();

        for(int i=0; i < max; i++)
        {
            text_amount_of_stuff_in_shop[i].text = amount_of_stuff_in_shop[i].ToString();
        }
    }
}
```

```

        }

audio_Player = Inventory_Canvas.GetComponent< AudioSource >();

if(SaveScript.class_Seller == true)
{
    SellerClassFeature();
}

}

public void Close()
{
    shop.SetActive(false);
    PlayerMovement.canMove = true;
}

public void BuyButton()
{
    if (canClick == true)
    {
        for (int i = 0; i < max; i++)
        {
            if (text_amount_of_stuff_in_shop[i] == compare)
            {
                max = i;
                if (amount_of_stuff_in_shop[i] > 0)
                {

                    if (isPub == true)
                    {
                        RefreshShopAmount();
                    }else if(isWizzardShop == true)
                    {
                        RefreshWizardShopAmount();
                    }
                    else if(isCraftsmenWorkshop == true)
                    {
                        RefreshCraftsMenShopAmount();
                    }
                }

                if (Inventory.gold >= cost_of_stuff_in_shop[i])
                {
                    if (inventory_items[i] == 0)
                    {

```

```

        Inventory.newIcon = element_number[i];
        Inventory.iconUpdated = true;
    }
    Inventory.gold -= cost_of_stuff_in_shop[i];

    //RANDOM SFX COIN
    int randomNumber = UnityEngine.Random.Range(1, 101);
    if (randomNumber > 0 && randomNumber < 33)
    {
        audio_Player.clip = Inventory_Canvas.GetComponent<Inventory>().coin_buy_SFX;
    } else if (randomNumber >= 33 && randomNumber < 66)
    {
        audio_Player.clip = Inventory_Canvas.GetComponent<Inventory>().coin2_buy_SFX;
    } else if (randomNumber >= 66 && randomNumber < 101)
    {
        audio_Player.clip = Inventory_Canvas.GetComponent<Inventory>().coin3_buy_SFX;
    }
    audio_Player.Play();
    //RANDOM SFX COIN

    if (isPub == true)
    {
        SetShopAmount(i);
    }
    else if (isWizzardShop == true)
    {
        SetWizzardShopAmount(i);
    }
    else if (isCraftsmenWorkshop == true)
    {
        SetCraftsMenShopAmount(i);
    }
}

void RefreshShopAmount()
{
    inventory_items[0] = Inventory.amount_of_bread;
    inventory_items[1] = Inventory.amount_of_cheese;
    inventory_items[2] = Inventory.amount_of_meat;
}

void RefreshWizardShopAmount()
{
}

```

```

inventory_items[0] = Inventory.amount_of_redPotion;
inventory_items[1] = Inventory.amount_of_bluePotion;
inventory_items[2] = Inventory.amount_of_lazurePotion;
inventory_items[3] = Inventory.amount_of_greenPotion;

inventory_items[4] = Inventory.amount_of_monsterEye;
inventory_items[5] = Inventory.amount_of_roots;
inventory_items[6] = Inventory.amount_of_leaf;

}

void RefreshCraftsMenShopAmount()
{
}

public void UpdateFinance()
{
    text_finance[0].text = Inventory.gold.ToString();
    text_finance[1].text = Inventory.diamond.ToString();
}

void SetShopAmount(int item)
{
    switch (item)
    {
        case 0:
            Inventory.amount_of_bread++;
            break;
        case 1:
            Inventory.amount_of_cheese++;
            break;
        case 2:
            Inventory.amount_of_meat++;
            break;
        default:
            break;
    }

    amount_of_stuff_in_shop[item]--;
    text_amount_of_stuff_in_shop[item].text = amount_of_stuff_in_shop[item].ToString();
    UpdateFinance();
    max = amount_of_stuff_in_shop.Length;
}

void SetWizzardShopAmount(int item)
{
}

```

```

switch (item)
{
    case 0:
        Inventory.amount_of_redPotion++;
        break;
    case 1:
        Inventory.amount_of_bluePotion++;
        break;
    case 2:
        Inventory.amount_of_lazurePotion++;
        break;
    case 3:
        Inventory.amount_of_greenPotion++;
        break;
    case 4:
        Inventory.amount_of_monsterEye++;
        break;
    case 5:
        Inventory.amount_of_roots++;
        break;
    case 6:
        Inventory.amount_of_leaf++;
        break;

    default:
        break;
}

amount_of_stuff_in_shop[item]--;
text_amount_of_stuff_in_shop[item].text = amount_of_stuff_in_shop[item].ToString();
UpdateFinance();
max = amount_of_stuff_in_shop.Length;
}

void SetCraftsMenShopAmount(int item)
{

}

void CheckAmount(int items_number_general)
{
    if (amount_of_stuff_in_shop[items_number_general] > 0)
    {
        canClick = true;
    }
    else

```

```

    {
        canClick = false;
    }

}

void CheckAmount_for_WizzardShop(int items_number_general_v2)
{
    if (amount_of_stuff_in_shop[items_number_general_v2] > 0)
    {
        canClick = true;
    }
    else
    {
        canClick = false;
    }
}

//for Shop basic
public void bread()
{
    compare = text_amount_of_stuff_in_shop[0];
    CheckAmount(0);
}

public void cheese()
{
    compare = text_amount_of_stuff_in_shop[1];
    CheckAmount(1);
}

public void meat()
{
    compare = text_amount_of_stuff_in_shop[2];
    CheckAmount(2);
}

//for Wizzard Shop
public void red_Potion()
{
    compare = text_amount_of_stuff_in_shop[0];
    CheckAmount_for_WizzardShop(0);
}

public void blue_Potion()
{
    compare = text_amount_of_stuff_in_shop[1];
    CheckAmount_for_WizzardShop(1);
}

public void lazure_Potion()
{
}

```

```

{
    compare = text_amount_of_stuff_in_shop[2];
    CheckAmount_for_WizzardShop(2);
}

public void green_Potion()
{
    compare = text_amount_of_stuff_in_shop[3];
    CheckAmount_for_WizzardShop(3);
}
public void monster_Eye()
{
    compare = text_amount_of_stuff_in_shop[4];
    CheckAmount_for_WizzardShop(4);
}
public void roots()
{
    compare = text_amount_of_stuff_in_shop[5];
    CheckAmount_for_WizzardShop(5);
}
public void leaf()
{
    compare = text_amount_of_stuff_in_shop[6];
    CheckAmount_for_WizzardShop(6);
}

public void SellerClassFeature()
{
    for(int i =0; i < cost_of_stuff_in_shop.Length; i++)
    {
        cost_of_stuff_in_shop[i] = (cost_of_stuff_in_shop[i] * 4) / 5; // 20 per cent lower price
        price_per_obj[i].text = cost_of_stuff_in_shop[i] + " coins";
    }
}
}

```

#### 1.4.2 Файл Buying\_Weapons.cs

```

public class Buying_Weapons : MonoBehaviour
{
    public Text finance_text_gold;
    public Text finance_text_diamond;

    public int weapon_index;
    public int armor_index;
    public int price;
    public GameObject Inventory_Canvas;
}

```

```

public AudioSource audio_Player;
public Text text_price;

// Start is called before the first frame update
void Start()
{
    finance_text_diamond.text = Inventory.diamond.ToString();
    finance_text_gold.text = Inventory.gold.ToString();
    audio_Player = Inventory_Canvas.GetComponent<AudioSource>();

    text_price.text = price.ToString() + " coins";

    if (SaveScript.class_Seller == true)
    {
        SellerClassFeature();
    }
}

public void BuyButton_Weapon()
{
    if(Inventory.gold >= price)
    {
        Inventory.gold -= price;
        Inventory_Canvas.GetComponent<Inventory>().weapons[weapon_index] = true;

        //RANDOM SFX COIN
        RandomAudio();
        //

        finance_text_diamond.text = Inventory.diamond.ToString();
        finance_text_gold.text = Inventory.gold.ToString();
    }
}

public void BuyButton_Armor()
{
    if (Inventory.gold >= price)
    {
        Inventory.gold -= price;
        SaveScript.index_of_equiped_armor = armor_index;
        SaveScript.should_change_armor = true;
        //RANDOM SFX COIN
        RandomAudio();
        //

        finance_text_diamond.text = Inventory.diamond.ToString();
    }
}

```

```

        finance_text_gold.text = Inventory.gold.ToString();
    }
}

public void RandomAudio()
{
    int randomNumber = UnityEngine.Random.Range(1, 101);
    if (randomNumber > 0 && randomNumber < 33)
    {
        audio_Player.clip = Inventory_Canvas.GetComponent<Inventory>().coin_buy_SFX;
    }
    else if (randomNumber >= 33 && randomNumber < 66)
    {
        audio_Player.clip = Inventory_Canvas.GetComponent<Inventory>().coin2_buy_SFX;
    }
    else if (randomNumber >= 66 && randomNumber < 101)
    {
        audio_Player.clip = Inventory_Canvas.GetComponent<Inventory>().coin3_buy_SFX;
    }
    audio_Player.Play();
}

public void SellerClassFeature()
{
    price = price * 4 / 5;
    text_price.text = price.ToString() + " coins";
}
}

```

## 1.5 Реалізація функціонали створення та використання магії.

### 1.5.1 Файл Particle\_Destroyer.cs

```

public class Particle_Destroyer : MonoBehaviour
{
    public float life_time_for_chest = 2.0f;
    // Start is called before the first frame update
    void Start()
    {
        Destroy(gameObject, life_time_for_chest);
    }
}

```

```
}
```

### 1.5.2 Файл Particle\_Point.cs

```
public class Particle_Point : MonoBehaviour
{
    public int damage = 30;
    public float speed = 1.0f;
    public bool should_rotate = false;
    public bool move_to_target = true;

    public GameObject object_triggered;

    // Update is called once per frame
    void Update()
    {
        if (should_rotate == true)
        {
            transform.Rotate(0, speed * Time.deltaTime, 0);
        }
        if(move_to_target == true)
        {
            transform.Translate(Vector3.forward * speed * Time.deltaTime);
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("enemy") && other.transform.gameObject != object_triggered)
        {
            Enemy_Type enemy_type = other.GetComponent<Enemy_Type>();
            if (SaveScript.class_Mage == true)
            {
                damage = damage * 6 / 5;
            }
            if (enemy_type.enemyType == Enemy_Type.EnemyType.Golem)
            {
                other.transform.gameObject.GetComponent<Golem_Movement>().full_HP -= (damage * 4)/5; // 20 peer
                cent magic decrease
                object_triggered = other.transform.gameObject;
            }
            else
            {
                other.transform.gameObject.GetComponent<EnemyMovement>().full_HP -= damage;
                object_triggered = other.transform.gameObject;
            }
        }
    }
}
```

```

        }
    }

}
}
```

### 1.5.3 Faül Particle\_Transform.cs

```

public class Particle_Transform : MonoBehaviour
{
    //flame nova/twist
    public GameObject target_point;
    public GameObject vfx_object_container;
    public float speed = 5.0f;
    public float duration_of_life = 1.5f;
    public float spell_mana_cost = 0.06f;

    private GameObject vfx_target_save;
    public GameObject player;
    //

    public bool enemy_search = false ;
    public bool non_moving = false;
    public bool support_spell_follow_player = false;
    public bool shield_spell = false;
    public bool power_stats_up_spell = false;
    public bool heal_magic = false;

    public bool invisibility_spell_is_active = false;

    public GameObject object_triggered;
    public int damage = 30;

    private void Start()
    {

        vfx_target_save = SaveScript.spell_target;
        player = GameObject.FindGameObjectWithTag("Player") ;
        if(invisibility_spell_is_active == true)
        {
            SaveScript.is_invisible = true;
        }

        if (shield_spell == true)
    }
```

```

    {
        SaveScript.is_shielf_active = true;
    }
    if(power_stats_up_spell == true)
    {
        SaveScript.strength_increase = 100;
    }

}

// Update is called once per frame
void Update()
{
    if(target_point != null) //avarage target spel position - worl*
    {
        transform.position = Vector3.LerpUnclamped(transform.position/*current pos*/,
target_point.transform.position/*target pos*/, speed * Time.deltaTime); //fuction to move between object a and b with speed c
(from current position to target with speed multiplied by delta time
    }
    if(enemy_search == true) //enemy search spell attack
    {
        if(vfx_target_save != null)
        {
            transform.position = Vector3.LerpUnclamped(transform.position, vfx_target_save.transform.position, speed
* Time.deltaTime);
        }
        else
        {
            transform.Translate(Vector3.forward * speed * Time.deltaTime);
        }
    }
    if(non_moving == true) //click on enemy magic
    {
        if(vfx_target_save != null)
        {
            transform.position = vfx_target_save.transform.position;
        }
        else
        {
            Destroy(vfx_object_container);
        }
    }
    if(support_spell_follow_player == true)
    {
}

```

```

        transform.position = player.transform.position;
        duration_of_life = 100;
        if(SaveScript.mana <= 0.02)
        {
            Destroy(vfx_object_container);
        }

    }

    if (heal_magic == true)
    {
        SaveScript.health += SaveScript.health_regeneration_skill * Time.deltaTime;
    }

    SaveScript.mana -= spell_mana_cost * Time.deltaTime;

    Destroy(vfx_object_container, duration_of_life);
}

private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("enemy") && other.transform.gameObject != object_triggered)
    {
        Enemy_Type enemy_type = other.GetComponent<Enemy_Type>();
        if (SaveScript.class_Mage == true)
        {
            damage = damage * 6 / 5;
        }
        if(enemy_type.enemyType == Enemy_Type.EnemyType.Golem)
        {
            other.transform.gameObject.GetComponent<Golem_Movement>().full_HP -= (damage * 4) / 5; // 20 peer
cent magic decrease
            object_triggered = other.transform.gameObject;
        }
        else
        {
            other.transform.gameObject.GetComponent<EnemyMovement>().full_HP -= damage;
            object_triggered = other.transform.gameObject;
        }
    }
}

```

## 1.6 Реалізація функціоналу взаємодії з ігровими об'єктами.

### 1.6.1 Файл ItemPickUp.cs

```

public class ItemPickUp : MonoBehaviour
{
    private bool can_pick_up = true;
    private WaitForSeconds pickUp_Pause = new WaitForSeconds(0.0001f);

    public int number_of_pickedUp_items;

    public bool is_redMushroom = false;
    public bool is_blueFlower = false;
    public bool is_whiteFlower = false;
    public bool is_purpleFlower = false;
    public bool is_redFlower = false;

    public bool is_roots = false;
    public bool is_leaf = false;
    public bool is_keySimp = false;
    public bool is_keyGold = false;
    public bool is_monsterEye = false;

    public bool is_bluePotion = false;
    public bool is_greenPotion = false;
    public bool is_lazurePotion = false;
    public bool is_redPotion = false;

    public bool is_bread = false;
    public bool is_cheese = false;
    public bool is_meat = false;

    public bool is_purpleMushroom = false;
    public bool is_orangeMushroom = false;

    public bool is_loot_coin = false;

    public static bool is_keySimp_exist = false;
    public static bool is_keyGold_exist = false;

    public GameObject Inventory_Canvas;
    public AudioSource audio_Player;

    private void Start()
    {
        Inventory_Canvas = GameObject.Find("Inventory");
    }
}

```

```

audio_Player = Inventory_Canvas.GetComponent<AudioSource>();
if(is_loot_coin == true) // only 10 sec to pick up loot coins from enemy
{
    Destroy(gameObject, 10);
}
}

private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player") && can_pick_up == true)
    {
        can_pick_up = false;

        audio_Player.clip = Inventory_Canvas.GetComponent<Inventory>().pick_UP_SFX;
        audio_Player.Play();
        if (is_redMushroom == true)
        {
            if (Inventory.amount_of_redMushrooms == 0)
            {
                DisplayIcons();
            }
            Inventory.amount_of_redMushrooms++;
            Destroy(gameObject);
        }
        else if (is_blueFlower == true)
        {
            if (Inventory.amount_of_blueFlowers == 0)
            {
                DisplayIcons();
            }
            Inventory.amount_of_blueFlowers++;
            Destroy(gameObject);
        }
        else if (is_whiteFlower == true)
        {
            if (Inventory.amount_of_whiteFlowers == 0)
            {
                DisplayIcons();
            }
            Inventory.amount_of_whiteFlowers++;
            Destroy(gameObject);
        }
        else if (is_purpleFlower == true)
    }
}

```

```

{
    if (Inventory.amount_of_purpleFlowers == 0)
    {
        DisplayIcons();
    }
    Inventory.amount_of_purpleFlowers++;
    Destroy(gameObject);
}

else if (is_redFlower == true)
{
    if (Inventory.amount_of_redFlowers == 0)
    {
        DisplayIcons();
    }
    Inventory.amount_of_redFlowers++;
    Destroy(gameObject);
}

else if (is_roots == true)
{
    if (Inventory.amount_of_roots == 0)
    {
        DisplayIcons();
    }
    Inventory.amount_of_roots++;
    Destroy(gameObject);
}

else if (is_leaf == true)
{
    if (Inventory.amount_of_leaf == 0)
    {
        DisplayIcons();
    }
    Inventory.amount_of_leaf++;
    Destroy(gameObject);
}

else if (is_keySimp == true)
{
    if (Inventory.amount_of_keySimp == 0 && is_keySimp_exist == false)
    {
        DisplayIcons();
        is_keySimp_exist = true;
    }
    Inventory.amount_of_keySimp++;
    Inventory.player_has_a_common_key = true;
    Destroy(gameObject);
}
}

```

```

else if (is_keyGold == true)
{
    if (Inventory.amount_of_keyGold == 0 && is_keyGold_exist == false)
    {
        DisplayIcons();
        is_keyGold_exist = true;
    }
    Inventory.amount_of_keyGold++;
    Inventory.player_has_a_gold_key = true;
    Destroy(gameObject);
}

else if (is_monsterEye == true)
{
    if (Inventory.amount_of_monsterEye == 0)
    {
        DisplayIcons();
    }
    Inventory.amount_of_monsterEye++;
    Destroy(gameObject);
}

else if (is_bluePotion == true)
{
    if (Inventory.amount_of_bluePotion == 0)
    {
        DisplayIcons();
    }
    Inventory.amount_of_bluePotion++;
    Destroy(gameObject);
}

else if (is_greenPotion == true)
{
    if (Inventory.amount_of_greenPotion == 0)
    {
        DisplayIcons();
    }
    Inventory.amount_of_greenPotion++;
    Destroy(gameObject);
}

else if (is_lazurePotion == true)
{
    if (Inventory.amount_of_lazurePotion == 0)
    {
        DisplayIcons();
    }
    Inventory.amount_of_lazurePotion++;
    Destroy(gameObject);
}

```

```

    }

else if (is_redPotion == true)
{
    if (Inventory.amount_of_redPotion == 0)
    {
        DisplayIcons();
    }

    Inventory.amount_of_redPotion++;
    Destroy(gameObject);
}

else if (is_bread == true)
{
    if (Inventory.amount_of_bread == 0)
    {
        DisplayIcons();
    }

    Inventory.amount_of_bread++;
    Destroy(gameObject);
}

else if (is_cheese == true)
{
    if (Inventory.amount_of_cheese == 0)
    {
        DisplayIcons();
    }

    Inventory.amount_of_cheese++;
    Destroy(gameObject);
}

else if (is_meat == true)
{
    if (Inventory.amount_of_meat == 0)
    {
        DisplayIcons();
    }

    Inventory.amount_of_meat++;
    Destroy(gameObject);
}

else if (is_purpleMushroom == true)
{
    if (Inventory.amount_of_purpleMushroom == 0)
    {
        DisplayIcons();
    }

    Inventory.amount_of_purpleMushroom++;
    Destroy(gameObject);
}

```

```

        }

        else if (is_orangeMushroom == true)
        {
            if (Inventory.amount_of_orangeMushroom == 0)
            {
                DisplayIcons();
            }

            Inventory.amount_of_orangeMushroom++;
            Destroy(gameObject);
        }

        else if (is_loot_coin == true)
        {
            Inventory.gold += Random.Range(10, 50);
            Destroy(gameObject);
        }

        else
        {
            DisplayIcons();
            Destroy(gameObject);
        }

        // Destroy(gameObject);
        StartCoroutine(Reset_PickUp());
    }
}

void DisplayIcons()
{
    Inventory.newIcon = number_of_pickedUp_items;
    Inventory.iconUpdated = true;
}

public static void DestroyIcon()
{
    Inventory.newIcon = 0;
    Inventory.iconUpdated = true;
}

IEnumerator Reset_PickUp()
{
    yield return pickUp_Pause;
    can_pick_up = true;
}
}

```

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

\_\_\_\_\_ Едуард ЖАРИКОВ

“\_\_\_” \_\_\_\_\_ 2024 р.

**ІГРОВИЙ ЗАСТОСУНОК МОДЕЛЮВАННЯ ПОВЕДІНКИ  
ІНТЕЛЕКТУАЛЬНИХ АГЕНТІВ У 3D RPG З ВИКОРИСТАННЯМ  
ІГРОВОГО РУШІЯ UNITY.**

**Програма та методика тестування**

КПІ.ІТ-0223.045440.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

\_\_\_\_\_ Олексій ФІНОГЕНОВ

Нормоконтроль:

\_\_\_\_\_ Максим ГОЛОВЧЕНКО

Виконавець:

\_\_\_\_\_ Максим ТЕРЕШКОВИЧ

Київ – 2024

**ЗМІСТ**

1	ОБ'ЄКТ ВИПРОБУВАНЬ.....	3
2	МЕТА ТЕСТУВАННЯ .....	4
3	МЕТОДИ ТЕСТУВАННЯ.....	5
4	ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ .....	6

## 1      ОБ'ЄКТ ВИПРОБУВАНЬ

Об'єктом випробування є мануальне тестування 3D RPG ігрового застосунку. Мосю метою є перевірка інтерактивності, геймплею та візуальної привабливості гри. У процесі тестування я зосереджуєсь на оцінці ігрових механік, таких як система бою, прокачки, класів, характеристик, ворогів, економіки і тд. Я також перевіряю інтуїтивність користувацького інтерфейсу та легкість навігації по ігровому світу. Я також досліджую поведінку NPC та інші елементи штучного інтелекту, щоб переконатися, що вони відповідають задуму гри. Нарешті, я оцінюю загальну атмосферу та історію гри, щоб забезпечити захоплюючий ігровий досвід. Всі ці тести виконуються з метою створення якісного ігрового продукту, який доставить задоволення гравцям та відповідатиме високим стандартам ігрової індустрії.

## 2 МЕТА ТЕСТУВАННЯ

Метою тестування є перевірка наступних аспектів:

- функціональність: програмне забезпечення повинно належним чином виконувати всі функції, передбачені для його роботи, а також ефективно обробляти запити клієнтів і надавати коректну інформацію у відповідях.;
- зручність використання: інтерфейс програми повинен бути простим і зрозумілим для користувачів, щоб користувачі могли легко скористатися всіма необхідними функціями без будь-якого надання додаткової інформації про них.

### 3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;

Мануальне тестування варто обрати через його гнучкість та здатність швидко адаптуватися до змін у проекті. Воно дозволяє нам інтуїтивно знаходити баги та перевіряти програму з точки зору кінцевого користувача, забезпечуючи високу якість користувацького досвіду. Крім того, мануальне тестування не потребує створення складних скриптів, що економить час та ресурси.

## 4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Процес тестування складається з одного етапу.

Основний і єдиний це мануальне тестування. Легкість використання дає нам змогу швидко та інтуїтивно оцінити коректність роботи програми визначивши її основні помилки та баги. Для того, щоб перевірити працевдатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- Вручне тестування: Виконання тестів вручну за допомогою реальних користувальницьких сценаріїв. Цей метод дозволяє отримати оцінку відповідності програмного забезпечення вимогам користувачів та оцінити зручність використання.
- Тестування в реальній середовищі: Проведення тестів у реальній середовищі, де програмне забезпечення буде використовуватись. Це може включати тестування на реальних пристроях, у реальних мережах або у реальних умовах роботи.
- Тестування продуктивності: Оцінка продуктивності програмного забезпечення, включаючи його швидкодію, витривалість, масштабованість та інші параметри. Це може включати тестування завантаження, стрес-тестування та інші види тестування продуктивності.

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

\_\_\_\_\_ Едуард ЖАРИКОВ

“\_\_\_” \_\_\_\_\_ 2024 р.

**ІГРОВИЙ ЗАСТОСУНОК МОДЕЛЮВАННЯ ПОВЕДІНКИ  
ІНТЕЛЕКТУАЛЬНИХ АГЕНТІВ У 3D RPG З ВИКОРИСТАННЯМ  
ІГРОВОГО РУШІЯ UNITY.**

**Керівництво користувача**

КПІ.ІТ-0223.045440.05.34

“ПОГОДЖЕНО”

Керівник проєкту:

\_\_\_\_\_ Олексій ФІНОГЕНОВ

Нормоконтроль:

\_\_\_\_\_ Максим ГОЛОВЧЕНКО

Виконавець:

\_\_\_\_\_ Максим ТЕРЕШКОВИЧ

Київ – 2024

**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 .....	5

## 1 ПРИЗНАЧЕННЯ ПРОГРАМИ

"Echoes of Eternity" - це ігровий застосунок у жанрі RPG з видом від 3 особи у 3D. Даний застосунок дозволяє користувачам зануритися у захоплюючий тривимірний світ RPG, де вони можуть взяти участь у неймовірних пригодах, взаємодіяти з різноманітними персонажами, об'єктами та досліджувати ігровий всесвіт. Гравці мають можливість створювати та налаштовувати своїх персонажів, розвивати їхні навички та здібності, а також збирати предмети та ресурси для подальшого прогресу в грі. Ігровий процес включає в себе виконання завдань, боротьбу з ворогами, знаходження різних предметів та використання всіх магічних та бойових можливостей, що дозволяє створити динамічний та захоплюючий ігровий досвід.

Програма призначена для покращення ігрового досвіду користувачів.

Основні можливості користувача «Echoes of Eternity» включають:

- Створення та використання магії.
- Взаємодія з ігровими об'єктами, ворогами та НПС.
- Використання досвідченого бойової системи та пізнавання різноманітної поведінки ворогів.
- Класова система та можливість прокачки персонажа.

## 2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

### 2.1 Системні вимоги для коректної роботи

Мінімальна конфігурація технічних засобів:

- Windows;
- Intel Core i5 або AMD Ryzen 5 з частотою не менше 3.0 GHz.
- об'єм ОЗП: 8 Гб.
- NVIDIA GeForce GTX 1060.
- SSD 10 GB.

Рекомендована конфігурація технічних засобів:

- тип процесору: Intel Core i9 або AMD Ryzen 9;
- об'єм ОЗП: 96 Гб;
- NVIDIA GeForce GTX 4090 Suprim Liquid X.
- SSD 2 TB.

### 2.2 Завантаження застосунку

Ігровий застосунок інсталяється за допомогою .exe файл.

### 2.3 Перевірка коректної роботи

У Для перевірки коректності роботи застосунку потрібно інсталювати його, відкрити – і якщо у відкритому меню є вибір «NewGame» тоді застосунок працює коректно.

### 3 ВИКОНАННЯ ПРОГРАМИ

При відкритті застосунку гравець бачить основне меню гри. (рисунок 3.1). В ньому є можливість обрати «New Game» або продовжити вже існуючу «Continue». Якщо збережень немає, то кнопка «Continue» буде не доступна.

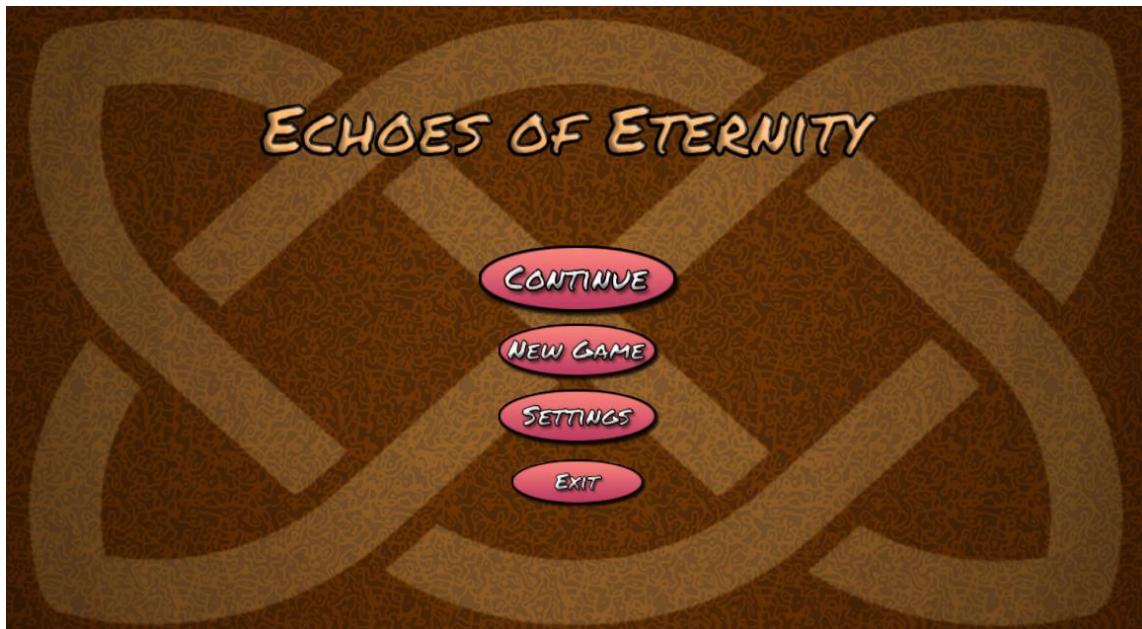


Рисунок 3.1 – Основне меню

Обирайте «New Game» у вас відкриється сцена вибору персонажа. (рисунок 3.1) В наявності є 6 різних персонажів та можливість задати кожному один з 3 класів. Кожен клас має свою унікальну можливість. Описожної з них також можна побачити на екрані. В ньому є можливість обрати «New Game» або продовжити вже існуючу «Continue». Якщо збережень немає, то кнопка «Continue» буде не доступна.



Рисунок 3.2 – Меню вибору персонажа

Після обрання персонажа та задання класу гравець потрапляє в ігровий світ. (рисунок 3.3).



Рисунок 3.3 – Ігровий світ

Після цього у вас є вибір робити все що вам забажається та досліджувати світ. В грі присутня міні-карта яка знаходиться у лівому верхньому кутку екрану. По центру знаходиться основні слоти UI гравця, в них можна вставляти створені гравцем заклинання та магію. Синя та Червона половина відповідає за параметри Здоров'я та Мани, а жовта полоса – це стаміна. Вона буде використовуватися для кожного удару гравця. Також в

лівому нижньому кутку екрану є книга. Натиснувши на ней відкривається основний інвентар гравця. (рисунок 3.4)



Рисунок 3.4 – Основний інвентар гравця

Ліворуч в слоти будуть з'являтися предмети, які були підібрані гравцем. Вони використовуються для створення заклинань. Проте з самого початку гри ця опція не доступна. Потрібно сходити в найближчий Паб та підібрати з стенду магічну книгу, яка дасть можливість створювати закляття. (рисунок 3.5). та (рисунок 3.6).



Рисунок 3.5 – Стенд з магічною книгою



Рисунок 3.6 – Відкрите меню створення заклинань за допомогою інгредієнтів

Для створення заклять треба обрати зазначені в книзі інгредієнти зі свого інвентарю. Якщо обрано коректний елемент – його іконка стане менш прозорою. (рисунок 3.7) При обранні всіх інгредієнтів – можна створити заклинання. Якщо ви випадково натиснули не на той елемент – він зникне але коли «Create», нічого не створиться, бо ви обрали не правильний варіант елементів. Тому будьте уважні – магія, це складна наука і не допускає помилок. Далі створена магія буде доступна у ячейках Spells (рисунок 3.8) звідки ви зможете додати її в основні ячейки шляхом натискання на закляття і обранням клавіш 1-8 (будь яку за бажанням) (рисунок 3.9). Але пам'ятайте – всі вони одноразові і після назначення клавіші – переназначити на іншу не вийде і треба буде шукати інгредієнти і створювати магію заново.



Рисунок 3.7 – Приклад відображення вже доданих інгредієнтів для закляття



Рисунок 3.8 – Створене закляття



Рисунок 3.9 – Додане заклинання на основну панель

В грі можна створити 6 видів заклинань та 7 видів магії. Різниця лише в тому що магія – багаторазова, в той час як заклинання лише одноразові. Але

багаторазова магія доступна лише після відкриття піктограми в найближчому підземеллі, де треба перемогти боса Голема. (рисунок 3.10)



Рисунок 3.10 – Голем та сама піктограма за ним

Після перемоги та підняття піктограми – відкриються відповідні слоти в інвентарі. (рисунок 3.11)

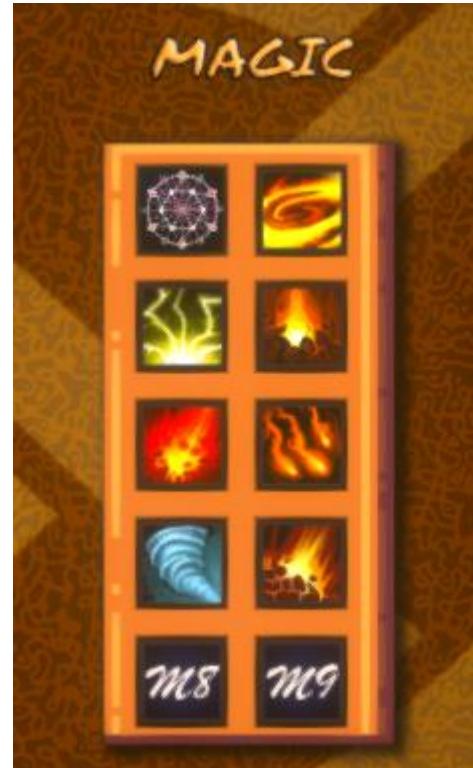


Рисунок 3.11 – Відкрита багаторазова магія

Наступний розділ інвентару відповідає за завдання «Task», які стоять перед гравцем. Їх задача направляти гравця в потрібні місця і за потрібними предметами.(рисунок 3.12) Для того щоб отримати завдання треба підійти до одного з НПС та обрати першу опцію в діалозі. (рисунок 3.13)



Рисунок 3.12 – Секція Завдань



Рисунок 3.13 – Діалогова варіація при спілкуванні з НПС

Наступний розділ інвентарю «Stats» відповідає за характеристики персонажа, зброю, рівень і тд. В ньому ви можете обрати, яку зброю використовувати та які характеристики прокачувати при піднятті рівня персонажа. Також там можна побачити кількість знищених ворогів, рівень персонажа та доступні бали прокачки характеристик.(рисунок 3.14)



Рисунок 3.14 – Секція характеристик та екіпірування

Остання секція – це карта світу «Мар». (рисунок 3.15) На ній відмічені всі основні місця цього світу, НПС, персонажи та активності. Гравець може спокійно орієнтуватися по світу за цими позначками.



Рисунок 3.15 – Карта світу

Подорожуючи світом на шляху гравця зустрічаються три різни види НПС персонажів. Кожен з них дає доступ до свого унікального магазину зі своїми цікавими товарами.

Перший з персонажів це – дівчина в Пабі. Вона продає інгредієнти, які відновлюють здоров'я гравця (рисунок 3.16).



Рисунок 3.16 – Магазин в Пабі

Другий з персонажів це – маг. Він продає інгредієнти, які використовуються для створення заклять (рисунок 3.17).



Рисунок 3.17 – Магазин чаклуна

Третій з персонажів це – чоловік в майстерні. Він продає зброю та екіпірування. Кожна зброя має свій стиль атаки, рівень шкоди, а броня захищає від шкоди на певний відсоток (легка – на 20% менше, важка – на 40% менше) (рисунок 3.18).



Рисунок 3.18 – Магазин в майстерні

В грі також присутні об'єкти з якими може взаємодіяти користувач. Це інгредієнти, скрині та ящики. Останні дві просто так підняті не вийде. Для скрині потрібен буде ключ, а ящик можна тільки розбити зброею. (рисунок 3.19)

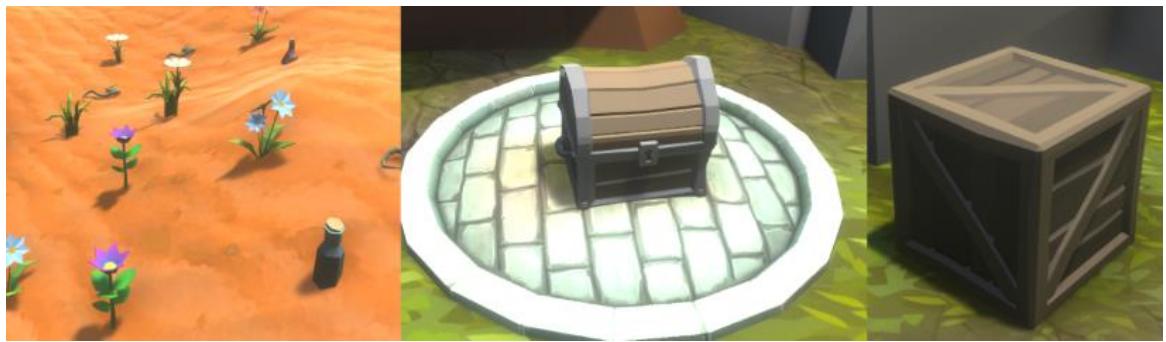


Рисунок 3.19 – Об’єкти для взаємодії

Подорожуючи світом ви будете зустрічати різних ворогів. Таких як Голем, Піглін, Скелет та Гоблін. Голем, як вже було показано, захищає підземелля та піктограму. Він агресивний та має широкий спектр атак. Скелет – одинак, який у разі відсутності навколо нього ворогів інших типів ворогів і будучи досить великим, викличе підмогу у вигляді двох маленьких скелетів(рисунок 3.20). Піглін – мирний, проте якщо гравець вдарить його першим – почне атакувати, але при втраті більшої частини здоров’я почне тікати від гравця, намагаючись врятувати своє життя(рисунок 3.21). Основна задача гобліна – патрулювання території. Він напевно самий небезпечний ворог – так, як він має здатність уникати атак персонажа шляхом перекатів і може помітити вас на досить великій відстані. (рисунок 3.22)

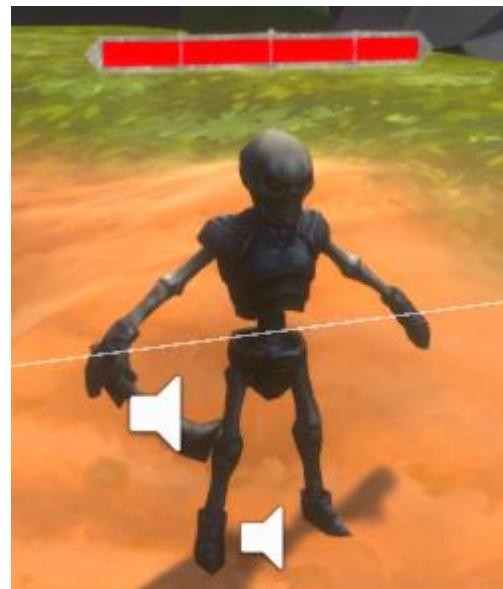


Рисунок 3.20 – Скелет



Рисунок 3.21 – Піглін



Рисунок 3.22 – Гоблін

Якщо гравець хоче зберегти гру – потрібно підійти до згарища та в меню на екрані обрати «так» при питанні «Чи хочете ви зберегти гру?» (рисунок 3.23)



Рисунок 3.23 – Меню збереження гри

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

\_\_\_\_\_ Едуард ЖАРИКОВ

“\_\_\_” \_\_\_\_\_ 2024 р.

**ІГРОВИЙ ЗАСТОСУНОК МОДЕЛЮВАННЯ ПОВЕДІНКИ  
ІНТЕЛЕКТУАЛЬНИХ АГЕНТІВ У 3D RPG З ВИКОРИСТАННЯМ  
ІГРОВОГО РУШІЯ UNITY**

**Графічний матеріал**

КПІ.ІТ-0223.045440.06.99

“ПОГОДЖЕНО”

Керівник проєкту:

\_\_\_\_\_ Олексій ФІНОГЕНОВ

Нормоконтроль:

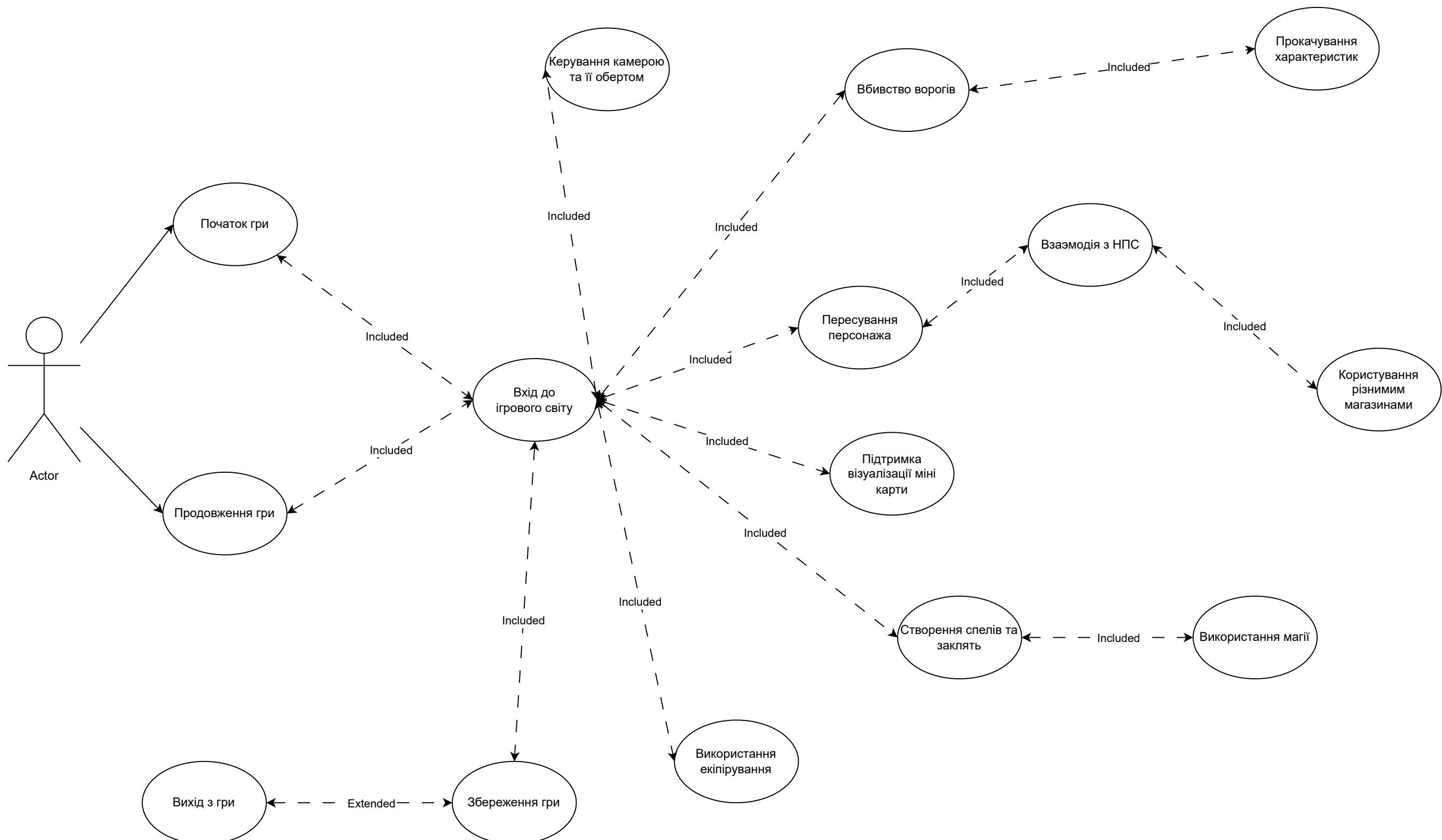
\_\_\_\_\_ Максим ГОЛОВЧЕНКО

Виконавець:

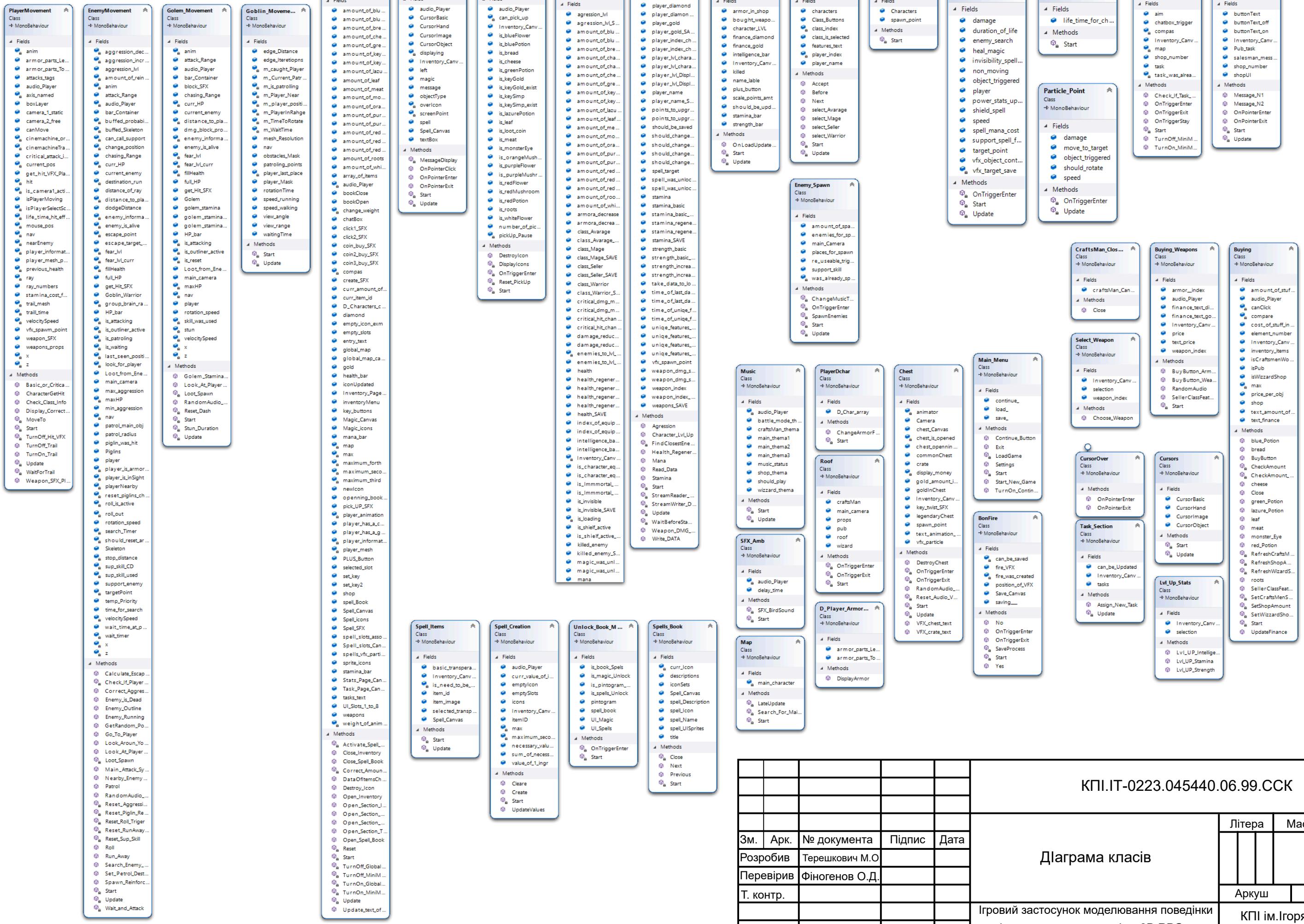
\_\_\_\_\_ Максим ТЕРЕШКОВИЧ

Київ – 2024

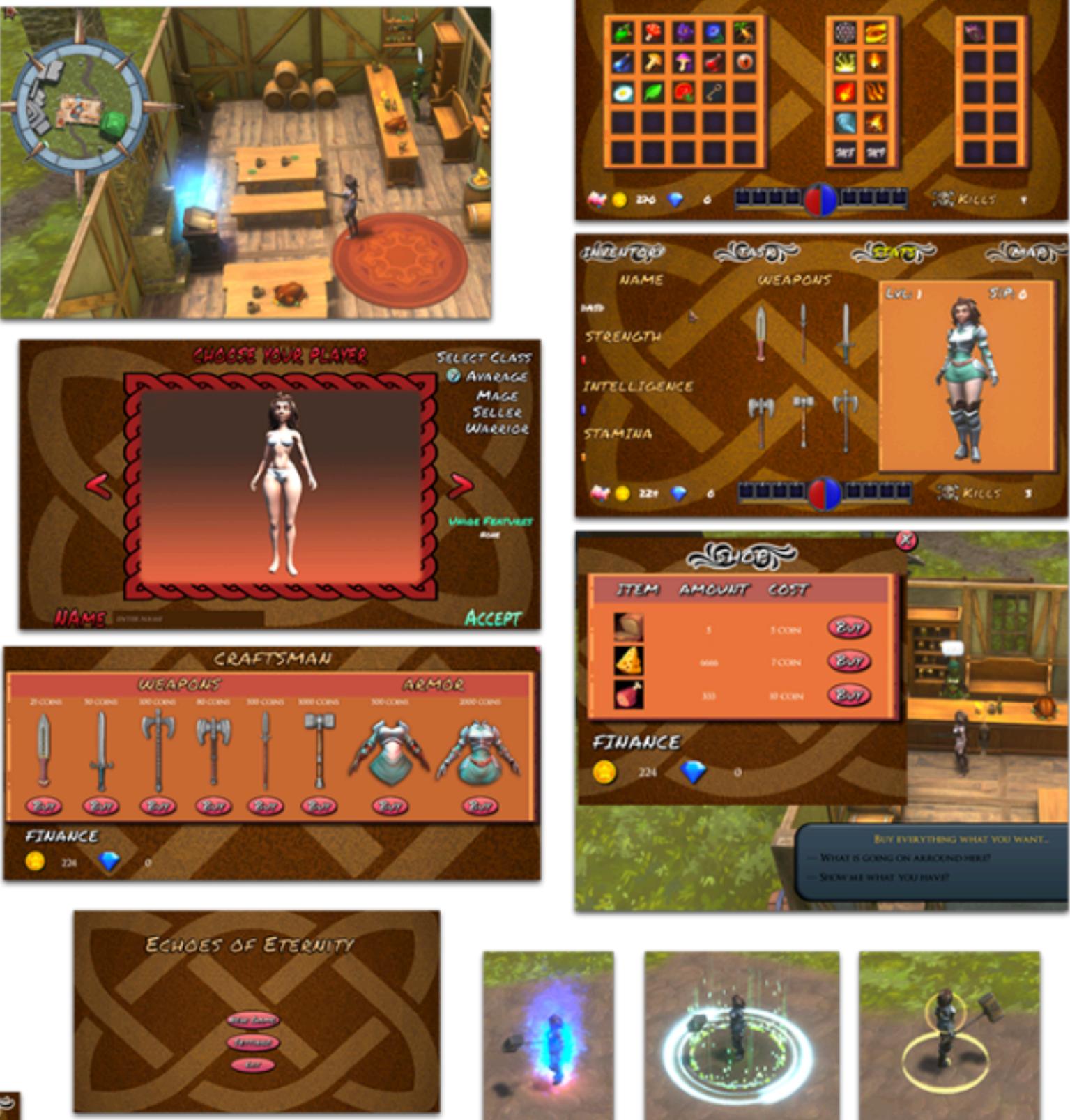
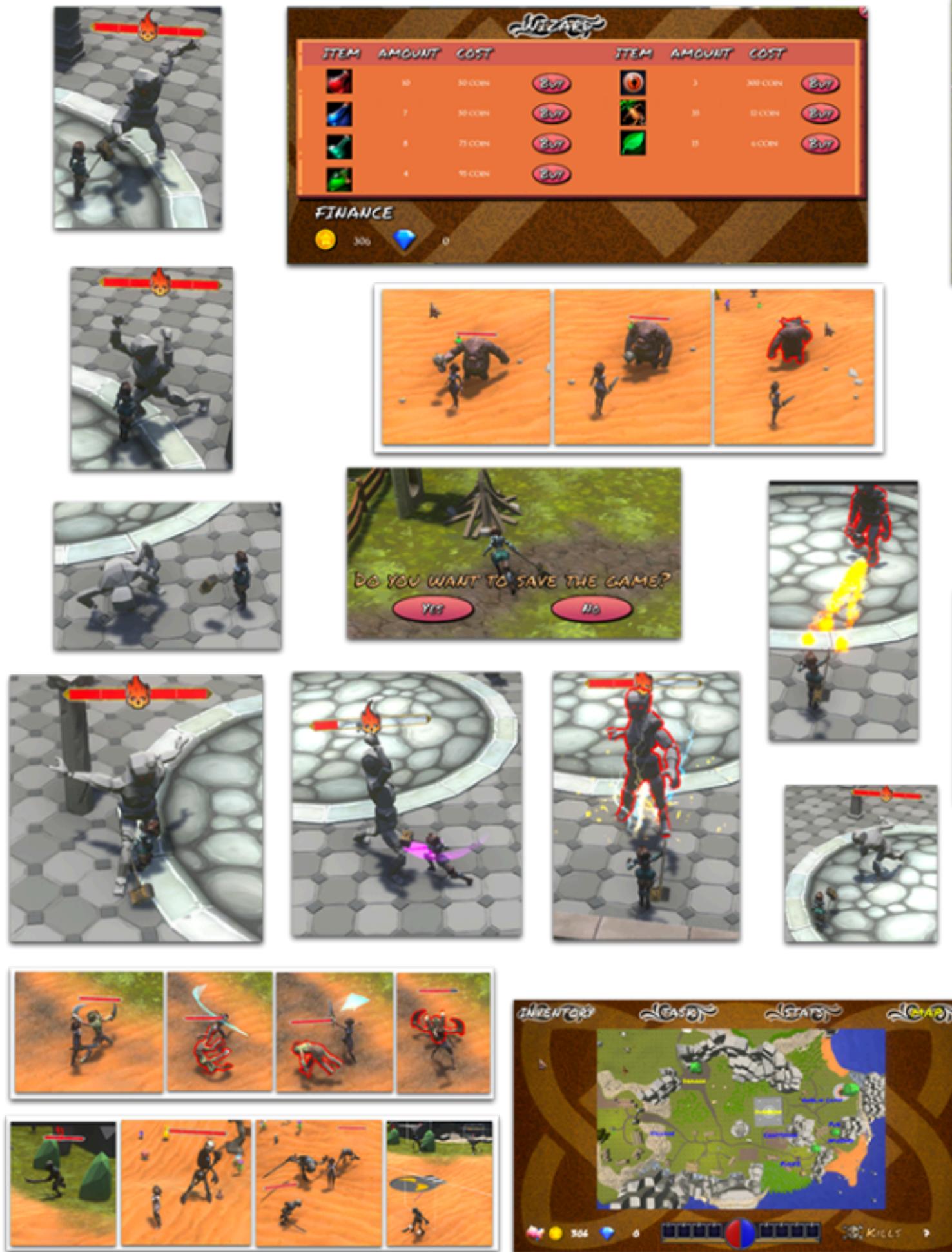
## RPG Echoes of Eternity



					КПІ.ІТ-0223.045440.06.99.ССВ		
Зм.	Арк.	№ документа	Підпис	Дата	Діаграма варіантів використань		
Розробив		Терешкович М.О.			Літера		
Перевірив		Фіоногенов О.Д.			Маса		
Т. контр.					Масштаб		
Н. контр.		Головченко М.М.			Аркуш		
Затвердив		Жаріков Е.В.			Аркушів		
Ігровий застосунок моделювання поведінки інтелектуальних агентів у 3D RPG з використанням ігрового рушія Unity						КПІ ім. Ігоря Сікорського Кафедра ІПІ гр. IT-02	



Зм.	Арк.	№ документа	Підпис	Дата
Розробив		Терешкович М.О		
Перевірив		Фіоногенов О.Д.		
Т. контр.				
Н. контр.		Головченко М.М		
Затвердив		Жаріков Е.В.		



КПІ.ІТ-0223.045440.06.99.КЕ

Зм.	Арк.	№ документа	Підпис	Дата	Скріни програми		Літера	Масштаб
					Скріни	Програми		
Розробив		Терешкович М.О.						
Перевірив		Фіоногенов О.Д.						
Т. контр.								
Н. контр.		Головченко М.М.					Аркуш	Аркушів
Затвердив		Жаріков Е.В.						

Ігровий застосунок моделювання поведінки інтелектуальних агентів у 3D RPG з використанням ігрового рушія Unity

КПІ ім. Ігоря Сікорського  
Кафедра ІПІ  
гр. IT-02