МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

на тему: «***Розробка гри в жанрі RPG на рушії Unity***»

Виконав: студент 4 курсу, групи КН-42
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Євтушок Максим Едуардович

Керівник: ***Жуковський В.В., кандидат технічних наук, доцент кафедри ЕММІТ***

Рецензент: *кандидат технічних наук, доцент, доцент кафедри прикладної математики та кібербезпеки Донецького національного університету імені Василя Стуса*
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від «30» травня 2024 р.

Острог, 2024

Міністерство освіти і науки України
Національний університет «Острозька академія»

Факультет: *економічний*

Кафедра: *економіко-математичного моделювання та інформаційних технологій*

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри економіко-математичного моделювання
та інформаційних технологій
_____ Ольга КРИВИЦЬКА
«____» _____20__ р.

ЗАВДАННЯ

на кваліфікаційну роботу студента

Євтушка Максима Едуардовича

1. Тема роботи: ***Розробка гри в жанрі RPG на рушії Unity.***

Керівник роботи: Жуковський В.В.., кандидат технічних наук, доцент кафедри ЕММІТ.

Затверджено наказом ректора НаУОА від 03.11.2023 р., № 98.

2. Термін здачі студентом закінченої роботи: 31 травня 2024 року.

3. Вихідні дані до роботи: Unity, C#, Visual Studio Code.

4. Перелік завдань, які належить виконати: обрати ассети для використання, створити концепт карти, створити механіку руху та додавання анімацій, реалізувати дизайн ігрових рівнів, створити механіки бойової системи, реалізувати ші ворогів, створити кат-сцени через timeline та cinemachine, створити та налаштувати локації, створення вогняного шару, створення системи збереження, покращення системи прогресу, відображення тексту від шкоди, створення полоси здоров’я, створення візуальних ефектів, додавання курсорів, покращення системи здоров’я, додавання саунд ефектів, використання UniTask, створення меню гри та меню паузи, створення системи діалогів.

5. Перелік графічного матеріалу: рисунки, таблиці, діаграми, лістинги.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Жуковський В.В.	01.12.2023	01.12.2023
2	Жуковський В.В.	01.12.2023	01.12.2023
3	Жуковський В.В.	01.12.2023	01.12.2023

7. Дата видачі завдання: 01.12.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Затвердження теми проєкту	до 31.10.2023	
2	Постановка технічного завдання	до 01.12.2023	
3	Створення концепту гри	до 10.12.2023	
4	Створення базового функціоналу	до 12.12.2023	
5	Написання скриптів для функціоналу гри	до 12.12.2023	
6	Створення вогняного шару	до 01.03.2024	
7	Створення системи збереження	до 01.03.2024	
8	Покращення системи прогресу	до 01.03.2024	
9	Відображення тексту від шкоди	до 01.03.2024	
10	Створення полоси здоров’я	до 20.03.2024	
11	Створення візуальних ефектів	до 20.03.2024	
12	Додавання курсорів	до 20.03.2024	
13	Додавання звукових ефектів	до 01.04.2024	
14	Реалізація UniTask	до 01.04.2024	
15	Створення Меню	до 01.04.2024	
16	Створення діалогів	до 01.04.2024	
17	Тестування гри	до 20.04.2024	
18	Виправлення помилок	до 01.05.2024	
19	Попередній захист кваліфікаційної роботи	до 31.05.2024	
20	Здача кваліфікаційної роботи на кафедру	31.05.2024	

Студент: _____ Максим ЄВТУШОК

Керівник кваліфікаційної роботи: _____ Віктор ЖУКОВСЬКИЙ

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: *Розробка гри в жанрі RPG на рушії Unity.*

Автор: *Євтушок Максим Едуардович*

Науковий керівник: Жуковський В. В., кандидат технічних наук, доцент кафедри ЕММІТ.

Захищена «……»…… 2024 року.

Пояснювальна записка до кваліфікаційної роботи: *74 с., 61 рис., 1 табл., 1 додаток, 35 джерел.*

Ключові слова: Unity, RPG, гра.

Короткий зміст праці:

У цій кваліфікаційній роботі було розроблено RPG гру на рушії Unity. У прогресі розробки було використано такі засоби як рушій Unity, засіб для написання коду Visual Studio Code, магазин ассетів Unity Asset Store, мову програмування C# та інструмент для організації проєктів Jira. Метою було створення гри для любителів жанру RPG та для новачків. У цій роботі було показано різні види механік та як вони створювались для цікавого геймплею та залучення гравців.

(підпис автора)

ANNOTATION
of qualification paper
for bachelor's degree

Theme: *Development of an RPG game on the Unity engine.*

Author: *Yevtushok Maksym*

Supervisor: *Zhukovskyi V. V., candidate of technical sciences, associate professor of the Department of EMMIT.*

Defended **"....."..... 2024.**

Explanatory note to the qualification work: *74 p., 61 pic., 1 table, 1 attachment, 35 sources.*

Keywords: *Unity, RPG, game.*

Summary of the work:

In this qualification work, an RPG game was developed on the Unity engine. In the development process, the Unity engine, Visual Studio Code, Unity Asset Store, C# programming language, and Jira project management tool were used. The goal was to create a game for fans of the RPG genre and for beginners. The report showed different types of mechanics and how they were created for interesting gameplay and player engagement.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

ШІ - штучний інтелект

RPG (Role-Playing Game) - рольова гра

ВСТУП

Сьогодні ігрова індустрія досягла масштабів, де кожен, навіть не граючи в комп'ютерні ігри, чув про існування ігор. За десятки років ця індустрія виросла з чогось невідомого для більшості, до індустрії гіганта, де створюються та розвиваються різноманітні компанії, заробляючи мільйони доларів на продажах, створюючи ігри для людей будь-якого віку. Ігрова індустрія тепер впливає на різні аспекти нашого життя. Саме завдяки цьому розвивається бізнес, з'являються робочі місця, розвиваються країни з ігровими компаніями, отримуючи величезні податки, та, головне, для чого створені ігри — це розважати людей. У майбутньому кількість ігор буде рости. Завдяки конкурентності та талановитості розробників індустрія буде розвиватися, привалюючи ще більше гравців. Створюючи гру, розробник буде думати в першу чергу, як зацікавити користувача придбати їхню гру. Створення ігор, на перший погляд, може здаватися доволі простою та примітивною, але це зовсім не так. Розробка поділяється на багато різних етапів та деталей. Сотні людей працюють над великими проєктами для успішного виконання всіх задач. Попри це, працюючи індивідуально можна розробити гру, яка принесе успіх та дозволить розробнику, наймати працівників для розвитку компанії.

Створити ігровий додаток в жанрі RPG з такими механіками, як: патруль ворогів, підбирання та використання зброї, завдання шкоди ворогам та головному персонажу, механікою Click Event та Cinemachine, використовуючи різноманітні ассети (будь який файл який використовується в проєкті в Unity) для зброї, анімації, моделей персонажів та середовища, створення візуальних ефектів, створення системи збереження, покращити різні механіки при потребі, відображення health bar та шкоди від зброї, додавання курсорів, додавання саунд ефектів, реалізація інструменту UniTask, створення меню та створення діалогів.

Об'єктом дослідження є Ігровий рушій Unity, система для написання коду Visual Studio Code, магазин ассетів Unity Asset Store, інструмент для організації проєктів Jira, мова програмування C# та механіки для завершення гри.

Предметом дослідження є створення гри в жанрі RPG на рушії Unity для PC.

Для досягнення поставленої мети потрібно виконати такі задачі:

- Опис предметного середовища
- Огляд наявних аналогів
- Постановку задачі
- Аналіз предметної області
- Архітектура гри
- Розробка та створення локацій
- Система меню
- Використання бібліотеки UniTask
- Засоби розробки
- Вимоги до технічного та програмного забезпечення
- Опис програмної реалізації

РОЗДІЛ 1
АНАЛІЗ РИНКУ ІГОР ЖАНРУ RPG ТА ОПИС СЕРЕДОВИЩ ДЛЯ РОЗРОБКИ ІГОР ЖАНРУ RPG

1.1. Опис предметного середовища

Перед початком розробки був вибір між трьома рушіями Unity, Unreal Engine та Godot. Після ретельного аналізу та набутих знань було вирішено вибрати рушій Unity, оскільки цей рушій має значні плюси в порівнянні з іншими рушіями та дозволяє нам добре реалізувати поставлені задачі.

Unity - це рушій для розробки ігор, який дозволяє розробникам створювати ігри на C# для таких платформ, як Xbox, Playstation, Android, MacOS і PC. Unity - це безкоштовний рушій, який дозволяє розробникам безперешкодно працювати і розвиватися в цій індустрії. Рушій надає всі можливості, необхідні для створення найрізноманітніших ігор: 3D та 2D візуалізація, музика, звукові ефекти, освітлення, анімація і т.д. Також Unity має свій магазин Unity Asset Store з різноманітними ассетами різної вартості, завдяки цьому, можна обрати тематику та моделі для втілення своїх ідей і гру відштовхуючись від свого бюджету.

Переваги використання Unity

1. Велика функціональність рушія. Unity має весь потрібний функціонал для створення своєї гри. Незалежно від розміру запланованої гри.
2. Безкоштовний доступ для початківців. В Unity є 2 безкоштовних тарифа це Student та Personal. Ці тарифи можна використовувати якщо ви отримуєте менше ніж 100 тис. доларів на рік та для особистого користування. Тарифи наведені у рисунку 1.1.

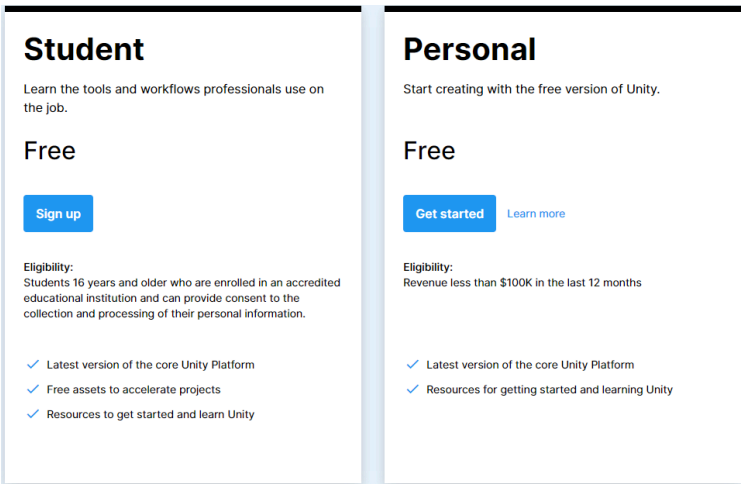

Рис. 1.1. Тарифи рушія*

Джерело: [1]

3. Велика спільнота та підтримка. Сьогодні Unity є одним із найпопулярніших рушіїв для розробки ігор. Завдяки цьому існує багато сайтів з розробниками даякі завжди готові допомогти, від новачка до професіонала, розв'язати їхню проблему.
4. Простота використання. Порівнюючи з головним конкурентом Unreal Engine, Unity є доволі простим для освоєння рушієм (див. рис. 1.2) оскільки має менше функціонала який зробить розробку важчою навіть для початківця.
5. Постійні оновлення та покращення. Щоб конкурувати з іншими рушіями, Unity потрібно завжди щось змінювати та покращуватись, тому для цього вони часто випускають нові версії з виправленням помилок та поліпшеннями використання рушія. Документацію Unity можна переглянути тут [2].

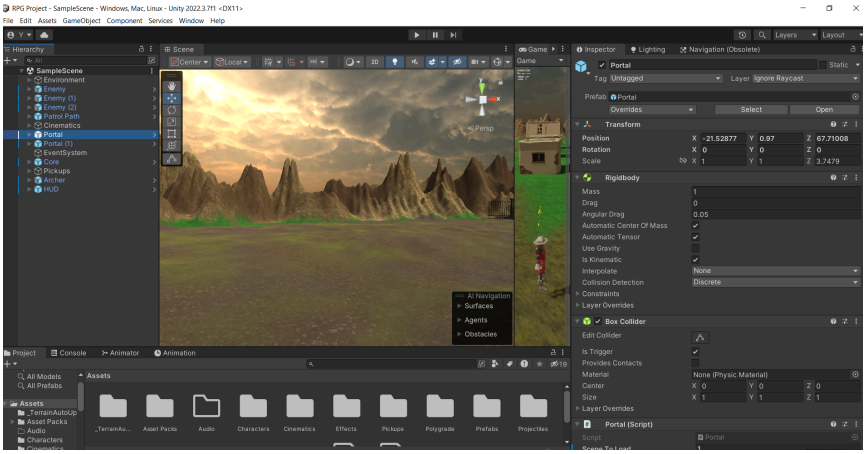

Рис. 1.2. Інтерфейс рушія

* створено автором

1.1.1. Мова програмування C#

C# - це об’єктно орієнтована мова програмування. C# походить від сімейства мов C. ЇЇ можна використовувати для створення багатьох різноманітних продуктів, від створення ігор до сайтів та додатків. C# є популярною тому, що ця мова програмування є універсально та досить легкою для вивчення. Microsoft створили її як просту, сучасну мову програмування для універсального призначення, яку можна використовувати для розробки компонентів програмного забезпечення, але протягом 22 років існування, вона стала використовуватись в різних сферах програмування. Приклад показаний на рисунку 1.3. Створення та використання C# [3].

Ключові особливості C#:

- Структурована мова програмування
- Великий вибір бібліотек
- Швидкість розробки
- Масштабування та оновлення
- Сумісність

```
public void EquipWeapon(Weapon weapon)
        {
            currentWeapon = weapon;
            Animator animator = GetComponent<Animator>();
            weapon.Spawn(rightHandTransform, leftHandTransform, animator);
        }
```

Рис. 1.3. Приклад C#

*створено автором

Переваги використання мови програмування C#

1. Велика спільнота та підтримка: C# має велику кількість користувачів які допомагають новачкам на різних сайтах спільнот у вирішенні проблем. Сьогодні існує сотні сайтів, де можна знайти відповіді для вирішення ваших питань. C# активно підтримується Microsoft, що сприяє оновленню різних технологій та інструментів для цієї мови.
2. Інтеграція з іншими технологіями: завдяки великій інтеграції з іншими мовами програмування, можна легко працювати з різними бібліотеками та системами.

3. Об’єктно-орієнтований підхід: C# підтримує об’єктно-орієнтований підхід, що спрощує розробку.
4. Платформна незалежність: C# була розроблена для платформи Microsoft .NET, що може використовуватись на таких операційних системах як Windows, Linux та MacOS.

1.1.2. Jira Software

Jira Software - це середовище для управління проєктами та відстеження помилок. Сьогодні Jira одна із найпопулярніших програм для керування проєктами. Jira дозволяє динамічно планувати, відстежувати та керувати своїми робочими процесами. Також Jira надає численні переваги, включно з можливістю впроваджувати спринти для керування Scrum та Kanban. Це допомагає ефективній організації проєктів (див. рис. 1.4), розподілу та визначення пріоритезації завдань. Крім того, Jira корисна для моніторингу прогресу, виявлення помилок і забезпечення повної прозорості протягом усього життєвого циклу проєкту. Головна сторінка Jira [4].

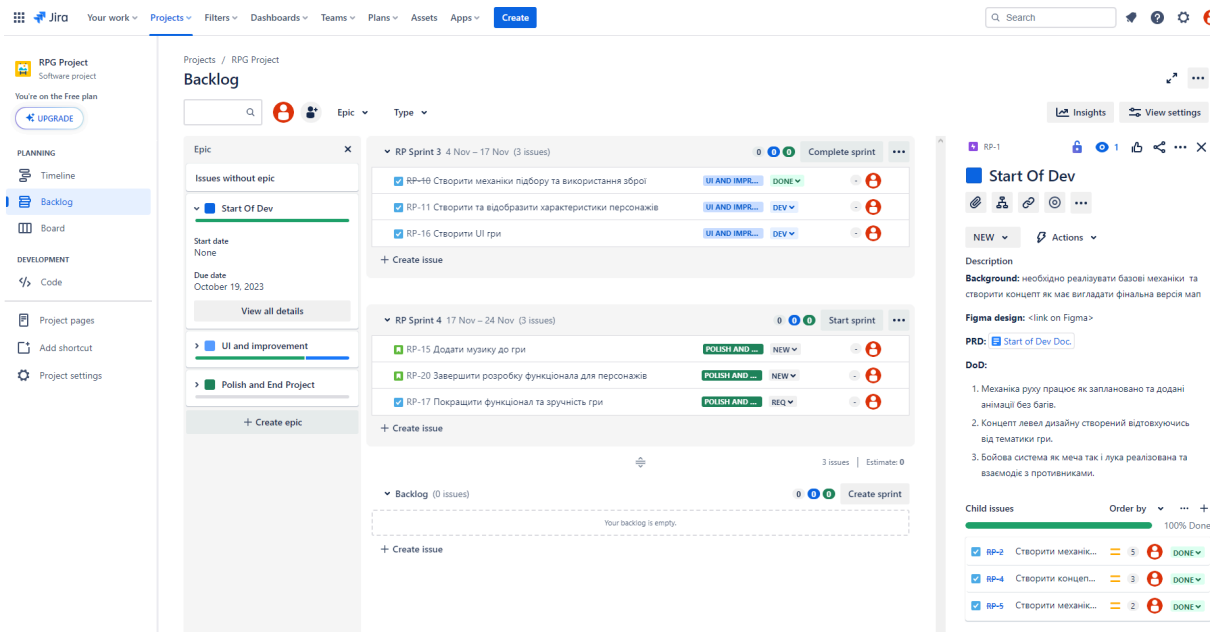

Рис. 1.4. Інтерфейс Jira

*створено автором в Jira

Переваги Jira

1. Зручні інструменти для призначення, створення, пріоритизації та відстеження завдань допомагають командам ефективно керувати роботою.
2. Jira дозволяє створювати власні типи задач, статуси, поля та робочі потоки.
3. Завдяки обговоренням у коментарях, відстеження змін та оновленням покращується співпраця в команді.
4. Jira може використовуватись для різноманітних проєктів - від розробки програмного забезпечення і розробки ігор до управління процесами та маркетингових кампаній.

1.1.3 Visual Studio Code

Visual Studio Code - це редактор коду, який полегшує виконання завдання, контроль версіями та налагодження. Розробники отримали інструмент, який їм потрібний для налагодження та швидкого циклу генерації коду. На відміну від використаного VS Code, Visual Studio IDE виконує більш складні процеси, але і працює повільніше. Приклад показаний на рисунку 1.5. Головна сторінка Visual Studio Code [5].

Ключові особливості Visual Studio Code:

- Комбінація клавіш за замовчуванням
- Налагодження
- Режим Дзен
- Інтеграція Git
- Зміна мовного режиму

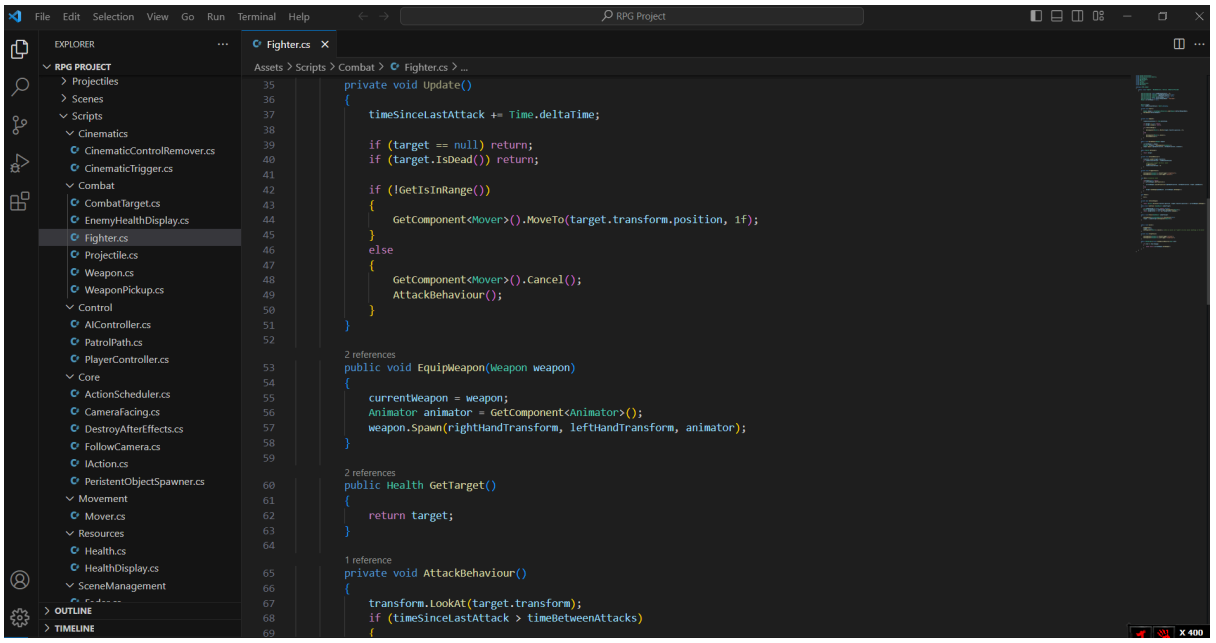

Рис. 1.5. Інтерфейс VS Code

*створено автором в VS Code

Переваги Visual Studio Code

1. Він є доступним для широкого кола розробників оскільки використовується на різних операційних системах.
2. Завдяки вбудованій підтримці Git ви можете легко керувати вашим репозиторієм.
3. Великою перевагою є швидкість та мінімальне споживання ресурсів, що дозволяє вам працювати більш продуктивно.
4. Синтаксичне підсвічування та автодоповнення допоможе вам швидше написати код та уникнути помилок.

1.1.4. Unity Asset Store

Unity Asset Store - це платформа де ви можете придбати як безплатно, так і платні ассети для вашої гри. Ассети включають різні види складових для гри, наприклад: анімації, моделі персонажів та об’єктів, музика та звукові ефекти. Приклад показано на рисунку 1.6.

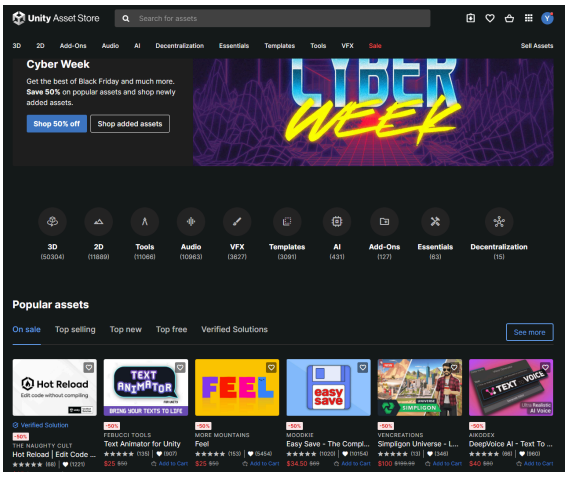

Рис. 1.6. Інтерфейс магазину

Джерело: [6]

Переваги Unity Asset Store

1. Зручний інтерфейс
2. Багато знижок для ассетів
3. Велика кількість авторів та матеріалу
4. Якісний безкоштовний матеріал для новачків

1.2. Огляд наявних аналогів

У найпопулярнішому ігровому магазині Steam можна знайти багато RPG-ігор. Сьогодні цей жанр є одним із найпопулярнішим у світі. Виходячи з жанру та схожих механік я знайшов найуспішніших конкурентів у цьому жанрі.

1. Diablo 4 - Комп’ютерна гра для PlayStation 5, Xbox One, Xbox Series, PlayStation 4, Microsoft Windows. Розроблена компанією Blizzard Entertainment, Inc.

Diablo 4 - це пригодницька RPG гра. Кампанію можна проходити самостійно або з друзями. Потрібно досліджувати відкритий світ, спостерігаючи за розгортанням захоплюючої історії. Завдяки кросплею (здатність гравців, які використовують різне обладнання для відеоігор, грати один з одним одночасно) та крос прогресії ви можете грати разом на всіх платформах разом.

У Diablo 4 дія відбувається через 50 років після подій Diablo 3, коли до світу Санктуаріуму повертається демониця Ліліт, дочка демона Мефісто, щоб перемогти свого батька та правити Пеклом. За її слідами вирушає головний герой під наставництвом янгола-відступника Інаріуса, аби завадити планам Ліліт. Фото обкладинки гри та приклад геймплею показано на рисунку 1.7 та 1.8.

Рис. 1.7. Diablo 4

Джерело: [7]

Рис. 1.8. Геймплей

Джерело: [7]

2. Baldur’s Gate 3

Baldur’s Gate 3 — це гостросюжетна рольова гра у всесвіті Dungeons & Dragons, де ваш вибір формує історію дружби та зради, виживання та жертви та спокуси абсолютної влади.

Baldur's Gate 3 отримала всесвітнє визнання та зібрала численні нагороди від ігрових критиків та гравців. Вона була удостоєна звання "Гри року" від Golden Joystick Awards і The Game Awards. Деякі профільні видання називають її грою десятиліття. Розробники залучили голлівудських акторів, таких як Джейсон Айзекс та Джонатан Сіммонс і відомих в ігровій індустрії акторів озвучування, серед яких Меттью Мерсер, Меггі Робертсон. Фото обкладинки гри та приклад геймплею показано на рисунку 1.9 та 1.10.

Рис. 1.9. Baldur’s Gate 3

Джерело: [8]

Рис. 1.10. Геймплей

Джерело: [8]

3. Hades

Hades - це рольова гра, де ви граєте за персонажа з греко - римської міфології, змагається проти богів, долаєте неймовірні складнощі та побачите чудовий сюжет.

У грі ви виступаєте у ролі Заги, сина бога підземелля, Гадеса. Головний герой намагається вибратися з царства підземелля і дістатися на Олімп, відомий як рід богів.

Головна мета гри - дістатися до кінця підземелля і перемогти самого батька, Гадеса. Але це важке завдання, яке вимагає від вас стратегічного мислення, вправності у бою та розвинутої тактики. Фото обкладинки гри та приклад геймплею показано на рисунку 1.11 та 1.12.

Рис. 1.11. Hades

Джерело: [9]

Рис. 1.12. Геймплей

Джерело: [9]

Для аналізу конкурентів доцільно використати методику SWOT-аналізу. SWOT розшифровується як Strengths, Weaknesses, Opportunities і Threats (сильні та слабкі сторони, можливості та загрози). Цей інструмент допомагає проаналізувати, що ви робите найкраще та розробити успішну стратегію для майбутнього. Також SWOT може виявити слабкі сторони компанії або проєкту та загрозу виявлення конкурентами. Результати SWOT-аналізу гри “Baldur’s Gate 3” наведено в таблиці 1.1.

Таблиця 1.1

SWOT-аналіз гри “Baldur’s Gate 3”*

Сильні сторони	Слабкі сторони
Красива графіка Великий функціонал Хороший сюжет	Важкі рівні Важкий геймплей
Можливості	**Загрози**
Додавання нового контенту Виправлення багів	Поява гри з більш цікавим сюжетом та зручнішим геймплеєм

* створено автором

1.3. Постановка задачі

Ця гра буде цікава для гравців різного віку та для гравців яким цікавий жанр RPG. Гравці можуть випробувати свої сили в проходженні захоплюючої гри з різноманітними механіками. Гра доступна на PC для Windows.

Для створення гри була використана мова програмування C# та рушій Unity. Ассети для гри були взяті з Unity Asset Store бесплатно. Концепт та фінальна версія була створення інструментом Unity. Гра розробляється для PC оскільки більшість гравців які зацікавлені в цьому жанрі більшість ігрового часу грають тільки на PC.

Завдяки широкому функціоналу гра має залучити різноманітних гравців, незалежно від віку.

Гра повинна містити наступний функціонал:

1. Створення концепту та реалізації карти - На початку розробки було використано інструмент Unity Terrain tool, завдяки цьому реалізовано концепт карти та використані різноманітний функціонал для успішного завершення.
2. Написання коду - в Unity було створення різноманітні скрипти які були додані в різні папки для кращої організації проєкту.
3. Створення механік - різноманітні механіки були створенні через скрипти та інструменти рушія. Завдяки успішній організації та поєднано цих факторів було виконано план на створення різних механік.
4. Додавання візуальних ефектів - візуальні ефекти були створені для кращого сприйняття гравців реалізованих механік.
5. Додавання саунд ефектів - завдяки зручному інтерфейсу Unity було легко додано різні звукові ефекти використавши безкоштовні ассети.
6. Реалізація штучного інтелекту - через написання скрипту та поєднання його з функцією NavMesh було створено штучний інтелект та доданий до ворогів.
7. Виправлення помилок та покращення функціоналу - В кінці виконання гри було виконано покращення коду для кращого функціоналу та організації гри.

Висновки до розділу 1

У першому розділі було описано використані предметні середовища, деталі переваг кожного з середовищ, аналізовано конкурентів, їхній успіх на міжнародному ринку, SWOT-аналіз, постановку різних задач для проєкту та опис середньостатистичного гравця для цієї гри.

В наступному розділі буде розглянуто вибір жанру для гри, визначення основних механік та особливостей потрібних для розробки рольової гри. Також буде описано архітектуру гри: діаграми меню та організацію папок в Unity.

РОЗДІЛ 2
ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Аналіз предметної області

Сьогодні існує багато ігор з різноманітними жанрами і кожен розробник хоче зацікавити користувача грати в його гру. У середовищі розробки ігор дуже важко виділитись в умовах постійної конкуренції, тому потрібно створювати щось унікальне та цікаве, те що зможе привернути увагу гравців. Для створення комерційно успішної гри, потрібно вирішити багато організаційних етапів розробки. Наприклад: концепція гри, управління маркетингом та контроль продажів, залучення гравців, створення технічної складової, розподіл бюджету, тестування гри перед випуском та її графік розробки. В рольові ігри , грають люди різного віку і зазвичай вони грають для того, щоб насолодитися історією, що запам'ятається на довгі роки. Однак, щоб створити цікаву історію, потрібно створити зручний геймплей який не буде негативно впливати на загальну репутацію гри. Для того, щоб розв'язувати цю проблему потрібно створити гру з наступним геймплеєм:

1. Головному персонажу потрібно потрапити в секретну локацію, обравши 1 дорогу з 3.
2. На кожній дорозі будуть зустрічатися вороги різного типу.
3. Для подолання ворогів можна підібрати різну зброю та вирушити в подорож.
4. В головного персонажа та різних ворогів є здоров’я, тому для потрапляння на секретну локацію потрібно орієнтуватися на рівень свого здоров’я та силу ворогів
5. Не всі вороги будуть ставитись до вас агресивно, якщо ви оберете правильний вибір в діалозі.

2.1.1. Визначення основних механік та особливостей геймплею.

Визначення основних механік та особливостей геймплею є важливою складовою планування успішного проєкту оскільки без правильного планування етап розробки перетвориться в хаос. Для розуміння потрібних механік потрібно

проаналізувати механіки відомих ігор в цьому жанрі та як вони залучили гравців. Кожна вікова група гравців мають свої уподобання тому для цього потрібно збалансувати наповнення ігрового світу потрібними механіками та особливостями.

В грі повинна бути достатня кількість різноманітних механік для виживання головного героя та для можливості гравцю насолоджуватись грою.

Отже головні задачі та механіки для проходження гри:

- Використання зброї. В грі присутні 3 види зброї не враховуючи рукопашний бій. Для використання цієї зброї потрібно підібрати її на карті. Також через деякий час вона може з’являтись знову на тому самому місці.
- Прогрес для покращення здібностей для виживання. Використовуючи зброю та долаючи різних ворогів можна покращувати своє здоров’я та рівень шкоди ворогам.
- Можливість збереження. Збереження дозволяє гравцеві зберегти свій прогрес та продовжувати гру не боячись за втрату прогресу. При втраті всього здоров’я гравець відправиться на останню точку збереження.

Використовуючи ці механіки гравець може пройти до потрібної локації. Також при конкретному проходженні можна пройти гру не здолавши ні єдиного ворога використовуючи діалог з ворогами. Вибір яким стилем має проходити гру гравець може обрати сам.

2.2. Архітектура гри

Для створення архітектури була створена ігрова діаграма з використанням різних пунктів гри, які також використані як папки для скриптів. В архітектурі було використано такі пункти як Control, Movement, Cinematics, Scene Management, Combat, Resources, Stats, Core та Dialogue. Кожен із пунктів відповідає за відповідну функціональність і залежить від іншого пункта.

Для правильного планування гри необхідно створити архітектурну складову для кращого розуміння, які етапи повинні виконуватись в першу чергу, а які не можуть продовжуватись без інших. Приклад такої архітектури показано на діаграмі

Також потрібно розуміти середовище, потрібне для успішного виконання завдання.

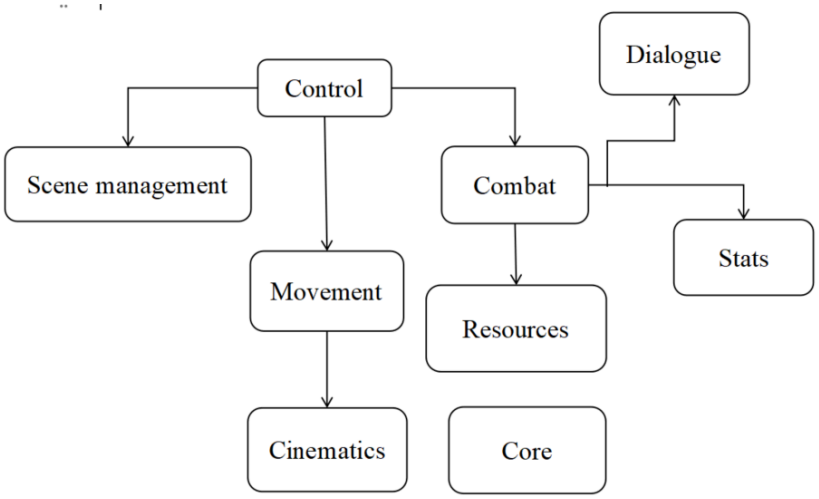

Рис. 2.1. Діаграма архітектури

*створено автором

Для організації скриптів було створено під папки де знаходяться всі скрипти. Кожна папка відповідає різним пунктами та функціональності гри. Cinematics - кат-сцени, Combat - бойова система, Control та Movement - рух під час гри, Core - рух камери, Dialogue - діалоги, Resources - здоров'я, Saving - система збереження, Stats - система прогресу, UI - інтерфейс. Приклад показаний на рисунку 2.2.

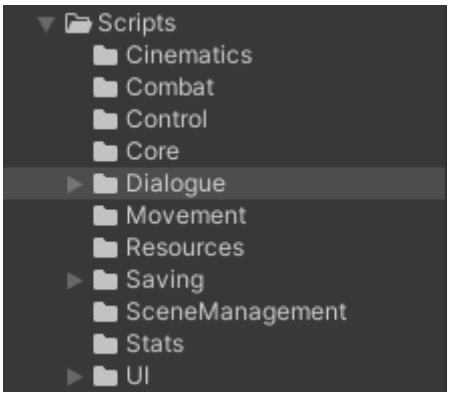

Рис. 2.2. Папка з скриптами

*створено автором на рушії Unity

Організація ассетів є важливою складовою розробки оскільки при правильній організації розробка гри робиться набагато швидше та продуктивніше. На рисунку 2.3 показано всі створенні папки для різноманітних ассетів з яких була створена гра.

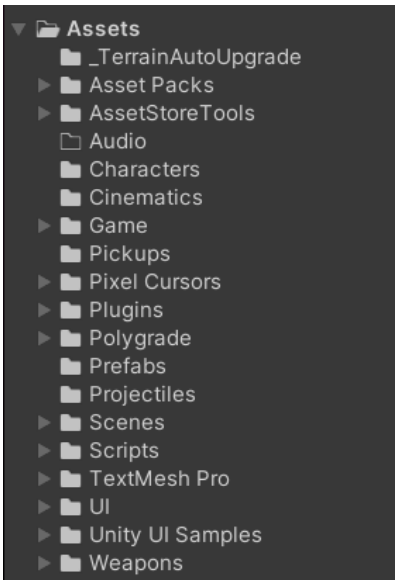

Рис. 2.3. Папки з ассетами

*створено автором

2.3. Розробка та створення локацій

Проектування локацій є важливим аспектом гри оскільки це впливає на час проходження, важкість проходження та на враження гравця. Тому важливо знайти потрібний баланс для жанру та тематики гри.

Для створення локацій було використано інструмент рушія Terrain tool. Цей інструмент дозволяє розробнику модифікувати та втілювати різні задачі в левел дизайні. На рисунку 2.4 показано результат концепту та використання цього інструмента який є дуже зручним та дієвим для розробки.

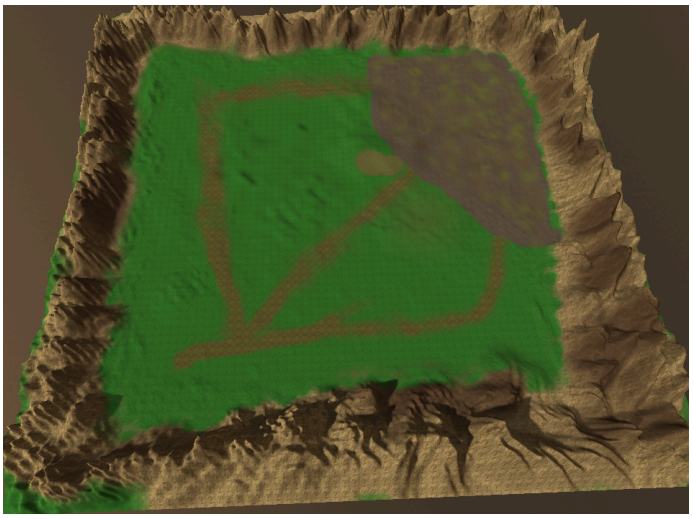

Рис. 2.4. Концепт 1 сцени

*створено автором

2.4. Система меню

Для створення головного меню потрібно визначити потрібний функціонал який ми маємо розробити. Для цього було створено діаграму послідовності переходів на головному меню. Приклад діаграми показаний на рисунку 2.5 та 2.6.

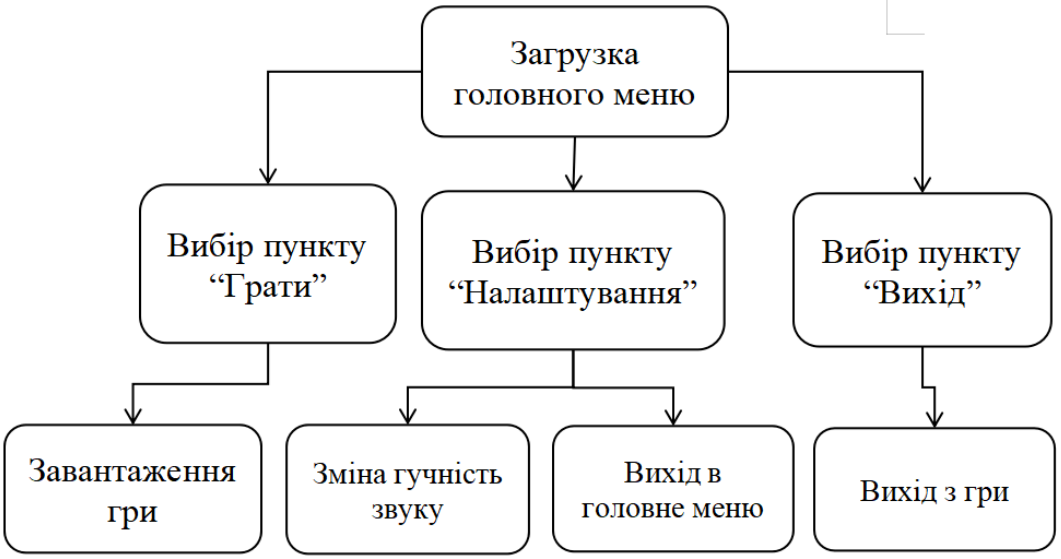

Рис. 2.5. Діаграма послідовності головного меню

*створено автором

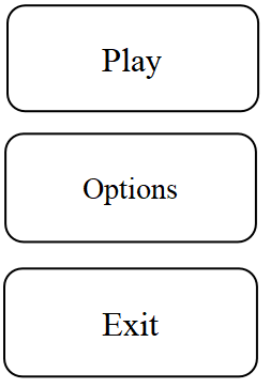

Рис. 2.6. Прототип головного меню

*створено автором

2.5. Використання бібліотеки UniTask

UniTask - це бібліотека для роботи з асинхроним (асинхронний код Unity дозволяє виконувати трудомісткі завдання (завантаження файлів, мережеві запити та обчислення), не блокуючи основний потік виконання, що забезпечує більш плавний та зручний інтерфейс користувача) кодом на рушії Unity. Вона допомагає розробникам, надаючи ефективний та зручний спосіб працювати з асинхронними операціями, такими як мережеві запити, зберігання даних та завантаження ресурсів.

Завдяки цьому, розробники можуть виконувати свою роботу більш ефективно та продуктивно, покращуючи швидкість розробки в проєктах різної складності.

Головні переваги UniTask:

1. UniTask легкий у використанні разом з існуючим кодом в Unity.
2. Він легкий для написання і розуміння оскільки дозволяє працювати з стандартними конструкціями async/await.
3. Підвищує швидкість та продуктивність гри завдяки високоефективний реалізації асинхронних операцій.
4. UniTask є популярною бібліотекою у спільноті Unity, тому під час виникнення питань можна без зволікань звернутися до підтримки на сайтах спільноти.

Приклад використання показаний в лістингу 2.1.

```
private async UniTask FadeRoutine(float target, float time,
CancellationToken token)
    {
        while (!Mathf.Approximately(canvasGroup.alpha, target))
        {
        canvasGroup.alpha = Mathf.MoveTowards(canvasGroup.alpha, target,
Time.deltaTime / time);
            await UniTask.Yield(PlayerLoopTiming.Update, token);
        }
    }
```

Лістинг 2.1. Приклад UniTask в скрипті Fader

* створено автором

Висновки до розділу 2

В цьому розділі було розглянуто проблематику створення рольової гри, та які геймплейні складові потрібно для залучення гравців, проаналізовано основні механіки гри, для правильного планування та описано чому це важливо. Описано кожен з головних етапів нашої гри, які відповідають за конкретний функціонал та в поєднанні, доповнюють загальне наповнення гри.

Також було описано, чому створення концепту є важливим та інструменти, які ми використовували для реалізації концепту, описано алгоритм виконання та прототип головного меню та яку бібліотеку було використано для покращення асинхроних методів.

РОЗДІЛ 3
ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Засоби розробки

Засобами розробки є : Unity, Unity Asset Store, Visual Studio Code, Jira, UniTask - для роботи з скриптами, інтерфейсом та бібліотеками.

Рушій Unity - один із найпопулярніших і найкращих програм для створення ігор, різноманітних додатків та створення дизайну для середовища.

Також Unity є потужним графічним рушієм оскільки він підтримує різні ефекти, включаючи освітлення, тіні, текстури та шейдери. Перевагою для розробників також є вбудована система фізики та система анімацій. Система анімацій має вбудовані інструменти для створення складних анімацій для персонажів, об’єктів та інших елементів гри. Завдяки вбудованій системі фізики розробник має можливість створювати реалістичні фізичні об’єкти, такі як гравітація, колізії та симуляція руху об’єктів.

3.2. Вимоги до технічного та програмного забезпечення

Для розробки гри було визначено такі вимоги до програмного забезпечення:

1. Unity -версія 2022.3.7f1.
2. C# - мова програмування.
3. Microsoft Visual Studio Code - середовище для написання коду.
4. Unity Asset Store - магазин ассетів, інструментів та бібліотек.

Вимоги до технічного забезпечення:

1. Процесор: Intel Core i5-11400H
2. Відеокарта: NVIDIA GeForce RTX 3050
3. Операційна система: Windows 10/11 64-розрядна
4. Оперативна пам’ять: 16 ГБ

3.3. Опис програмної реалізації

3.3.1 Unity Asset Store

Проаналізувавши магазин Unity Asset Store та його безкоштовні ассети було обрано тему середньовіччя, яку добре можна реалізувати з доступними безкоштовними ассетами.

Головними критеріями для вибору цієї тематики були модель персонажів, середовище, моделі будинків, саунд візуальні ефекти. Обрані ассети в грі було обрано в офіційному магазині Unity - Unity Asset Store, посилання на них можна переглянути нижче в додатку.

3.3.2. Terrain tools

Для створення концепту карти було обрано використовувати інструмент Terrain tool. Terrain tool - це інструмент для створення та редагування ландшафтів. Використання цього інструменту надає розробнику можливість зручно створювати рельєфи, озера та гори. Приклад використання показаний на рисунку 3.1. Весь можливий функціонал інструменту Terrain Tool описаний в офіційній документації Unity [10].

Особливості інструмента:

1. Легкість створення та налаштування розмірів.
2. Можливість додавання дерев, кущів, каменів що робить гру більш реалістичним
3. Налаштування текстур для різних частин карт створених інструментом.

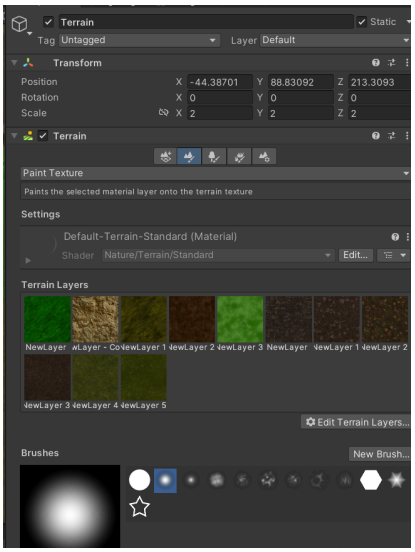

Рис. 3.1. Інструмент Terrain

* створено автором

Для створення концепту карти було обрано гірний ландшафт та 3 дороги, на якій будуть знаходиться селища, через які буде на вибір проходити персонаж для проходження рівня. Приклад показано на рисунку 3.2.

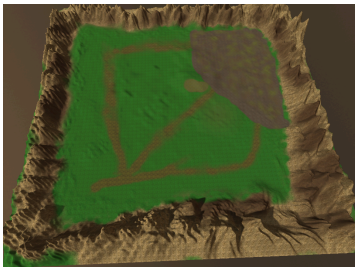

Рис. 3.2. Концепт карти

*створено автором

Після закінчення концептів було додано ассети з моделями та освітленням для створення повноцінної карти. Приклади карт показано на рисунках 3.3 та 3.4. Офіційний ассет вітряка було взято з посилання [11].

Рис. 3.3. Завершена 2 карта

*створено автором

Рис. 3.4. Завершена 1 карта

*створено автором

3.3.3. SkyBox

Для зміни неба було використано наявний skybox який змінює текстуру неба та створює нову атмосферу яку підходить грі. Для підключення skybox потрібно в голоній камері змінити clear flags на skybox, після цього маючи наявні варіанти неба (див. рис. 3.5) ми можемо створювати різні версії неба для нашої гри.

Рис. 3.5. Головна камера

*створено автором

На рисунку 3.6 показано всі 19 можливих skybox які доступні для вибору

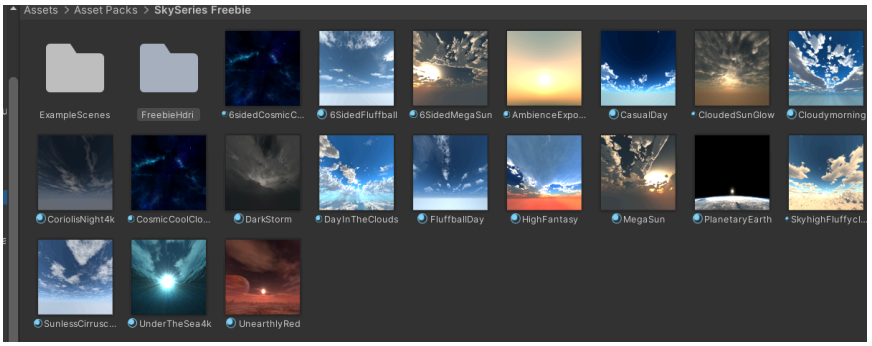

Рис. 3.6. Вибір skybox

*створено автором

Після детального аналізу було обрано цей skybox який найбільше підходив під тематику гри та створював унікальну атмосферу. Приклад показано на рисунку 3.7 та використаний ассет за посиланням [12].

Рис. 3.7. Обраний SkyBox

*створено автором

Таким чином маючи широкий вибір скайбоксів ми можемо підлаштувати наше оточення в грі під потрібну атмосферу. Документація по використанню SkyBox [13].

3.3.4. Використання NavMesh

NavMesh - це інструмент, який використовується для навігації території у середовищі де головний герой може рухатись. Його основна мета - визначити найефективніший маршрут до визначеного пункту призначення, оминаючи будь-які перешкоди, що зустрічаються на цьому шляху, і сприяти безперебійній навігації для штучного інтелекту.

Демонстрація, зображена на рисунку 3.8 , демонструє практичне застосування NavMesh. Після виконання всіх необхідних конфігурацій виділена синя область стає потрібним шляхом, яким персонаж може вільно рухатися. Однак важливо зазначити, що будь-яка область за межами синьої області обмежує рух персонажа. Це навмисне обмеження гарантує, що персонажі залишаються в межах карти та не зможуть подолати непрохідну місцевість. Документація NavMesh [14].

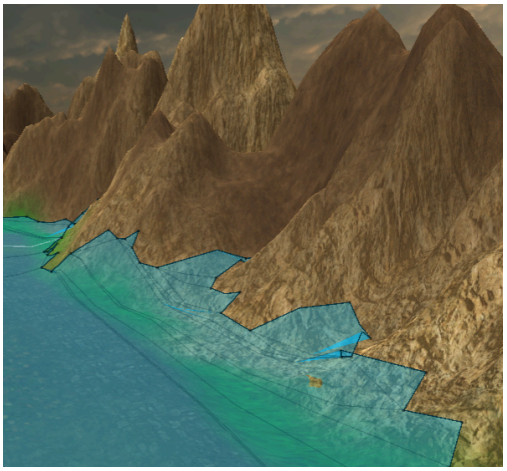

Рис. 3.8. NavMesh

*створено автором

Так виглядає одна з налаштувань для NavMesh де можна налаштувати висоту, градус підйомів тощо. Для того, щоб налаштування запрацювали потрібно натиснути кнопку Bake.

На рисунку 3.9 відображається вигляд однієї з конфігурацій для NavMesh, демонструючи параметри зміни висоти, рівня підйому та інших змінних. Щоб активувати ці налаштування, потрібно натиснути кнопку Bake.

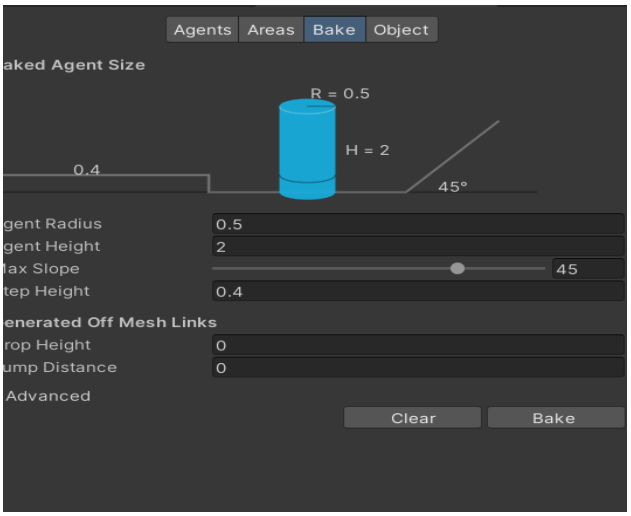

Рис. 3.9. Налаштування NavMesh

*створено автором

Налаштування Nav mesh Agent дозволяє вашому персонажу рухатись до конкретної локації уникаючи перешкод. Приклад показано на рисунку 3.10. більше про використання Nav Mesh Agent [15].

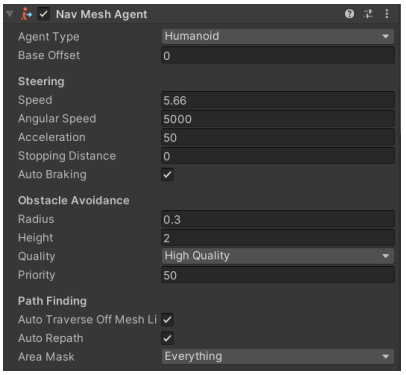

Рис. 3.10. Налаштування NM Agent

*створено автором

Було написано скрипт Mover.cs який знаходиться в папці Movement, він завершує реалізацію NavMesh в проекті. В лістингу 3.1 знаходиться головна частина коду.

```
void Update()
      {
         navMeshAgent.enabled = !health.IsDead();
         UpdateAnimator();
      }
      public void MoveTo(Vector3 destination, float speedFraction)
      {
         navMeshAgent.destination = destination;
                                    navMeshAgent.speed    =    maxSpeed    *
Mathf.Clamp01(speedFraction);
         navMeshAgent.isStopped = false;
      }

      public void Cancel()
      {
         navMeshAgent.isStopped = true;
      }
```

Лістинг 3.1. Реалізація NavMesh

* створено автором

3.3.5. Механіка руху Click Event

Механіка Click Event - це базова механіка створена для того щоб персонаж рухався до точки натискання мишки.

Для Використання механіки Click Event було створено метод Interact With Movement яка дозволяє рухати гравцем клікнувши на любу частину карти. Приклад показано в лістингу 3.2. Більше про використання Click Event [16].

```
private bool InteractWithMovement()
        {
            RaycastHit hit;
            bool hasHit = Physics.Raycast(GetMouseRay(), out hit);

            if (hasHit)
            {
                if (Input.GetMouseButton(0))
                {
                    GetComponent<Mover>().StartMoveAction(hit.point, 1f);
                }
                return true;
            }
            return false;
        }
```

Лістинг 3.2. Реалізація Click Event

* створено автором

3.3.6. Створення камери яка рухається за персонажем

Щоб створити цю механік, було написано скрипт FollowCamera.cs у папці Core. Використовуючи метод Late Update, успішно виявлено ціль камери та встановили шлях її руху. У результаті камера тепер безперешкодно відстежуватиме рухи нашого персонажа протягом гри, незалежно від їхнього напрямку. Більше про рух камери можна переглянути тут [17].

3.3.7. Додавання моделей персонажів

У проєкт було додано безкоштовні ассети де ми обрали потрібні моделі для різноманітних задач. Моделі персонажів чудово вписуються під тематику середньовіччя та мають різноманітні кольори костюмів. Після встановлення їх потрібно імпортувати в проєкт. Після імпортування потрібно перенести префаб

персонажа, підключити до нього анімації через Animator та додати скрипти, колайдери тощо. На рисунку 3.11 та 3.12 показано фінальну версію головного персонажа за якого гравець буде грати та ворога. Фінальна версія персонажа в грі та початково префаба відрізняється тим, що до персонажа підключені різноманітні анімації, Capsule Collider (невидима форма яка використовується щоб обробляти фізичні зіткнення для об'єкта), Rigidbody (тіло, яке ні за яких умов не деформується і за всіх умов відстань між двома точками якого залишається постійною), Nav Mesh Agent та 10 скриптів: Mover, Action Scheduler, Health, Fighter, Base Stats, Player Controller та Experience, Saveable Entity, Player Conversant. Також в скрипті Fighter було реалізування підбирання різної зброї в праву та ліву руки. Посилання на використаний ассет [18].

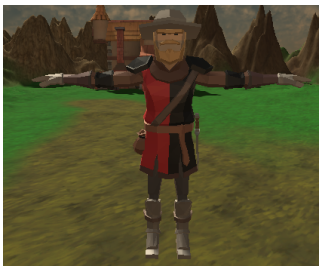

Рис. 3.11. Головний персонаж

*створено автором

Порівняно з персонажем, налаштування ворога доволі схожі, але зміни які роблять з них два різних об'єкти. На відміну від головного персонажа, ворог не має скриптів Player Controller та Experience, але має AI Controller та Combat Target, Saveable Entity.

Рис. 3.12. Модель ворога

*створено автором

Для навколишнього середовища були створені різні ассети, включаючи будівлі та звичайними речами. Всі вони розміщені по всій карті і добре доповнюють атмосферу гри. Приклад показано на рисунку 3.13. Використаний ассет було взято з офіційного сайту за посиланням [19].

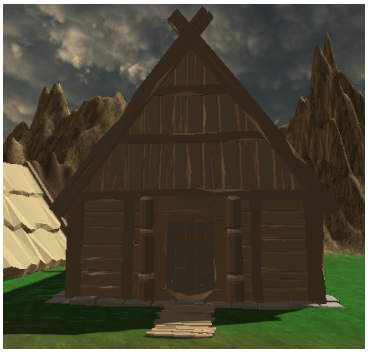

Рис. 3.13. Модель будинку

*створено автором

3.3.8. Використання анімацій

Для створення анімації були використані Blend Tree. Він дозволяє послідовно виконувати різні анімації на одному об'єкті. Наприклад: Idle > Walk > Run. Для цієї послідовності було створено Blend Tree під назвою Locomotion. Приклад показано на рисунку 3.14. Використаний ассет з анімаціями було взято за посиланням [20].

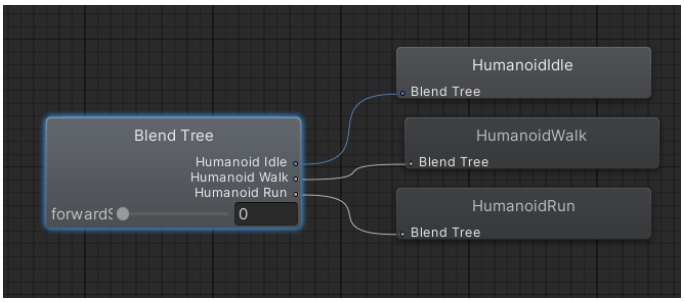

Рис. 3.14. Blend Tree

*створено автором

В розділі Motion можна міняти різні ассети, протягнувши з файлу або просто нажати на панель щоб обрати можливі варіанти. Вище в Parameter можна побачити як три анімації переходять з однієї до іншої.

На рисунку 3.15 налаштований Blend Tree, який дозволяє контролювати порядок анімацій і швидкість переходів між ними. У розділі Motion ви можете змінювати різні об'єкти, перетягуючи їх з файлу або клацнувши на панелі і вибравши потрібну опцію. У розділі Parameter вище, ви можете побачити, як відбувається перехід між трьома анімаціями.

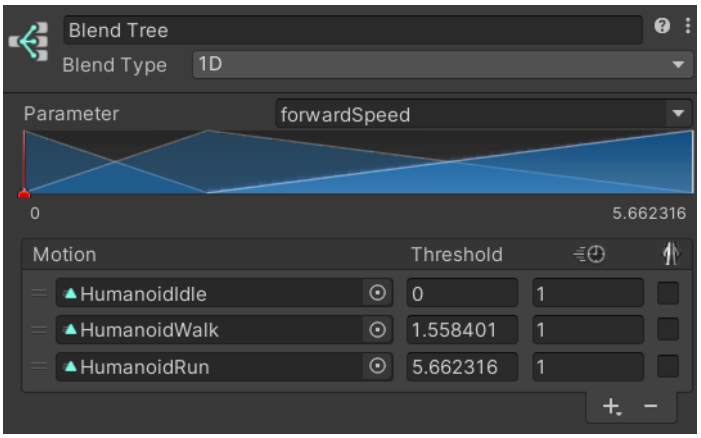

Рис. 3.15. Налаштування Blend Tree

*створено автором

Після додавання анімацій, написано код який реалізував функціонування анімацій. Ось приклад того як була реалізована смерть персонажа використовуючи анімації та зв’язки з іншими скриптами. Приклад показано в лістингу 3.3.

```
private void Die()
        {
            if (isDead) return;
            isDead = true;
            GetComponent<Animator>().SetTrigger("die");
            GetComponent<ActionScheduler>().CancelCurrentAction();
        }
```

Лістинг 3.3 Анімація смерті

* створено автором

Метод EquipWeapon дозволяє персонажу піднімати різну зброю, залежно від підходящої руки для зброї. Приклад показано в лістингу 3.4.

```
public void EquipWeapon(Weapon weapon)
        {
            currentWeapon = weapon;
            Animator animator = GetComponent<Animator>();
            weapon.Spawn(rightHandTransform, leftHandTransform, animator);
        }
```

Лістинг 3.4 Підбір зброї

* створено автором

Було створено два методи для зупинки та атаки персонажем, реалізовуючи механіку реагування тригера для зупинки атаки та для початку атаки в залежності від прописаної ситуації.

Для управлінням анімаціями та переходами між ними було використано Animator. На рисунку 3.16 показано переходи між анімаціями Attack та Locomotion яка включає в себе вище згаданий Blend tree, а також перехід між всіма станами та Death. Більше про використання анімації [21].

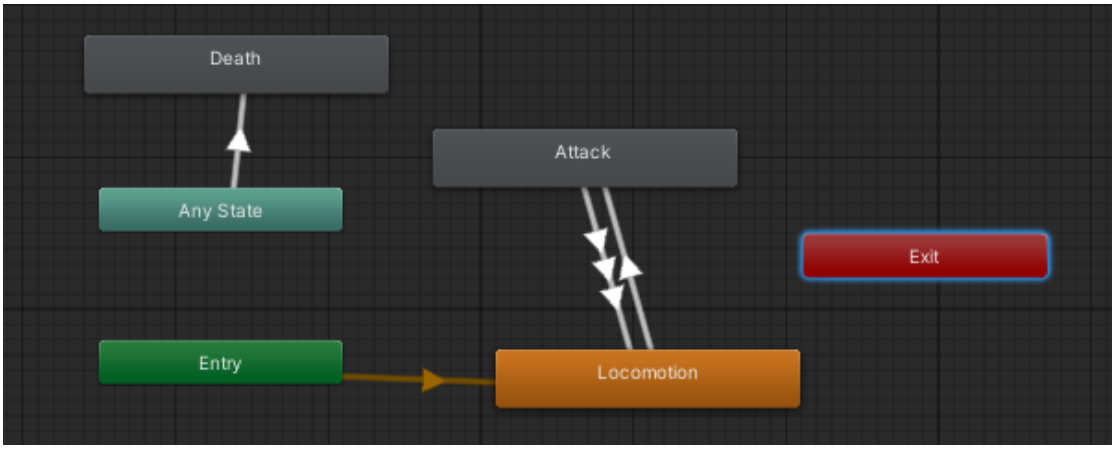

Рис. 3.16. Animator

*створено автором

3.3.9. Бойова система

При натисканні лівої кнопки миші на ворога, персонаж рухається до нього і починає атакувати. Для цього було створено метод Interact With Combat. Приклад показано в лістингу 3.5.

```
if (Input.GetMouseButtonDown(0))
                {
                      GetComponent<Fighter>().Attack(target.gameObject);
                }
```

Лістинг 3.5. Управління мишкою

* створено автором

Для реалізування бойової системи було створено різноманітні скрипти, які дозволяють персонажам брати різну зброю і атакувати різною зброєю з різною шкодою. Також існують стріли ворогів та головного персонажа, наприклад, ворогів - червоні, сині - головного персонажа. У класі Mover було створено можливість регулювати швидкість відповідно до загальної швидкості гри, щоб не порушувати баланс.

Також було створено метод CanAttack, який перевіряє, чи ворог вже мертвий, чи потрібно зупинитися і продовжувати атакувати, поки ціль не буде нейтралізована. Приклад показано в лістингу 3.6.

```
public bool CanAttack (GameObject combatTarget)
        {
            if (combatTarget == null) {return false;}
            Health targetToTest = combatTarget.GetComponent<Health>();
            return targetToTest != null && !targetToTest.IsDead();
        }
```

Лістинг 3.6. Перевірка ворога

* створено автором

3.3.10. Здоров’я

Для відображення здоров'я ворогів та персонажів у грі було створено скрипти Enemy Health Display та Health Display. Також було створено Canvas для відображення різних елементів гри у вигляді тексту, таких як здоров'я персонажа, здоров'я ворога, бали за вбивство і рівні, які збільшуються залежно від кількості балів. Приклад показано в лістингу 3.7.

```
string healthValue = String.Format("{0:0}", experience.GetPoints());
healtText.SetText(healthValue);
```

Лістинг 3.7. Відображення здоров’я

* створено автором

Після перемоги над ворогом, ви отримуєте винагороду відповідно до рівня складності. Наприклад, 10 балів за ворога 1-го рівня, 20 балів за ворога 2-го рівня і т.д. Цей скрипт був створений для реалізації цієї механіки. На початку було вирішено відображати здоров’я у відсотках. Однак з розвитком системи здоров'я, відсотки були переведені в звичайні числа, щоб гравець краще розумів прогрес і погіршення стану здоров'я. Приклад показано в лістингу 3.9.

```
healthPoints = Mathf.Max(healthPoints - damage, 0);
      if (healthPoints == 0)
      {
          Die();
          AwardExperience(instigator);
      }
```

Лістинг 3.8. Отримання нагороди

* створено автором

При вбивстві ворога, здоров’я відновлюється до 70. Приклад реалізації можна побачити в наступному методі. Він призначений для того, щоб заохотити гравця продовжувати грати, оскільки ворогів багато і стандартного здоров’я може не

вистачити. Після покращення системи здоров'я персонаж відновлює здоров'я конкретним відсотком від загального здоров'я. Приклад показано в лістингу 3.9.

```
float regenHealthPoints GetComponent<BaseStats>().GetStat(Stat.Health) * (regenerationPercentage / 100);
healthPoints = Mathf.Max(healthPoints, regenHealthPoints);
```

Лістинг 3.9. Регенерація здоров'я

* створено автором

Коли бойова система тільки створювалася, була виявлена помилка, коли персонаж не дивився в бік ворога при атаці, тому був написаний скрипт, щоб виправити цю помилку і змусити персонажа дивитися в бік цілі. Приклад показано на рисунках 3.17 та 3.18.

Рис. 3.17. До підвищення рівня

*створено автором

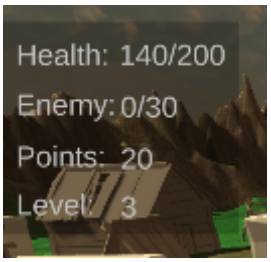

Рис. 3.18. Після підвищення рівня

*створено автором

3.3.11. AI

Для цікавого геймплею та функціоналу був створений ворожий патруль. Це реалізовано за допомогою скрипту, який визначає наступну точку, яку ворог повинен патрулювати. У режимі розробника також створено видимий об'єкт, що показує патрульні групи. Патрульна група - це група точок, до яких ворог рухається поступово, як показано на рисунку 3.19). Для візуалізації патрулів були створені Gizmos. Також був використаний інструмент NavMesh та NavMeshAgent, що дозволяють рухатися ворогу по конкретній території та допомогло для створення патруля.

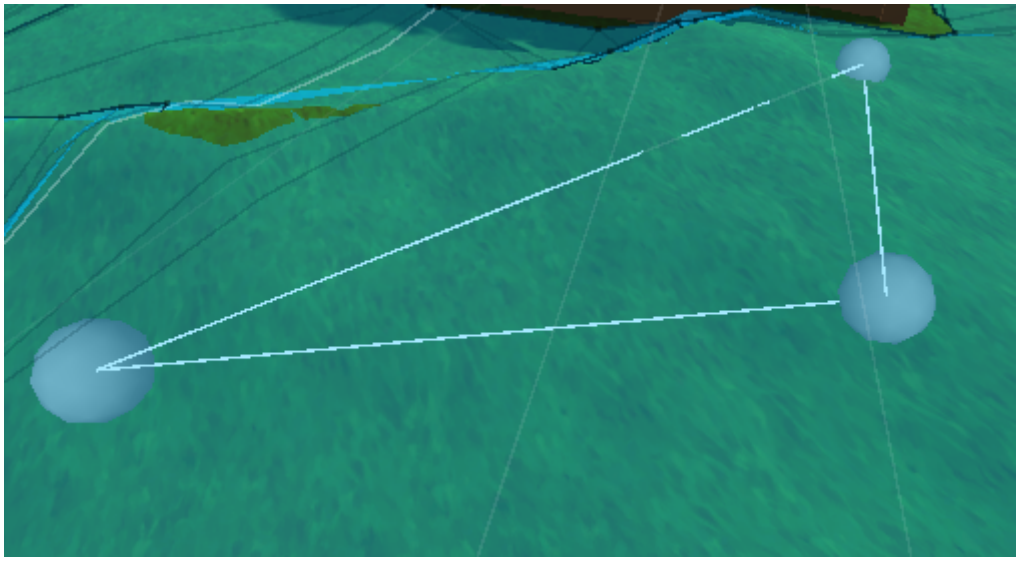

Рис. 3.19. Візуалізація патруля

*створено автором

Через оператор if було реалізовано механіку патруля. В разі, якщо персонаж заходить в зону патруля ворога, то ворог починає атакувати, в іншому випадку спрацьовує інший метод де ворог повертається на початкову позицію.

3.3.12. Scene Management

Для переходу між сценами було створено два портали. Для цього було створено скрипт Portal: коли гравець з тегом Player торкається порталу, починається перехід між сценами та ефект відбілювання на екрані - Fader. Після завершення ефект зупиняється. Після цього з'являється можливість переходити між двома порталами, з порталу A на A і з B на B. Щоб ефект Fader працював належним чином, було створено скрипт Fader, де прописано різницю в часі коли побіління має завершитись після переходу між рівнями. Приклад показано на рисунку 3.20. Більше про управління сценами [22].

Рис. 3.20. Портал на 1 сцені

*створено автором

3.3.13. Кат сцени

Для реалізації катсцен було використано Cinemachine Virtual Camera (це система управління камерами Unity, яка дозволяє легко створювати кінематографічні камери та камерну анімацію), Playable Director та Timeline. В самому рушії було створено 4 об’єкти: Intro Sequence, CM Reveal 1, CM Dolly 1, Dolly Track. Кожен з цих об'єктів відповідає за реалізацію кат-сцен. В Intro Sequence було використано Playable Director. Це дає змогу використовувати Timeline який управляє CM Reveal1, CM Dolly 1 та Dolly Track. Ці об'єкти самі мають віртуальну камеру Cinemachine (див. 3.21), з якої створюються катсцени. Більше про створення кат-сцен можна переглянути тут [23].

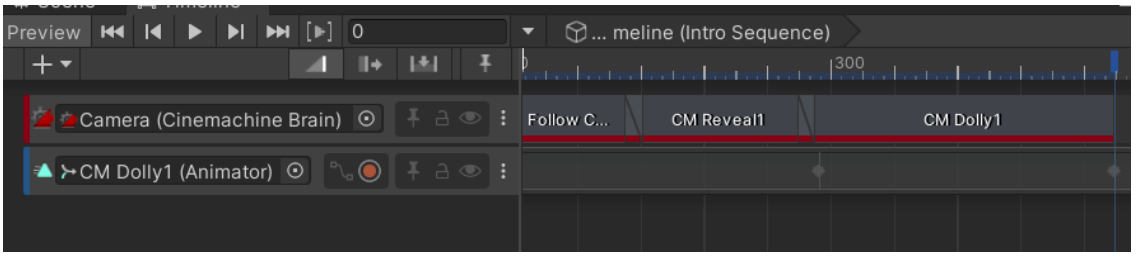

Рис. 3.21. Timeline

*створено автором

Використавши Timeline та створивши скрипт Cinematic Trigger було створено функціональну та візуальну версію кат-сцени. (див. рис. 3.22).

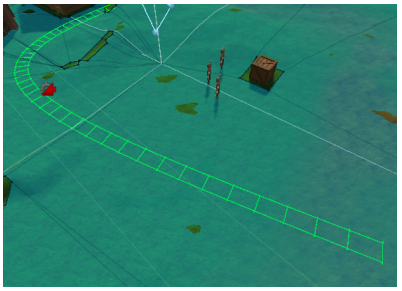

Рис. 3.22. Траєкторія кат-сцени

*створено автором

3.3.14. Base Stats

Base Stats - це розділ у налаштуваннях ворога та героя, де знаходяться різні характеристики, які були описані в скрипті Base Stats та Progression. Почнемо з того, що система прогресії була створена для того, щоб дозволити кастомізувати кожен рівень кожного персонажа. Наприклад, для прикладу є об’єкт гравця, який вимагає встановлення низки характеристик, які покращуватимуться з підвищенням вашого рівня. Приклад як це реалізовано (див. рис. 3.23). Три характеристики, які можна налаштувати, - це "сила", "досвід підвищення рівня" та "кількість зароблених балів". Крім того, Experience Reward та Experience To Level Up пов'язані між собою, оскільки чим більше балів досвіду(Experience Reward), тим краще буде рівень (Experience To Level Up). Завдяки цій структурі, ми маємо можливість змінювати цю систему під різні потреби.

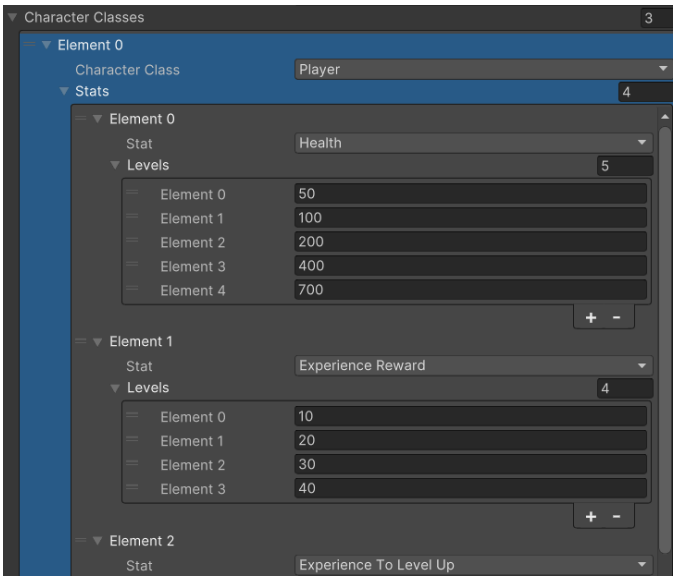

Рис. 3.23. Налаштування прогресу

*створено автором

Пізніше було додано показник шкоди. Він нічим не відрізняється від інших показників маючи такий самий алгоритм прогресу. Гравець розвиваючи свій рівень може покращити свій рівень шкоди. Приклад показано на рисунку 3.24.

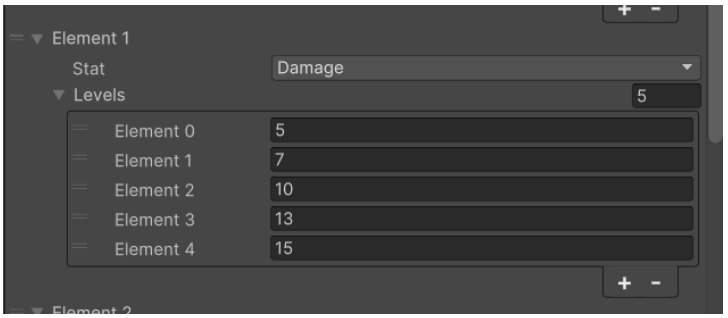

Рис. 3.24. Система прогресу шкоди

*створено автором

В лістингу 3.10 показано як ми додали шкоду до вже існуючих характеристик до скрипту Stat.cs. Також в майбутньому цей скрипт може наповнюватись новими характеристиками які будуть наповнювати систему прогресу.

```
public enum Stat
    {
        Health,
```

```
        ExperienceReward,
        ExperienceToLevelUp,
        Damage
    }
```

Лістинг 3.10.

* створено автором

3.3.15. Використання підібраної зброї

Для цього було створено скрипт WeaponConfig. Він дозволяє налаштовувати дані напряму в рушії. Наприклад: вибір руки, вибір базового префаба, шкоду від цієї зброї, дальність ураження та контролер, який підключає анімацію потрібного руху. Приклад показано на рисунку 3.25.

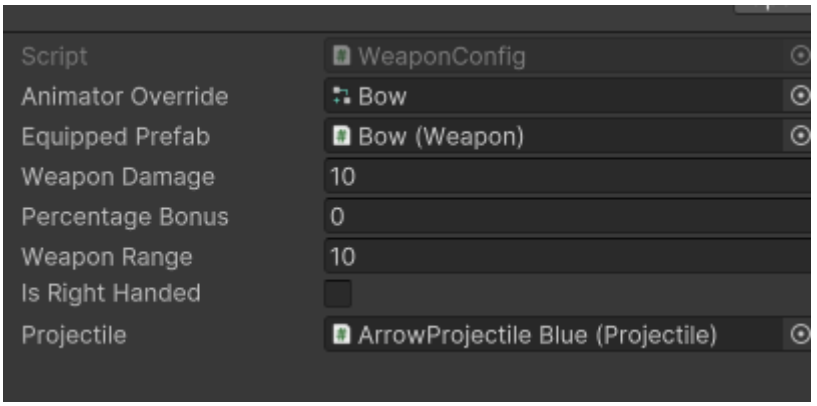

Рис. 3.25. Інтерфейс налаштувань

*створено автором

Загалом ми маємо 4 вида зброї: лук, вогняний шар, меч та рукопашний бій. Кожен з них має свої кастомізовані характеристики (див. рис. 3.26). Маючи готову зброю ми можемо в любий момент змінити: анімації, скрипт, нанесення шкоди, відсоток від надання шкоди, дальність пострілу або удару, вибір руки та візуальні ефекти.

Рис. 3.26. Scriptable Objects

*створено автором

Розглянемо один з прикладів розробки зброї. Для прикладу візьмемо вогняний шар. Для початку було створено новий Animator Override Controller де додано анімацію кидка вогняного шару. Приклад показано на рисунку 3.27.

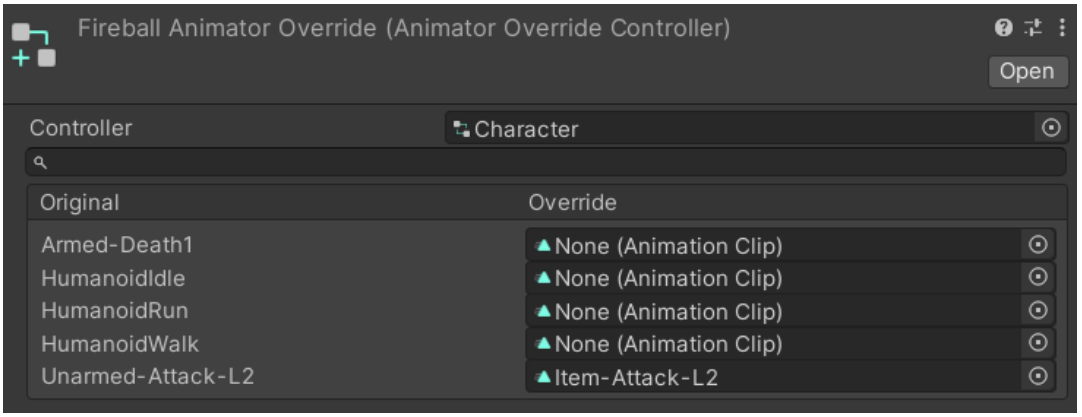

Рис. 3.27. Animator Override Controller

*створено автором

Також в Weapon config було кастомізовано шкоду яку наносить вогняний шар, бонусний відсоток, дальність пострілу та скрипт, завдяки якому вогняний шар функціонує. Для вогняного шара я використав безкоштовний ассет Simple FX - Cartoon Particles. Приклад показано на рисунку 3.28.

Рис. 3.28. Fireball

*створено автором

Після готової анімації та характеристик потрібно обрати модель, яка буде знаходитись на карті і яку персонаж зможе підібрати. Для цього було обрано квітку, яка буде з’являтися після зникнення кожні 5 секунд. Приклад показано на рисунку 3.29.

Рис. 3.29. Fireball Pickup

*створено автором

3.3.16. Ефекти для стріл

Кожен персонаж, що володіє луком, має колір стріл, залежно від того, ворог він чи союзник. Наприклад, вороги мають червоні стріли, а союзники - сині. Приклад показано на рисунку 3.30. Більше про використання інструмента для ефектів можна переглянути тут [24].

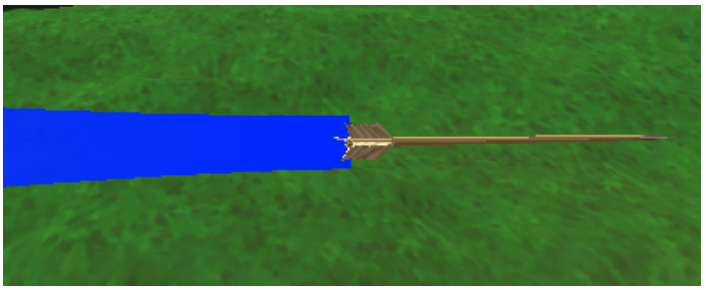

Рис. 3.30. Стріли союзника

*створено автором

Для того, щоб зробити цей ефект потрібно використати Trail Renderer, де можна міняти колір, змінювати перехід між кольорами, довжина ефекта, що вимірюється в секундах, ширину ефекту в точці розташування об'єкта, ширина ефекта на його кінці, Мінімальна відстань між точками привязки ефекта, увімкнути для самознищення об'єкта після бездіяльності протягом Time секунд.

Для створення цього ефекту потрібно скористатися Trail Renderer. Конструктор дозволяє змінювати колір, переходи між кольорами, тривалість ефекту (у секундах), ширину ефекту в точці розташування об'єкта, ширину в кінцевій точці ефекту, мінімальну відстань між опорними точками ефекту, а також самознищення об'єкта після певного періоду бездіяльності. Приклад показано на рисунку 3.31.

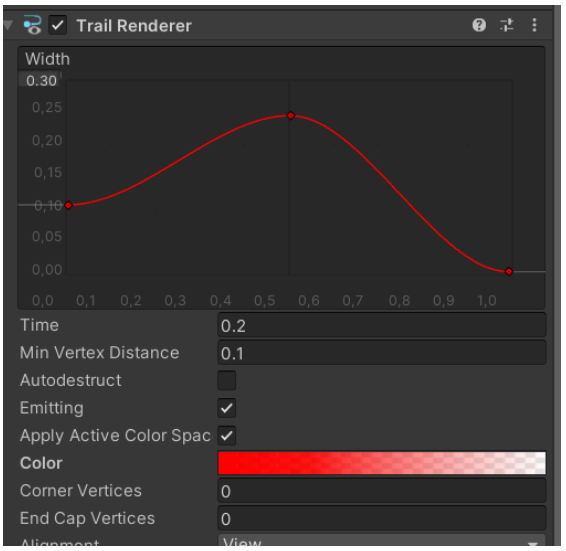

Рис. 3.31. Налаштування кольору

*створено автором

3.3.17. Система збереження

Для реалізації системи збереження було створено кілька скриптів, зокрема Saving Wrapper, Saving System, Saveable Entity та ISaveable, які зберігають переходи та прогрес між мапами. Saving Wrapper - дозволяє при натисканні конкретних клавіш зберегти, загрузити та видалити дані гри. В Saving System прописано функціонал всіх дій системи збереження. Saveable Entity створює унікальний ідентифікатор та показує це повідомленням після конкретних дій. ISaveable - це інтерфейс який дозволяє в інших класах захоплювати та відновлювати дані під час збереження та завантаження. Система збереження допомагає гравцеві зберегти свій прогрес, тим самим полегшити складність проходження. Для збереження прогресу потрібно натиснути S, для загрузки попереднього збереження L та для видалення збереження Delete. Приклад показано в лістингу 3.11. та за посиланням [25].

```
public void Save(string saveFile)
        {
            Dictionary<string, object> state = LoadFile(saveFile);
            CaptureState(state);
            SaveFile(saveFile, state);
        }
```

Лістинг 3.11. Приклад частини коду з Saving System

* створено автором

В лістингу 3.12. продемонстровано реалізацію кнопок в скрипті. Після кожного збереження на вашому комп’ютері створюється файл який зберія всю інформацію про важе збереження (див. рис. 3.32) і при потребі оновлює дані або видаляється.

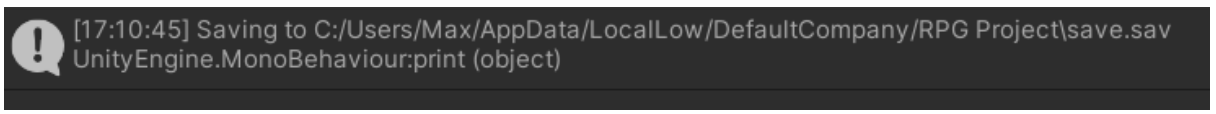

Рис. 3.32. Сповіщення збереження

*створено автором

```
private void Update() {
            if (Input.GetKeyDown(KeyCode.S))
            {
                Save();
            }
            if (Input.GetKeyDown(KeyCode.L))
            {
                Load();
            }
            if (Input.GetKeyDown(KeyCode.Delete))
            {
                Delete();
            }
        }
```

Лістинг 3.12. Приклад частини коду з Saving Wrapper

* створено автором

Скрипт Saveable Entity.cs дозволяє нам вписувати унікальний ідентифікатор який буде потім зберігатись в файлі збереження. В разі не заповнення ідентифікатора, скрипт створить автоматичний ідентифікатор який можна буде замінити в майбутньому. Приклад показано на рисунку 3.33.

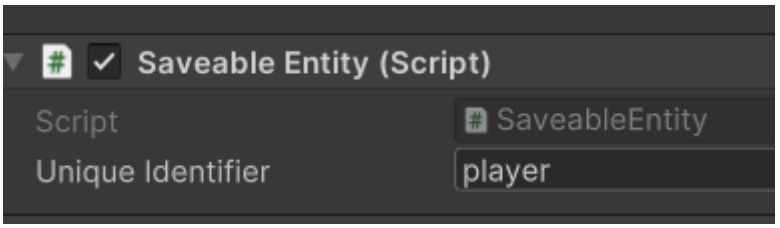

Рис. 3.33. Скрипт Saveable Entity

*створено автором

3.3.18. Інтерфейс

Для початку було створено анімацію з появою отриманої шкоди. Для цього ми створили в Animation звичайний текст та створили його появу як анімацію. В

скрипті було створено функцію яка висвітлювала число тіки потрібною стороною до камери. Приклад показано на рисунку 3.34.

Рис. 3.34. Анімація тексту

*створено автором

На рисунку 3.35 можна побачити, що на початку 0 секунді починається анімація маючи першу з трьох позначок. На 0,15 секунді створено анімацію підйому числа над головою ворога і на 0,30 секунді повертається в початкову позицію, таку саму як і на 0 секунді.

Рис. 3.35. Приклад анімації появлення тексту шкоди

*створено автором

В лістингу 3.13. показаний приклад як реалізована поява правльної кількості шкоди через скрипт.

```
public void Spawn(float damageAmount)
    {
        DamageText instance = Instantiate<DamageText>(damageTextPrefab,
transform);
```

```
        instance.SetValue(damageAmount);
    }
```

Лістинг 3.13. Скрипт DamageTextSpawner

* створено автором

Далі було створено полосу здоров’я для відображення залишку здоров'я. Спочатку було створено порожній об'єкт куди було додано скрипт Health Bar. Він дозволяв прикріпити canvas яким і є health bar реалізовуючи функціонал кольорового рядка. Приклад показано на рисунку 3.36. та за посиланням [26].

Рис. 3.36. Полоса здоров’я

*створено автором

Після цього, використовуючи інструмент Particle System було створено та налаштовано візуальний ефект під час попадання по ворогу. Під час створення було відредаговано: початковий розмір ефекта, тривалість ефекту, розмір та колір. Приклад показано на рисунку 3.37.

Рис. 3.37. Ефект отримання шкоди

*створено автором

До скрипту Player Controller була додана можливість показу курсора. В Cursor Mappings було реалізовано можливість вибору різних курсорів для різних потреб. Зараз в наявності 7 курсорів які відображаються, якщо навести на конкретний об’єкт. На рисунку 3.38. детально показано різноманітність курсорів та їх призначення. Також приклад використання можна переглянути за посиланням [27].Використаний ассет для курсорів було взято за посиланням [28].

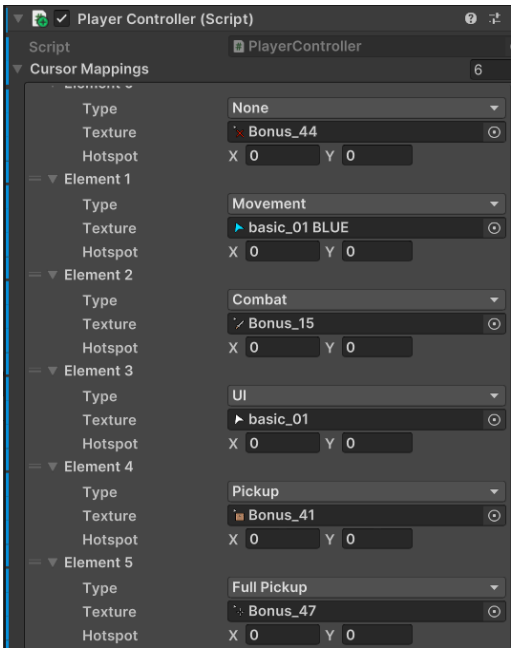

Рис. 3.38. Курсори

*створено автором

В класі PlayerController було створено метод Interact With Component для реалізації курсорів та взаємодії на ігрових об’єктах. Цей метод є дуже важливим оскільки завдяки йому ми маємо розуміння на який об’єкт ми зараз навелись і чи можемо ми з ним взаємодіяти. Приклад показно в лістингу 3.14.

```
private bool InteractWithComponent()
        {
            RaycastHit[] hits = RaycastAllSorted();
            foreach (RaycastHit hit in hits)
            {
                                        IRaycastable[]   raycastables   =
```

```
hit.transform.GetComponents<IRaycastable>();
                foreach (IRaycastable raycastable in raycastables)
                {
                    if (raycastable.HandleRaycast(this))
                    {
                        SetCursor(raycastable.GetCursorType());
                        return true;
                    }
                }
            }
            return false;
        }
```

Лістинг 3.14. Клас PlayerController

* створено автором

Наостанок було створено головне меню та меню паузи. Для цього було створено окрему сцену де ми мали змогу створювати весь інтерфейс меню, від кнопок до тексту. На рисунку 3.39 та 3.40 показано головне меню та меню паузи для якого ми обрали безкоштовний фон, дизайн та текст. Через скрипт Main Menu.cs та функцію OnClick в Unity було створено працюючу взаємодію між меню та грою. Приклад створення показано за посиланням [29]. Використаний шрифт тексту було взято за посиланням [30].

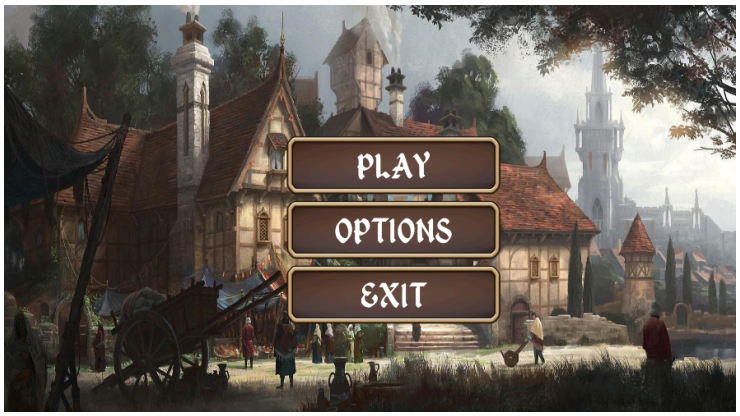

Рис. 3.39. Головне меню

*створено автором

Рис. 3.40. Меню паузи

*створено автором

3.3.19. Звукові ефекти

Звукові ефекти були додані для кращого подання бою та занурення в атмосферу гри. Для реалізації звукових ефектів було додано в скрипт Health можливість програвання звуку при конкретній дії. Тепер коли гравець отримує або задає шкоду ворогу, включається звуковий ефект Damage Sound та Die Sound. Приклад показано на рисунку 3.41. та приклад використання за посиланням [31]

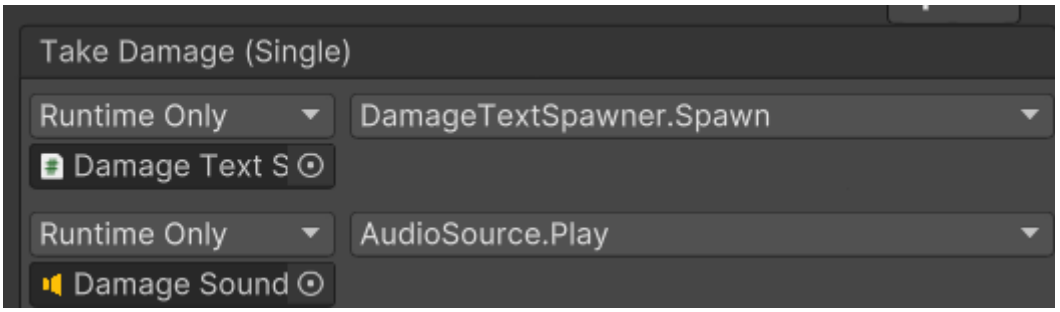

Рис. 3.41. Звукові Ефекти в Unity

*створено автором

3.3.20. UniTask

Було використано бібліотеку UniTask оскільки вона спростила роботу з асинхронним кодом і зробила розробку в Unity більш продуктивною та ефективною. Використання бібліотеки полегшило керування асинхронними завданнями, підвищуючи якість і швидкість розробки, особливо таким складним проєктом. В лістингу 3.15. використано частину коду Weapon Pickup та перероблено

використовуючи бібліотеку UniTask, яка підвищила якість та швидкість коду. Приклад використання UniTask показано за посиланням [32] та документацію за посиланням [33].

```
private async UniTaskVoid HideForSeconds(float seconds)
        {
            ShowPickup(false);
            await UniTask.Delay((int)(seconds * 1000));
            ShowPickup(true);
        }
```

Лістинг 3.15. Weapon Pickup

* створено автором

3.3.21. Система Діалогів

Для створення функціонуючої системи діалогів було створено різноманітні скрипти. Перш за все було створено Редактор діалогів де б ми могли створювати нові діалоги та поєднувати їх один з одним. Для цього були створені скрипти Dialogue.cs, DialogueEditor.cs, DialogueTrigger.cs, AIConvertsant.cs та PlayerConversant.cs. в класі Dialogue.cs прописана можливість створення діалога через інтерфейс рушія та можливість видаляти, з'єднувати та редагувати діалоги. DialogueTrigger.cs надає можливість запускати діалоги в конкретний момент в грі. AIConversant.cs дозволяє при наявності запускати діалог, натиснувши конкретною клавішею. PlayerConversant.cs надає можливість вибирати конкретний вибір в діалозі, починати і завершувати його. DialogueEditor.cs відповідає за зовнішній вигляд діалогів в режимі розробника та їхнє розташування. Створивши всі ці скрипти ми маємо вручно створений редактор діалогів з чудовим функціоналом, інтерфейс діалогів та взаємодія їх з навколишнім середовищем. Приклад показано на рисунку 3.42, 3.43 та за посиланням [34] .

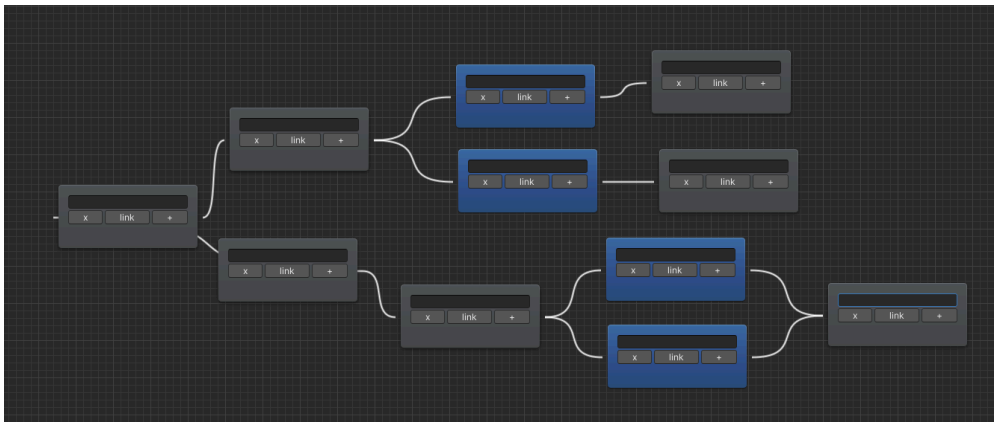

Рис. 3.42. Редактор діалогів

*створено автором

Рис. 3.43. Тестовий приклад діалогу в грі

*створено автором

Висновки до розділу 3

В цьому розділі було описано які засоби розробки було використано для створення гри та які системні та технічні вимоги потрібні були використані протягом роботи над грою.

Також було описано як гра була реалізована, описуючи такі етапи розробки: Unity Asset Store, Terrain tools, SkyBox, NavMesh, Click-To-Move, створення камери яка рухається за персонажем, додавання моделей персонажів, використання анімацій, бойова система, здоров’я, AI, Scene Management, кат сцени, Base Stats, використання підібраної зброї, ефекти для стріл, система збереження, інтерфейс , звукові ефекти, UniTask та система діалогів.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи, перш за все було проаналізовано ризики створення ігор та ігоргової індустрії, визначено мету розробки гри, описавши що для цього нам потрібно. Для розробки такої гри потрібно мати перш за все ідею гри та правильне планування розробки, після цього можна слідувати плану та створювати концепти, додавати нові ассети та покращувати їх різноманітними анімаціями, кольорами та функціональністю поєднуючи це з написанням скриптів та організацією проєкта в самому Unity.

У першому розділі було описано використані предметні середовища, деталі переваг кожного з середовищ, аналізовано конкурентів, їхній успіх на міжнародному ринку, SWOT-аналіз, постановку різних задач для проєкту та опис середньостатистичного гравця для цієї гри.

У другому розділі було розглянуто проблематику створення рольової гри, та які геймплейні складові потрібно для залучення гравців, проаналізовано основні механіки гри, для правильного планування та описано чому це важливо. Описано кожен з головних етапів нашої гри, які відповідають за конкретний функціонал та в поєднанні, доповнюють загальне наповнення гри.

У третьому розділі було описано які засоби розробки було використано для створення гри та які системні та технічні вимоги потрібні були використані протягом роботи над грою. Головним етапом розробки було описати реалізацію технічні етапи розробки такі як: Unity Asset Store, Terrain tools, SkyBox, NavMesh, Click-To-Move, створення камери яка рухається за персонажем, додавання моделей персонажів, використання анімацій, бойова система, здоров’я, AI, Scene Management, кат сцени, Base Stats, використання підібраної зброї, ефекти для стріл, система збереження, інтерфейс , звукові ефекти, UniTask та система діалогів.

Після виконання проєкту стало зрозуміло, що розробка ігор - це важка праця, яка в разі самостійної розробки, потребує правильного планування, набутих базових знань про використані програми та хорошої підготовки. Проте, незважаючи на складні випробування, старанна праця винагороджується успіхом і похвалою гравців. Весь проєкт був завантажений на GitHub [35].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Unity products [Електронний ресурс] URL: https://unity.com/products
   (дата звернення 02.12.2023)
2. Unity. Unity User Manual 2022.3 (LTS) [Електронний ресурс] URL: http://surl.li/uawyi
   (дата звернення 02.12.2023)
3. Scripting. Creating and Using Scripts [Електронний ресурс] URL: http://surl.li/uawyu
   (дата звернення 03.12.2023)
4. Jira Atlassian [Електронний ресурс] URL: https://jira.atlassian.com
   (дата звернення 12.10.2023)
5. Visual Studio Code. [Електронний ресурс] URL: https://code.visualstudio.com
   (дата звернення 03.12.2023)
6. Unity asset Store. [Електронний ресурс] URL: https://assetstore.unity.com
   (дата звернення 03.12.2023)
7. Diablo 4 [Електронний ресурс] URL: http://surl.li/uawyx
   (дата звернення 02.12.2023)
8. Baldur’s Gate 3 [Електронний ресурс] URL: http://surl.li/uawza
   (дата звернення 02.12.2023)
9. Hades [Електронний ресурс] URL: http://surl.li/uawzd
   (дата звернення 02.12.2023)
10. Terrain tools. Unity Manual URL: http://surl.li/uawzi
    (дата звернення 02.12.2023)
11. Medieval Windmill asset. [Електронний ресурс] URL: http://surl.li/uawzm
    (дата звернення 12.12.2023)
12. SkyBox. Sky Manual [Електронний ресурс] URL: http://surl.li/uawzq
    (дата звернення 10.12.2023)
13. SkyBox Series free asset. [Електронний ресурс] URL: http://surl.li/uawzw
    (дата звернення 12.12.2023)

14. Scripting API NavMesh [Електронний ресурс] URL: http://surl.li/uawzy (дата звернення 12.12.2023)
15. Scripting API NavMesh Agent. [Електронний ресурс] URL: http://surl.li/uaxab (дата звернення 12.12.2023)
16. Scripting API Click Event. [Електронний ресурс] URL: http://surl.li/uaxag (дата звернення 12.12.2023)
17. Scripting API Follow Camera. [Електронний ресурс] URL: http://surl.li/uaxaj (дата звернення 12.12.2023)
18. Characters assets and weapons. [Електронний ресурс] URL: https://syntystore.com (дата звернення 12.12.2023)
19. Buildings - RPG Poly Pack - Lite [Електронний ресурс] URL:http://surl.li/uaxam (дата звернення 12.12.2023)
20. Scripting API Animator. [Електронний ресурс] URL: http://surl.li/uaxaq (дата звернення 12.12.2023)
21. RPG Character Mecanim Animations asset. [Електронний ресурс] URL: http://surl.li/uaxar (дата звернення 12.12.2023)
22. Scripting API Scene Management. [Електронний ресурс] URL:http://surl.li/uaxat (дата звернення 12.12.2023)
23. Кат Сцена. Class Cinemachine Virtual Camera [Електронний ресурс] URL: http://surl.li/uaxay (дата звернення 12.12.2023)
24. Manual: Trail Renderer (Ефекти для стріл). [Електронний ресурс] URL: http://surl.li/uaxbb (дата звернення 12.12.2023)
25. Saving and loading system. [Електронний ресурс] URL: http://surl.li/uaxbi (дата звернення 15.02.2024)
26. Health Bar. [Електронний ресурс] URL: http://surl.li/uaxbm (дата звернення 16.03.2024)
27. Custom Cursor with Input System. [Електронний ресурс] URL: http://surl.li/uaxbp (дата звернення 18.03.2024)

28. Pixel Cursor. [Електронний ресурс] URL: http://surl.li/uaxbt
(дата звернення 18.03.2024)
29. Main Menu. [Електронний ресурс] URL: http://surl.li/uaxbx
(дата звернення 19.03.2024)
30. Fonts. [Електронний ресурс] URL: http://surl.li/uaxbz
(дата звернення 12.12.2023)
31. Sounds and music. Sound Effects. [Електронний ресурс] URL: http://surl.li/uaxcb
(дата звернення 11.03.2024)
32. How to make async operations properly in Unity. UniTask. [Електронний ресурс] URL: http://surl.li/uaxcd
(дата звернення 15.03.2024)
33. UniTask documentation GitHub. [Електронний ресурс] URL: https://github.com/Cysharp/UniTask
(дата звернення 15.03.2024)
34. Dialogue System (Editor). [Електронний ресурс] URL: http://surl.li/uaxcf
(дата звернення 10.03.2024)
35. GitHub project. [Електронний ресурс] URL: https://github.com/Max355/RPG-Project
(дата звернення 01.12.2023)

ДОДАТОК

Fighter.cs - метод AttackBehaviour спрямовує об’єкт атаки на ціль, потім перевіряє, чи минув достатній час між атаками, і якщо так, викликає атаку і оновлює час до наступної атаки.

```
private void AttackBehaviour()
        {
            transform.LookAt(target.transform);
            if (timeSinceLastAttack > timeBetweenAttacks)
            {
                TriggerAttack();
                timeSinceLastAttack = 0;
            }
        }
```

Projectile.cs - метод LaunchProjectile відповідає за створення і запуск снаряду з визначеними параметрами, такими як ціль, точка випуску і потенційна шкода.

```
public void LaunchProjectile(Transform rightHand, Transform leftHand,
Health target, GameObject instigator)
        {
            Projectile projectileInstance = Instantiate(projectile,
GetTransform(rightHand, leftHand).position, Quaternion.identity);
            projectileInstance.SetTarget(target, instigator, weaponDamage);
        }
```

CinematicTrigger.cs - коли гравець торкається тригера, якщо це відбувається вперше, відтворюється кінематографічна сцена, і подальше торкання тригера не призведе до повторного відтворення сцени.

```
public class CinematicTrigger : MonoBehaviour
    {
        bool alreadyTriggered = false;
        private void OnTriggerEnter(Collider other)
        {
            if(!alreadyTriggered && other.gameObject.tag == "Player")
            {
                alreadyTriggered = true;
                GetComponent<PlayableDirector>().Play();
            }
        }
    }
```

PatrolPath.cs - цей клас дозволяє візуалізувати та працювати з маршрутом патрулювання, розташованому у вигляді дочірніх об'єктів у батьківському об'єкті PatrolPath

```
public class PatrolPath : MonoBehaviour
    {
        const float waypointGizmoRadius = 0.3f;

        private void OnDrawGizmos() {
            for (int i = 0; i < transform.childCount; i++)
            {
                int j = GetNextIndex(i);
                Gizmos.DrawSphere(GetWaypoint(i), waypointGizmoRadius);
                Gizmos.DrawLine(GetWaypoint(i), GetWaypoint(j));
            }
        }

        public int GetNextIndex(int i)
        {
            if (i + 1 == transform.childCount)
            {
                return 0;
            }
            return i + 1;
        }

        public Vector3 GetWaypoint(int i)
        {
            return transform.GetChild(i).position;
        }
    }
```

CameraFacing.cs - об'єкт завжди повернутий в тому ж напрямку, що й головна камера.

```
public class CameraFacing : MonoBehaviour
    {
        void LateUpdate()
        {
            transform.forward = Camera.main.transform.forward;
        }
    }
```

DialogueNode.cs - дозволяє запускати його тільки тільки під час роботи у редакторі Unity та потрібні для редагування діалогу.

```
#if UNITY_EDITOR
        public void SetPosition(Vector2 newPosition)
        {
            Undo.RecordObject(this, "Move Dialogue Node");
            rect.position = newPosition;
        }

        public void SetText(string newText)
        {
            if (newText != text)
            {
                Undo.RecordObject(this, "Update Dialogue Text");
                text = newText;
            }
        }

        public void SetPlayerIsSpeaking(bool newIsPlayerSpeaking)
        {
            Undo.RecordObject(this, "Changed Dialogue Node Speaker");
            isPlayerSpeaking = newIsPlayerSpeaking;
            EditorUtility.SetDirty(this);
        }

        public void AddChild(string childID)
        {
            Undo.RecordObject(this, "Add Dialogue Link");
            children.Add(childID);
        }

        public void RemoveChild(string childID)
        {
            Undo.RecordObject(this, "Remove Dialogue Link");
            children.Remove(childID);
        }
#endif
```

Health.cs - цей клас виконує всі дії коли об’єкт помирає, винагороджує гравця досвідом за вбивство ворога та відновлює здоров’я об’єкта після певної дії.

```
 private void Die()
```

```
        {
            if (isDead) return;

            isDead = true;
            GetComponent<Animator>().SetTrigger("die");
            GetComponent<ActionScheduler>().CancelCurrentAction();
        }

        private void AwardExperience(GameObject instigator)
        {
            Experience experience = instigator.GetComponent<Experience>();
            if (experience == null) return;


experience.GainExperience(GetComponent<BaseStats>().GetStat(Stat.Experience
Reward));
        }

        private void RegenerateHealth()
        {
           float regenHealthPoints  =
GetComponent<BaseStats>().GetStat(Stat.Health) * (regenerationPercentage /
100);
           healthPoints = Mathf.Max(healthPoints, regenHealthPoints);
        }
```

Progression.cs - дозволяє легко управляти і зберігати дані прогресії для різних класів персонажів у грі.

```
[CreateAssetMenu(fileName = "Progression", menuName = "Stats/New
Progression", order = 0)]
    public class Progression : ScriptableObject
    {
        [SerializeField] ProgressionCharacterClass[] characterClasses =
null;

        Dictionary<CharacterClass, Dictionary<Stat, float[]>> lookupTable =
null;

        public float GetStat(Stat stat, CharacterClass characterClass, int
level)
        {
            BuildLookup();
```

```
            float[] levels = lookupTable[characterClass][stat];
            if (levels.Length < level)
            {
                return 0;
            }

            return levels[level - 1];
        }

        public int Getlevels(Stat stat, CharacterClass characterClass)
        {
            BuildLookup();
            float[] levels = lookupTable[characterClass][stat];
            return levels.Length;
        }

        private void BuildLookup()
        {
            if (lookupTable != null) return;

            lookupTable = new Dictionary<CharacterClass, Dictionary<Stat,
float[]>>();

            foreach (ProgressionCharacterClass progressionClass in
characterClasses)
            {
                var statLookupTable = new Dictionary<Stat, float[]>();

                foreach (ProgressionStat progressionStat in
progressionClass.stats)
                {
                    statLookupTable[progressionStat.stat] =
progressionStat.levels;
                }

                lookupTable[progressionClass.characterClass] =
statLookupTable;
            }
        }

        [System.Serializable]
        class ProgressionCharacterClass
        {
```

```
        public CharacterClass characterClass;
        public ProgressionStat[] stats;
        //public float[] health;
    }

    [System.Serializable]
    class ProgressionStat
    {
        public Stat stat;
        public float[] levels;
    }

}
```