МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Розробка комп'ютерної рольової фентезі-гри з використанням Unity**

Виконав: студент 4 курсу, групи 6.1210-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
І.О. Красножон
(ініціали та прізвище)
Керівник доцент каф. ЕІС та ПЗ, к.ф.-м.н., доцент
В.І. Попівщий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дісітел» П. О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення

Рівень вищої освіти _____перший (бакалавський)_____

Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)

Освітня програма ___Програмне забезпечення систем_____
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Тетяна КРИТСЬКА
“_01_” __березня_____2024_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

_____Красножону Івану Олеговичу_____
(прізвище, ім’я, по батькові)

1.Тема роботи: **Розробка комп’ютерної рольової фентезі-гри з використанням Unity**

керівник роботи Попівщий Василь Іванович_____
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від ___26.12.2023 № 2215-с_____

2. Строк подання студентом кваліфікаційної роботи_____07.06.2024_____

3. Вихідні дані кваліфікаційної роботи

- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми створення відеоігор на Unity;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- тестування програмної системи та розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата
		Завдання прийняв

7. Дата видачі завдання 01.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи бакалавра	Примітка
1	Аналіз предметної області	05.03 — 10.03.2024	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з викладачем	11.03 — 15.03.2024	виконано
3	Аналіз існуючих рішень	16.03 — 20.03.2024	виконано
4	Дослідження ігрового рушія Unity	21.03 — 31.03.2024	виконано
5	Визначення основних систем для реалізації в застосунку	01.04 — 10.04.2024	виконано
6	Узгодження подальших дій з науковим керівником	11.04 — 20.04.2024	виконано
7	Створення відеогри з реалізацією визначених основних систем	21.04 — 28.05.2024	виконано
8	Представлення отриманих результатів науковому керівнику та узгодження подальшого плану дослідження	29.05 — 30.05.2024	виконано
9	Усунення додаткових проблем та «багів»	01.06 — 02.06.2024	виконано
10	Оформлення звіту	02.06 — 07.06.2024	виконано

Студент _____ І.О. Красножон
(підпис) (прізвище та ініціали)

Керівник роботи _____ В.І. Попівший
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок — 78

Рисунків — 18

Джерел — 17

Красножон І.О. Розробка комп’ютерної рольової фентезі-гри з використанням Unity: кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник В. І. Попівщий. Запоріжжя : ЗНУ, 2024. 78 с.

Мета полягає в створенні прототипу відеогри жанру фентезі-рпг з використанням ігрового рушія Unity для операційної системи Windows, котра буде складатися з основних систем, котрі можна використовувати для додання нового контенту до відеогри. Наприклад зброю, ворогів, нові предмети для використання, предмети з якими може взаємодіяти гравець.

У процесі розробки була розглянута проблема розробки відеоігор на Unity. В результаті був розроблений прототип відеогри жанру фентезі-рпг з основними системами. Даний застосунок має в собі системи, котрі можна розширювати та доповнювати, а саме: систему нанесення шкоди, створення ворогів, створення різноманітних ворогів, різноманітної зброї, ігровий магазин, інвентар, перемикання анімацій для гравця і ворогів та багато іншого.

Ключові слова: *unity, інді гра, прототип, система, вороги, данжі, інвентар, зброя, предмети, C#.*

ABSTRACT

Pages — 78

Pictures — 18

Sources — 17

Krasnozhon I.O. Development of a computer role-playing fantasy game using Unity: qualification thesis of the bachelor of specialty 121 "Software engineering" / Science. supervisor V. I. Popivchshiy. Zaporizhzhia : ZNU, 2024. 78 p.

The goal is to create a prototype of a fantasy RPG video game using the Unity game engine for the Windows operating system, which will consist of core systems that can be used to add new content to the video game. For example, weapons, enemies, new items to use, items that the player can interact with.

In the development process, the problem of developing video games on Unity was considered. As a result, a prototype of a fantasy RPG video game with basic systems was developed. This application has systems that can be expanded and supplemented, namely: damage system, creation of enemies, creation of various enemies, various weapons, in-game store, inventory, switching animations for the player and enemies, and much more.

Keywords: *unity, indie game, prototype, system, enemies, dungeons, inventory, weapons, items, C#.*

ЗМІСТ

ВСТУП

Актуальність теми

В сучасному світі розробка комп’ютерних інді ігор, зокрема фентезі-рпг проектів, вже давно перестала бути невеликим хоббі, а враховуючи величезний наплив нових гравців з 2020 року, пов’язаний з поширенням ковіду, ще більше людей почало грати в них, а також створювати їх самостійно чи в невеликих командах, створюючи сміливі проекти, в яких експерементують та іноді створюють унікальні проекти, які можуть “вистрелити” на своєму виході саме через те, що вони приносять щось нове в ігрову індустрію.

Сам жанр ігор фентезі-рпг теж є досить актуальною темою дослідження, яка ще довго не буде втрачати свою популярність. Великими прикладами таких ігор є серія ігор TES, Відьмак, Baldur`s Gate, Dark Souls і багато інших. Всі з них надають гравцю можливість спробувати себе в ролі героя, який проживає унікальну історію.

Також дані ігри надають гравцю можливість підвищувати рівень героя, відкриваючи нові можливості та уміння, котрі додають гравцю ще більшого відчуття прогресу.

На даний момент поєднуючи високий інтерес людей до інді ігор та популярність жанру фентезі-рпг, створення гри стає актуальною та захоплюючою темою для дослідження.

В сьогодення в інді-розробників можуть виникнути різноманітні труднощі при створенні своєї гри, так-як вони мають обмежений чи нульовий бюджет та невелику команду або і зовсім розробляють гру сам-на-сам. Але деяка частина проблем компенсується наявністю великої кількості безкоштовних інструментів та технологій, за допомогою яких можна зробити майже будь-який елемент відео-гри. Але навіть з наявністю таких плюсів сьогодення при розробці інді-гри в жанрі фентезі-рпг можуть виникнути такі проблеми:

1. Відсутність досвіду. Створення відео-гри в жанрі фентезі-рпг на ігровому рушії може бути складним завданням, так-як необхідно знати те, як можуть взаємодіяти компоненти ігрового рушія. Без досвіду програмування, створення ігрових механік буде складною задачею. Також за відсутності навичок в інших програмах, котрі можуть допомогти зробити контент для своєї гри, створення стилізованого ігрового світу стане ще більшим викликом, так-як у безкоштовному доступі не завжди можна знайти саме такі ігрові ресурси, які потрібні.

2. Малий бюджет. Розробці відеогри в жанрі фентезі-рпг може потребувати великих витрат на ігрові асети. Наприклад готові моделі, анімації, звуки, текстури, шейдери або навіть готові механіки можуть суттєво прискорити процес створення власної гри, але за відсутністю чи за наявності малого бюджету може виникнути необхідність створювати ресурси гри власноруч.

Мета дослідження

Головною метою даного дослідження є розробка прототипу рольової фентезі гри за допомогою ігрового рушія Unity та мови програмування C#. Цей проект спрямований на створення невеликої гри в жанрі фентезі-рпг з цікавим геймплеєм, який може зацікавити деяких гравців.

Завдання дослідження

У процесі дослідження будуть вирішуватися такі завдання, як:

- Аналіз літературних джерел про створення ігор на ігровому рушії Unity. Дане завдання дозволить зрозуміти основні труднощі, які можуть виникнути в процесі розробки гри, а також дізнатися більше про взаємодію компонентів ігрового рушія, що поєднуються в геймплеї.
- Аналіз існуючих ігор в жанрі фентезі-рпг для визначення основних механік геймплею. Дане завдання дозволить виділити особливості

жанру фентезі-рпг, котрі можна буде використати при доданні контенту в гру.

- Реалізація визначених основних ігрових механік та систем, які актуальні для жанру фентезі-рпг, а саме: інвентар гравця, його переміщення по ігровому світі, створення штучного інтелекту ворогів, створення системи покращення головного героя з підвищенням ігрового рівня, а також система данжів, в якій головний герой зможе зустрітися сам на сам з ворогами, та отримати за це деяку нагороду.
- Для створення гри використовувати ігровий рушій Unity та мову програмування C#

Об’єкт дослідження

Об’єктом дослідження являється процес розробки ігрових механік, систем для створення комп’ютерної гри в жанрі фентеі-рпг.

Предмет дослідження

Предметом дослідження є ігрові механіки, елементи та системи, які притаманні іграм в жанрі фентезі-рпг.

Методи дослідження

В процесі дослідження використовувались такі методи:

- Аналіз успішних існуючих ігор в жанрі фентезі-рпг
- Програмування з використанням Unity фреймворку та мови програмування C#.
- Емпіричні дослідження створених ігрових механік та систем в процесі створення продукту.

Практичне значення існуючих результатів

Отриманий результат дослідження, а саме комп’ютерна гра в жанрі фентезі-рпг може стати цікавим та захоплюючим продуктом на ринку інді ігор,

який може стати популярним серед гравців, цікавим прикладом для розробників інді ігор, а також може принести розробнику деякий фінансовий успіх.

Глосарій

Інді ігри (англ. Indie games) — це невеликі ігри, в розробці яких зазвичай приймають участь від одного до десяти розробників без фінансової підтримки видавництв та спонсорів.

Ігровий рушій (англ. Game engine) — це програмна частина, яка створена для прискорення розробки ігор, спрощення самої розробки відеоігор.

Рендер пайплайн (англ. Render Pipeline) — це деякий набір кроків, які проходять дані для того, щоб отримати вихідне зображення.

Квест (англ. Quest) — це деяке завдання, за виконання якого гравець може отримати ту чи іншу винагороду.

Unity — це один з найпопулярніших ігрових рушіїв, що використовується для розробки ігор та інтерактивних додатків для багатьох платформ, таких як: персональні комп'ютери, мобільні телефони на базі операційної системи Андроїд та iOS, PlayStation, Xbox, Nintendo Switch та інші. На даний момент для програмування в даному ігровому рушії використовується мова програмування C#, а також з недавніх пір має свою мову візуального програмування.

Фентезі-рпг — це один з багатьох жанрів комп'ютерних ігор, який відображає світ з фантастичними елементами, де гравці може спробувати себе в ролі персонажу, який взаємодіє зі світом даної гри.

Мова програмування C# — об'єктно орієнтована мова програмування з безпечною типізацією, яка створена компанією "Майкрософт". Синтаксис даної мови схожий на C++, а ще більше на Java. Дана мова програмування має строго статичну типізацію.

Шейдер (англ. Shader) — це деяка програма, яка виконується на графічному процесорі, котра тим чи іншим чином впливає на зображення, використовуючи пікселі, вершини, текстури і так далі.

Данжі (англ. Dangeon) — це деякі ігрові локації, на яких гравець зустрічається з одним або бореться супротивниками, котрих необхідно побороти. Зазвичай успішне проходження данжів нагороджується ігровою валютою та/або ігровим досвідом для підвищення рівня героя.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ЖАНРУ ФЕНТЕЗІ

1.1 Визначення жанру фентезі-рпг

Фентезі-рпг — це один з багатьох жанрів комп’ютерних ігор, дія якої відбувається в вигаданому світі, гравець має можливість підвищувати свій рівень та покращувати за рахунок підвищення рівня свої характеристики та відкривати уміння, а також може зустріти фантастичних створінь, наприклад ельфів чи гоблінів.

Основні механіки даного жанру включають в себе:

1. Переміщення головного героя по ігровій мапі
2. Інвентар гравця, який зберігає всі його речі, такі як зброю, їжу, зілля і так далі.
3. Прокачка головного героя, котра дозволяє йому підвищити свої шанси на виживання, а також відкривати нові уміння, котрі полегшують проходження гри.
4. Система квестів, за які головний герой може отримувати різноманітні нагороди або досвід для підвищення рівня головного героя.

Фантастичний світ даного жанру може включати в себе такі сутності, які можуть бути відсутніми або присутніми за задумкою автора:

1. Гобліни — невеликі швидкі істоти, зазвичай з зеленою шкірою та довгими вухами, які можуть бути як ворогами, так і союзниками.
2. Демони — потойбічні істоти, які в більшості випадків мають злі наміри на надприродні сили.
3. Ліч — одночасно живі та неживі істоти, які часто виступають як вороги. Їх поява в фентезі світі пов’язана з некромантією або магією.
4. Ельфи — довгожителі фентезі світу, які часто в фентезі іграх володіють сильною магією, або стрільбою з лука.
5. Дракони — могутні містичні істоти, які часто виступають в грі як боси, або сильні вороги. В деяких випадках — союзники.

Цей список можна продовжувати дуже довго, але види створінь, які зустрічаються найчастіше були наведені.

1.2 Огляд існуючих рішень відеоігор в жанрі фентезі-рпг

За останні два десятиліття вийшла велика кількість ігор в даному жанрі. У кожної з них своя історія, свій стиль та інші особливості, котрі роблять їх унікальними. Гравці полюбили ігри жанру фентезі-рпг і таким чином на світ з’явилось багато хороших ігор в жанрі фентезі-рпг. Деякі з найпопулярніших ігор даного жанру:

- The Elder Scrolls V: Skyrim.
- The Witcher 3: Wild Hunt.
- Elden Ring.
- Starfield.

1.2.1 The Elder Scrolls V: Skyrim

Розглянемо The Elder Scrolls V: Skyrim як приклад гри, котру гравці люблять та готові перепроходити багато разів уже багато років.

«Скайрім» вийшла на світ 11 листопада 2011 року компанією «Bethesda Game Studios» одразу на трьох платформах, а саме на персональних комп’ютерах на базі Windows, Play Station 3 та Xbox 360. Також дана гра перевипускалась декілька разів з невеликими покращеннями графіки та з деякими доповненнями.

Провівши огляд даної гри можна виділити такі особливості:

- Персонажа гри можна покращувати, шляхом отримання навичок у різних областях (див. Рис 1), та підвищуючи рівень самого персонажу. Навички в різних областях підвищуються шляхом частого виконання деяких дій. Наприклад персонаж гри може ходити навприсядки та бути таким чином в режимі маскування. Якщо робити це часто, то персонаж

буде все краще і краще маскуватися і його менше будуть помічати вороги.

- Також головний герой може покращувати свої навички виконуючи різноманітні дії і отримувати уміння для деяких гілок навичок, наприклад гілки бойових навичок, магічних навичок, злочинних навичок і так-далі.

Рисунок *1 — Підвищення рівня навички підкрадання, а також підвищення загального рівня персонажу в TES: Skyrim*

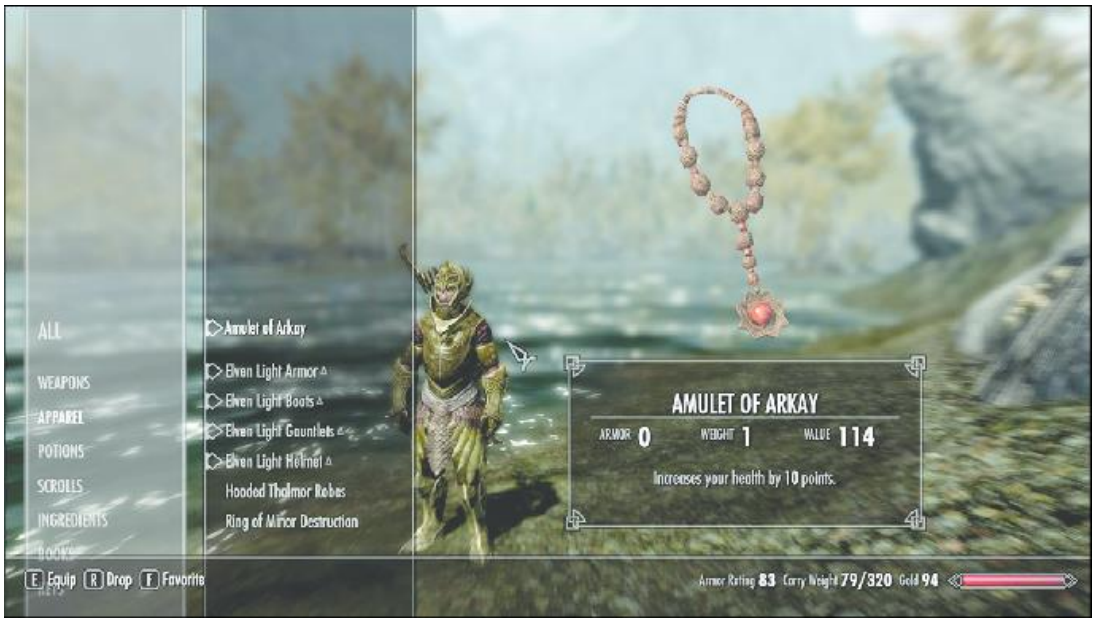

Рисунок *2 — Інвентар гравця в TES: Skyrim*

- У персонажа є інвентар (див. Рис. 2), в котрому він зберігає різноманітні предмети. Наприклад зілля, артефакти, зброю їжу і так далі. Головний герой може використовувати дані предмети з інвентаря в процесі гри. Для прикладу гравець може екіпірувати зброю, яка є в інвентарі.

Рисунок *3 — Дракон як один з представників фантастичних істот в TES: Skyrim*

- “Скайрім” також має фантастичних істот. Наприклад, дракони (див. Рис. 3) — котрі виступають як ключові антагоністи, вурдалаки — це перетворені вовколюди, які є ворожими до гравця. Вони можуть бути як і звичайними вовколюдами, але вночі перетворюються в агресивних вурдалаків. Також в даній грі є гобліни, орки, тролі, ліч та інші фантастичні істоти, котрі роблять цей світ фантастичним.

Це лише основні особливості гри, котрі полюбилися гравцям. Завдяки унікальному стилю гри, TES: Skyrim отримала високі оцінки критиків та зібрала ряд нагород, в тому числі нагороду на звання кращої гри 2011 року Spike Video Game Award, звання «гра року» та «рольова гра року» від IGN та Gamespot. Завдяки широкому розголосу про дану гру вона полюбилась фанатами та притягнула багато нових гравців до себе і таким чином «Скайрім» здобула великі продажі, коли в 2014 році продажі даної гри

досягли 29 мільйонів копій, а після перевипуску з покращеннями в 2016 продажі гри перевалили за 30 мільйонів копій, а в 2023 році TES: Skyrim увійшла в топ—10 ігор з найбільшими продажами, так-як копію даної гри купили уже 60 мільйонів разів.

Завдяки великому успіху дана гра вийшла також на таких платформах, як Play Station 4 та 5, Xbox Series One, S та X, а також була портована на портативну консоль Nintendo Switch.

1.3.2 The Witcher 3: Wild Hunt

The Witcher 3: Wild Hunt, а в перекладі з англійської Відьмак 3: Дике Полювання вийшла 19 травня 2015 року компанією CD Project на таких платформах, як персональні комп'ютери на базі Windows, Play Station 4 та Xbox One, а пізніше в 2019 році була перевипущена на Play Station 5, Xbox Series S/X та на портативній консолі Nintendo Switch.

При розгляді гри The Witcher 3: Wild Hunt, то побачимо співпадіння особливостей з попередньою грою, а саме:

- Інвентар головного героя (див. Рис 4), в якому «Геральт», головний герой гри, зберігає предмети

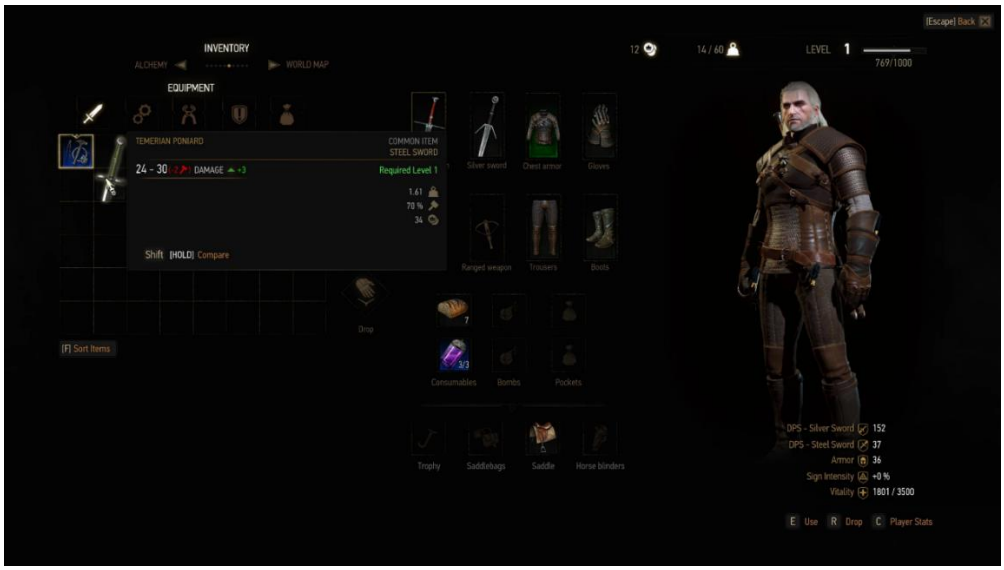

Рисунок *4 — Інвентар гравця в The Witcher 3: The Wild Hunt*

- Підвищення рівня та покращення навичок, котрі створюють відчуття прогресії для гравця та полегшують проходження гри. Підвищення рівня дивитись на рисунку 5.

Рисунок 5 — *Отримання нового рівня персонажу в The Witcher 3: The Wild Hunt*

- Наявність фантастичних істот. Для прикладу наведений грифон (див. Рис. 6).

Рисунок 6 — *Грифон в грі The Wither 3: The Wild Hunt*

Гра також отримала велику популярність серед любителів жанру фентезі-рпг та отримала велику кількість нагород на деяких виставках та великих подіях. Також хочу сказати, що історія даної гри взята із серії одноіменних книг, а також через великий успіх третьої гри серії «Відьмак» вийшов однойменний серіал від **Netflix**, що спровокувало нову хвилю нових гравців, котрі хотіли побачити історію уже знайомих їм персонажів більш детально.

1.3.3 Elden Ring

Elden Ring вийшла в 2022 році студією **FromSoftware**. Дана гра надає гравцю можливість подорожувати по відкритому світу з шести основних зон, паралельно б'ючись з різноманітними противниками, кожен з яких має свої унікальні атаки та тактики, що робить бойову систему даної гри дуже цікавою, а також в даній грі зустрічаються боси, котрі привносять в гру додаткові можливості гравцям перевірити свої уміння.

При огляді гри ми зможемо знайти такі основні особливості гри:

1. Інвентар головного героя на рисунку 7.

Рисунок *7 — Інвентар головного героя*

2. Підвищення рівня героя на рисунку 8.

Рисунок *8* — *Система підвищення рівня головного героя*

3. Фантастичні істоти. Для прикладу наведений один з босів гри на рисунку 9.

Рисунок *9* — *Бос «Grafted Scion»*

Дана гра є дуже різноманітною та має велику кількість інших фантастичних істот, котрих можна перечислювати дуже довго. Загалом дана

гра, як і попередні здобула велику популярність серед гравців, а також запам’яталась своєю великою складністю, за допомогою якої гравцю доводиться для кожного нового супротивника знаходити свій підхід.

Даний список чудових прикладів фентезі ігор можна продовжувати дуже довго, але ще необхідно сказати про ще одну невід’ємну частину фентезі-рпг, а саме — бойова система, яка зазвичай включає в себе систему ближнього бою, дальнього бою, а також використання магії, як для посилення самого головного героя, так і для самого бою, як наприклад Геральт з гри «The Witcher 3: Wild Hunt» може використовувати різні заклинання, коли деякі з них можуть надавати йому тимчасовий щит, котрий захищає його від наступної пропущеної атаки і водночас він має заклинання, котре може тимчасово привести ворога в стан приголомшення, під час якого ворог не може нічого робити декілька секунд, що відкриває для головного героя вікно для атаки.

Аналіз показав, що в залежності від того яку атмосферу ігровий дизайнер хоче передати гравцю, то він може змінювати наявність тої чи іншої фантастичної істоти, наявність деяких умінь у головного героя, котрі можуть допомогти йому в бою, а також можливість підвищувати рівень головного героя, що дає гравцю відчуття прогресу, не забуваючи про надану можливість гравцю придбати або знайти в ігровому світі різноманітні предмети, котрі також можуть допомогти в проходженні даної гри.

1.4 Висновки до першого розділу

В ході огляду існуючих рішень дослідили найпопулярніші ігри жанру фентезі-рпг та були визначені основні особливості даного жанру.

2 МЕТОДИ ТА ЗАСОБИ ВИКОРИСТАНІ ПРИ ВИКОНАННЯ РОБОТИ

2.1 Огляд літературних джерел

В сучасній ігровій індустрії ігровий рушій грає важливу роль, так-як він забезпечує розробників деяким набором інструментів, котрий може сильно прискорити та спростити розробку відеоігор. Unity — один з найактуальніших ігрових рушіїв, котрі використовуються в найрізноманітніших командах починаючи від інді розробників, закінчуючи великими студіями розробки ігор.

Основною метою при підготовці дипломної роботи є огляд літературних джерел, так-як це може розширити розуміння обраної теми та розглянути ігри жанру фентезі-рпг в сучасній ігровій індустрії.

При написанні дипломної роботи велика увага приділяється збору і аналізу літературних джерел, котрі становлять фундаментальну основу дипломної роботи. При аналізі літературних джерел використовувались онлайн-бібліотеки, офіційна документація Unity та академічні журнали.

При аналізі літературних джерел велика увага була зосереджена на основних темах галузі розробки відеоігор на ігровому рушії Unity, основних інструментах та використання фреймворку Unity для створення ігрових механік. Для створення різноманітних ігрових механік можуть використовуватися різні інструменти з урахуванням того, як це впливає на геймплей.

Під час аналізу літературних джерел були визначені наступні основні теми:

1. Опис жанру відеоігор фентезі-рпг, які його основні особливості, найпопулярніші представники даного жанру, історія зародження жанру.
2. Дослідження ігрового рушія Unity, його основні інструменти, історія зародження, популярні проекти створені на основі ігрового рушія Unity.
3. Використання фреймворку ігрового рушія Unity для створення ігрових механік та їх поєднання для створення геймплею гри.

Жанр відеоігор фентезі-рпг може бути досить складним в реалізації, так як даний жанр може поєднувати в собі велику кількість ігрових механік, котрі повинні взаємодіяти один-з-одним. Аналіз представників жанру допоможе краще виразити основні особливості жанру, щоб в подальшому було від чого відштовхуватись при доданні нової механіки.

Дослідження ігрового рушія Unity та аналіз інструментів допоможе зрозуміти основні можливості даного ігрового рушія, щоб в подальшому знати які елементи є і які компоненти необхідно використовувати для виконання деяких дій та для створення ігрових механік. Також при аналізі основних інструментів ігрового рушія є можливість зрозуміти за що відповідає кожен компонент ігрового двигуна.

Використання ігрового рушія Unity має під собою необхідність використання його фреймворку для взаємодії компонентів ігрового рушія між собою. Повне розуміння того, як можуть взаємодіяти різноманітні компоненти ігрового рушія за допомогою його фреймворку допоможе пришвидшити створення нових ігрових механік та полегшити створення гнучкої архітектури, котру можна буде зручно підтримувати.

Великою проблемою під час створення інді фентезі-рпг на Unity є те, що не завжди можна знайти безкоштовні ресурси на бажану тематику для додання їх в гру. Дану проблему розробник може нівелювати при наявності деяких навичок наприклад в створенні 3D моделей, запису та редагуванні звуків, анімацій і так далі, але не завжди у розробника є час на освоєння нових умінь для створення своїх ігрових ресурсів.

Також ще одною проблемою може стати складність розробки ігрових механік та поєднання їх в архітектурі. Дана складність більш помітна в командах з однієї людини, коли їй приходиться реалізувати велику кількість механік. Саме поєднання даних ігрових механік може стати випробуванням для невеликих команд, так як без хорошої архітектури проєкту, даний проєкт буде складно відлагоджувати та підтримувати.

На сьогоднішній день у людей досить часто змінюються інтереси до різноманітних жанрів відеоігор і часто інді-розробники та невеликі команди не вгадують з тим, чого саме хочуть геймери, і створені проєкти, скільки б часу та грошей не було в них вкладено, часто стають не поміченими, або і зовсім не доходять до свого релізу.

Але не дивлячись на всі ці проблеми, котрі можуть виникнути в процесі розробки фентезі-рпг гри на ігровому рушії Unity можуть нівелюватися можливістю виникнення таких подій:

1. Можливість створити чудовий продукт та стати поміченим або навіть знайти інвестора та/або видавця, котрий може взяти на себе деякі витрати на рекламу продукту та на підтримку проєкту.
2. Якщо проєкт цього заслуговує, то можна активувати збір на Kick-Starter і якщо все буде добре, то гра може отримати деяку кількість фанатів, котрі можуть підтримати проєкт фінансово, чи особисто. Наприклад — допомогти в розробці тих чи інших елементів гри.
3. Якщо гра так сказати вгадає те, що може “вистрелити” при публікуванні гри, то вона може швидко здобути свою аудиторію та почати приносити деякі фінанси.

2.2 Ігровий рушій Unity

Unity — це ігровий рушій, розроблений і підтримуваний компанією Unity Technologies. Він використовується для створення ігор, мобільних додатків, веб-додатків, створення анімацій та інших інтерактивних додатків.

Unity має широкий набір інструментів для розробки ігор. Наприклад графічний рушій, котрий чудово працює як з 2D графікою, так і з 3D графікою, фізичний рушій, систему штучного інтелекту, систему звуку, систему створення анімацій та багато інших інструментів, котрі мають низький поріг входження та об’ємну документацію, що дозволяє розробникам швидко приступити до створення різноманітних ігор з різними механіками та

захоплюючим геймплеєм. А завдяки тому, що Unity вміє працювати як з двовимірною графікою, так і з тривимірною графікою, то мало що зможе зупинити розробника на шляху створення стилізованих або реалістичних ігор.

В ігровому рушії Unity використовується мова програмування C#, а також Visual Scripting. Мова Visual Scripting дозволяє розробникам, котрі мало знайомі з C# створювати гру без написання коду.

При розробці ігор на Unity розробник взаємодіє з графічним інтерфейсом ігрового рушія, котрий дозволяє керувати об’єктами на сцені, створювати компоненти, редагувати об’єкти і так далі. На Рисунку 10 зображено графічний інтерфейс ігрового рушія Unity.

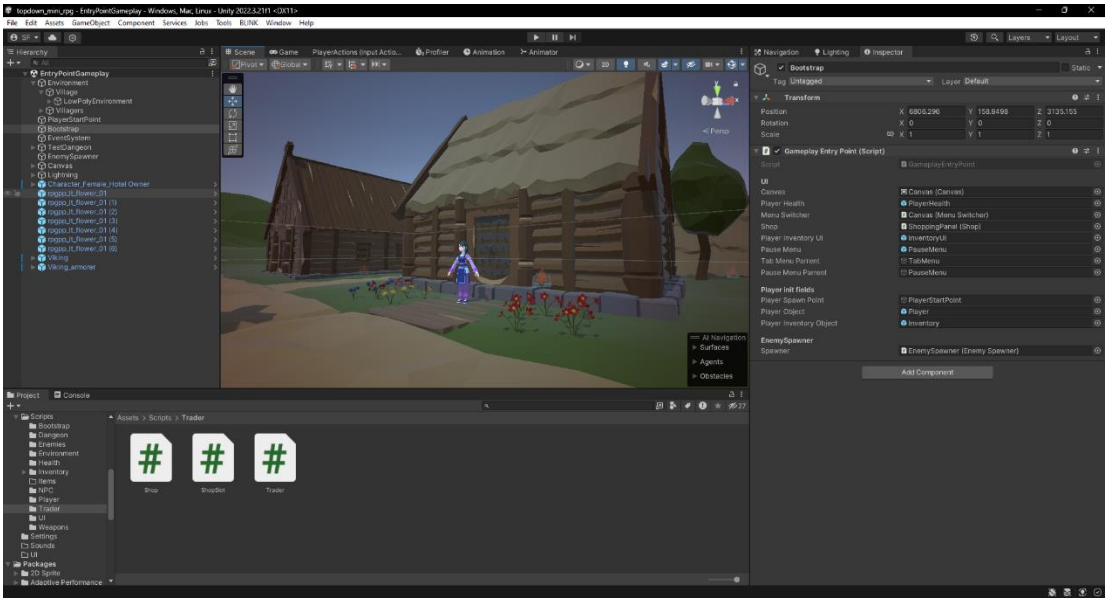

Рисунок *10 — Графічний інтерфейс ігрового рушія Unity*

Даний ігровий рушій має багату систему для знаходження проблем гри, а саме Profiler, котрий дозволяє почати захоплення ігрових кадрів та проаналізувати їх, щоб знайти можливі місця для оптимізації гри. Наприклад виконання деякого скрипту може займати багато часу і це можна побачити в профайлері, або деяке джерело світла може мати незвичайні параметри та потребувати великого часу обчислення. Загалом це дуже гнучкий інструмент, котрий вміє працювати також з Android та iOS пристроями, для котрих можна

збирати збірки для розробки та вмикати автоматичне під'єднання до профайлеру, котре дозволить знайти причини лагів в додатку.

Також говорячи про інструменти відлагоджування необхідно згадати про вбудовану можливість використання LogCat для андроїд пристроїв. LogCat використовується для отримання доступу до логів додатку. Даний інструмент допоможе в знаходженні помилок при виконанні додатку на цільовому пристрої.

Також Unity має вбудовану систему редагування шейдерів Shader Graph, котра дозволяє створювати чудові ігрові ефекти не використовуючи стару мову шейдерів Shaderlab, а використовувати редактор у вигляді графу. На Рисунку 11 зображений інструмент Shader Graph.

Продовжуючи розповідь про широкі вбудовані можливості необхідно також згадати про наявність в останній версії даного ігрового рушія вбудовано асистента зі штучним інтелектом – Muse. Даний вбудований штучний інтелект має широкий спектр можливостей, котрий може допомогти розробникам пришвидшити створення прототипу гри різноманітними способами. Наприклад Muse AI може генерувати спрайти та текстури для використання в грі, додання нових анімацій для людиноподібних персонажів, генерування заготовок коду за допомогою чату з ШІ та навіть можливість налаштування персонажів за допомогою Muse Behavior. Користувач має можливість описати потрібну поведінку і Unity Muse створить дерева поведінки в ігровому рушії Unity[12]. Також при генерації спрайтів розробник має можливість редагувати їх з допомогою генеративного заповнення, коли розробник виділяє зону на спрайті та описує, що в ній повинно з'явитися.

Muse AI не єдиний інструмент зі штучним інтелектом, котрий можна використовувати в Unity. Unity також має доступ до багатьох інших інструментів, котрі створені його спільнотою. Одні з найпопулярніших це:

- Convai – дозволяє швидко зробити ігрових персонажів живими надавши їм можливість відповідати на запитання гравця, відтворюючи анімацію лиця дуже близьку до природної. Також з

даним інструментом створені ігрові персонажі можуть відтворювати інші анімації тіла

- Inworld – має схожий функціонал як у попереднього як написано на сайті [13] — даний інструмент використовується щоб «надавати гравцям новаторські ігрові механіки, динамічних неігрових персонажів та світи які розвиваються з кожною подією».
- Layer – використовується для генерування різноманітних спрайтів в різних стилях, таких як: реалізм, піксель-арт, мультяшний, комічний та в багатьох інших. Даний інструмент, як говориться на головній сторінці даного інструменту [14] є завіреним компанією Unity.
- Polyhive – інструмент, який використовується для високоякісного текстурування об’ємних моделей за допомогою штучного інтелекту [15].
- LMNT – використовується для генерування реалістичної людської мови з можливістю додавати емоції.

Існує ще багато інструментів, які часто використовуються з ігровим рушієм Unity і якщо повернутись до інструментів, які розробляються та тестуються самою Unity, то необхідно згадати про Unity Sentis [16]. Даний інструмент, як говориться на його головній сторінці [16] – «Сентіс швидким темпом вносить використання інноваційних моделей штучного інтелекту в розробку ігор. Sentis надає можливість імпортувати свої моделі штучного інтелекту та використовувати їх прямо в Unity». Також необхідно сказати про те, що Сентіс запускається та використовується на комп’ютерах клієнтів, а не в хмарі, «усуває складну хмарну інфраструктуру, затримки мережі і періодичні витрати на підтримку системи» [16]. Можна привести декілька прикладів використання Sentis AI в своїх іграх, а саме використання моделей штучного інтелекту для отримання поточного знаку користувача, який він показує в камері або можна використовувати дані моделі для перетворення природної мови в текст, створення інтерактивних ігрових персонажів, читати рукописи,

ідентифікувати об’єкти через камеру пристроя клієнта або через ігрову камеру або оцінки глибини світу при використанні з AR, щоб правильно розміщати та відображати віртуальні об’єкти. Всі ці приклади описані тут [16].

Якщо ще більше зануритися в те, які можливості представляє ігровий рушій Unity, то необхідно згадати про платформу від розробників Unity Learn, котра допоможе починаючим розробникам швидше поринути в світ розробки своїх ігор. Дана платформа має можливість користувачам оглянути майже всі інструменти в юніті читаючи короткий опис з прикладами, а також маючи можливість продивитись відео з використанням даних інструментів для глибшого розуміння їх можливостей та призначення.

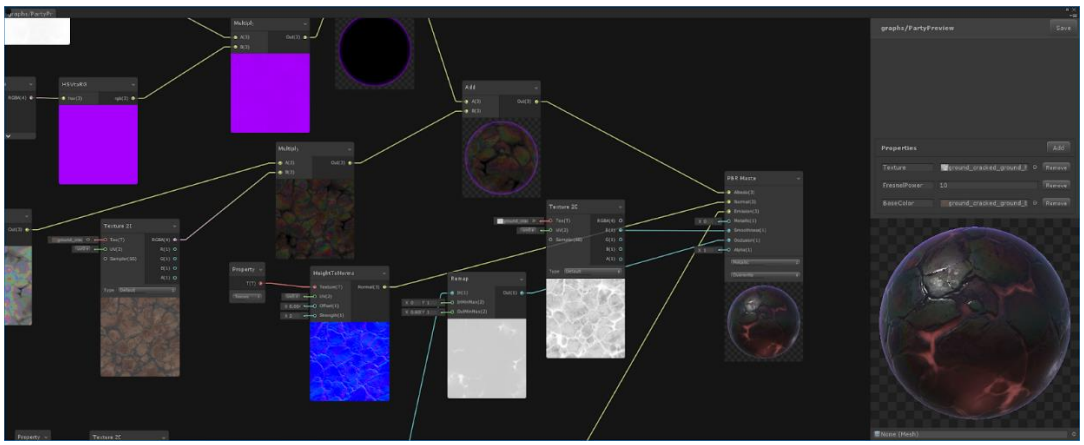

Рис. 11 Інтерфейс Shader Graph

Ігровий рушій Unity добре себе зарекомендував у створенні різноманітних шедеврів ігрової індустрії, таких як Cities: Skylines, Beat Saber, Rust, Genshin Impact, Phasmophobia, Escape from Tarkov та сотні інших. Ігровий рушій має одну з найбільших спільнот серед ігрових рушіїв, що породжує велику кількість інформації, відповідей на запитання, різноманітних плагінів для ігрового рушія, графічних та звукових ресурсів.

Unity має вбудовану систему мультиплатформенності, що дозволяє зібрати білд своєї гри майже під будь-яку популярну платформу, а саме ПК, Android, PS, Xbox, WebGL, Nintendo Switch, TV OS, VisionOS, iOS та лінукс. Це дає розробникам створювати різноманітні ігри під ті платформи на яких їм

буде найзручніше. Також ігровий рушій Unity є найкращим вибором, якщо говорити про мобільні ігри та ігри для не потужних комп'ютерів, але водночас дозволяє створювати ігри з реалістичною графікою.

Те, що Unity став найпопулярнішим ігровим рушієм на мобільних пристроях породило те, що юніті має вбудовану систему аналітики, реклами для мобільних пристроїв. Також даний ігровий рушій має різноманітні дебагери, котрі дозволяють на ранніх стадіях розробки виявляти та виправляти помилки.

Unity має вбудовані функції для роботи з тривимірною та двовимірною графікою, звуком, шейдерами, матеріалами, штучним інтелектом, анімацією та багатьма іншими аспектами гри.

Описуваний ігровий рушій має підтримку розробки AR та VR ігор, що робить Unity ще більш універсальним інструментом для розробки фантастичних ігор. Прикладом інструменту для реалізації ігор з використанням AR є поєднання ARCore разом з Geospatial Creator, котрий надає можливість використовувати реальний світ як ігровий використовуючи дані про геолокацію користувача та про направлення компасу його пристрою. Прикладом використання даного поєднання є уже стара гра Pokémon Go в якій гравцю необхідно подорожувати реальним світом, щоб спробувати себе в колекціонуванні покемонів.

Сьогодні Unity використовується широким колом користувачів по всьому світу, починаючи з людей, для яких створення ігор на даному ігровому рушії просто хобі або вони займаються цим для навчання, завершуючи користувачами з комерційних організацій, котрі варіюються від так званих студій в "гаражі" або інді розробників до великих інтернаціональних студій, котрі використовують Unity для створення величезної кількості ігор.

Також на сьогоднішній день в Unity підтримуються три рендер пайплайни [17], а саме Built-In Render Pipeline, Universal Render Pipeline та High Definition Render Pipeline. Пройдемося по основним особливостям кожного з них:

1. Built-In Render Pipeline — це найперший рендер пайплайн, котрий зазвичай використовується для слабких пристроїв, в якому немає можливості використовувати найсучасніші можливості юніті.
2. Universal Render Pipeline — це другий рендер пайплайн в юніті. Даний рендер пайплайн є золотою серединою між Built-In Render Pipeline та High Definition Render Pipeline, котрий дозволяє використовувати більшість нових інструментів, нові графічні елементи і так далі, але разом із цим дозволяє створювати оптимізовані ігрові світи для для великої різноманітності платформ, починаючи з мобільних пристроїв, Wii, Nintendo Switch аж до потужних консолей та ПК.
3. High Definition Render Pipeline — це найпотужніший рендер пайплайн, котрий підтримує всі найновіші технології в юніті. Наприклад освітлення за допомогою трасування променів, відображення за домопогою того ж трасування променів, разноманітні Post-ефекти, котрі в поєднанні з вбудованими кінематографічними ефектами, трасуванням шляху та іншими неназваними технологіями можуть дати досить реалістичне зображення в вашій грі. Також даний рендер пайплайн разом з універсальним рендер пайплайном використовується в архітектурних візуалізаціях, симуляціях, фільмах та на живих виступах.

Включаючи все описане раніше Unity є потужним інструментом для розробки ігор з великою кількістю вбудованих інструментів, що дозволяє розробникам не витрачати купу часу на створення своїх інструментів, а зосередитися на створенні якісних, різноманітних ігор для багатьох платформ.

2.3 Visual Studio

Visual Studio — це середовище програмування, підтримуване компанією Microsoft, котре дозволяє писати програми на різних мовах програмування та має багато влаштованих інструментів, котрі пришвидшують написання коду, можуть підвищити його якість та багато інших речей. З основних влаштованих

інструментів це автозавершення коду, виконання збірки проєкту, інструменти для відлагодження, тестування та інші. Visual Studio добре себе зарекомендувала в відлагодженні ігрових механік в Unity.

Також дане середовище програмування дозволяє пришвидшити створення графічних інтерфейсів на різних технологіях, таких, як MAUI, WPF, Xamarin та інших.

2.4 Висновки до розділу 2

Оглянуті основні методи та інструменти, які будуть використані при створенні відеогри, такі як Unity, її можливості, інструменти, а також Visual Studio.

3 ОПИС СТВОРЕННЯ ВІДЕОГРИ ЖАНРУ ФЕНТЕЗІ-РПГ З ВИКОРИСТАННЯМ ІГРОВОГО РУШІЯ UNITY

3.1 Опис предметної області роботи та її мета

Предметна область цієї роботи включає в себе розробку відеогри з використанням ігрового рушія Unity, а саме створення ігрових механік, створення ігрового світу з поєднанням створених механік. Дана предметна область використовує навички користування ігровим рушієм Unity та навички програмування з використанням мови програмування C#.

При розробці відеогри на ігровому рушії Unity дуже важливо розбити систему на підсистеми, щоб в подальшому було зручно підтримувати та розширювати систему без внесення великих змін в системі.

Дипломна робота зосереджується на створенні ігрових механік притаманних жанру фентезі-рпг з використанням ігрового рушія Unity. Використання ігрового рушія Unity оправдане відносно низьким порогом входження до створення ігор з якісною та оптимізованою графікою, котра зможе добре працювати навіть на слабких пристроях з використанням джерел світла, тіней, текстур, шейдерів, скайбоксів. Також даний ігровий рушій підтримує додання ефекту «об’ємний туман» котрий привносить «глибину» в ігровий світ, що сприяє кращому зануренню гравця в ігровий процес.

Також в даній роботі розглянуться основні архітектурні рішення при розробці відеоігор на основі ігрового рушія Unity.

Виконання даної роботи сприяє покращенню навичок розробки відеоігор загалом та підвищенню навичок при розробці відеоігор з використанням ігрового рушія Unity.

Мета кваліфікаційної роботи — це створення прототипу відео-гри жанру фентезі-рпг з використанням ігрового рушія Unity котрий в майбутньому можна буде розширяти та доповнювати новим контентом. Наприклад новою зброєю, предметами, ворогами і так далі.

3.2 Завдання кваліфікаційної роботи

Постановка завдання для виконання даної кваліфікаційної роботи включає в себе такі пункти:

1. Аналіз потреб і вимог: Визначення основних потреб і вимог, котрі повинна задовольняти створена відео-гра. Це включає визначення основних механік, котрі необхідно реалізувати, для того, щоб в поєднанні визначених механік вийшла гра в жанрі фентезі-рпг, визначення апаратних вимог.
2. Створення архітектури, котра зможе поєднати потрібні механіки. Створена архітектура повинна бути гнучкою та легко підтримуватись, щоб в подальшому не виникало проблем при доданні нових механік та доданні нового контенту.
3. Створення ігрових механік. Реалізація основних ігрових механік визначених в жанрі фентезі-рпг. А саме можливість переміщення головного героя, можливість атакувати ворогів з різноманітною зброєю, котра зберігається в інвентарі, а також наявність фантастичних ворогів, котрі зможуть давати гравцю опір при виконанні завдань. Гравець повинен отримувати нагороду за знищення ворогів та/або за успішне завершення квестів. Також необхідно реалізувати можливість придбати різні предмети у торговця.
4. Створення демонстраційного рівня в якому використовуються створені механіки, де можна побачити використання кожної з них, а саме можливість переміщення гравця, бойова система, інвентар гравця, зміна поточної зброї гравця з використанням інвентаря, ігровий магазин в якому гравець може придбати предмети і тестовий данж, де гравець може поборотись з ворогами та продемонструвати можливість вбивства ворогів та можливість своєї смерті.

3.3 Опис систем котрі необхідно реалізувати

3.3.1 Система ініціалізації об’єктів

Необхідно реалізувати систему ініціалізації об’єктів, котра дозволить при запуску ігрового рівня правильно ініціалізувати ігрові об’єкти з необхідними залежностями. Наприклад ініціалізація інтерфейсу відображення здоров’я гравця з залежністю від об’єкта здоров’я гравця тоді коли здоров’я гравця вже пройшло ініціалізацію.

3.3.2 Система переміщення гравця

Гра повинна надавати можливість гравцю переміщуватись по ігровому світу з використанням клавіатури. Дана система повинна забезпечувати такі можливості, як:

- Восьми стороннє переміщення гравця.
- Можливість переміщатись в режимі бігу.
- Можливість робити ривок в сторону погляду гравця, що дозволить гравцю наприклад уникнути отримання шкоди від ворогів або просто переміщатись швидше по ігровому світу.

Створення даної механіки надає гравцю можливість досліджувати світ та можливість втекти або добратись до ворогів коли це потрібно і просто мати більш «живого» персонажа, яким можна цікаво переміщатися.

3.3.3 Система ігрового інвентаря головного героя

Інвентар гравця в грі жанру фентезі-рпг має далеко не останню роль, так як за допомогою інвентаря гравець може збирати предмети в ігровому світі.

Дана система дає можливість гравцю носити предмети та використовувати їх при потребі. Інвентар повинен мати такі обов’язкові механіки, а саме:

- Можливість додати та видалити предмет з інвентаря.
- Інтерфейс інвентаря, який відображає предмети в ньому.

- Можливість переміщення предметів між слотами в UI за допомогою мишки.
- Деякі предмети повинні мати можливість знаходитись в деякій кількості в одному слоті.

Можливість мати активний слот, котрий відповідає за те який предмет може бути використаний або яка зброя повинна бути екіпірована.

3.3.4 Система зміни зброї

Дана система щільно співпрацює з можливістю зміни поточного предмету з активного слоту. Система зміни зброї повинна виконувати такі функції:

- Відображення поточної зброї в руках. Коли в руках гравця знаходиться предмет, котрий не являється зброєю, тол відображати пусті руки.
- При зміні зброї повинні виконуватись саме атаки тою зброєю, яка обрана гравцем.

3.3.5 Різновиди зброї

В грі повинні бути різні види зброї, такі, як:

- Кулаки, котрі використовуються, коли в активному слоті інвентаря знаходиться предмет відмінний від зброї.
- Меч.
- Лук.
- Магічна палиця.

Система різновидів зброї повинна щільно співпрацювати з інвентарем та системою зміни зброї забезпечуючи гравця можливістю використовувати їх окремо. Кожна зброя повинна мати свої значення параметрів, а саме:

- Шкода від базових та вторинних атак.
- Час перезарядки між базовими та вторинними атаками.

- Радіус базових та вторинних атак.
- Таймінги на внесення шкоди, котрі налаштовуються відповідно до анімацій атаки гравця, щоб точніше відображати момент внесення шкоди по противникам.

Також повинна бути можливість кожному виду зброї реалізовувати способи атаки по своєму, щоб можна було створювати зброю з різними можливостями. Це може буде корисно для створення різноманітних атак. Наприклад при атаці магічною палицею можна створювати різноманітні атаки, такі як атака по зоні або все інше, що може впасти на думку.

Кожен вид зброї повинен мати свій унікальний зовнішній вигляд та мати анімацію атаки гравцем зброєю. При атаці поточною зброєю необхідно активувати анімацію поточного виду зброї.

3.3.6 Система ігрового магазину

Дана система забезпечить гравця можливістю отримувати ігрові предмети до інвентаря для подальшого їх використання.

Система ігрового магазину повинна бути прикріпленою до ігрового персонажу так, щоб гравець мав можливість придбати предмети лише біля торговця. В даній системі повинні реалізовуватись такі можливості:

- Відображення іконки предметів, їх назви та ціни.
- При відкритті магазину повинен відкриватися і інвентар гравця, щоб гравець наглядно бачив отримані речі в ньому.
- Ігровий магазин повинен співпрацювати з гаманцем гравця.

3.3.7 Гаманець гравця

Гаманець гравця використовується для купівлі зброї та інших предметів в ігровому магазині. Даний гаманець повинен мати такі функції:

1. Можливість додати кошти.
2. Можливість забрати кошти.

Гравець повинен отримувати нагороду за вбивство ворогів і зачищення данжів в виді підвищення грошей на балансі гаманця.

3.3.8 Система створення ворогів

Система створення ворогів повинна відповідати таким вимогам:

1. Можливість звернутися статично до системи створення ворогів, щоб створити ворогів.

2. Можливість створювати об’єкти ворогів на заданих координатах за ідентифікатором ворога.

3. Система створення ворогів повністю відповідає за ініціалізацію ворогів.

3.3.9 Система оновлення анімацій гравця

Щоб гравець виглядав живішим як і геймплей він повинен мати можливість змінювати свої анімації в залежності від виконуваних дій чи подій. Аніматор гравця повинен мати такі стани:

1. Стан покою, коли гравець просто стоїть.

2. Стан ходьби, коли гравець переміщається по ігровому світу зі звичайною швидкістю.

3. Стан бігу, коли гравець переміщається по ігровому світу бігом.

4. Стан «ривка», коли гравець робить «ривок».

5. Стан смерті, коли гравець помер.

6. Стан виконання базової атаки кулаком.

7. Стан виконання вторинної атаки кулаком.

8. Стан виконання базової атаки мечем.

9. Стан виконання вторинної атаки мечем.

10. Стан атаки луком.

11. Стан атаки магічною зброєю.

12. Стан отримання шкоди.

Всі ці стани дозволять головному герою виглядати живішим на сцені за наявності різноманітності атак, стану смерті, покою, переміщення та ривка. Їх наявність також дозволить гравцю краще розуміти, що зараз відбувається на сцені.

3.3.10 Система ворогів

Вороги в даній грі мають значущу роль, так як без них можливість гравця атакувати була б безкорисною. Поява ворогів повністю залежить від системи створення ворогів, як і їхня ініціалізація. Вороги повинні мати такі можливості:

1. Переміщатись по ігровому світу прокладаючи маршрут до цілі.
2. Переслідувати головного героя, якщо він підійшов достатньо близько.
3. Атакувати гравця, якщо головний герой став достатньо близько.
4. Різноманітні атаки, а саме базова та вторинна.

У кожного різновиду ворога можуть бути власні параметри, такі як:

- Швидкість переміщення ворога.
- Радіус «зору» ворога, в якому він починає переслідувати гравця.
- Радіус атаки ворога, в якому він починає атакувати гравця.
- Шкода від базових атак.
- Шкода від вторинних атак.
- Кількість здоров’я.
- Перериви між базовими та вторинними атаками.
- Часові проміжки базової та вторинної атаки, щоб гравець розумів, коли може отримати шкоду від ворога.
- Нагорода за вбивство ворога, котра використовується для додання нагороди гравцю.

3.3.11 Взаємодія головного героя з ігровим світом

Головний герой повинен мати можливість взаємодіяти з ігровим світом, щоб мати можливість користуватись інтерактивними об’єктами гри. Загалом дана система повинна виконувати одну функцію, а саме взаємодію з навколишнім світом. Дана система повинна використовуватися наприклад для активації данжу та відкриття магазину.

3.3.12 Система зміни стану UI та керування головним героєм

Необхідно релізувати систему, котра буде керувати вводом керування та ввімкненням елементів інтерфейсу, такими, як інвентар, меню паузи та екран смерті. Дана система необхідна для контролю над станом керування головним героєм в залежності від стану UI та від інших факторів. Дана система повинна мати такі функції:

- Відкриття та закриття меню інвентаря.
- Відкриття та закриття меню паузи.
- Увімкнення та вимкнення керування гравцем.
- Керування ввімкненням екрану смерті гравця.

3.3.13 Система внесення шкоди об'єктам

Дана система впливає на об'єкти, які можуть отримувати шкоду від гравця або від ворогів. Система внесення шкоди повинна реалізувати такі потреби:

- Можливість нанесення шкоди.
- Можливість лікування.
- Події смерті та зміни здоров'я.

3.4 Вимоги до апаратного забезпечення системи

Ігри на юніті для комфортної гри можуть потребувати комп’ютери різної потужності в залежності від налаштувань якості графіки гри, кількість

сутностей зі штучним інтелектом на сцені та від обраного для гри рендер пайплайну. В моєму випадку я обрав «золоту середину», щоб більша кількість користувачів могла пограти в створену гру.

Для ефективної роботи описуваної системи в якій в створену відеогру на ігровому рушії Unity можна буде не тільки запустити, а і зіграти з комфортом комп’ютер користувача повинен відповідати таким системним параметрам:

Рекомендовані характеристики апаратного забезпечення для комфортної гри в роздільній здатності монітору 1920x1080 пікселів:

1. Процесор: Intel Core i5 3450 3.1Ghz.
2. Графічний процесор: Nvidia GT 1030 2GB.
3. RAM: 8GB 2400Mhz.
4. Жорсткий диск або SSD: 1GB.
5. Звукова карта Realtek HD Audio.

3.5 Вимоги до програмного забезпечення системи

Основні вимоги до програмного забезпечення включають наступні пункти:

1. Операційна система Windows 10 x64 і вище.
2. Для графічних процесорів Nvidia драйвер версії 500.0 і вище.

3.6 Вимоги до користувачів відеогри

Головною метою цього аналізу було з’ясування потреб та очікувань цільової аудиторі, а також визначення вимог, за яких гравці могли б комфортно грати в розроблену відеогру.

З’ясувалося, що головними користувачами даної гри будуть гравці котрі ставлять в пріоритеті зручне керування в грі, приємну графічну складову та хороший геймплей. Також дана гра повинна давати деяку частину випробувань для гравців з різним ігровим досвідом.

Було визначено таку основну роль користувачів розробленої відеогри:

- Гравець.
  - Функції: Проходження гри, взаємодія з елементами керування в грі, з клавіатурою та мишкою, налаштування параметрів гри.
  - Можливості системи: Запуск відеогри, налаштування параметрів гучності, якості графіки, керування грою.
  - Вимоги: Мінімальні або відсутність ігрових навичок на комп’ютері, розуміння взаємодії з типічними ігровими пристроями, знання типічних основ геймплею.
- Тестувальник.
  - Функції: Проходження гри, знаходження багів при типічній поведінці гравця, пошук багів при нетипічній поведінці гравця, передача знайдених багів та помилок до розробників.
  - Можливості системи: Запуск відеогри, керування грою, налаштуваннями гри.
  - Вимоги: Середні ігрові навички, креативне та критичне мислення.
- Розробник.
  - Функції: Створення гри, додання контенту, виправлення багів, додання нових механік, квестів та все інше, що можна рахувати за підтримку та оновлення гри.
  - Можливості системи: Повний контроль над проектом відеогри, її контентом, механіками, архітектурою.
  - Вимоги: Впевнене користування ігровим рушієм Unity, середні навички мови програмування C#, розуміння принципів роботи з компонентами в Unity.

3.7 Реалізація відеогри на ігровому рушії Unity

3.7.1 Система ініціалізація ігрових об'єктів

При розробці відеоігор з великою кількістю підсистем, котрі так чи інакше пов’язані одне-з-одним може виникнути проблема початкової ініціалізації ігрових об’єктів. Для прикладу інтерфейс відображення поточного здоров’я гравця залежить від об’єкта здоров’я гравця, щоб мати можливість при зміні значення здоров’я гравця оновити його на екрані або магазин ігрових предметі залежить від інвентаря гравця, щоб мати можливість додавати предмети до нього.

Для повного управління порядком ініціалізації об’єктів популярним рішенням є створення об’єкта, котрий буде являтись ініціалізатором ігрових об’єктів. Зазвичай об’єкти котрі відповідають за таку можливість містять в своїй назві термін «EntryPoint» або «Bootstrap», так як таким чином вони відображають сутність цього класу, так як переклад даних термінів з англійської мови — «Точка входу».

Для ініціалізації головних аспектів геймплею необхідно створити вхідну точку, в якій буде можливість ініціалізувати об’єкти ігрової сцени з використанням необхідних залежностей.

Загалом в даному проекті був створений клас котрий відповідає за ініціалізацію таких компонентів:

- Додання UI елементів, а саме відображення поточного здоров’я гравця, меню інвентаря, екран смерті та меню паузи.
- Ініціалізація слотів інвентаря.
- Ініціалізація об’єкта гравця та додання його на ігрову сцену на його стартову позицію.
- Ініціалізація UI здоров’я гравця з відповідним екземпляром здоров’я гравця та ініціалізація контролера зміни ігрових меню.
- Ініціалізація спавнера ворогів.
- Увімкнення керування гравця.

- Ініціалізація ігрового магазину з інвентарем гравця та його гаманцем.

В реалізації даний клас має такі поля та методи як на Лістингу 1.

Лістинг *1 Клас «GameplayEntryPoint»*

```
public class GameplayEntryPoint : MonoBehaviour
{
private Canvas canvas
private GameObject playerHealth
private MenuSwitcher menuSwitcher
private Shop shop
private GameObject playerInventoryUI
private GameObject pauseMenu
private GameObject tabMenuParrent
private GameObject pauseMenuParrent
private GameObject playerSpawnPoint
private GameObject playerObject
private GameObject playerInventoryObject
private EnemySpawner spawner
private Inventory playerInventory
private Player player
private void Start()
private void InitializeShop()
private void InitializeUI()
private void InitializeInventory()
private void InitializePlayer()
private void InitializePlayerHealth()
private void EnablePlayerControls()
private void InitializeEnemySpawner()
}
```

Розглянемо поля більш детально:

- Canvas canvas – канвас на якому відображаються UI елементи.
- GameObject playerHealth – UI котрий відповідає за відображення здоров’я гравця.
- MenuSwitcher menuSwitcher – відповідає за уввімкнення та вимкнення різноманітних об’єктів.
- Shop shop – ігровий магазин, котрий потребує при ініціалізації інвентар гравця для подальшого використання при придбанні предметів.
- GameObject playerInventoryUI – інтерфейс інвентаря гравця.
- GameObject pauseMenu – інтерфейс меню паузи.
- GameObject tabMenuParrent – об’єкт до якого прикріпляється меню інвентаря гравця.
- GameObject pauseMenuParrent – об’єкт до якого прикріпляється меню паузи.
- GameObject playerSpawnPoint – точка створення гравця.
- GameObject playerObject – префаб об’єкта гравця.
- GameObject playerInventoryObject – префаб інвентаря гравця.
- EnemySpawner spawner – посилання на спавнер ворогів.
- Inventory playerInventory – посилання на інвентар гравця.
- Player player – посилання на об’єкт гравця.

Також розглянемо методи даного класу:

- void Start() – викликається автоматично при завантаженні гри перед першим кадром.
- void InitializeShop() – метод для ініціалізації магазину.
- private void InitializeUI() – метод для створення інтерфейсу для відображення здоров’я, меню паузи та інвентаря гравця.
- private void InitializeInventory() – метод для створення та ініціалізації інвентаря гравця.

- private void InitializePlayer() – метод для ініціалізації об’єкта гравця на відповідній позиції зі створеним інвентарем.
- private void InitializePlayerHealth() – метод для ініціалізації інтерфейса здоров’я.
- private void EnablePlayerControls() – метод для ввімкнення керування гравця.
- private void InitializeEnemySpawner() – метод для ініціалізації спавнера ворогів, котрий відповідає за процес створення та ініціалізації ворогів.

Виклики даних методів виконується в методі Start, котрий викликається на початку гри. На Лістингу 2 можна побачити в якому порядку викликаються описані методи.

Лістинг *2 Процес ініціалізації*

```
private void Start()
 {
     InitializeUI();
     InitializeInventory();
     InitializePlayer();
     InitializeEnemySpawner();
     InitializePlayerHealth();
     InitializeShop();
     EnablePlayerControls();
 }
```

3.7.2 Реалізація системи переміщення гравця

Для реалізації простої системи переміщення гравця в ігровому рушії Юніті зазвичай використовують вбудований клас CharacterController, котрий дозволяє з невеликими зусиллями реалізувати переміщення ігрового персонажу. Загалом даний клас має такі вбудовані функції:

- Налаштування висоти кроку персонажу, що дозволяє налаштовувати те на які виступи він зможе зайти без стрибка, а на які ні.
- Налаштування колайдера гравця, а саме його позицію відносно об’єкта на якому він закріплений, радіус та висоту.

Також для реалізації переміщення гравця необхідно зчитувати клавіші з клавіатури та миші. За таку можливість в Unity відповідає система InputSystem, котра дозволяє контролювати мапи подій, а також додавати нові події, котрі викликаються з відповідністю прив’язаних клавіш. На Рисунку 11 відображено інтерфейс інструмента InputSystem.

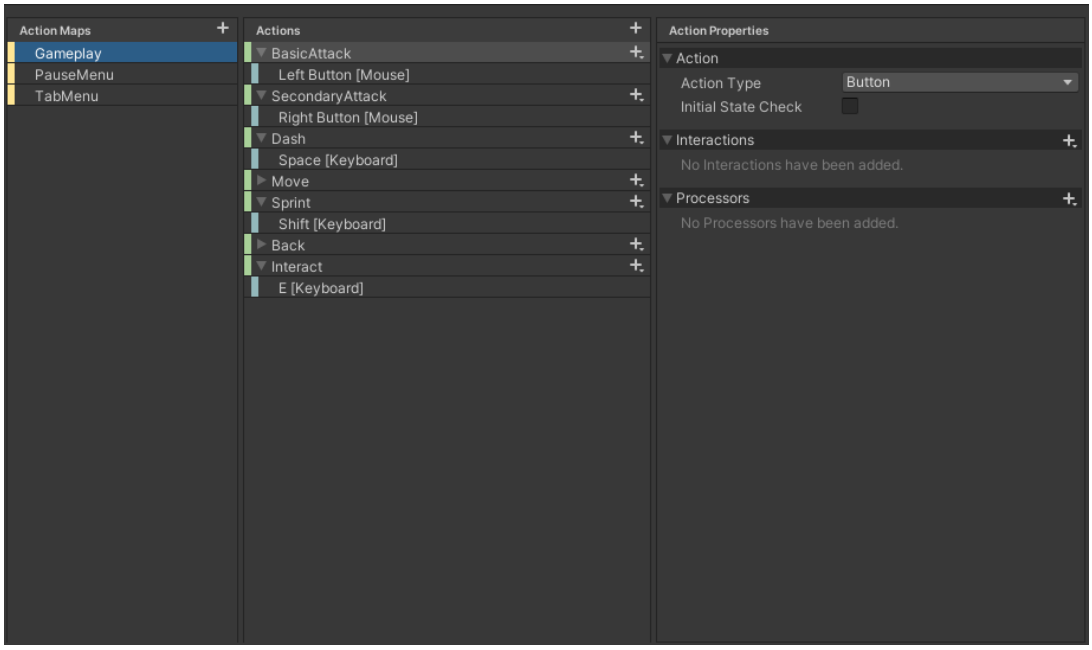

Рисунок *11 — Інтерфейс інструмента «InputSystem»*

З представленого рисунку видно, що даний інтерфейс дозволяє створювати мапи подій(зліва), котрі відповідають за те, які можливості для керування буде мати гравець в різних станах гри. Посередині відображені різноманітні прив'язки клавіш до подій, а справа можна налаштовувати тип події та інші корисні функції.

Опишемо те, як в даному проекті реалізовано переміщення гравця. Поперше для правильної роботи переміщення гравця необхідно підписатися на події, а також додати підписки та відписки в методи OnEnable() та OnDisable(), щоб при ввімкненні та вимкненні компонента переміщення гравця він відписувався та підписувався на події. Приклад підписки можна побачити на Лістингу 3.

Лістинг *3 Ініціалізація, підписка та відписка від подій «InputSystem»*

```
        public void InitializeInput()
        {
            playerActions                                    =
PlayerInputController.instance.playerActions;
            OnEnable();
            currentSpeed = moveSpeed;
        }
        private void OnEnable()
        {
            if (playerActions != null)
            {
                playerActions.Gameplay.Move.Enable();
                playerActions.Gameplay.Sprint.performed   +=
Sprint_performed;
                playerActions.Gameplay.Sprint.canceled    +=
Sprint_performed;
                playerActions.Gameplay.Move.performed     +=
UpdateMovementVector;
                playerActions.Gameplay.Move.canceled      +=
StopMovement;
            }
        }
```

З представленого лістингу видно, що в системі переміщення головного героя використовується мапа подій **Gameplay** і різноманітні події з ней, такі як **Move, Sprint,** а також вид події – performed та canceled, котрі викликаються відповідно тоді коли клавіша була натиснута та віджата. Підписані методи виконують читання поточних значень вводу. Для прикладу на Лістингу 4 представлено те, як читається контекст для визначення направлення переміщення гравця з події Move.

Лістинг *4 Читання значень з події «Move»*

```
    private void
UpdateMovementVector(InputAction.CallbackContext context)
        {
            inputVector = context.ReadValue<Vector2>();
        }
```

Потім прочитані дані з InputSystem використовуються для переміщення головного героя. Реалізацію переміщення можна побачити на Лістингу 5. З даного лістингу видно, що переміщення головного героя виконується в методі Update(), що викликається кожен ігровий кадр.

Лістинг *5 Переміщення головного героя*

```
    private void Update()
    {
        if (!enabled | playerActions == null |
playerActions == null | inputVector == null)
        {
            return;
        }
        //reseting player fall speed when is grounded
        if (controller.isGrounded && velocity.y < 0)
        {
```

```
            velocity.y = -2f;
        }
        Vector3 moveDirection = new(-inputVector.y, 0,
inputVector.x);
        controller.Move(currentSpeed * Time.deltaTime *
moveDirection);
        //Debug.Log($"ControllerMagnitude:
{controller.velocity.magnitude}");
        OnMove.Invoke(controller.velocity.magnitude);
        //applying gravity to player body
        velocity.y += gravity * Time.deltaTime;
        controller.Move(velocity * Time.deltaTime);
        if (moveDirection != Vector3.zero)
        {
            //rotating player body to player movement
direction
            Quaternion targetRotation =
Quaternion.LookRotation(moveDirection, Vector3.up);
            playerBody.transform.rotation =
Quaternion.Slerp(playerBody.transform.rotation,
targetRotation, Time.deltaTime * 10f);
        }
    }
```

Розглянемо метод Update() класу PlayerMovement докладніше. На початку перевіряються різноманітні поля на те, чи ініційовані вони. Потім створюється вектор на який переміститься CharacterController і викликається метод Move, в який передається швидкість(float), Time.deltaTime(час, який пройшов з моменту виклику попереднього методу Update() та сам вектор направлення переміщення. Після цього відбувається керування базовою гравітацією головного героя, а також поворот тіла головного героя в сторону переміщення.

Також клас PlayerMovement керує вмиканням бігу, в якому в залежності від того натиснута чи відпущена клавіша змінюється поточна швидкість відповідно на швидкість бігу та швидкість звичайного переміщення.

Необхідно згадати, що для реалізації системи переміщення головного героя також було створено ще один клас — PlayerDash, котрий відповідає за виконання головним героєм ривку в сторону його переміщення. Загалом підписка та відписка від InputSystem відбувається так-само, як і в попередньому описаному класі, але є сенс пояснити як відбувається сам «ривок». Реалізацію ривка можна побачити на Лістингу 6.

Лістинг *6 Реалізація «ривка» головного героя*

```
public void OnDash()
{
     if (Time.time < lastDashTime + dashCooldown)
         {
             return;
         }
         if (GetMoveDirection() == Vector3.zero)
         {
             return;
         }

         Debug.Log("Dash!");
         OnDodge.Invoke();
         lastDashTime = Time.time;
         StartCoroutine(Dash());
}

private IEnumerator Dash()
{
         float startTime = Time.time;
         while (Time.time < startTime + dashTime)
```

```
            {
                controller.Move(dashSpeed * Time.deltaTime
* GetMoveDirection());
                yield return null;
            }
    }
```

В наведеному лістингу видно, що спочатку перевіряється чи пройшло достатньо часу після останнього ривка, потім береться вектор направлення руху. Якщо він нульовий, то немає сенсу робити ривок. Далі стартує корутина, котра плавно робить ривок в сторону направлення головного героя з визначеною швидкістю впродовж часу виконання ривка, після чого корутина завершається.

В попередньому абзаці згадувався термін корутина, котрий буде згадуватись і в наступних системах, тому є сенс описати, що таке корутина.

Корутина — це деяка функція, котра повертає тип IEnumerator та дозволяє зручно виконувати дії, котрі виконуються в продовж деякого часу. Корутини дозволяють зручніше виконувати дії, котрі виконуються впродовж більше ніж один кадр для забезпечення плавності візуалізації чогось або для інших цілей, коли може знадобитись виконання коду в із затримкою.

3.7.3 Реалізація системи інвентаря головного героя

Інвентар являється однією з основних механік даної гри, тому він використовується в багатьох інших класах. На рисунку 12 можна побачити діаграму класів, в яких використовується інвентар.

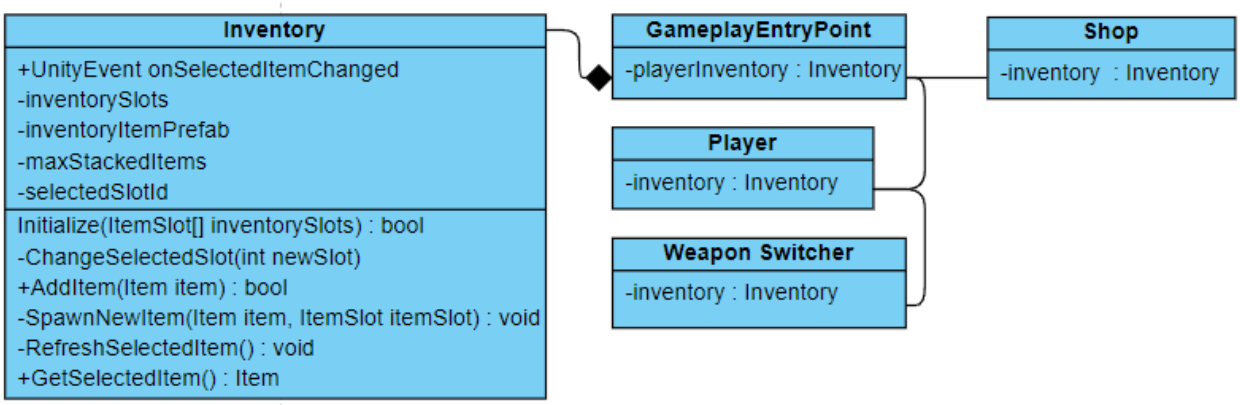

Рисунок *12 — Діаграма класів, котрі використовують інвентар*

Як бачимо, інвентар спочатку з'являється в класі «GameplayEntryPoint», а потім екземпляри інвентаря використовуються в інших класах, а саме в класі гравця, для ініціалізації класу «WeaponSwitcher» та в магазині для можливості додавання придбаних предметів.

Інвентар в іграх жанру фентезі-рпг є досить важливою складовою, так-як вона дозволяє гравцю використовувати різноманітні предмети, котрі він може тим чи іншим чином знайти в ігровому світі. Для прикладу це може бути зброя, їжа, бинти або якісь зілля. В Unity прийнято використовувати Scriptable Objects, котрі дозволяють зберігати налаштування предметів в файлах і потім використовувати їх в грі. Для створення таких файлів з налаштуваннями предметів створимо клас, який буде наслідуватись від ScriptableObject та матиме публічні поля, котрі і будуть налаштуваннями предметів інвентаря. Створений клас можна побачити на Лістингу 7.

Лістинг *7 Клас Item для створення предметів інвентаря*

```
    [CreateAssetMenu(fileName = "ItemData", menuName=
"Inventory/Item")]
    public class Item : ScriptableObject
    {
        public string Id;
```

```
    public string Title;
    public Sprite Icon;
    public ItemType type;
    public float Price;
    public bool stackable = true;
    public Weapon Weapon;
}
```

В даному класі бачимо, що спочатку використовується атрибут класу — [CreateAssetMenu(fileName = "ItemData", menuName= "Inventory/Item")], котрий використовується для додання даного типу ScriptableObject до контекстного меню в ігровому рушії юніті. Створене меню виглядає як на Рисунку 13. На ньому бачимо, що створилось меню як вказано в параметрі menuName, а саме Inventory і в ньому Item.

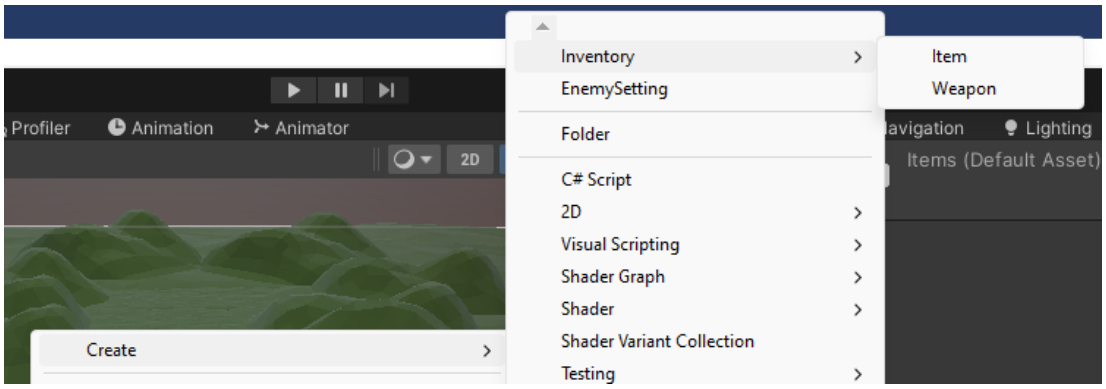

Рисунок *13 — Відображення використаного атрибуту «CreateAssetMenu»*

Далі клас Item наслідується від ScriptableObject, та йдуть поля даного класу. Як видно з Лістингу 7 даний клас має такі публічні поля:

- Ідентифікатор предмета, котрий буде використовуватись для додання предмета.
- Ім’я предмета.

- Спрайт icon, котрий відповідає за те, яким зображенням буде відображений даний предмет в інвентарі.
- Ціна предмета при купівлі
- Чи може даний предмет складуватись разом з таким-же предметом.
- Поле ItemType, котрий відповідає за те чи являється даний предмет бронею, предметом для звичайного використання, зброєю ближнього бою, магічною палицею чи луком.
- Поле Weapon, клас якого також являється Scriptable Object, котре використовується при зміні зброї, якщо тип даного предмету є одним з видів зброї, щоб знати якою зброєю являється даний предмет.

На лістингу 8 можна побачити перечислення ItemType, яке було описано вище.

Лістинг *8 Категорії предметів інвентаря*

```
public enum ItemType
{
    Item,
    Melee,
    MagicWand,
    Bow,
}
```

Далі необхідно створити модель інвентаря, котра буде мати такі поля, як на Лістингу 9, де бачимо такі поля:

- public UnityEvent onSelectedItemChanged — подія, яка «підіймається» тоді, коли змінюється обраний предмет в активному слоті.
- private ItemSlot[] inventorySlots — масив слотів інвентаря з інтерфейсу користувача, котрий опишеться нижче.

- private GameObject inventoryItemPrefab — префаб(заготовка) об’єкта предмета, котрий буде додаватись до вільного слоту.
- private int maxStackedItems — відповідає за максимальну кількість предетів одного виду в одному слоті.
- private int selectedSlotId — ідентифікатор обраного активного слоту інвентаря.

Лістинг *9 Поля класу «Inventory»*

```
public UnityEvent onSelectedItemChanged;
private ItemSlot[] inventorySlots;
private GameObject inventoryItemPrefab;
private int maxStackedItems = 16;
private int selectedSlotId = -1;
```

3.7.4 Реалізація різновидів зброї

Для реалізації різновидів зброї необхідно створити абстрактний клас, котрий уже буде мати абстрактні та звичайні поля методи котрі будуть актуальні для використання в для будь-якої зброї. Для даної реалізації був створений абстрактний клас **WeaponBase,** котрий буде являтись базовим для будь-якого виду зброї. На рисунку 14 можна побачити діаграму класів з WeaponBase.

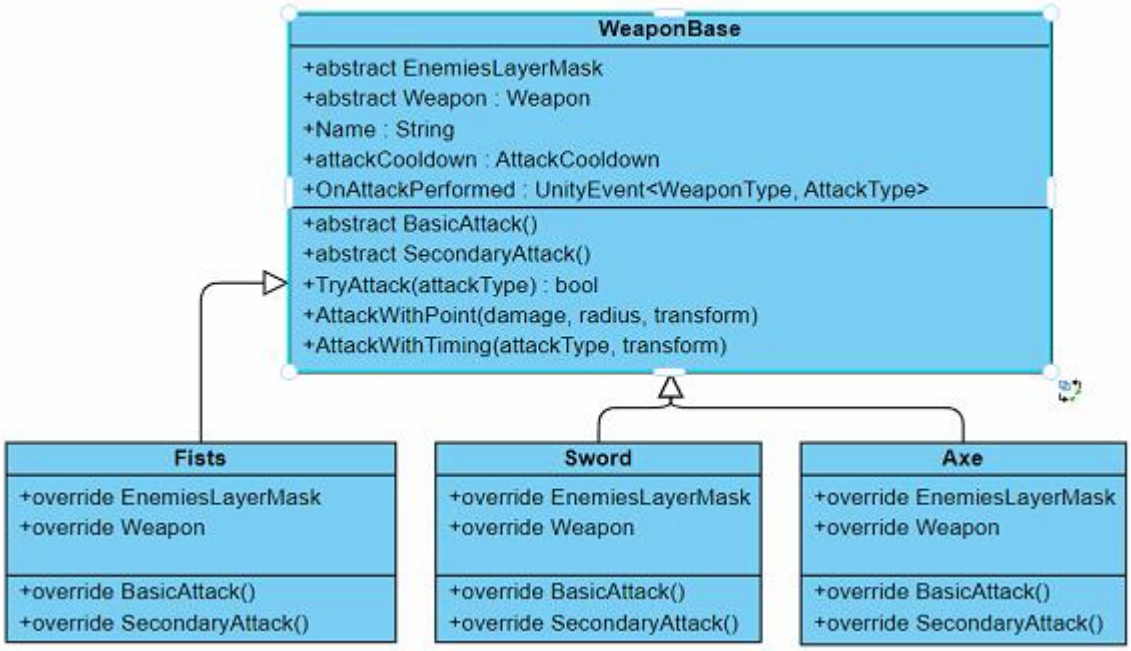

Рисунок *14 — Діаграма класів наслідників від «WeaponBase»*

З наданої діаграми класів бачимо, що абстрактний клас **WeaponBase**, є батьківським для класів зброї, таких як «Fists», «Sword» та «Axe».

Розглянемо поля класу **WeaponBase**, що відображені на Лістингу 10.

Лістинг *10 Поля абстрактного класу «WeaponBase»*

```
public abstract LayerMask EnemiesLayerMask { get; }
public abstract Weapon Weapon { get; }
public string Name() => Weapon.Name;
public readonly AttackCooldown attackCooldown = new AttackCooldown();
public UnityEvent<WeaponType, AttackType> OnAttackPerformed = new UnityEvent<WeaponType, AttackType>();
public abstract void BasicAttack();
public abstract void SecondaryAttack();
```

З наведеного лістингу бачимо такі поля:

- public abstract LayerMask EnemiesLayerMask { get; } — відповідає за «шар» ігрових об’єктів, котрі може атакувати дана зброя. Дану властивість має кожна сутність на сцені Unity.
- public abstract Weapon Weapon { get; } — Scriptable Object, котрий відповідає за налаштування параметрів зброї.
- public string Name() => Weapon.Name — гетер для швидкого отримання імені зброї.
- public readonly AttackCooldown attackCooldown — відповідає за відтворення переривів між атакамми.
- public UnityEvent<WeaponType, AttackType> OnAttackPerformed — подія, яка викликається при успішному виконанні атаки.

Також в лістингу бачимо такі абстрактні методи:

- public abstract void BasicAttack() — абстрактний метод для унікальної реалізації базової атаки для зброї.
- public abstract void SecondaryAttack() — абстрактний метод для унікальної реалізації вторинної атаки для зброї.

Також в класі «WeaponBase» реалізовані такі методи для запобігання дублювання коду, які описані на Лістингу 11.

Лістинг *11 Методи класу «WeaponBase»*

```
    public bool TryAttack(AttackType attackType);
    public void AttackWithPoint(int damage, float radius,
Transform attackPoint);
    public void AttackWithTiming(AttackType attackType,
Transform attackPoint);
    private IEnumerator
PointAttackWithTimingEnumerator(AttackType attackType,
Transform attackPoint);
```

Розглянемо кожен з представлених вище методів:

- public bool TryAttack(AttackType attackType) — метод, який повертає true або false на відповідь того чи була атака успішною чи ні, в залежності від того, що поверне метод **AttackAvailable()** класу **AttackCooldown,** а також при успішності атаки викличе метод **Attack(float cooldown)** та підніме подію **OnAttackPerformed.**
- public void AttackWithPoint(int damage, float radius, Transform attackPoint) — виконує атаку в деякій **точці** світу, наносячи деяку **шкоду** в конкретному **радіусі,** виконуючи атаку по об'єктам шару, визначеного в **EnemiesLayerMask.**
- public void AttackWithTiming(AttackType attackType, Transform attackPoint) — використовується для швидкого запуску корутини атаки з відкладеним нанесенням шкоди. Це може бути корисно для того, щоб момент нанесення шкоди по ворогам був більш явним для гравця.
- private IEnumerator PointAttackWithTimingEnumerator(AttackType attackType, Transform attackPoint) — корутина, котра запускається попереднім описаним методом **AttackWithTiming.**

Також необхідно розглянути поля класу Weapon, котрий унаслідується від Scriptable Object. Даний клас можна побачити на Лістингу 12.

Лістинг *12 Клас «Weapon»*

```
[CreateAssetMenu(fileName = "ItemData", menuName = "Inventory/Weapon")]
public class Weapon : ScriptableObject
{
    public WeaponType type;
    public string Name;
    public int basicAtackDamage;
    public int secondaryAtackDamage;
```

```
    public float basicAttackRadius;
    public float secondaryAttackRadius;
    public float basicAttackTiming;
    public float secondaryAttackTiming;
    public float basicAttackCooldown;
    public float secondaryAttackCooldown;
}
public enum WeaponType
{
    Melee,
    MagicWand,
    Bow,
    Fists
}
```

Розглянемо окремо кожне поле даного класу:

- public WeaponType type — один з перечислених типів зброї.
- public string Name — унікальне ім’я зброї.
- public int basicAtackDamage — шкода при виконанні базової атаки.
- public int secondaryAtackDamage — шкода при виконанні вториної атаки.
- public float basicAttackRadius — радіус базової атаки. Для прикладу це може бути атака по площі або по невеликій точці.
- public float secondaryAttackRadius — радіус вторинної атаки.
- public float basicAttackTiming — **час** нанесення шкоди при спробі нанести базову атаку з використанням методу **AttackWithTiming**.
- public float secondaryAttackTiming — **час** нанесення шкоди при спробі нанести вторинну атаку з використанням методу **AttackWithTiming**.
- public float basicAttackCooldown — **час** перериву після виконання базової атаки.

- public float secondaryAttackCooldown — **час** перериву після виконання вторинної атаки.

Також в представленому Лістингу 12 представлено перечислення типів зброї, де вказано такі типи:

- Melee — це тип для групування зброї ближнього бою. Наприклад мечі, дубини, сокири і так далі.
- MagicWand — магічні палиці.
- Bow — лук.
- Fists — кулаки головного героя.

Для прикладу неведене використання базового абстрактного класу в реалізації меча на Лістингу 13.

Лістинг *13 Клас «Sword»*

```
public class Sword : WeaponBase
{
    public override Weapon Weapon => weapon;
    public override LayerMask EnemiesLayerMask => 
attackLayerMask;
    [SerializeField]
    private Weapon weapon;
    [SerializeField]
    private LayerMask attackLayerMask;
    [SerializeField]
    private Transform attackPoint;
    public override void BasicAttack()
    {
        if (TryAttack(AttackType.Basic))
        {
            AttackWithTiming(AttackType.Basic, 
attackPoint);
```

```
            }
        }

        public override void SecondaryAttack()
        {
            if (TryAttack(AttackType.Secondary))
            {
                AttackWithTiming(AttackType.Secondary,
attackPoint);
            }
        }
    }
```

В наведеному Лістингу 13 бачимо, що для реалізації атак використовуються вже створені методи, а саме **TryAttack** та **AttackWithTiming** класу «WeaponBase». Також були реалізовані абстрактні поля **Weapon** та **EnemiesLayerMask.**

Створений клас **WeaponBase** дозволяє реалізувати різноманітні типи зброї, уникнувши великого дублювання коду.

Вважаю необхідним привести код класу **AttackCooldown,** котрий можна побачити на Лістингу 14.

Лістинг *14 Клас «AttackCooldown»*

```
public class AttackCooldown
{
    private float attackCooldown = 0;
    private float lastTimeAttack = 0;
    public bool AttackAvailable()
    {
        return Time.time > lastTimeAttack +
attackCooldown;
    }
```

```
    public void Attack(float attackCooldown)
    {
        this.attackCooldown = attackCooldown;
        lastTimeAttack = Time.time;
    }
}
```

Як бачимо в даному класі є два методи та два поля. Розглянемо їх:

- private float attackCooldown — це поле відображає час, котрий повинен пройти для того, щоб наступна атака була доступною.
- private float lastTimeAttack — це поле відображає час, коли відбулася остання атака.
- public bool AttackAvailable() — повертає булеве значення, чи пройшло достатньо часу з виконання останньої атаки.
- public void Attack(float attackCooldown) — виконує оновлення часу останньої атаки та час перериву до наступної атаки.

3.7.5 Реалізація системи зміни зброї

Для реалізації системи зміни зброї необхідно реалізувати клас WeaponSwitcher, котрий буде тісно співпрацювати з інвентарем. Поля даного класу можна побачити на Лістингу 15.

Лістинг *15 Поля класу «WeaponSwitcher»*

```
public UnityEvent<WeaponBase> onWeaponChanged;
private WeaponBase [] meleeWeapons;
private WeaponBase [] magicWands;
private WeaponBase [] bows;
private GameObject fists;
private Dictionary<string, GameObject> weapons;
private Inventory inventory;
private GameObject currentWeapon;
```

Оглянемо поля даного класу:

- UnityEvent<WeaponBase> onWeaponChanged — подія, яка викликається при зміні зброї.
- WeaponStruct[] meleeWeapons — масив зі зброєю ближнього бою.
- WeaponStruct[] magicWands — масив із магічною зброєю.
- WeaponStruct[] bows — масив з луками.
- GameObject fists — кулаки головного героя.
- Dictionary<string, GameObject> weapons — словник зі зброєю для можливості швидкого пошуку зброї за ідентифікатором.
- Inventory inventory — посилання на інвентар гравця
- GameObject currentWeapon — ігровий об’єкт поточної активованої зброї.

При ініціалізації зброя з масивів заноситься в словник, де ключом є поле **Id** Scriptable Object`a Weapon. На Лістингу 16 представлені методи класу «Weapon Switcher».

Лістинг *16 методи класу WeaponSwitcher*

```
public void Initialize(Inventory inventory)
 {
     this.inventory = inventory;

this.inventory.onSelectedItemChanged.AddListener(OnItemSwit
ched);
     weapons = new Dictionary<string, GameObject>();
     InitializeWeaponsDictionary(meleeWeapons);
     InitializeWeaponsDictionary(magicWands);
     InitializeWeaponsDictionary(bows);
     currentWeapon = fists;
     EnableFists();
 }
```

```
    private void InitializeWeaponsDictionary(WeaponBase[]
weapons);
    private void OnItemSwitched();
    private void EnableFists();
    private void EnableWeapon(string id, Dictionary<string,
GameObject> weapons);
```

З лістингу бачимо, як ініціалізується даний клас, а також такі методи:

- private void InitializeWeaponsDictionary(WeaponBase[] weapons) — додає до словника зі зброєю об’єкти з переданого масиву.
- private void OnItemSwitched() — викликається при зміні активного предмету інвентаря і змінює поточну зброю, якщо це необхідно.
- private void EnableFists() — вмикає кулаки головного героя.
- private void EnableWeapon(string id) — вмикає зброю за ідентифікатором, якщо така існує в словнику зі зброєю.

3.7.6 Гаманець гравця

Гаманець гравця використовується для придбання предметів в ігровому магазині. На Лістингу 17 можна побачити його реалізацію.

Лістинг *17 Класс «Wallet»*

```
public class Wallet
{
    private float money = 0;
    public void Initialize(float startMoney)
    {
        this.money = startMoney;
    }
    public void AddMoney(float money)
    {
        if (money < 0)
```

```
            {
                return;
            }
            this.money += money;
        }
        public bool TakeMoney(float money)
        {
            if (money < 0 | this.money - money < 0)
            {
                return false;
            }
            this.money -= money;
            return true;
        }
    }
```

З представленого лістингу бачимо, що в гаманці гравця реалізовані такі методи:

- public void Initialize(float startMoney) — метод для ініціалізації гаманця гравця з початковою кількістю грошей.
- public void AddMoney(float money) — метод для додання грошей до гаманця.
- public bool TakeMoney(float money) — метод для віднімання грошей з гаманця.

3.7.7 Система ігрового магазину

Система ігрового магазину потребує інтерфейсу користувача, в якому гравець може побачити які предмети є в продажі та придбати їх. Був створений такий інтерфейс гравця як на Рисунку 15.

Рисунок *15 Інтерфейс гравця*

Для реалізації купівлі предметів в магазині був створений клас Shop. Діаграму класів з класом Shop можна побачити на рисунку 16.

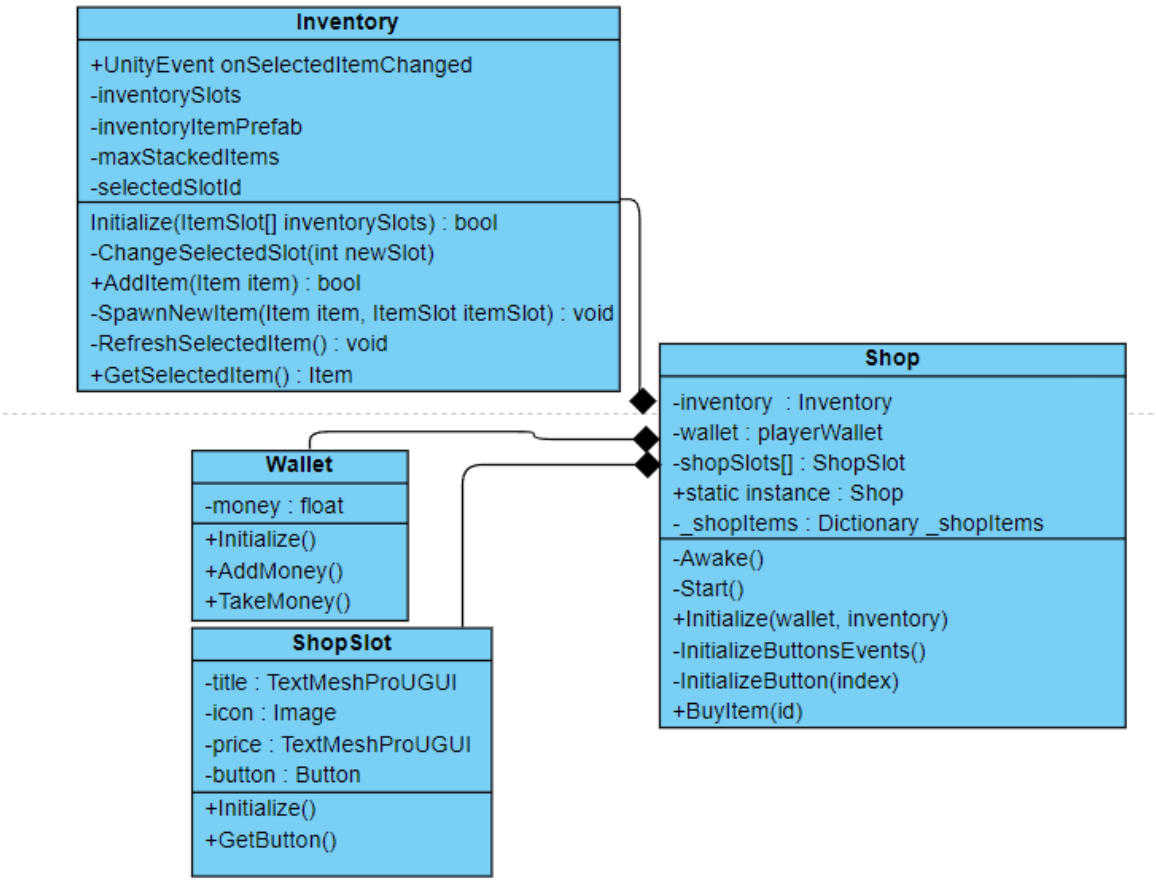

Рисунок *16 — Діаграма класів з класом «Shop»*

З діаграми класів видно, що магазин має в собі посилання на інвентар гравця, його гаманець, а також посилання на слоти магазину, кнопки котрих викликають метод **BuyItem,** завдяки підписки класу **Shop** своєї події для кожної з них.

Метод придбання предмету класу **Shop** зображений на Лістингу 18.

Лістинг *18 Клас «Shop»*

```
public void BuyItem(string id)
{
    Debug.Log($"Trying to buy item {id}");
    Item item = _shopItems[id];

    if(item == null )
    {
        return;
    }

    if(!playerWallet.TakeMoney(item.Price))
    {
        return;
    }

    inventory.AddItem(item);
}
```

Як видно з Лістингу 18 спочатку перевіряється чи є предмет з таким ідентифікатором в магазині, потім відбувається перевірка чи достатньо грошей в гаманці гравця і додається предмет до інвентаря гравця.

3.7.8 Система створення ворогів

Система створення ворогів повинна мати метод для створення ворогів на визначеній позиції за ідентифікатором ворога. Реалізований метод для створення ворогів в класі «EnemySpawner» показаний на Лістингу 19.

Лістинг *19 Метод створення противника на переданій позиції*

```
        public GameObject SpawnEnemy(string id, Transform
position)
        {
            GameObject enemyToSpawn =
enemiesDictionary[id];
            if (enemyToSpawn == null || position == null)
            {
                return null;
            }

            GameObject newEnemy = Instantiate(enemyToSpawn,
position.position, Quaternion.identity);

            if (newEnemy.TryGetComponent<Enemy>(out var
enemy))
            {
                enemy.Initialize(target);
            }

            Debug.Log($"[EnemySpawner] Spawned enemy
{newEnemy.transform.name} at
{newEnemy.transform.position}");
            return newEnemy;
        }
```

З наведеного лістингу бачимо, що спочатку перевіряється, чи є противник з переданим ідентифікатором в спавнері, потім створюється противник на переданій позиції і виконується ініціалізація ворога.

3.7.9 Система оновлення анімацій гравця

Система оновлення анімацій гравця працює з системою переміщення та атаки гравця, а також системою здоров'я, котра активує анімації смерті та отримання шкоди.

Для реалізації даної системи був створений клас «AnimationUpdater», котрий має посилання на класи переміщення гравця, атакера та на здоров'я гравця. При початку гри даний клас додає слухачі до потрібних подій, а потім в методах, які підписав на події виконує ввімкнення потрібних анімацій. Поля та ініціалізацію даного класу можна побачити на Лістингу 20.

Лістинг *20 Поля та ініціалізація класу «AnimationUpdater»*

```
private PlayerMovement playerMovement;
private PlayerDash playerDash;
private CharacterController characterController;
private PlayerAttacker attacker;
private Animator animator;
private Health health;
private void Start()
{
     playerMovement.OnMove.AddListener(UpdateMoveSpeed)
;
       playerDash.OnDodge.AddListener(OnDodge);
       attacker.OnAttack.AddListener(OnAttackAnimation);
       health.onHealthChanged.AddListener(OnDamage);
       health.onDeath.AddListener(OnDeath);
}
```

На лістингу 20 можна побачити компоненти на які має посилання система оновлення анімацій гравця.

3.7.10 Система ворогів

Для зміни станів ворогів необхідно використовувати Finite State Machine(скінченний автомат). Діаграма станів ворогу зображена на Рисунку 17.

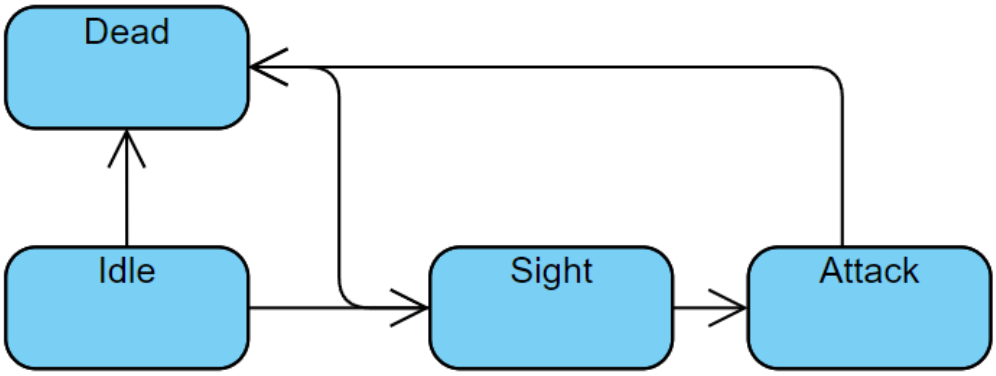

Рисунок *17 — Діаграма станів ворогів*

Як видно з діаграми станів видно, що ворог має такі стани:

- Idle — стан спокою, коли ворог не бачить поряд гравця і знаходиться на місці створення.
- Death — стан смерті ворога.
- Sight — стан переслідування цілі, коли гравець достатньо близько для переслідування, але далеко для атаки.
- Attack — стан атаки цілі, коли гравець достатньо близько для атаки

Даний автомат повинен використовувати стани, які представлені на Лістингу 21.

Лістинг *21 Стани ворогів*

```
public enum EnemyState
{
```

```
        Uninitialized,
        Idle,
        GoingToIdle,
        Chasing,
        Attacking,
        Died
    }
```

За переміщення ворога в ігровому світі з можливістю прокладання шляху до цілі з врахуванням перешкод в ігровому рушії юніті відповідає вбудований компонент AI navigation. Для використання даного компоненту спочатку необхідно додати до сцени об’єкт з комнонентом **NavMeshSurface,** налаштувати його та «запекти» поверхню, котра враховує перешкоди та використовується з компонентом **NavMeshAgent**, котрий використовують вороги. Налаштування компоненту **NavMeshAgent** зображені на Рисунку 15.

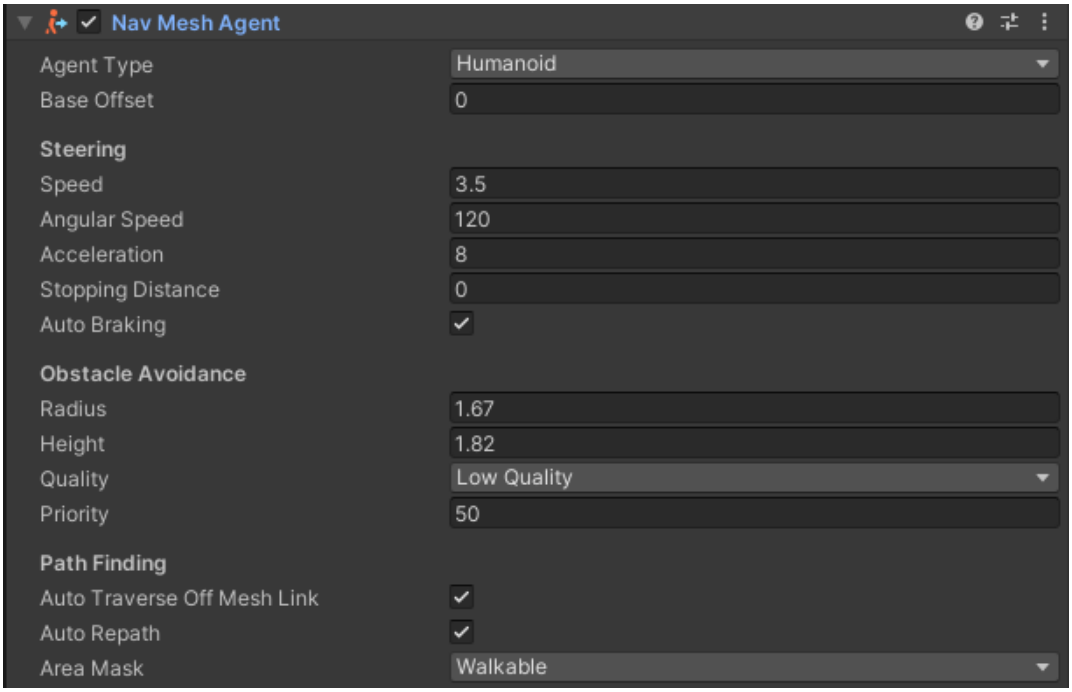

Рисунок *18 Налаштування компоненту «NavMeshAgent»*

Даний компонент використовує клас «EnemyPathFinder», котрий має такі методи:

- public void Initialize(Transform target) — ініціює клас для пошуку шляху з ціллю, котрою являється позиція гравця.
- public bool FindPathToTarget() — метод для знаходження шляху до цілі.
- public bool FindPathToPosition(Vector3 position) — метод для знаходження шляху до іншої позиції.

Скінченний автомат в класі «EnemyStateMachine» працює за таким принципом:

- Якщо ворог живий і гравець не знаходиться в радіусі переслідування, то він залишається на місці.
- Якщо гравець підійшов занадто близько, то ворог починає переслідувати його.
- Якщо гравець в радіусі атаки ворога, то ворог зупиняється та виконує атаку.
- Якщо гравець вийшов з радіусу переслідування, то ворог повертається на свою початкову позицію та стоїть там.

Для майбутнього створення різноманітних ворогів був створений абстрактний клас **EnemyBase**, котрий має реалізацію методів для атаки в точці, атаки з затримкою в точці, метод для виконання комбо атак, а також вимагає від класів, котрі його реалізовують реалізацію методів базової атаки та вторинної атаки.

На базі класу **EnemyBase** був створений клас ворога **Spider** котрий реалізує атаки через метод **AttackWithTiming**, котрий дозволяє атакувати ціль з затримкою.

3.7.11 Взаємодія головного героя з ігровим світом

Для взаємодії гравця з ігровим світом був створений інтерфейс **IInteractable,** котрий вимагає від класів, котрі наслідуються від нього реалізацію метода Interact(). Даний метод, для прикладу, реалізує торговець для відкритя магазину, що можна побачити в класі **Trader** на Лістингу 22.

Лістинг *22 Клас «Trader»*

```
using UnityEngine;

public class Trader : MonoBehaviour, IInteractable
{
    [SerializeField]
    private GameObject shopMenu;

    [SerializeField]
    private LayerMask playerLayermask;

    public void Interact()
    {
        shopMenu.SetActive(true);

PlayerInputController.instance.DisableGameplay();
    }

    private void FixedUpdate()
    {
        if (!shopMenu.activeSelf)
        {
            return;
        }

        if(!Physics.CheckSphere(transform.position,  3,
playerLayermask))
        {
            shopMenu.SetActive(false);

PlayerInputController.instance.EnableGameplay();
        }
    }
```

```
}
```

Як бачимо при взаємодії з торговцем відкривається інвентар гравця та саме меню торгівлі, а також, якщо гравець відійде від торговця достатньо далеко, то відкриті меню закриються.

3.7.12 Система зміни стану UI

Для зручного ввімкнення та вимкнення меню паузи, інвентаря, екрану смерті а також для наявності одного екземпляру **PlayerActions** був створений клас MenuSwitcher з необхідними методами на зміну стану керування та ввімкнених меню, а саме було реалізовано такі методи:

- public void InitializeDeathScreen(Health health) — ініціює екран смерті гравця посиланням на здоров’я гравця для використання події смерті гравця.
- public void InitializeInput() — ініціює поля даного класу.
- private void OnDeath() — вмикає екран смерті при смерті гравця.
- private void OnPause(InputAction.CallbackContext context) — вмикає меню паузи при натисненні відповідної клавіші.
- private void OnInventoryOpen(InputAction.CallbackContext context) — відкриває інвентар гравця при натисненні на відповідну клавішу
- public void SwitchMenu(GameObject menu) — змінює стан на протилежний від стану переданого меню.

3.7.13 Система внесення шкоди об'єктам

Для реалізації системи внесення шкоди об’єктам був створений клас **Health** з методами для отримання здоров'я, шкоди та подіями, котрі підіймаються коли сутність отримує шкоду або вмирає. Дані методи враховують передані їм значення котрі вони повинні додати чи відняти від здоров’я гравця враховуючи такі аспекти:

- При отриманні шкоди — шкода не може бути нульовою або від’ємною.
- Гравець не може отримати шкоду, якщо його здоров’я нижче або дорівнює нулю.
- При отриманні здоров’я — значення отримуваного здоров’я не може бути від’ємним і після отримання здоров’я, поточне здоров’я не може бути вище максимального.

3.8 Висновок до розділу 3 та пропозиції для покращення

В ході виконання даного розділу було визначено системи, котрі необхідно реалізувати, які реалізована гра має апаратні потреби та програмні потреби, а також було описано які системи були реалізовані та які компоненти були використані.

Для покращення результату даної роботи необхідно створити більш детальний світ, створити систему нанесення шкоди зброєю дальньої дії, додати різноманітних ворогів та створити систему квестів, щоб гра стала повноціннішою та могла вважатися більше ніж прототипом.

ВИСНОВКИ

1. За отриманими результатами дослідження під час виконання даної роботи можна сказати про те, що реалізація навіть невеликих ігор жанру фентезі-рпг є складною задачею через велику кількість базових механік, необхідних для даного жанру відеоігор.
2. Проведено аналіз існуючих аналогічних рішень, а саме ігор в жанрі фентезі-рпг, таких, як: TES: Skyrim, The Witcher 3: Wild Hunt та Elden Ring. Визначені основні особливості кожної з них, як гри жанру фентезі-рпг. Проведений аналіз літературних джерел на тему розробки відеоігор на ігровому рушії Unity.
3. Було визначено які основні системи необхідно реалізувати, визначені апаратні вимоги та вимоги до програмного забезпечення.
4. В результаті виконання кваліфікаційної роботи було створено прототип відеогри жанру фентезі-рпг з основними механіками. Відеогра була створення за допомогою Unity та мови програмування C#

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. C# Data Structures and Algorithms: Harness the Power of C# to Build a Diverse Range of Efficient Applications, лютий 2024 р. С. 374.
2. Learning C# by Developing Games with Unity: Get to Grips with Coding in C# and Build Simple 3D Games in Unity 2022 from the Ground Up, листопад 2022 р. С. 460.
3. Level Up!: The Guide to Great Video Game Design, квітень 2014 р. С. 535.
4. Rules of Play: Game Design Fundamentals (Mit Press), вересень 2003 р. С. 688.
5. The Elder Scrolls Wiki : веб-сайт. URL: https://elderscrolls.fandom.com/ (дата звернення: 07.05.2024).
6. The Witcher Wiki : веб-сайт. URL: https://witcher.fandom.com/uk/wiki/ (дата звернення: 07.05.2024).
7. Unity Learn : веб-сайт. URL: https://learn.unity.com/ (дата звернення: 07.05.2024).
8. Google Images : веб-сайт. URL: https://images.google.com/ (дата звернення: 07.05.2024).
9. GameShare: 30 найкращих фентезі ігор для ПК : веб-сайт. URL: https://gameshare.com.ua/63-30-luchshix-fentezi-igr-dlya-pk.html (дата звернення: 07.05.2024).
10. Unity Documentation. URL: https://docs.unity.com/ (дата звернення: 07.05.2024).
11. Figma: the collaborative interface design tool. : веб-сайт. URL: https://www.figma.com/ (дата звернення: 05.05.2024).
12. Unity Muse AI : веб-сайт. URL: https://unity.com/en/products/muse (дата звернення: 10.06.2024).
13. Inworld AI : веб-сайт. URL: https://inworld.ai/ (дата звернення: 10.06.2024).
14. Layer AI. URL: https://www.layer.ai/ (дата звернення: 10.06.2024).

15. Polyhive AI: веб-сайт. URL: https://polyhive.ai/ (дата звернення: 10.06.2024).
16. Unity Sentis: веб-сайт. URL: https://unity.com/en/products/sentis (дата звернення: 10.06.2024).
17. Unity Render Pipelines: веб-сайт. URL: https://docs.unity3d.com/Manual/render-pipelines.html (дата звернення: 10.06.2024).

Декларація

академічної доброчесності

здобувача ступеня вищої освіти ЗНУ

Я, Красножон Іван Олегович, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz20bd-104@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Розробка комп'ютерної рольової фентезі-гри з використанням Unity**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 17.06.2024 Підпис _____ Красножон Іван Олегович

(студент)

Дата 18.06.2024 Підпис _____ Попівщий Василь Іванович

(науковий керівник)