

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)
на тему:

«Розробка візуальної та UI частин для гри у жанрах
Metroidvania і Soulslike»

Виконала студентка групи КН-20-1:
ДЯЧЕНКО Валерія Володимирівна

Керівник роботи:
МАРЧУК Галина Вікторівна

Рецензент:
КУЗЬМЕНКО Олександр Вікторович

Житомир – 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СПЕЦІАЛЬНІСТЬ 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

Марина ГРАФ

«25» січня 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)

Здобувач вищої освіти: ДЯЧЕНКО Валерія Володимирівна

Керівник роботи: МАРЧУК Галина Вікторівна

Тема роботи: «Розробка візуальної та UI частин для гри у жанрах Metroidvania і Soulslike»,
затверджена наказом закладу вищої освіти від «25 січня» 2024 р., №22/с

Термін здачі закінченої роботи: «06» червня 2024 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження): використання графічних інструментів та ігрового двигуна Unity для створення візуальної/UI частин гри у жанрі Metroidvania та Soulslike.

Консультанти з кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв
1	Марчук Галина Вікторівна	26.03.2024	15.04.2024
2	Марчук Галина Вікторівна	15.04.2024	16.05.2024
3	Марчук Галина Вікторівна	16.05.2024	20.05.2024

Календарний план

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд, аналіз доступного інструментарію та аналогічного ПЗ.	15.09.23 – 25.09.23 26.09.23 – 01.10.23 02.10.23 – 05.10.23	Виконав
2	Створення схем та структури додатку.	06.10.23 – 06.11.23	Виконав
3	Розробка програмної частини.	07.12.23 – 21.04.24	Виконав
4	Тестування.	22.04.24 – 15.05.24	Виконав
5	Попередній захист.	06.06.2024	Виконав

Здобувач вищої освіти Валерія ДЯЧЕНКО

Керівник Галина МАРЧУК

РЕФЕРАТ

Кваліфікаційна робота складається з програмного додатку гри та пояснювальної записки

Пояснювальна записка до кваліфікаційної роботи складається з реферату, вступу, основної частини, загального висновку, списку використаної літератури та додатку.

Текстова частина викладена на 66 сторінках друкованого тексту.

Пояснювальна записка має 37 сторінок додатків. Список використаних джерел містить 22 найменування, і займає 2 сторінки.

В роботі наведено 59 рисунків та 6 таблиць. Загальний обсяг роботи – 103 сторінки.

Ключові слова: ГРА, METROIDVANIA, SOULSLIKE, UNITY, 2D, ПЛАТФОРМЕР, UI, URP, ASEPRITE, ГЕЙМДИЗАЙН.

					КН.КР.Б – 122 – 24 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	«Розробка візуальної та UI частин для гри у жанрах Metroidvania і Soulslike» Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		Дяченко В.В.						
Керівник		Марчук Г. В					2	103
Реценз.		Кузьменко О.В.				Житомирська політехніка, група КН-20-1		
Н. Контр.								
Затверд.		Граф М.С.						

ABSTRACT

The graduation thesis of the bachelor consists of a software application of the game and an explanatory note.

An explanatory note to the final thesis consists of an abstract, introduction, main part, general conclusion, a list of used literature and an appendix.

The text part is laid out on 66 pages of printed text.

The explanatory note has 37 pages of annexes. The list of used sources contains 22 names and occupies 2 page.

The work contains 59 figures and 6 tables. The total volume of work is 103 pages.

Keywords: GAME, METROIDVANIA, SOULSLIKE, UNITY, 2D, PLATFORMER, UI, URP, ASEPRITE, GAME-DESIGNE.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г. В				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ	
ВИРІШЕННЯ ЗАДАЧІ	9
1.1 Постановка задачі	9
1.2 Аналіз існуючого програмного забезпечення за тематикою кваліфікаційної роботи	10
1.3 Концепт-документ (Concept Document)	15
1.4 Обґрунтування вибору інструментальних засобів	18
Висновки до першого розділу	20
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО	
ЗАБЕЗПЕЧЕННЯ	21
2.1 Проєктування архітектури ігрового додатку	21
2.2 Розробка інтерфейсу користувача	29
2.3 Розробка програмної частини додатку	33
2.4 Реалізація візуальних ефектів, графічних елементів та анімацій	44
Висновки до другого розділу	53
РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО	
ТЕСТУВАННЯ	54
3.1 Порядок встановлення та налаштування ігрового додатку	54
3.2 Інтерфейс та порядок роботи	54
3.3 Тестування роботи ігрового додатку	62
Висновки до третього розділу	64
ВИСНОВКИ	65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	66
ДОДАТКИ	68

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

GUI	Graphic User Interface, графічний інтерфейс, елементи за допомогою яких користувач взаємодіє з програмою, отримує данні
ПЗ	Програмне забезпечення
HUD	Head-Up-Display , частина графічного інтерфейсу користувача для відображення різноманітної ігрової інформації без виклику додаткових вікон або меню
ПП	Програмний продукт
RPG	Role-Playing Game, жанр відеоігор, де основна частина процесу проходження полягає в управлінні персонажем, який досліджує світ, виконує різноманітні завдання та розвивається, слідуючи сюжету
AAA проект	Умовна підмножина відеоігор, створюваних й розповсюджуваних середніми й великими видавництвами, що зазвичай мають більше коштів на розробку й рекламу

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми. За останні десятиріччя ігрова індустрія безперечно зазнала змін і без перебільшень досягла свого розквіту. Попит на ігрові програмні продукти зріс завдяки доступності серед людей, що забезпечено швидким розвитком технологій. Кожен другий користувач може активно використовувати ігрові програмні продукти користуючись навіть звичайним мобільним телефоном. Разом з тим, карантинні обмеження, запроваджені у зв'язку з пандемією, значно змінили пейзаж індустрії в цілому. За цей період часу, коли багато людей були змушені залишатися вдома, ігрова індустрія зазнала ще більшого піднесення.

Ігри – це не лише спосіб скоротати час без користі, це окремий світ зі своїми порядками і правилами, а у сучасних реаліях досить актуальний варіант для розслаблення та підтримання морального духу.

Саме тому не дивно, що наразі існує безліч ігрових жанрів, які задовольняють смак та примхи кожного користувача. Створюваний програмний продукт є комбінацією таких жанрів як Metroidvania та Soulslike. Вони поєднують в собі цікаві механіки та атмосферу, що вкрай приваблює любителів важких випробувань у віртуальних світах.

Мета кваліфікаційної роботи - створення ігрового програмного продукту і таких його складових як: головне та внутрішньо-ігрове меню, інтерфейс для взаємодії з користувачем та передачею необхідної інформації, локації з пропрацьованим оточенням, а також візуальний стиль і анімації ворогів. Проект буде включати технології поліпшення візуальної частини, зручний інтерфейс та власні графічні елементи.

Об'єктом дослідження є методи та інструменти розробки програмного забезпечення для геймдевелопменту. Дослідження можливостей та особливостей ігрового рушія Unity та фреймворків для роботи з активними графічними елементами та візуальними ефектами.

Предметом дослідження є розробка візуальної та UI (англ. User Interface) складових для ігрового додатку, що охоплює створення та оптимізацію

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

графічних елементів, визначення способів їх взаємодії з користувачем та розробку ефективного і зрозумілого інтерфейсу в межах визначених жанрів гри.

В рамках теми кваліфікаційної роботи було опубліковано тези:

Марчук Г.В., Дяченко В.В. Важливість візуальної частини ігор. Комп'ютерні технології: інновації, проблеми, рішення: Тези VI Всеукраїнської науково-технічної конференції, 29-30 листопада 2023 року. Житомир: «Житомирська політехніка», 2023. С.153-155.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1 Постановка задачі

Основною метою розробки візуальної та UI частин для гри у жанрах Metroidvania [1] і Soulslike [2] є створення захоплюючого імерсивного досвіду для гравців, який поєднує в собі елементи пригодницьких випробувань, складних головоломок та стратегічної боротьби. Тобто створюваний проєкт буде збалансованою комбінацією двох жанрів, що мають власні особливості та механіки.

Жанр Metroidvania з'явився шляхом об'єднання різних ідей, що в основі своїй були не лінійні та відрізнялись від інших ігор свого часу. Гравець повинен був повертатися туди, де бував, і отримувати постійно діючі призи — на противагу класичним рівневим платформерам, де призи лише тимчасові. Саме унікальність в свій час подарувала жанру не аби яку славу і зараз вона посіла своє місце серед значущих представників ігрової індустрії. Навіть великі AAA - проєкти позичають дані ідеї та механіки. Більшість нових компаній віддають перевагу створенню невеликих інді - ігор у цьому жанрі для створення іміджу та імені.

Жанр Soulslike відзначається своєю високою складністю та вимагає від гравців сконцентрованості, уваги та технічних навичок. Ці ігри часто відображають атмосферні, але жорстокі світи, де кожна помилка може призвести до втрати прогресу або навіть головного героя. Основні елементи жанру включають стратегічну боротьбу, складність управління та велику кількість секретів і пасток, які чекають на гравця на кожному кроці.

Розробка програмного забезпечення має включати не лише естетичний аспект, але і зручність, і ефективність при використанні кінцевим користувачем. Створювані елементи будуть включати усі інтерактивні елементи, графічні та візуальні аспекти і відповідно кодову реалізацію для їх

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

коректної роботи. Тож, у підсумку, для реалізації поставленої задачі необхідно виконати наступні етапи:

1. *Створення внутрішньоігрових меню:*
 - Аналіз існуючих аналогів.
 - Визначення структури меню, його взаємодії та вигляду.
 - Проєктування та реалізація кодової частини у Unity.
2. *Розробка користувацького інтерфейсу (UI):*
 - Проєктування інтерфейсу, забезпечуючи зручність для гравців.
 - Розміщення та стилізація ігрових елементів на екрані.
3. *Візуальні ефекти:*
 - Аналіз існуючого інструментарію для роботи з пост-обробкою.
 - Практичне використання ефектів на сцені.
 - Використання анімацій, переходів, системи частинок (particles) та інших засобів для покращення візуального сприйняття.
4. *Створення графічних елементів:*
 - Забезпечення відповідності загальному стилю гри та концепції Metroidvania і Soulslike.
 - Розробка та інтеграція графічних елементів, таких як кнопки, іконки, оточення, фонові зображення, персонажі та інші декоративні складові.

Дані пункти охоплюють широкий спектр завдань – від розробки структури і кодової частини меню до створення ефектів, спрямованих на покращення візуального сприйняття. Органічна взаємодія усіх компонентів та правильний аналіз особливостей жанрів дозволить створити високоякісний та привабливий ігровий інтерфейс.

1.2 Аналіз існуючого програмного забезпечення за тематикою кваліфікаційної роботи

Перед початком проєктування необхідно чітко визначити, які складові включає в себе інтерфейс та яких правил слід дотримуватись, щоб забезпечити

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

ефективність та зручність для користувача. Планування елементів, які будуть відображені на екрані через HUD (англ. Head-Up-display) [3], а також урахування їх логічної та естетичної взаємодії, визначить якість та сприйняття інтерфейсу гравцем.

GUI (англ. Graphical User Interface) [4], або графічний інтерфейс користувача - це спосіб взаємодії з комп'ютерною програмою за допомогою графічних об'єктів, таких як кнопки, панелі, вікна, меню та інші візуальні елементи.

Проектування інтерфейсів для ігор передбачає виділення одного екрану для кожного завдання. Розробники уникають "приховування" інших функцій, оскільки це призводить до значних ризиків непорозуміння. Чітке визначення "одного завдання" також створює труднощі, оскільки практично будь-який процес включає кілька етапів.

Важливо організувати простір екрану так, щоб геймер міг інтуїтивно використовувати інтерфейс гри, розуміючи призначення кожного елемента. Це стосується усіх типів інтерфейсів: HUD і меню.

Розглянемо декілька існуючих представників жанру для кращого розуміння як інші розробники вирішують питання розташування та взаємодії елементів. Занотуємо успішні приклади та кращі практики, щоб вдосконалити свій власний дизайн інтерфейсу гри. Першим із них буде «Blasphemous» [5] - Metroidvania від студії The Game Kitchen, випущена у вересні 2019 року. Відзначається своєю темною атмосферою та унікальним візуальним стилем, який поєднує ретро-пиксельну графіку з макабре та релігійними мотивами. В основному гра отримала позитивні відгуки від гравців та критиків за свою атмосферу, візуальний наратив та зручність.

Наступний представник жанру - «Hollow Knight» [6] розроблений і виданий незалежною студією Team Cherry у лютому 2017 року. В даному ігровому продукті гравець виступає у ролі безіменного лицаря-комахи, який відправляється на дослідницький похід по підземному світу Гнилосо́рдія. Під час подорожі він зустрічає різноманітних персонажів, збирає зброю та навички,

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

і бореться з загрозами, які оточують його. Завдяки вражаючій музичній атмосфері та запаморочливому дизайну рівнів, «Hollow Knight» здобула велику популярність серед геймерів і критиків. Окрім цього також отримала визнання за свій детально вивершений світ та глибокий геймплей.

Одним із основоположників складних RPG (англ. Role-Play game) вважається наступна розглянута серія ігор – «Dark Souls» [7], розроблених і виданих компанією FromSoftware. Перша гра «Dark Souls», була випущена у 2011 році, а її продовження, II та III частини, у 2014 і 2016 відповідно. Серія відома своїм високим рівнем виклику та глибокими механіками геймплею. Гравці відправляються в темні та жорстокі світи, де кожен крок може бути останнім. Битви з ворогами вимагають від гравців точності, стратегії та терпіння. Крім того, гра пропонує складні головоломки та велику свободу в облаштуванні персонажа та експлорації світу.

Отримавши загальне розуміння про наведені проекти необхідно детальніше розглянути конкретні елементи кожного з них.

Головне меню. Як можна побачити (рис. 1.1 - 1.3) основні елементи головного меню – кнопки, що виконують відповідні дії. Починають гру, запускають продовження, показують досягнення тощо. Візуальний стиль – простий, виконаний у темних/понурих тонах. Всю увагу на себе бере фонове зображення, що може мати свій візуальний стиль. В першому варіанті це головний герой, у другому та третьому виділена назва гри.



Рисунок 1.1 - Головне меню гри Blasphemous

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		12



Рисунок 1.2 - Головне меню гри Dark Souls



Рисунок 1.3 - Головне меню гри Hollow Knight

Меню паузи. Меню паузи також буває декількох типів (рис. 1.4 - 1.6). Це може бути звичайне вікно, на кшталт головного, в іншому випадку воно стає не лише інструментом навігації, але і необхідною складовою ігрового процесу. Його функціонал розширюється, включаючи статистику, систему прокачки, інвентар та інші важливі елементи, схожі на те, як це реалізовано в грі «Dark Souls». Такий підхід робить меню паузи більш універсальним та зручним для гравців, а також вирішує проблему «одного завдання» згадану раніше. Проте у вищезгаданій хардкорній серії відсутня пауза взагалі. Що в свою чергу не дуже характерно для жанру Metroidvania.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		13



Рисунок 1.4 - Меню паузи гри Blasphemous



Рисунок 1.5 - Меню гри Dark Souls



Рисунок 1.6 - Меню паузи гри Hollow Knight

Head-Up Display. HUD ігор не перевантажений та зазвичай містить коротку інформацію про статистику гравця у верхній частині екрану (рис. 1.7- 1.9). Туди входить – здоров'я, мана, енергія, лікувальні предмети, валюта тощо.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		14



Рисунок 1.7 - HUD гри Blasphemous



Рисунок 1.8 - HUD гри Dark Souls



Рисунок 1.9 - HUD гри Hollow Knight

1.3 Концепт-документ (Concept Document)

1.1 Загальний опис продукту

Створюваний ігровий продукт «Dethrone» є грою жанру Metroidvania, що відзначається використанням деяких механік, натхнених серією «Dark Souls».

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		15

Гра пропонує гравцям глибокий і загадковий світ, де кожен крок може бути вирішальним. Повна назва проєкту – «Dethrone». Включає у себе:

- дві повноцінні локації, які мають бос-битви;
- 10 різних видів ворогів, кожен з яких має свій унікальний дизайн;
- розгалужену систему активних та пасивних навичок;
- різноманітні стилі володіння зброєю, що дає гравцям велику

свободу в налаштуванні свого персонажа.

Окрім цього, гра має досить заплутаний всесвіт, що включає взаємодію з непередбачуваними персонажами, які розкривають нові аспекти історії та допомагають гравцеві у його пригоді.

1.2 Жанр та аудиторія

Продукт «Dethrone» націлений на аудиторію, що включає фанатів ігрових жанрів, таких як Metroidvania та Soulslike. Основною групою користувачів є підлітки віком 16 років та старше, які прагнуть не лише динамічного геймплею, але і глибокого занурення в історію гри. Ринок продукту охоплює Європу та США, де англійська мова буде основною для взаємодії з гравцями.

Таблиця 1.1

Портрет потенційного клієнта

Показник	Опис
Вік	18-40 років
Стать	Будь-яка (проте переважно чоловіки)
Географія	Україна та країни Європи, Америки (англомовні користувачі)
Спосіб життя	Помірний, значна частина часу проводиться за комп'ютером, активний в ігровому середовищі
Соціальний клас	Середній і вище
Переконання	Відкритий до нових ідей, цінує творчість і інновації
Інтереси	Геймінг, віртуальні світи, фантастика/фентезі, техніка

Показник	Опис
Взаємодія з продуктом	Активно досліджує різноманітні віртуальні світи, взаємодіє з ігровими спільнотами

Як і кожен ігровий додаток «Dethrone» має викликати зацікавленість користувачів за рахунок сюжету, візуальної складової і головне – механік. Для жанру Metroidvania (і відповідного ігрового додатку) притаманні:

– *Нелінійність та дослідження*: відкриті рівні та знаходження секретів, обов’язкове повернення у попередні локації для відкриття всього контенту.

– *Прокачування персонажа*: різні механіки прокачування зброї, талантів, активних/пасивних навичок.

– *Збалансованість та контроль*: збалансовані вороги та головний персонаж, достатня к-ть предметів, ключових подій і секретів на локації. Зручне керування з можливістю персонального налаштування

1.3 Основні функції та унікальна пропозиція гри

Створюваний ігровий додаток повинен надавати гравцю можливості: дослідження (відкриття карти цілісного світу), прокачування персонажу (навички, зброя), унікальну бойову систему і різні типи ворогів (комбо), інтерактивність зі світом (діалоги). Таким чином додаток створить виклик для гравців будь-якого рівня навичок та надасть унікальний ігровий досвід.

1.4 Платформа

Продукт представляє із себе офлайн 2D гру для ПК, яку перед початком роботи необхідно встановити на жорсткий диск. Вона не вимагає високої потужності як сучасні AAA проекти, тому для гри можна обрати комп’ютер середнього класу. Чотирьохядерний або шестиядерний процесор з базовою частотою більше 2.5 ГГц, відеокарти серій NVIDIA GTX 1050 або AMD Radeon RX 560, 8 ГБ оперативної пам’яті. Щодо зберігання, можна обрати як HDD так і SSD.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

1.4 Обґрунтування вибору інструментальних засобів

Візуальний стиль з'єднує усі компоненти, що бачить гравець: персонажів, ігровий світ, анімацію, інтерфейс та меню. Вони повинні бути як ідентичними, так і унікальними, щоб додаток впізнавався лише по шматочку інтерфейсу. У той же час, потрібно розуміти, що різні типи ігор вимагають унікального підходу до створення світу, та різного типу дизайну. Тому що графіка визначає емоційний вплив на гравця та відображає сутність гри [8].

Залежно від жанру, обирається найкращий стиль, що передасть не тільки атмосферу, але і головну думку всього проєкту. У 2D іграх вони бувають абсолютно різні: художні (картунстичні), піксельні, сітчасті або скелетні. Для жанру Metroidvania добре підходить піксельний стиль, що зазвичай намальовано від руки [8].

Зазначимо, що саме піксельний стиль найкраще підходить для створюваного проєкту, через його ідеальне походження, що бере коріння у класичних ретро-іграх минулого десятиліття, які також стали основою для жанру Metroidvania [8].

Якщо більш детально, то піксельний стиль - це вид графіки, де зображення створюються шляхом розміщення пікселів на растровій сітці. Кожен піксель представляє окремий колір, і вони розташовані поруч один з одним, утворюючи зображення. Зображення в піксельному стилі часто мають низьку роздільну здатність та потребують менше ресурсів і обчислювальної потужності, що робить їх ідеальними для розвитку ігор в інді-гейм-розробці [8].

Існує безліч програм, що використовуються для створення піксельної графіки, тому провівши аналіз кожної з них результат було зведено до порівняльної таблиці 1.2.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

Таблиця 1.2

Можливості та особливості програм

Назва Програми	Основні Функції	Підтримка Платформ	Ціна	Особливості
Aseprite	Редагування та анімація піксельної графіки	Windows, macOS, Linux	Платна, пробний період	Висока ефективність анімації, різноманітні інструменти
Pxel Edit	Редагування та анімація піксельної графіки	Windows, macOS	Платна, пробний період	Легкість використання, інтеграція з різними платформами
GraphicsGale	Редагування та анімація піксельної графіки	Windows	Безкоштовна, Про версія - платна	Спеціалізований інтерфейс для анімації, функції обробки палітри
Pixaki (iPad)	Редагування та анімація піксельної графіки	iPad	Платна	Адаптована до роботи на планшеті, інтуїтивний інтерфейс
Piskel	Онлайн-редактор для створення та анімації піксельної графіки	Веб	Безкоштовна	Доступ з будь- якого пристрою, колективне редагування
Pro Motion NG	Редагування та анімація піксельної графіки	Windows	Платна	Інструменти для професійного дизайну, велика кількість шарів

У підсумку, враховуючи основні характеристики різних програм для, можна зазначити, що майже всі з них є платними, а безкоштовні альтернативи часто обмежені функціоналом. У зв'язку з цим, вибір спрямується на Aseprite [9, 10] як на найбільш відому та зручну програму для нових користувачів на сучасному ринку редагування піксельної графіки. Її висока ефективність у створенні анімацій, підтримуваний формат у Unity та різноманітні інструменти роблять Aseprite оптимальним вибором для успішного втілення нашого проєкту. Безпосередньо в Unity планується використання фреймворків, які надають необхідний функціонал та інструменти:

- UI 2D – забезпечує ефективне створення та управління елементами користувацького інтерфейсу у двовимірному просторі.
- URP 2D (англ. Universal Render Pipeline 2D) – використовується для оптимізації та відтворення 2D-графіки. Надає можливості налаштування ефектів пост-обробки та світла.
- Sprites 2D – необхідний для роботи з 2D-спрайтами, включаючи їх форматування, зміну кольору/розміру/центрування.
- А також Cinemachine – забезпечує динамічне налаштування та зміну камери.

Висновки до першого розділу

В першому розділі, була визначена основна мета кваліфікаційної роботи, як результат - поставлена та проаналізована задача. Були розглянуті відомі представники жанру створюваного програмного забезпечення, дві гри у жанрі Metroidvania та серія ігор “Dark Souls”. На основі їх аналізу було зроблено висновки щодо правильної структури та необхідних елементів розробки.

Також створено концепт-документ, в якому визначено жанр, аудиторію, особливості та платформу. В кінці розділу було обґрунтовано вибір інструментарію, наведені його недоліки та переваги.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		20

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Проєктування архітектури ігрового додатку

Для зручного відображення логічної частини додатку, усі її можливості та будова представлені у вигляді відповідних схем та діаграм.

Діаграма варіантів використання (англ. Use Case Diagram) (рис. 2.1) представляє як користувач може взаємодіяти з початковим головним меню та якими базовими можливостями володіє. А саме:

- Початок гри – перехід до сцени з першим рівнем.
- Продовження гри – перевірка наявних збережень та їх завантаження.
- Змінення налаштувань – налаштування розміру екрану, режиму вікна та гучності звуку.
- Перегляд відкритих досягнень – список отриманих досягнень, їх опис та відображення тих що наразі заблоковані.
- Завершення гри – вимкнення додатку.



Рисунок 2.1 - Діаграма варіантів використання головного меню

Діаграми активності (англ. Activity diagram) є візуальним представленням динамічних операцій та зв'язків між ними. У даному випадку рисунки 2.2-2.3 ілюструють роботу внутрішньоігрового меню, а саме кожної його сторінки.

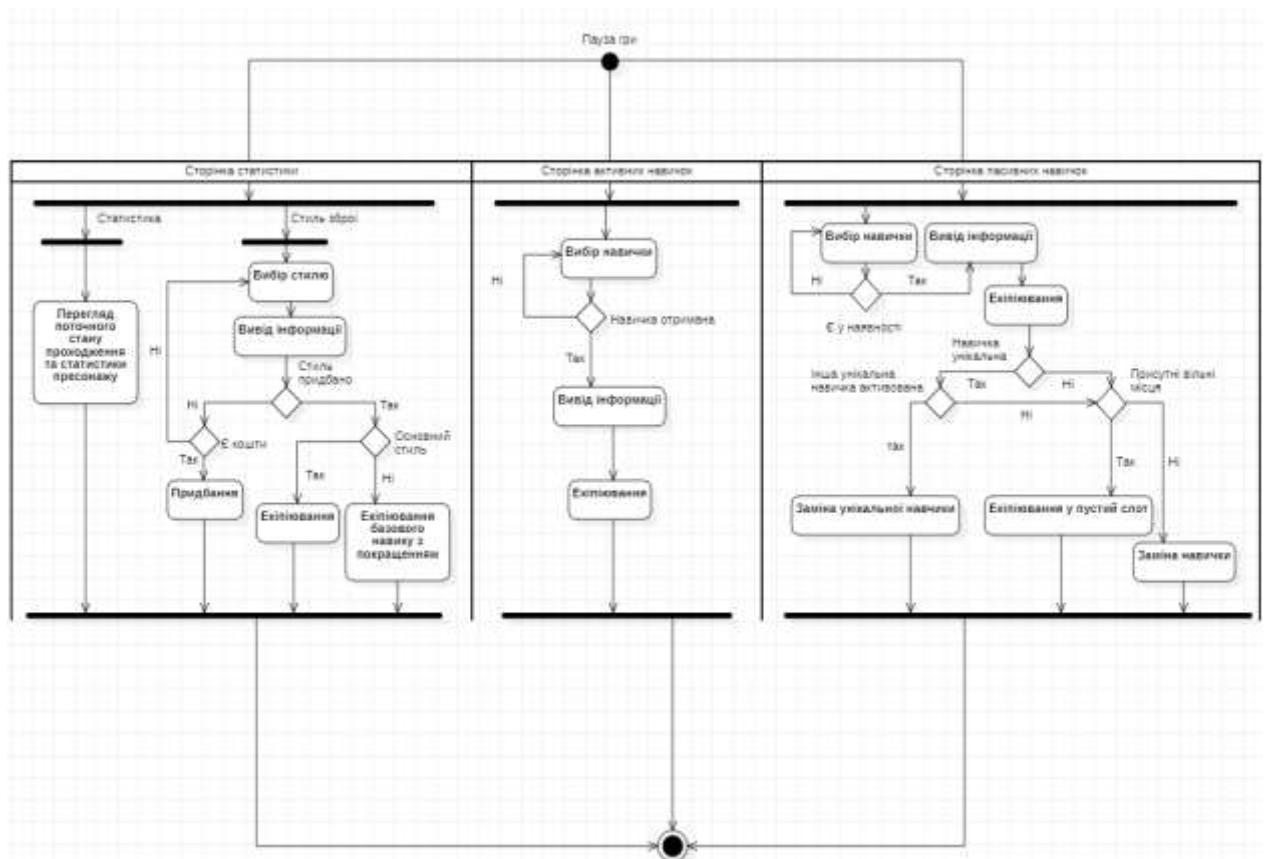


Рисунок 2.2 - Діаграма активності для статистики і навичок

На першій схемі (див. рис. 2.2) представлена робота таких механік як: прокачування обраного стилю зброї, екіпіювання пасивної або активної навички, а також вивід інформації по статистиці або вмінню.

Існує 3 основних стилі володіння зброєю (червоний, синій та зелений), кожен із них має два ступеня покращення. Спочатку при натисканні на іконку ми отримуємо інформацію про нього, надалі здійснюється перевірка, чи придбаний стиль. Якщо ні – відповідно нам пропонують купити його за кошти, які ми збираємо вбиваючи ворогів. В іншому випадку відбувається наступна перевірка – є навичка покращенням або основною навичкою. Якщо це базове вміння ми можемо його активувати. У випадку з покращенням отримуємо інформацію про нього, так як після покупки воно автоматично активується з базовим вмінням.

У випадку з екіпіюванням активної навички перевіряється лише її наявність. Отримати таку можна лише як нагороду здолавши боса локації.

Пасивні навички також можна знайти, отримати у нагороду або купити в NPC (англ. non playable character). Вони бувають двох типів: унікальні або звичайні. Активними одночасно можуть бути лише 3 навички, і тільки одна з них унікальна. Якщо навичка була отримана (тобто є у наявності) при натисканні на іконку спочатку виводиться інформація про неї. Далі йде перевірка, чи унікальна вона. При позитивному варіанті перевіряється чи не активовано іншу унікальну навичку – за результатом її або замінюють, або шукають вільний слот. Зі звичайними вміннями простіше, у такому випадку пропускається перевірка на унікальність і одразу шукається слот для екіпіювання або заміни останнього обраного.

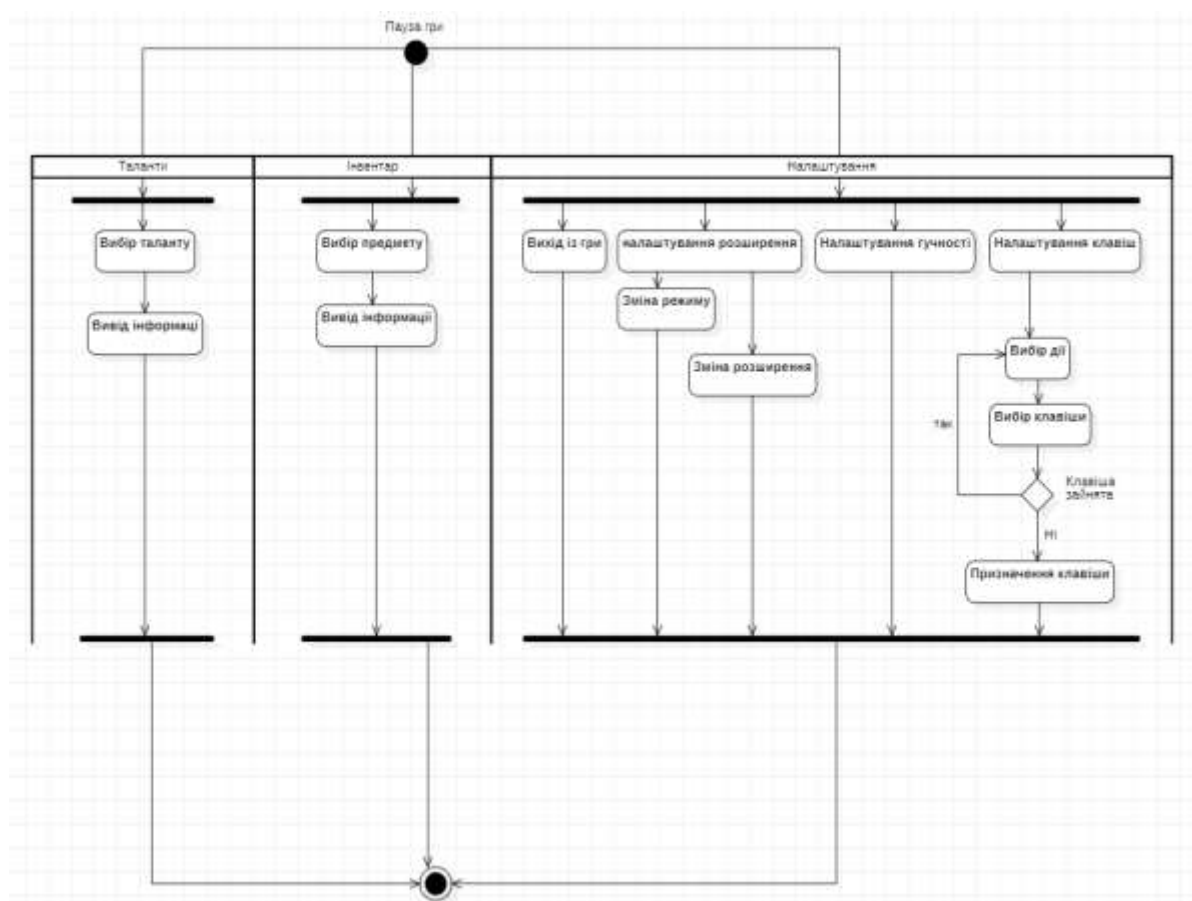


Рисунок 2.3 - Діаграма активності для талентів, інвентарю, налаштувань

На другій частині діаграми (див. рис. 2.3.) продемонстрована робота талентів, інвентарю та внутрішньоігрових налаштувань.

Таланти – це пасивні навички які не потрібно обирати, тобто після отримання вони завжди будуть доступні для використання. Їх робота досить схожа на інвентар, так як предмети використовуються автоматично у потрібному місці (за наявності). При натисканні на іконку виводиться інформація про вміння або предмет.

Що до налаштувань вони майже ідентичні до головного меню, проте мають додаткову можливість зміни клавіш. Можна перепризначити заздалегідь прив'язані клавіші дій на інші кнопки за бажанням. Спочатку обирається дія, наприклад стрибок або біг, далі натискається клавіша на клавіатурі. Якщо вона ще не зайнята – значення перепишеться.

Діаграма класів (англ. Class Diagram) відображає внутрішню структуру додатку. Статичні елементи, такі як: класи, типи даних, їх зміст та відношення. Представлені класи поділені на дві частини (рис. 2.4. – 2.5.).

Перша частина включає класи відповідальні за роботу головного меню, графічного інтерфейсу користувача, паралаксу (англ. Parallax) [11, 12] та тригерів, пов'язаних із переміщенням камери або програванням анімацій.

Конкретніше про кожен клас та його поля/методи:

- Клас MainMenu: відповідає за роботу усіх сторінок головного меню. Перемикання між ними та функціонал. Включає поля із панелями (GameObject), стандартні налаштування (масив розширень екрану та його режим, гучність). Також використовує структуру Achievement для збереження даних про досягнення гравця. Методи реалізують роботу з графічним інтерфейсом та зберігають налаштування користувача.

- Класи BossExit та DoorTrigger: включають у себе об'єкти HUD та оточення, які необхідно змінювати при взаємодії з персонажем. Клас CameraTrigger містить основну та альтернативну камери і слугує тригером для їх перемикання на одній сцені. В усіх класах міститься метод перевірки колізії з гравцем.

- Класи BarsController та BossGUI: відповідають за роботу HUD гравця і боса відповідно. Відображають зміну статистики. Містять поля

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

GameObject, що необхідно змінювати та дані (здоров'я, ману), які отримуються з класів наслідуваних від CharacterStats (клас для роботи із внутрішньою логікою).

- Класи ParallaxBehaviour та MapBehaviour: реалізують роботу паралаксу та мапи. Перший містить метод для оновлення положення камери, другий для відображення мапи світу по натисканню клавіші.

- Класи SpheresGUI та FlaskGUI: необхідні для роботи з динамічними предметами які містяться у HUD. Їх оновлення залежить не лише від характеристик персонажу, але і активованих навичок та стилю зброї. Вони включають методи для поступового або негайного вимкнення/ввімкнення, змінення елементів, що присутні на HUD гравця. Для їх реалізації було використано поведінковий патерн проєктування, а саме – шаблонний метод (англ. Template Method) [13]. Він визначає кістяк алгоритму, перекладаючи відповідальність за деякі його кроки на підкласи, дозволяє підкласам перевизначати кроки алгоритму, не змінюючи його загальної структури. Тому спочатку було реалізовано абстрактний клас GUIController, а вже від нього наслідувано SpheresGUI та FlaskGUI.

Друга частина діаграми складається із класів, що забезпечують роботу внутрішньоігрового меню. Кожна з них використовує відповідну структуру для збереження даних, а також породжувальний патерн проєктування "Одинак" (англ. Singleton) [14] для її передачі до інших частин програми. Він, гарантує, що клас має лише один екземпляр, та надає глобальну точку доступу до нього.

- Клас PauseMenu: є основою взаємодії з внутрішньоігровим меню. Він реалізує роботу перемикання між сторінками, а також ввімкнення/вимкнення необхідних елементів при розгортанні та згортанні UI. Отримує дані про наявну валюту і оновлює її. Містить Set/Get методи, а також обраховує індекс активної панелі для перемикання між ними.

- Клас CharacterPanel: працює зі статистикою персонажа та стилями зброї. Дозволяє переглянути поточний прогрес проходження, а також придбати або екіпірувати навички. Щоб отримати інформацію про персонажа звертається

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

до наслідуваного від CharacterStats класу, а саме PlayerStats, і виводить дані у вигляді тексту в відповідних полях. Для активації навички зброї використовує методи з класу WeaponStyles, що включає логіку для усіх вмінь. Містить поля з навичками, які реалізовані у вигляді масиву типу WeaponSkill, і тимчасовими змінними як наприклад currentStyle, що допомагають виконувати перевірки зміни або оновлення даних.

- Клас AbilityPanel: відповідає за активацію ультимативних навичок. Включає у себе перевірки на наявність та доступність екіпірування. Для активації ефекту передає доступ через Singleton до класу Skills.

- Клас CharmsPanel: реалізує роботу з пасивними навичками. Має методи для купівлі (PurshcaseCharm), екіпірування (EquipCharm) та зняття (UnequipCharm) навичок. Також включає перевірки на наявність, унікальність навички і кількість вільних слотів для активації. Поля включають: масиви з усіма та обраними навичками, дані для перевірок, поля для оновлення інформації.

- Класи TalantPanel, InventoryPanel: потрібні для відображення даних про знайдені таланти та предмети відповідно. Також вони реалізують методи знаходження даних об'єктів і оновлення меню. Це використовується у класах TalantCollision та InventoryCollision, що перевіряють колізію з гравцем. Також сторінка інвентарю за допомогою методу FindItems() динамічно додає предмети у порядку їх знаходження.

- Клас MovementKeys реалізує прив'язку клавіш до відповідних дій персонажа. Містить словник, що зберігає значення та корутину, що очікує натискання.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

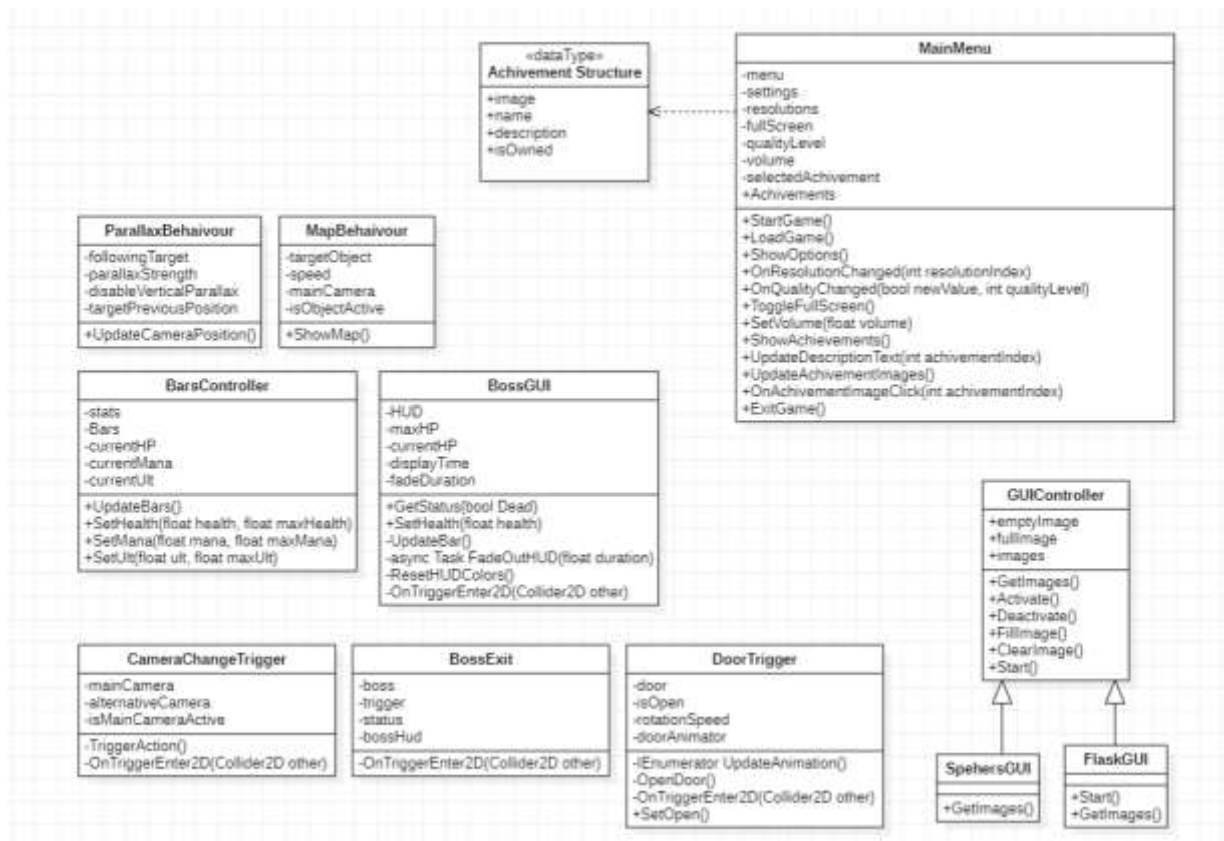


Рисунок 2.4 - Діаграма класів

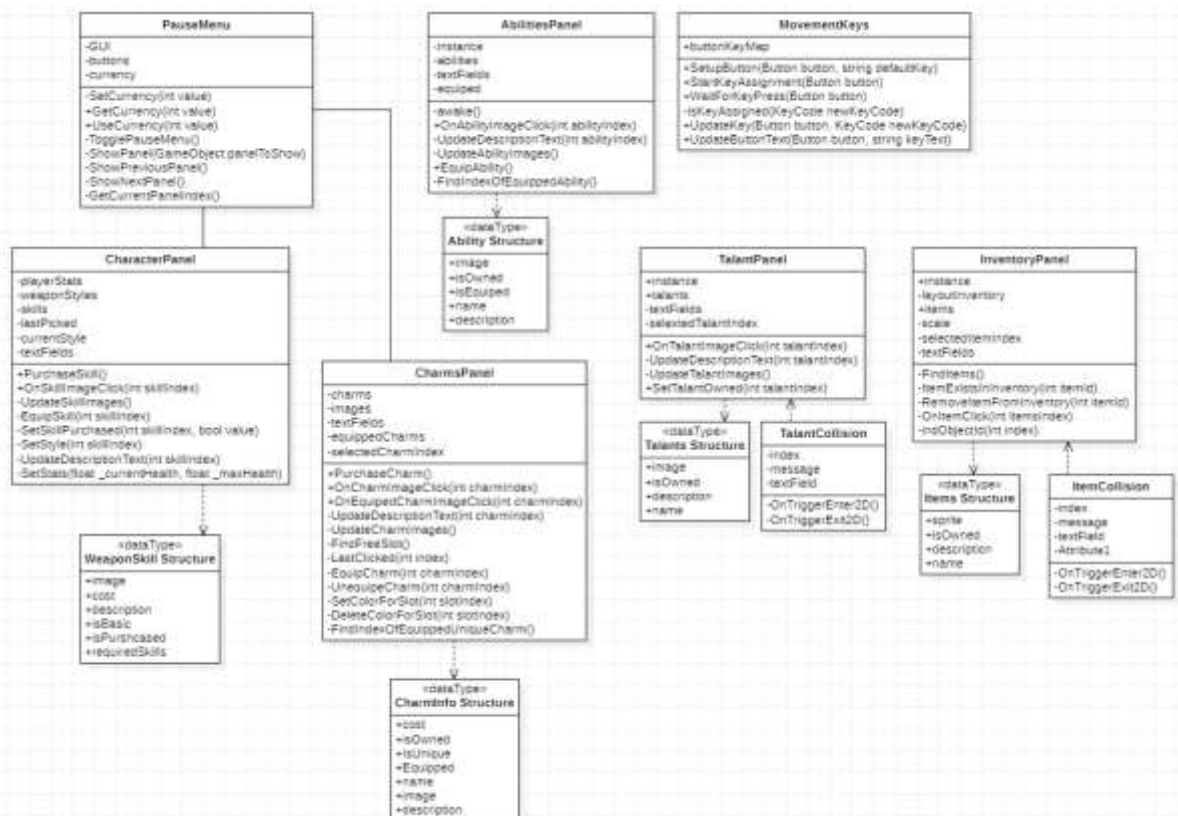


Рисунок 2.5 - Діаграма класів – продовження

Діаграма компонентів (англ. Component diagram) відображає організацію програми, залежності та зв'язки між елементами. На рисунку 2.6 зображено основні складові інтерфейсу гри, що включають: головне/внутрішньоігрове меню, HUD гравця та позначки ворогів, модуль карти і графічний модуль (який в свою чергу складається з фреймворків і художніх елементів).

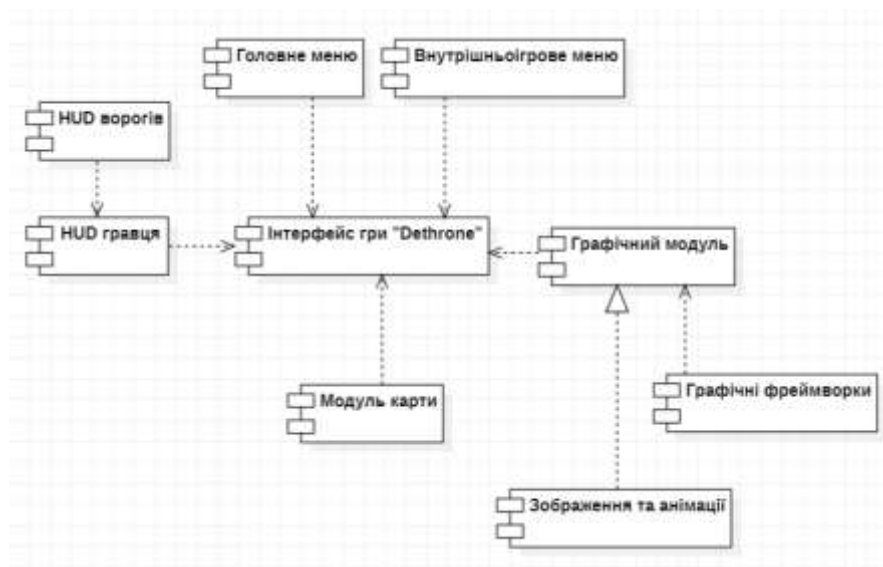


Рисунок 2.6 - Діаграма компонентів

На діаграмі розгортання (англ. Deployment diagram) зображено фізичне обладнання та компоненти програмного забезпечення системи. На рисунку 2.7 візуалізовано розгортання створюваної гри на апаратному забезпеченні, а саме ігровий двигун, готовий build проєкту та користувацький ПК.

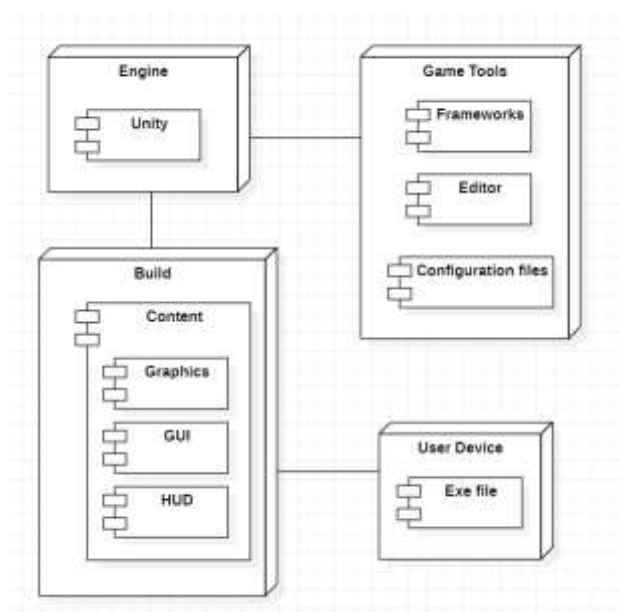


Рисунок 2.7 - Діаграма розгортання

2.2 Розробка інтерфейсу користувача

Наступним етапом розробки є проєктування UI. На основі проведеного раніше аналізу існуючих програмних продуктів, представлено концептуальні макети для створення власного інтерфейсу (рис. 2.8 - 2.10):

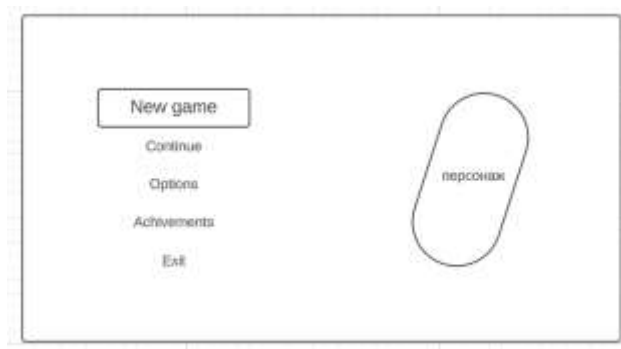


Рисунок 2.8 - Загальний макет головного меню



Рисунок 2.9 - Загальний макет меню паузи

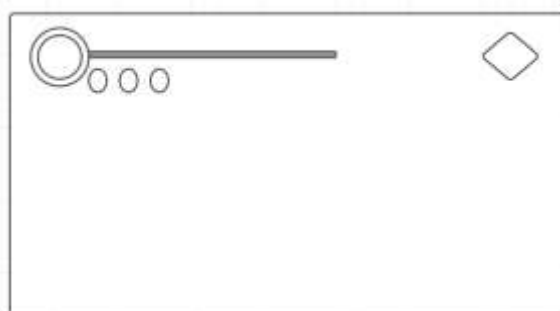


Рисунок 2.10 - Макет HUD

Усі вони не лише візуально, але і функціонально відповідають поставленим вимогам та надають зазначені у пункті 2.1 можливості гравцю. Тепер отриманий результат необхідно перенести у Unity.

Основою для створення UI є Canvas [15, 16]. Саме він дозволяє організувати і взаємодіяти з UI-компонентами у двовимірному просторі гри. У

процесі розробки використовуються такі елементи, як кнопки, текстові поля, панелі, слайдери та інші які представлені на рисунку 2.11.

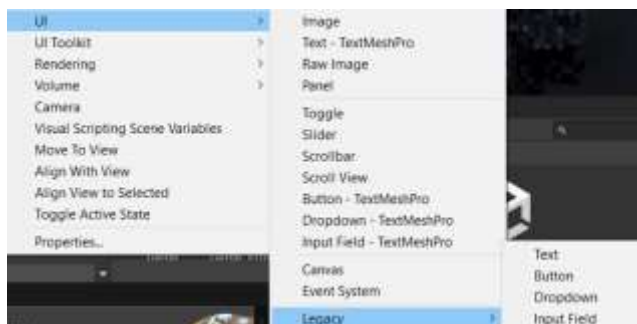


Рисунок 2.11 - UI-компоненти доступні у Unity

Для забезпечення відображення об'єктів на екрані спочатку необхідно налаштувати параметри рендер камери та розмір Canvas. Це необхідно для збереження коректного розміру та положення компонентів при зміні ширини і висоти екрану. Результат такого налаштування представлено на рисунку 2.12.

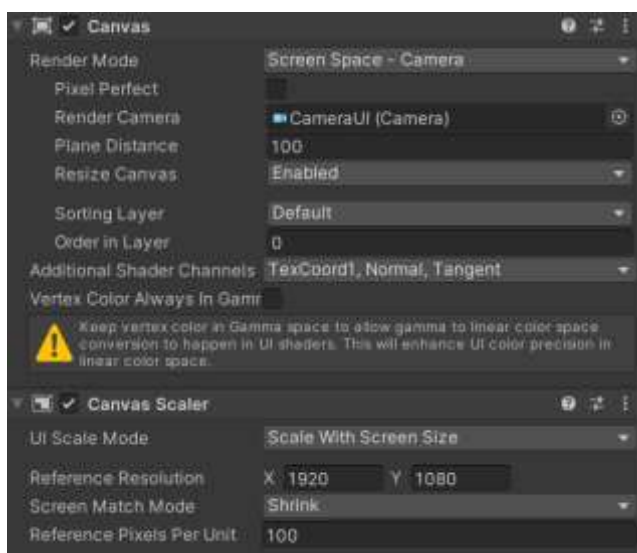


Рисунок 2.12 - Налаштування полотна

Щоб правильно розташувати об'єкти та забезпечити їх рівномірний розподіл, використовуються Layout-групи, такі як Vertical Layout Group, Horizontal Layout Group або Grid Layout Group. Ці інструменти Unity дозволяють автоматизувати розміщення та відступи одразу для групи обраних об'єктів замість одного. При створенні HUD використовуємо filled images type для відображення шкали здоров'я, мана, витривалості. Через код у відсотках сюди будуть передаватись показники персонажа (рис. 2.13).

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

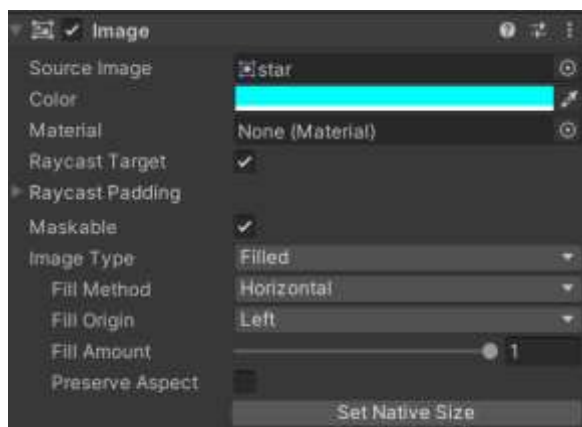


Рисунок 2.13 - Налаштування зображень з основними показниками

Для роботи з камерою використовується фреймворк Cinemachine, він дозволяє не лише задавати границі (рис. 2.14) пересування камери та перемикатись між існуючими за потреби, але і має функцію слідування за цільовим об'єктом. З його допомогою у проєкті досить зручно створювати динамічну та кінематографічну камеру. Приклад використання такої зображено на рисунку 2.15.

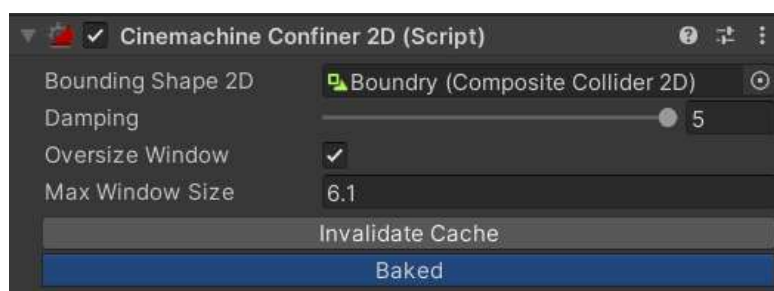


Рисунок 2.14 - Налаштування колайдери - границі камери

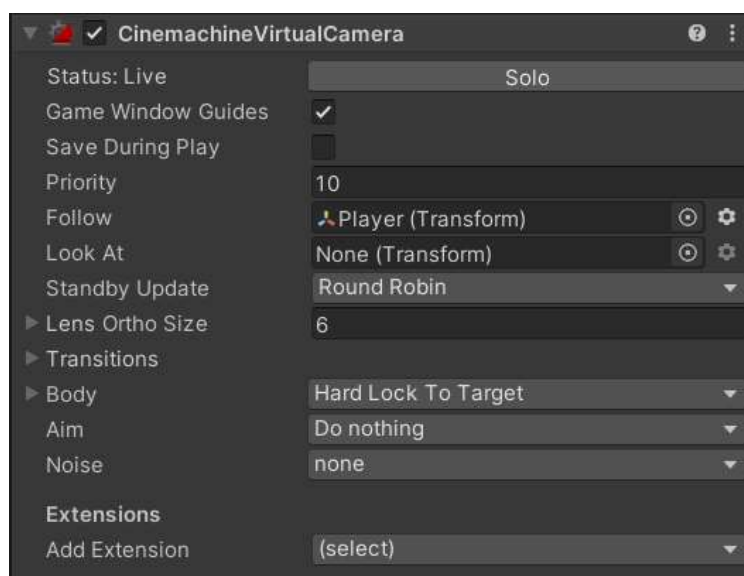


Рисунок 2.15 - Віртуальна камера та її налаштування

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

Для покращення зовнішнього вигляду використаємо спрайти, анімації та шрифт у відповідному стилі. Фінальний результат створення представлено на рисунках 2.16. – 2.18.



Рисунок 2.16 - Головне меню



Рисунок 2.17 - HUD гравця



Рисунок 2.18 - Меню паузи, початкова сторінка

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

2.3 Розробка програмної частини додатку

Після завершення роботи з розстановкою UI елементів можна перейти до написання програмної частини додатку. Кожна панель меню повинна мати відповідні обробники, що будуть викликатися при взаємодії. Наприклад можливості головного меню, які описані на діаграмі варіантів використання (рис. 2.1.), мають працювати за натисканням відповідної кнопки або слайдеру. Найважливіші створені функції з MenuController.cs:

У меню реалізовані методи для налаштування, що включають можливості зміни розширення, режиму екрану, рівня звуку. Розширення зберігаються у масиві resolutions в форматі “ширина x висота”. Щоб задати роздільну здатність та режим відображення використовуються функції із класу Screen який надає доступ до властивостей і методів керування параметрами екрану.

```
// Зміна розширення екрану.  
string selectedResolution = resolutions[resolutionIndex];  
string[] resolutionParts = selectedResolution.Split('x');  
  
int width = int.Parse(resolutionParts[0]);  
int height = int.Parse(resolutionParts[1]);  
Screen.SetResolution(width, height, Screen.fullScreen)  
  
// Перемикання віконного режиму.  
isFullScreen = !isFullScreen;  
Screen.fullScreen = isFullScreen;
```

Для зміни якості зображення використовується клас QualitySettings, який надає доступ до налаштувань графіки додатку. Функція SetQualityLevel(level) автоматично встановлює всі параметри графіки, які визначені для заданого рівня якості. Індексування починається з 0.

```
// Встановлення значення якості.  
  
lowQualityToggle.isOn = currentQualityLevel == 0;  
mediumQualityToggle.isOn = currentQualityLevel == 2;  
highQualityToggle.isOn = currentQualityLevel == 5;  
  
// Вибір рівню якості.  
currentQualityLevel = qualityLevel;  
QualitySettings.SetQualityLevel(currentQualityLevel);
```

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

Початок нової гри та вихід працюють досить просто. На відміну від них завантаження додатково реалізує перевірку на наявність збереження і підтягує дані гравця із файлу.

```
var data = SaveManager.Load<SaveData.PlayerPos>(saveKey2);  
sceneIndex = data._sceneIndex;  
SceneManager.LoadScene(sceneIndex);
```

Робота з досягненнями, включає у себе оновлення зображень після відкриття та вивід інформації про виконані умови. Якщо індекс досягнення в межах існуючих значень – отримуємо опис в текстовому полі.

```
// Оновлення опису досягнення (при натисканні)  
if (achivementIndex >= 0 && achivementIndex <  
    achievements.Length && achievements[achivementIndex].isOwned)  
    descriptionText.text = achievements[achivementIndex].description;
```

Додатково в коді реалізовані перевірки станів чекбоксів, розблокованих досягнень та активної панелі меню. Заздалегідь задані значення, такі як список розширень, рівень якості та масив досягнень, що необхідні для забезпечення консистентності і належного функціонування гри. Досягнення, в свою чергу, реалізовані у вигляді структури з необхідною інформацією: зображенням, описом, назвою та наявністю.

```
struct Achivement  
{  
    public Image image;  
    public bool isOwned;  
    public string description;  
    public string name;  
}
```

У наступній сцені, тобто в основній грі, меню представляє із себе не лише стандартний набір налаштувань, але і важливу частину геймплею без якої неможлива взаємодія користувача та основних механік. В нього включено 6 сторінок: статистика персонажа та стиль зброї, активні навички, пасивні навички, таланти, інвентар та налаштування.

Для керування сторінками необхідний окремий скрипт, що буде основою меню, він же PauseMenuController.cs. До нього з UI прив'язується низка кнопок за допомогою яких можна переходити по чергові від сторінки до сторінки, або

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

обирати потрібну нам одразу. Також інтерфейс включає кнопку повернення до гри і відображення валюти, що має гравець. Створені функції:

Реалізація паузи гри. Включає у себе приховання непотрібних елементів GUI та зупинку часу.

```
if (pauseMenu != null)
{
    bool isMenuActive = pauseMenu.activeSelf;
    bool isGuiActive = gui.activeSelf;
    bool isPostActive = postProcessing.activeSelf;
    pauseMenu.SetActive(!isMenuActive);
    gui.SetActive(!isGuiActive);
    postProcessing.SetActive(!isPostActive);
    // Pause or resume the game based on the menu's visibility
    Time.timeScale = isMenuActive ? 1f : 0f;
}
```

Навігація по сторінкам меню. Метод ShowPanel перевіряє відповідність кожної панелі до значення panelToShow, яке отримується шляхом перемикання між сторінками та перерахуванням їх індексу.

```
// Ввімкнення необхідної панелі.
void ShowPanel(GameObject panelToShow)
{
    foreach (var panel in menuPanels)
    {
        panel.SetActive(panel == panelToShow);
    }
}

// Повернення до попередньої панелі.
void ShowPreviousPanel()
{
    int currentIndex = GetCurrentPanelIndex();
    int previousIndex = (currentIndex - 1 + menuPanels.Length);
    ShowPanel(menuPanels[previousIndex]);
} ...
```

Робота з валютою. Для приховання значення і уникнення непередбачених змін, використовуємо не прямий доступ, а приватні поля. Шляхом використання методів Get та Set можна контролювати та вносити зміни у значення валюти зі збереженням правильної архітектури програми.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

```

public int SetCurrency(int value) {
    int currency = value;
    return currency;
}

public int GetCurrency(int value)
{
    value = Currency;
    return value;
}

```

Дані з наступних панелей для зручності зберігаються у вигляді структур, що мають усю необхідну інформацію про навичку або предмет. Також є низка методів, що однакові для усіх сторінок. Загалом це методи з оновлення опису, зміни зображення або пошук індексу елемента. Наприклад:

```

public struct Ability
{
    public Image image;
    public bool isOwned;
    public string description;
    public string name;
    public bool isEquiped;
}

```

Оновлення опису навички за індексом:

```

if (abilityIndex >= 0 && abilityIndex < abilities.Length)
{
    descriptionText.text = abilities[abilityIndex].description;
}

```

Оновлення кольору зображення в залежності від наявності:

```

for (int i = 0; i < array.Length; i++)
{
    array [i].image.color = abilities[i].isOwned ? Color.white : Color.black;
}

```

Передача інформації до інших класів відбувається за допомогою Singleton:

```

public static AbilitiesPanelScript instance;
private void Awake()
instance = this;

```

Перша сторінка меню під назвою “Character” включає у себе основну статистику персонажа (тобто рівень здоров'я, мана, відсоток відкриття карти, кількість знайдених секретів), а також 3 гілки стилів володіння зброєю. Кожна із них має одну активну навичку та два поліпшення. Скрипт CharacterPanel.cs

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

реалізує відповідні методи для отримання та виведення статистики персонажу, придбання та екіпіювання навичок, перевірки наявності поліпшень, а також оновлення інформації.

Робота зі статистикою персонажу і прогресом проходження. Для кожного типу статистики викликаються відповідні методи `SetStatTextFloat` та `SetStatTextInt`. Вони перевіряють, чи індекс знаходиться в межах допустимого діапазону, після чого оновлюють текстове значення за допомогою `currentValue` та `maxValue`.

```
// Метод для передачі статистики
public void SetStats(int _currentHealth, int _maxHealth, int _currentMana,
int _maxMana, int _collectedCollectibles ...)
{
    SetStatTextFloat(0, _currentHealth, _maxHealth); // Health
    SetStatTextFloat(1, currentMana, maxMana); // Mana
    SetStatTextInt(2, collectedCollectibles, maxCollectibles);
    SetStatTextInt(3, completionPercentage, 100);
}

// Встановлення змінних типу float
void SetStatTextFloat(int index, float currentValue, float maxValue)
{
    if (index >= 0 && index < statTextsFloat.Length)
    {
        statTextsFloat[index].text = string.Format("{0}/{1}", currentValue, maxValue);
    }
} ...
```

Оновлення візуальних елементів (іконок та елементів HUD).

```
// Оновлення кольору іконки
foreach (var skill in skills)
{
    if (skill.isPurchased)
    {
        skill.image.color = Color.white;
    }
    else
    {
        skill.image.color = Color.gray;
    }
}

// Зміна HUD в залежності від гілки стилю
switch (skillIndex) {
    case 0:
        style.color = Color.red;
        break;
}
```

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

Також перша панель включає роботу зі стилями зброї, перевірки наявності і тип. Основні із них зазначені на діаграмі активності (рис. 2.2).

```
// Придбання навички
private void SetSkillPurchased(int skillIndex, bool value)
{
    WeaponSkill[] updatedSkills = skills;
    updatedSkills[skillIndex].isPurchased = value;
    skills = updatedSkills;
}

// Екіпіювання навички
foreach (var skill in skills)
{
    if (skill.isPurchased && skill.requiredSkill.Length > 0)
    {
        if (skills[skillIndex + 1].isPurchased)
        {
            isUpgradedOnce = true;

            if (skills[skillIndex + 2].isPurchased)
            {
                isUpgradedOnce = false;
                isUpgradedTwice = true;
            } ...
        }
    }
}
```

Наступна панель відповідає за активні навички які ми отримуємо після перемоги над босом локації. Дає можливість обрати одну з них та активувати. Існує три випадки взаємодії:

- Ні одне вміння ще не було екіпіровано.

```
if (equiped == false)
{
    abilities[abilityIndex].isEquiped = true;
    equiped = true;
    nameText.text = abilities[abilityIndex].name;
    promptText.text = "equiped " + nameText.text;
    equipedText.text = "Equiped: " + nameText.text;
}
```

- Раніше гравець вже обирав вміння, і наразі хоче його замінити.

```
if (abilities[abilityIndex].isOwned && !abilities[abilityIndex].isEquiped)
{
    abilities[abilityIndex].isEquiped = true;
    abilities[equippedAbilityIndex].isEquiped = false;
    nameText.text = abilities[abilityIndex].name;
    promptText.text = "replaced with " + nameText.text;
    equipedText.text = "Equiped: " + nameText.text;
}
```

- Обрана навичка активована і гравцю необхідно відімкнути її дію.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

```

{
    abilities[abilityIndex].isEquiped = false;
    equipped = false;
    promptText.text = "unequiped " + nameText.text;
    equippedText.text = "Equiped: " ;
}

```

Аналогічним чином працює і наступна сторінка. А саме екіпіювання «чармів», вони ж пасивні навички. Єдина відмінність у їх кількості та типі. Вони поділяються на унікальні та звичайні. Пасивні навички можуть мати комбінації з трьох одночасно активних елементів. Метод їх екіпірування також можна умовно розділити на декілька блоків:

- Раніше вже була обрана унікальна навичка, заміна на нову.

```

if (charms[charmIndex].isUnique)
{
    int equippedUniqueIndex = FindIndexOfEquippedUniqueCharm();
    int previousIndex = FindIndexOfEquippedUniqueCharmInList();
    if (equippedUniqueIndex != -1)
    {
        Debug.Log("found unique");
        charms[previousIndex].isEquiped = false;
        equippedCharms[equippedUniqueIndex] = charms[charmIndex];

        charms[charmIndex].isEquiped = true;
        SetColorForSlot(equippedUniqueIndex);
        UpdateDescription(equippedUniqueIndex);
    }
}

```

- Обрана навичка звичайна або перша унікальна, присутній вільний слот (не обрано три).

```

if (freeSlot != -1)
{
    equippedCharms[freeSlot] = charms[charmIndex];

    charms[charmIndex].isEquiped = true;
    SetColorForSlot(freeSlot);
    UpdateDescription(freeSlot);
}

```

- Обрана навичка звичайна або перша унікальна, проте вільних місць немає. Заміна першої навички або обраного слоту.

```

if (lastClickedIndex != -1)
{
    charms[lastClickedIndex].isEquiped = false;
    equippedCharms[lastClickedIndex] = charms[charmIndex];

    charms[charmIndex].isEquiped = true;
    SetColorForSlot(lastClickedIndex);
}

```

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

```

        UpdateDescription(lastClickedIndex);

        lastClickedIndex = -1;
    }

    else
    {
        charms[0].isEquiped = false;
        equippedCharms[0] = charms[charmIndex];

        charms[charmIndex].isEquiped = true;
        SetColorForSlot(0);
        UpdateDescription(0);
    }

```

Деактивація пасивної навички, по своїй суті, еквівалентна її видаленню зі списку обраних. Для цього з масиву типу Charms видаляються усі дані про вміння і залишається пустий слот.

```

Charm currentCharm = charms[charmIndex];
charms[charmIndex].isEquiped = false;

for (int i = 0; i < equippedCharms.Length; i++)
{
    if (currentCharm.description == equippedCharms[i].description)
    {
        equippedCharms[i].image= null;
        equippedCharms[i].description = "";
        equippedCharms[i].isEquiped = false;
        equippedCharms[i].isUnique = false;
        equippedCharms[i].isOwned = false;
        equippedCharms[i].cost = 0;
        DeleteColorForSlot(i);
    }
}

```

Досить просто працюють таланти та інвентар. На даних сторінках не потрібно активувати елементи вручну (через натискання кнопки). Усі таланти доступні одразу після знаходження і так само є різновидом пасивних вмінь. Предмети з інвентарю в свою чергу використовуються автоматично при знаходженні відповідного триггеру, і додаються динамічно у порядку знаходження. У меню є можливість ознайомитись з їх детальним описом.

Метод динамічного додавання предметів проходиться по всім елементам масиву типу items та перевіряє, чи об'єкт вже знайдено і чи він ще не був доданий до інвентарю. Якщо обидві умови виконуються - створюється новий об'єкт типу GameObject з іменем "Item_" і індексом поточного елементу. До нього додається компонент Image для відображення, а також компонент Button, для обробки натискання і виклику методу відображення опису.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

```

private void FindItems()
{
    for (int i = 0; i < items.Length; i++)
    {
        if (items[i].isOwned && !ItemExistsInInventory(i))
        {
            GameObject newItemObject = new GameObject("Item_" + i);
            Image newItemImage = newItemObject.AddComponent<Image>();
            Button newItemButton = newItemObject.AddComponent<Button>();

            newItemImage.overrideSprite = items[i].sprite;
            newItemImage.rectTransform.localScale = scale;

            newItemImage.rectTransform.sizeDelta = imageSize;

            newItemObject.transform.SetParent(layoutInventory.transform);
            newItemImage.rectTransform.localScale = scale;
            newItemImage.rectTransform.localPosition.y, 0f);

            int index = i;
            newItemButton.onClick.AddListener(() => OnItemClick(index));
        } else if (!items[i].isOwned)
        {
            RemoveItemFromInventory(i);
        }
    }
}

```

Остання сторінка реалізує налаштування. Окрім усіх доступних налаштувань з головного меню тут також присутня система призначення клавіш. Будь яку дію персонажа можна прив'язати до зручної користувачу клавіші на клавіатурі.

При кліку на кнопку викликається метод StartKeyAssignment і запускає процес очікування. Це корутин, що чекає затискання клавіші, після чого призначає її для відповідної кнопки дії. Якщо клавіша обрана і не призначена, то currentKey записується у поточний keyCode зі словнику.

```

void StartKeyAssignment(Button button)
{
    StartCoroutine(WaitForKeyPress(button));
}

IEnumerator WaitForKeyPress(Button button)
{
    KeyCode currentKey = KeyCode.None;

    while (currentKey == KeyCode.None)
    {
        yield return null;
    }
}

```

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

```

foreach (KeyCode keyCode in System.Enum.GetValues(typeof(KeyCode)))
{
    if (Input.GetKeyDown(keyCode) && !IsKeyAssigned(keyCode))
    {
        currentKey = keyCode;
        UpdateKey(button, currentKey);
        break;
    } ...
}

```

Окрім роботи внутрішньоігрового меню на сцені необхідно оновлювати значення HUD. Персонаж гравця має власні характеристики, що зберігаються у класі PlayerStats. Для їх візуального відображення використовуються Set методи, які викликаються при зміні значень.

```

void UpdateBars()
{
    BarHP.fillAmount = currentHP / stats.GetMaxHealthValue();
    BarMana.fillAmount = currentMana / stats.GetMaxManaValue();
    BarUlt.fillAmount = currentUlt / stats.GetMaxUltValue();
}

public void SetHealth(float health, float maxHealth)
{
    currentHP = Mathf.Clamp(health, 0f, maxHealth);
    UpdateBars();
} ...

```

У деяких ситуаціях для зміни стану середовища або взаємодії з об'єктами необхідні тригери. Наприклад для зміни зовнішнього вигляду HUD:

```

if (currentHP <= 0 && HUD.activeSelf)
{
    UpdateBars();
    isDead = true;
    victoryText.SetActive(true);

    await FadeOutHUD(displayTime);

    HUD.SetActive(false);
}

```

Отримання предмета або навички:

```

if (isPlayerNearby && Input.GetKeyDown(KeyCode.E))
{
    inventoryPanelScript.SetItemOwned(itemIndex);
    Destroy(gameObject);
    message.text = "";
}
}

```

Або програвання анімацій:

```

private IEnumerator UpdateAnimation()
{
    yield return new WaitForSeconds(1f);
}

```

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

```

        doorAnimator.SetBool("isFalling", false);
    }

```

Відображення динамічних елементів HUD реалізовано за допомогою шаблонного методу. Батьківський абстрактний клас реалізує основну логіку роботи методів. Дочірні класи перезаписують їх за потреби:

```

public abstract class GUIController : MonoBehaviour
{
    [SerializeField] public Sprite emptyImage;
    [SerializeField] public Sprite fullImage;
    public Image[] images;

    public abstract Image[] GetImages();

    public virtual void Start()
    {
        images = GetImages();
        Deactivate();
    }

    public virtual void Activate()
    {
        foreach (Image img in images)
        {
            img.enabled = true;
        }
    }

    public virtual void Deactivate()
    {
        foreach (Image img in images)
        {
            img.enabled = false;
        }
    }
}

```

Іншим важливим елементом користувацького інтерфейсу у іграх є карта. Жанри на кшталт Metroidvania використовують загальну мапу рівня із позначками персонажу, NPC, босів. Даний скрипт реалізує переміщення іконки гравця на мапі відповідно до його розташування на рівні, а також активацію карти при натисканні клавіші “М”.

```

Vector2 targetPosition = targetObject.position;
Vector3 direction = (targetPosition - (Vector2)transform.position).
normalized;
float distanceToMove = speed * Time.deltaTime;
Vector3 newPosition = transform.position + direction * distanceToMove;

if (Input.GetKeyDown(KeyCode.M))
{
    isObjectActive = !isObjectActive;
}

```

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

```
mainCamera.SetActive(isObjectActive);
}
```

Система діалогів [17] (англ. Dialogue Editor) – фреймворк у Unity, що допомагає візуально будувати структуру розмови з персонажем (рис. 2.19), а також викликати необхідні методи при певних відповідях. Для коректної роботи системи використовується `ConversationManager`, що містить методи початку/кінця діалогу, призначення змінних та перевірки статусу розмови.

Початок необхідного діалогу:

```
ConversationManager.Instance.StartConversation(dialogue);
```

Призначення змінної:

```
ConversationManager.Instance.SetBool("isEnough", false);
```

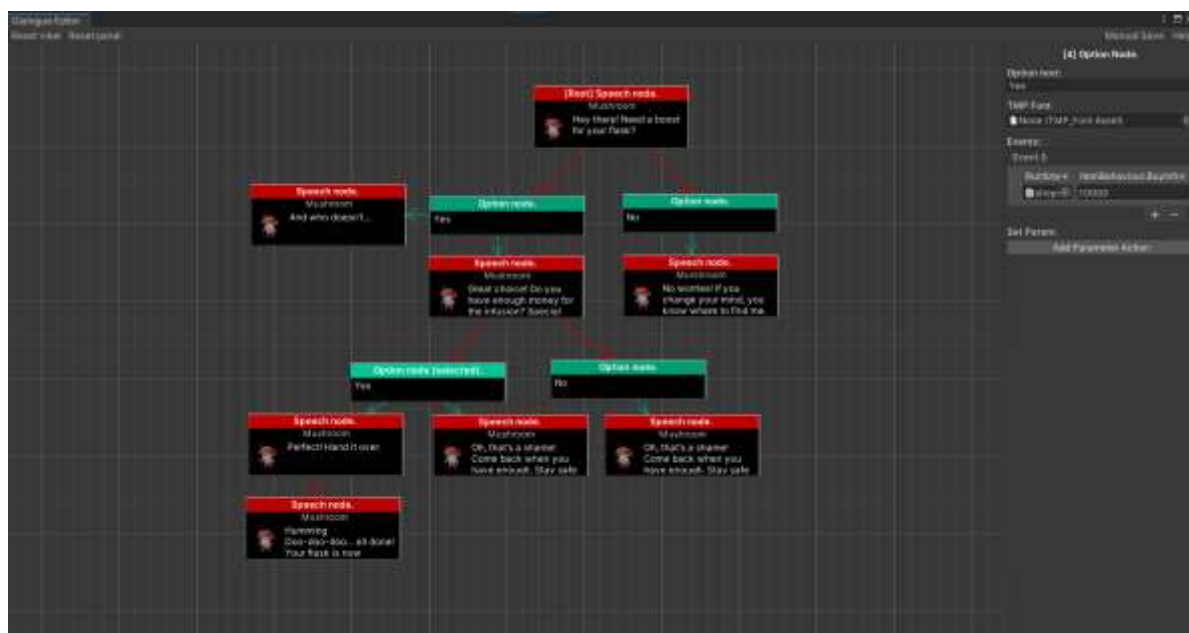


Рисунок 2.19 - Налаштований діалог з неігровим персонажем «Mushroom»

2.4 Реалізація візуальних ефектів, графічних елементів та анімацій

Щодо графічного оформлення, для поліпшення візуальної частини була використана система часток **[18, 19]** (англ. Particle System). За її допомогою можна створювати ефекти розсіювання, обертання або руху частинок. Це поліпшує не лише зовнішній вигляд, але і атмосферу занурення та інтерактивність загалом. Таким чином у головному меню було досягнуто ефекту «зоряного неба» (рис. 2.20), а безпосередньо у грі ефекти «падаючого

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				44
Змн.	Арк.	№ докум.	Підпис	Дата		

листя» та «горіння вогню» (рис. 2.21 - 2.22). На рисунку 2.23 детально відображено процес налаштування Particle System.

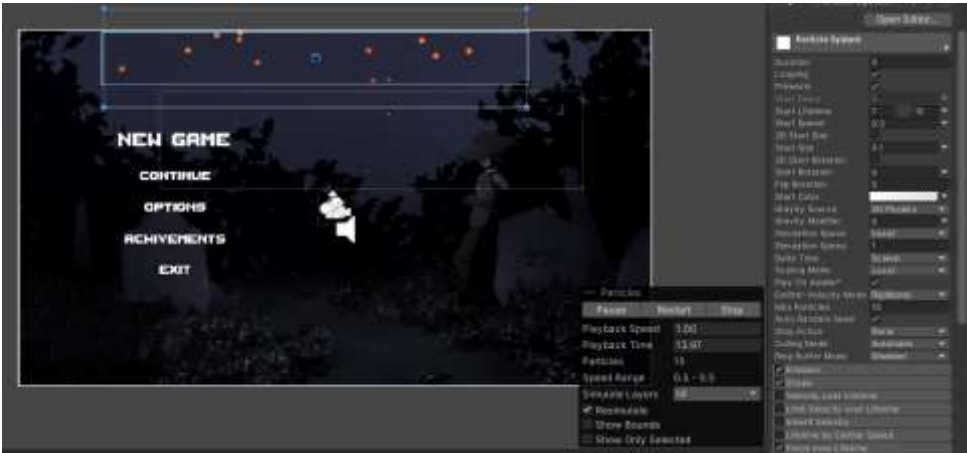


Рисунок 2.20 - Реалізація системи часток у головному меню

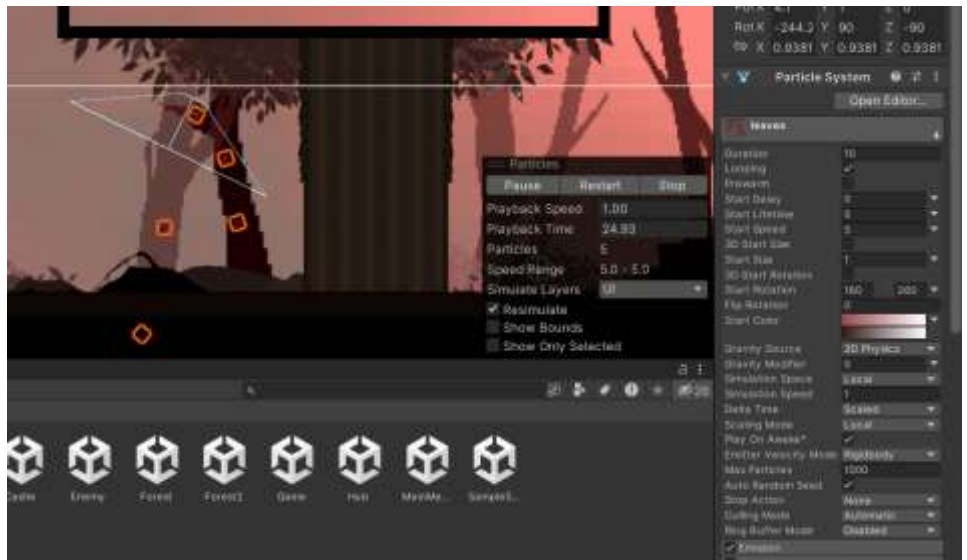


Рисунок 2.21 - Реалізація системи часток у локації «Ліс»

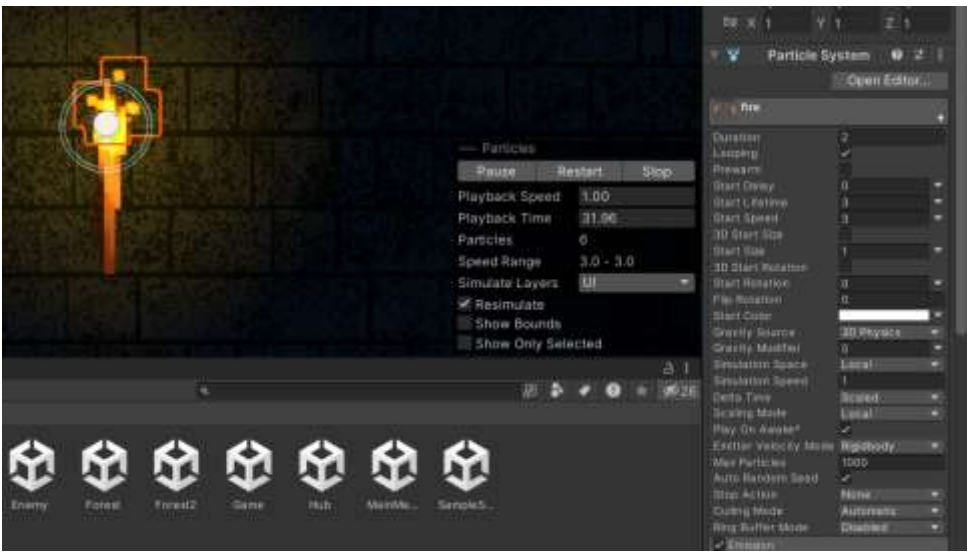


Рисунок 2.22 - Реалізація системи часток у локації «Замок»

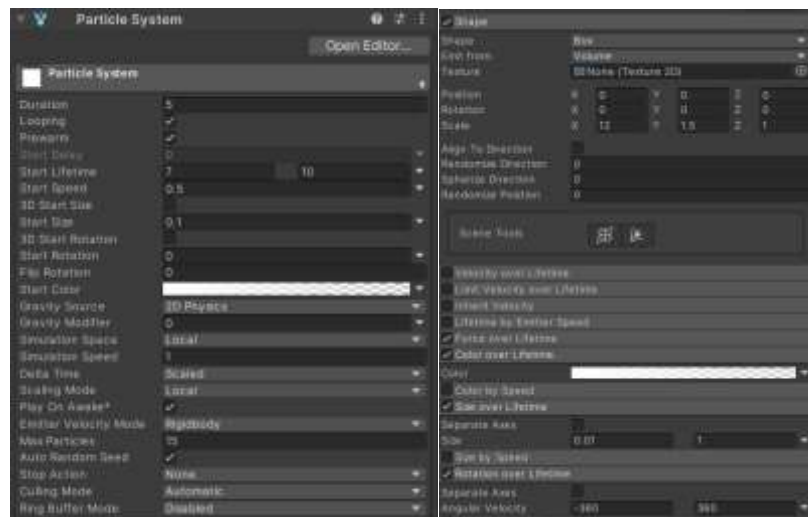


Рисунок 2.23 - Налаштування системи частинок

У процесі роботи були відредаговані такі параметри як повторюваність, максимальна кількість часток, «тривалість життя», початковий розмір, колір та швидкість. Також вказана форма площини де вони мають з'являтися, зміна параметрів з часом, кут обертання. Як результат налаштовано ефект часток, що з'являються у різний час, різного розміру та кольору і обертаються навколо своєї осі.

Для роботи з освітленням та пост-обробкою [20] камери було використано фреймворк Universal Render Pipeline 2D. А саме об'єкти light, shadow caster 2d та global volume.

Освітлення та види його джерел бувають різні. Існує два основних типи – глобальне та локальне. Глобальне освітлення працює на всій сцені та об'єктах, в той час коли локальне лише у певній області та на вказані слої. Локальне освітлення може мати багато форм, наприклад: спрайту, точкову або вільну. Усі види освітлення, що доступні у редакторі, зображені на рисунку 2.24. Перед початком роботи із ним необхідно обов'язково додати світло-відбиваючий матеріал на ігрові об'єкти.

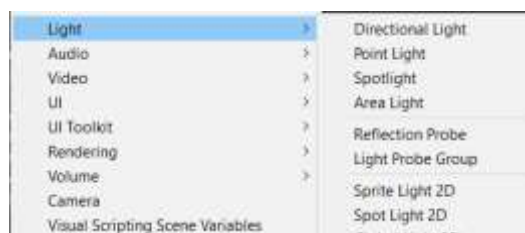


Рисунок 2.24 - Види освітлення

Для більшого реалізму відображення світла необхідно створити тіні. У 2D проєктах вони можуть одразу виключати із себе освітлення або просто перекривати об'єкти за собою. Форма тіні налаштовується за допомогою колайдери, а скрипт додається до необхідного об'єкту як компонент із фреймворку URP. Рисунок 2.25 відображає приклад налаштування такого компоненту.

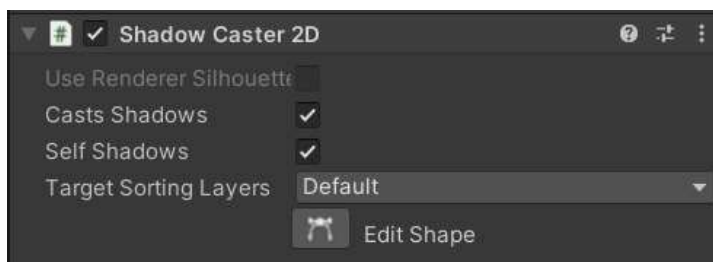


Рисунок 2.25 - Налаштування тіней

Безпосередньо на рівнях, налаштування світла було використано для створення темних локації, таких як секретне підземелля, а також реалізовано роботу предмету «Lantern» (рис. 2.26).

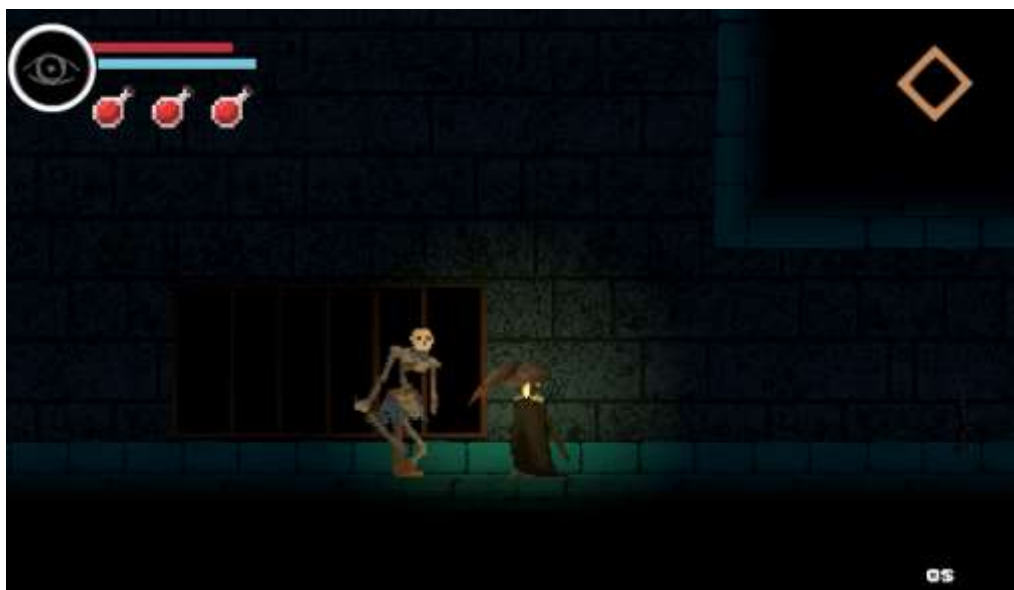


Рисунок 2.26 - Використання світла для поліпшення візуальної складової рівня

Що до пост-обробки, спочатку необхідно створити URP Asset та render data, в яких зберігаються основні налаштування графіки для проєкту (рис. 2.27). Після цього необхідно створити Volume-об'єкт який буде зберігати обрані фільтри камери. В даному випадку підсвічування, корекцію кольору і шум.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

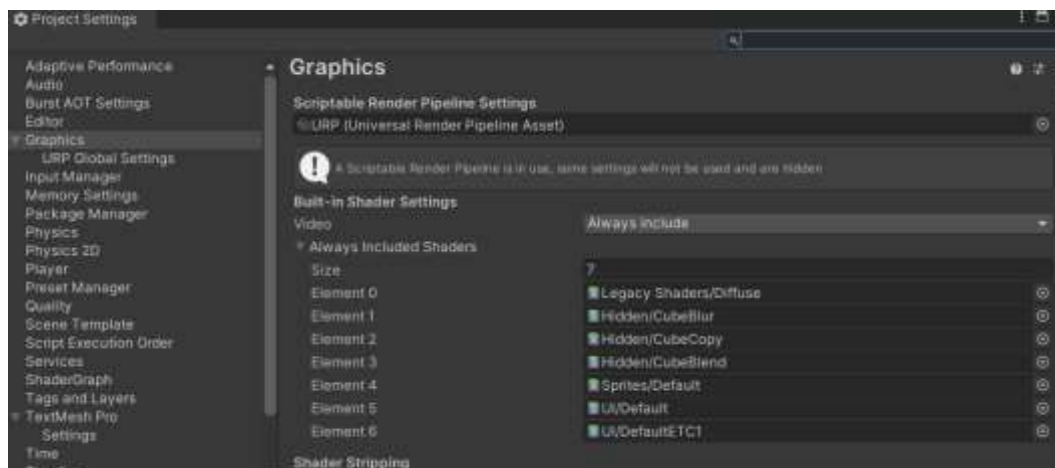


Рисунок 2.27 - Додавання асету у налаштуваннях проєкту

В основній грі для створення ефекту руху та глибини фону, зображення не просто розділяються на слої, але і мають ефект паралаксу. Основна задумка полягає в тому, що об'єкти ближчі до гравця рухаються швидше і мають темнішу текстуру. Дальні зображення навпаки сповільнені та бліді. У коді це прописано наступним чином:

```
var delta = followingTarget.position - targetPreviousPosition;
if (disableVerticalParallax)
    delta.y = 0;

targetPreviousPosition = followingTarget.position;
transform.position += delta * parallaxStrength;
```

Камера реагує на рух цільового об'єкта (в даному випадку гравець). Вектор delta представляє різницю позицій між поточною та попередньою. Потім ця різниця масштабується за допомогою параметру parallaxStrength і додається до поточної позиції камери, забезпечуючи ефект руху фону з швидкістю інакшою від цільового об'єкту.

Наступний крок у розробці – створення спрайтів персонажів, анімацій та паліти для Tilemap [21, 22]. Графічні об'єкти малюються вручну у програмі Aseprite, та переносяться до ігрового двигуна у підтримуваному форматі ase, так як він підтримує слої та кадри (рис. 2.28).

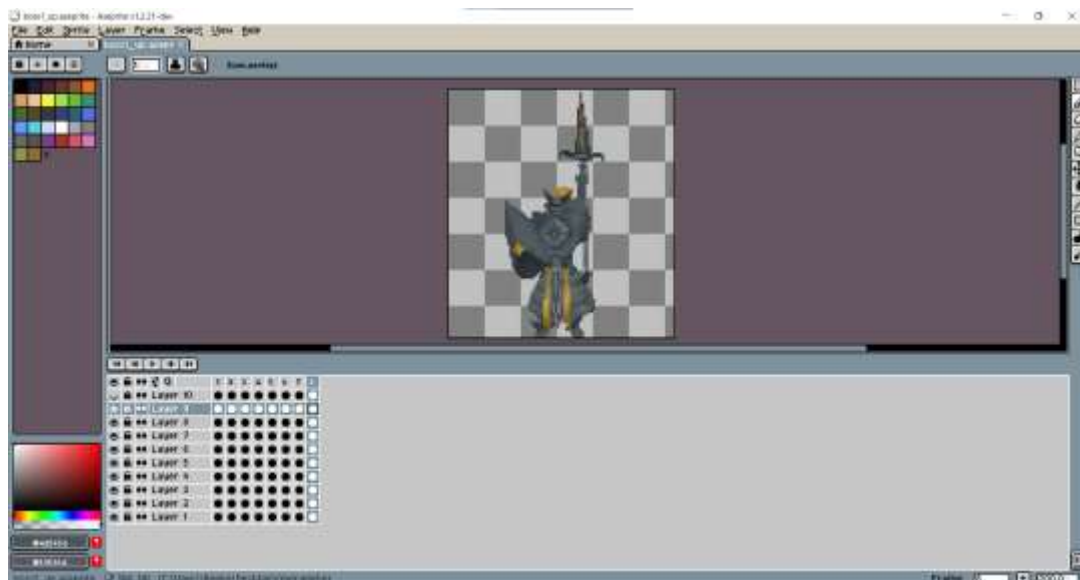


Рисунок 2.28 - Інтерфейс програми Aseprite

Усі розроблені зображення наведено у Додатку Е.

Кожен персонаж включає відповідну низку анімацій. Головний герой має найбільшу кількість. Вороги мають стандартні стейти, що відрізняються однією-двома анімаціями. Перелік дружніх персонажів зведено до таблиці 2.1, ворожих до таблиці 2.2.

Таблиця 2.1

Перелік анімованих фалів дружніх ігрових та неігрових персонажів

Ім'я персонажу	Тип	Перелік анімацій
Main character	Ігровий персонаж	<i>Idle, Move, Climb, Dash, Hang, Fall, Jump</i> – пересування <i>Attack_one, Attack_two, Attack_three</i> – серія атак
Main character	Ігровий персонаж	<i>Block, Counterattack</i> – блокування шкоди, нанесення контр атаки <i>Hit, Death</i> – отримання шкоди, смерть <i>Catch, Throw, Rotate</i> – використання активних навичок
Mushroom npc	NPC	<i>Idle, talk</i> – взаємодія
Fake npc	NPC	
Knight npc	NPC	
Snake npc	NPC	

Таблиця 2.2

Перелік анімованих фалів ворожих неігрових персонажів

Ім'я персонажу	Унікальні анімації	Загальні анімації
Archer	<i>Back</i> – ухиляння <i>Melee _attack</i> – додаткова атака ближнього бою <i>Arrow</i> – стріла	<i>Idle, Walk</i> – пересування <i>Attention, Attack</i> – пошук гравця та атака <i>Death, Hit, Stun</i> – отримання шкоди, стан та смерть
Assassin	<i>Attack_two, Attack_three</i> – додаткові анімації ударів	
Debuffer	<i>Appearance</i> – поява (телепортація)	
Giant Eye	<i>Laser</i> – лазер	
Wolf	<i>Attack_two</i> – сильний удар	
Wood-spirit	<i>Fireball</i> – шар для удару магією	
Forest Boss	<i>Up, Roll, Spit</i> – унікальні атаки босу	
Castle Boss	<i>Shield_block, Magic_attack</i> – унікальні атаки босу	
Dog	—	
Skeleton		
Spider		
Ghost		

Окрім анімованих елементів у грі також присутні зображення для GUI, HUD, локацій і візуальних ефектів (наприклад системи частинок). Їх узагальнений перелік зведено до таблиці 2.3.

Таблиця 2.3

Перелік створених зображень

Тип зображення	Призначення
Іконки	Зображення для досягнень, стилів зброї, активних/пасивних навичок, талантів, предметів.
Кнопки, слайдери, випадаючі списки	Кнопки для керування у головному та внутрішньому ігровому меню.

Тип зображення	Призначення
Фонові зображення	Зображення для головного та внутрішнього ігровому меню, локації «Ліс», локації «Замок», секретної локації.
Елементи декору	Зображення лісу, елементи замку, елементи секретної локації.
HUD	Смужки статистики, стилю зброї.

Загальна кількість статичних зображень (фонів, іконок, UI елементів) налічує більше 200 елементів, анімації – 123 файли (рис. 2.29).



Рисунок 2.29 - Частина створених графічних файлів

Коли графічні елементи створені, можна перейти до їх редагування та обробки у Unity. Для початку необхідно розрізати складені спрайти на частини. Це робиться за допомогою Sprite Editor. (рис. 2.30).

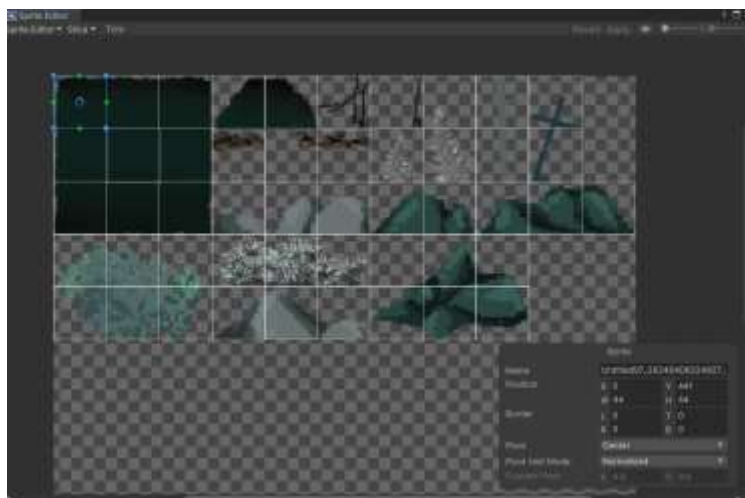


Рисунок 2.30 - Приклад створення multiple зображення, редагування розміру

Наступним кроком є реалізація палітри елементів та налаштування «правил». Для цього використовується фреймворк Tilemap 2D, який має розширення, що завантажується із пакетів ігрового двигуна Unity. Він дозволяє створювати двовимірні карти, які складені з малих прямокутних фрагментів. Їх називають плитками (англ. Tiles). Цей підхід досить поширений у розробці комп'ютерних ігор, так як є найзручнішим при побудові великих рівнів. На рисунку 2.31 зображено створену таким чином палітру для локації «Замок».

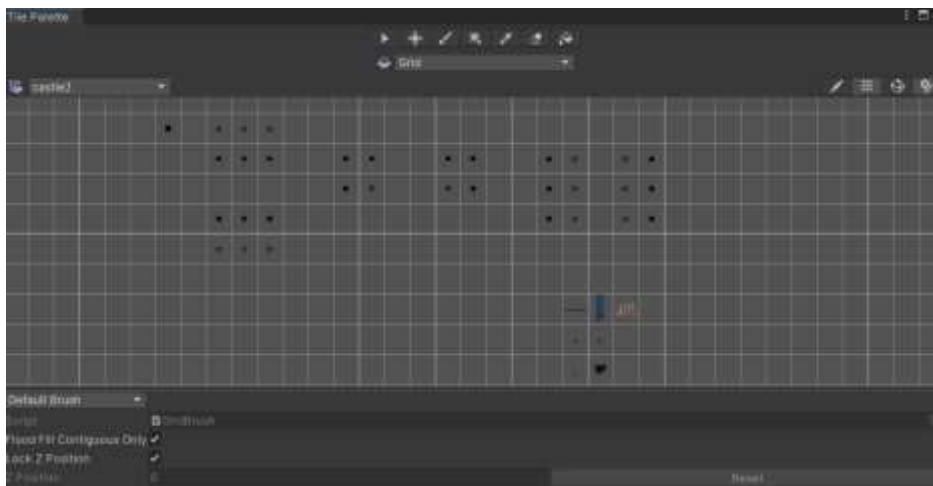


Рисунок 2.31 - Створена палітра для локації «Замок»

Щоб спростити процес моделювання рівня можна використати додатковий інструмент під назвою Tile Rule, він же Autotiling. З назви зрозуміло, що цей метод розташовує плитки автоматично відповідно до заданих правил (рис. 2.32). Вони визначають які плитки можна розміщувати поряд із заданою, а які ні. Таким чином при роботі з пензлем, Unity автоматично вираховує який Tile має бути наступним.

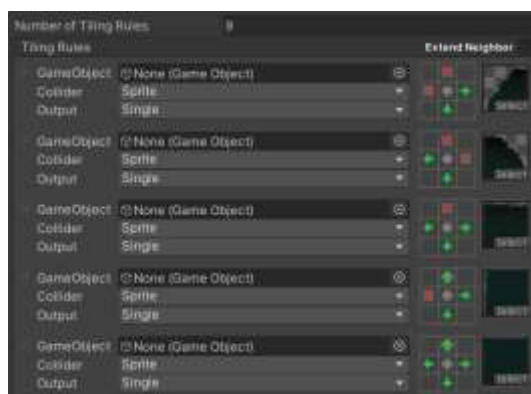


Рисунок 2.32 - Створене правило для першої палітри локації «Ліс»

Висновки до другого розділу

В другому розділі було спроектовано архітектуру ПП з використанням схем: варіантів використання, активності, компонентів, розгортання, а також діаграми класів. Також було спроектовано та розроблено інтерфейс користувача.

Була проведена розробка програмної частини додатку, описані основні методи та реалізація патернів. Після цього була описана реалізація візуальних ефектів, графічних елементів та анімацій. Інформація узагальнена у формі таблиць.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ

3.1 Порядок встановлення та налаштування ігрового додатку

Програмний продукт під назвою «Dethrone» було створено за допомогою ігрового рушія Unity, що дозволяє виконати збірку для різних пристроїв. У даному випадку це ПК з операційними системами – Windows, Mac, Linux.

Щоб отримати доступ до гри, спочатку необхідно завантажити файл із інтернету або лаунчера, наприклад Steam, після чого встановити його на свій ПК. Як було зазначено у концепт-документі, мінімальні налаштування для запуску додатку включають:

- Процесор із базовою частотою вище 2,5 ГГц.
- Відеокарту серії GTX.
- 8 Гб оперативної пам'яті.
- HDD або SSD, що має 5 Гб вільного місця.

3.2 Інтерфейс та порядок роботи

Перше вікно, що зустрічає користувача при запуску гри – головне меню (рис. 3.1). Управління відбувається за допомогою миші. Гравець може переходити до гри, налаштувань, досягнень або завершити роботу додатку. Якщо почати або продовжити гру – завантажиться сцена із рівнем, в залежності від наявності останнього збереження. При переході до налаштувань (рис. 3.2) гравець має змогу взаємодіяти із перемикачем режиму вікна, випадаючим списком розширень та слайдером гучності. Проте усі налаштування не можуть бути активними одночасно. У вікні досягнень (рис. 3.3) гравець може переглянути відкриті досягнення та їх опис, шляхом натискання на одну з іконок.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		54



Рисунок 3.1 - Головне меню

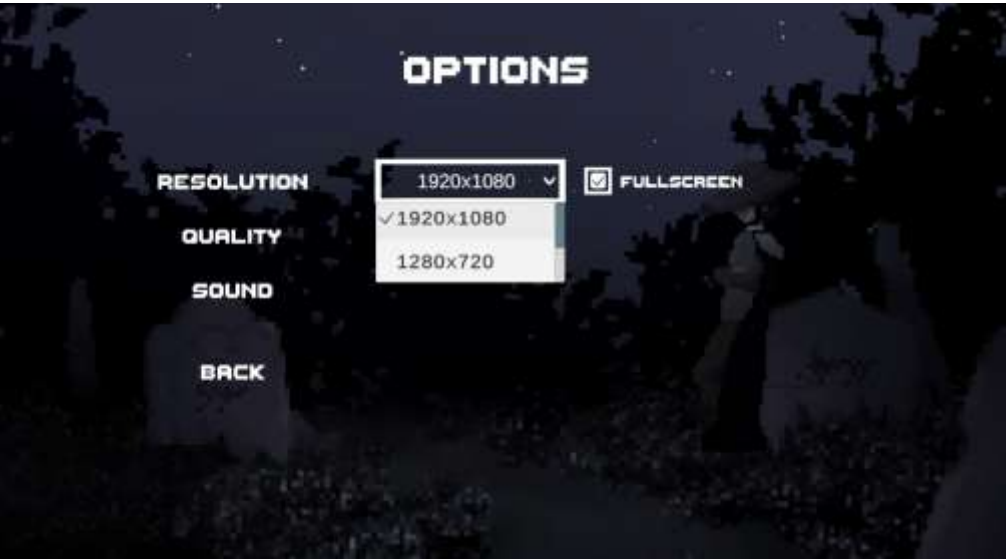


Рисунок 3.2 - Налаштування, зміна розширення та режиму екрану



Рисунок 3.3 - Вікно досягнень

При запуску рівня користувач опиняється на сцені, де присутні такі частини інтерфейсу як HUD, внутрішньоігрове меню та мапа. Перше, що можна побачити це індикатори здоров'я, мана, лікувальних засобів, а також шкалу заповнення ультимативного вміння (рис. 3.4). Вони відображають числові значення характеристик персонажу зручним та візуально зрозумілим способом. Також при використанні деяких із вмінь можуть з'являтися або зникати додаткові індикатори, такі як: колір обраної гілки стилю зброї або сфери (рис. 3.5). На початку нової гри користувач має змогу ознайомитись із керуванням за допомогою спеціального вікна, що зображене на рисунку 3.6. Якщо персонаж взаємодіє з NPC або знаходить предмет HUD також відображає це, як показано на рисунках 3.7 – 3.8.



Рисунок 3.4 - Ігровий інтерфейс гравця



Рисунок 3.5 - Додаткові індикатори HUD – сфери

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		56



Рисунок 3.6 - Навчання, показ клавiш для керування

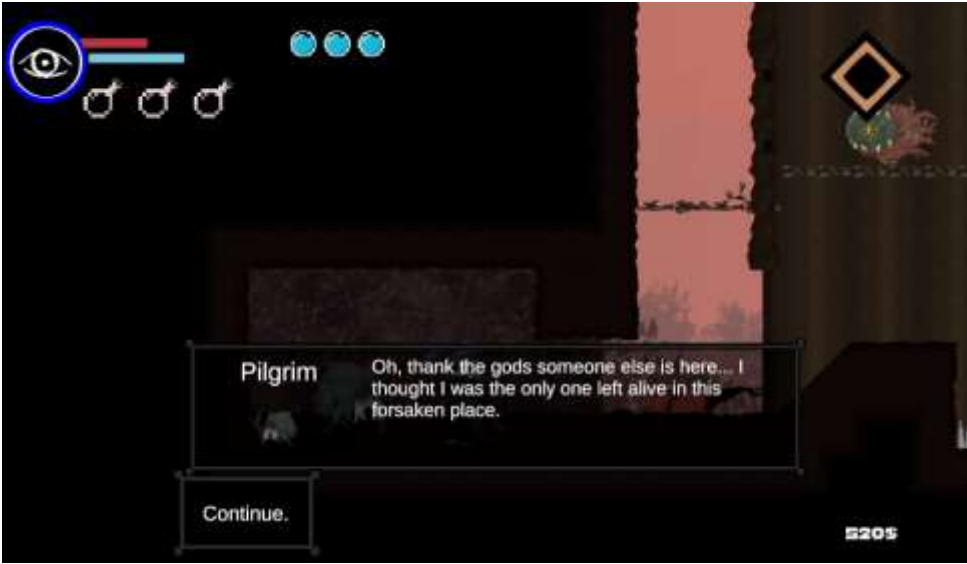


Рисунок 3.7 - Дiалог з NPC



Рисунок 3.8 - Знаходження предмету

Внутрішньоігрове меню відкривається та закривається при натисканні клавіші «Esc», або відповідної кнопки у формі «хрестика», та дозволяє обрати будь-яку наявну навичку, переглянути інформацію або налаштувати керування персонажем. Усі сторінки використовують клік миші та мають описи/кнопки для взаємодії з елементами (рис. 3.9 - 3.14). Усі елементи інтерфейсу позначені різними іконками (зображення зброї, навички або предмету).



Рисунок 3.9 - Сторінка характеристики та стилів зброї



Рисунок 3.10 - Сторінка активних навичок



Рисунок 3.11 - Сторінка пасивних навичок



Рисунок 3.12 - Сторінка талентів



Рисунок 3.13 - Інвентар



Рисунок 3.14 - Внутрішньоігрові налаштування, прив'язка клавіш

Мапа відкривається на клавішу «М» та відображає поточне положення персонажу на локації (рис. 3.15). Якщо герой знаходить місце збереження, не ігрового персонажа або боса – з'являється відповідна позначка на мапі. Також при переході до наступного рівня з'являється можливість перегляду повної карти світу. Тепер при натисканні клавіш «1» та «2» відповідно будуть змінюватись режими з локального на глобальний (рис. 3.16).

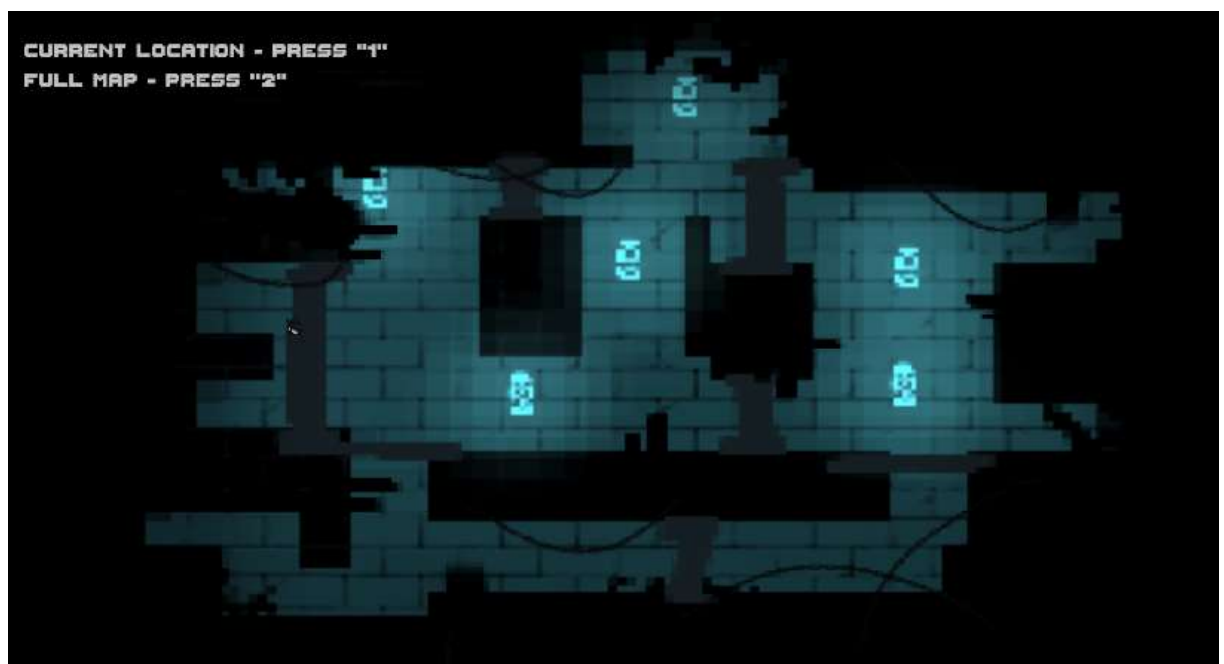


Рисунок 3.15 - Карта рівня із позначкою гравця

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		60



Рисунок 3.16 - Режим повної карти після відвідання другого рівня

Якщо гравець втрачає усе здоров'я – відображається «екран смерті», що пропонує нам повернутись до останнього збереження (рис. 3.17).

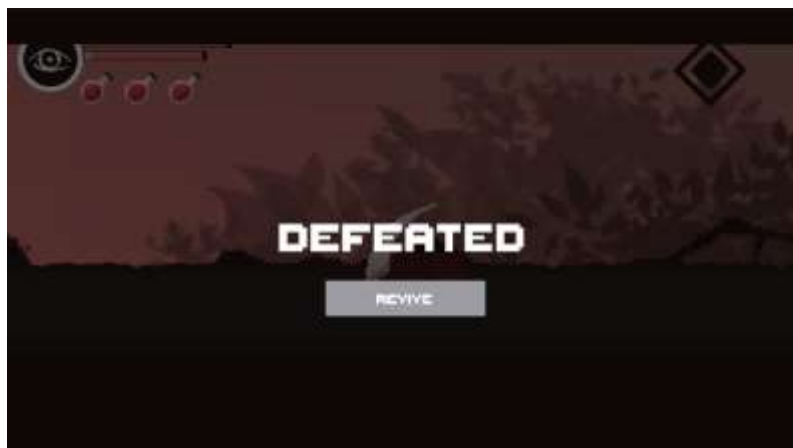


Рисунок 3.17 - Екран смерті

Боси також мають свій інтерфейс, а саме ім'я та здоров'я (рис. 3.18). Після перемоги він зникає.



Рисунок 3.18 - Інтерфейс боса

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

3.3 Тестування роботи ігрового додатку

Основною метою тестування продукту "Dethrone" є забезпечення високої якості гри, зменшення кількості помилок та недоліків перед релізом. Щоб досягти цього, необхідно проводити різні види тестувань, включаючи функціональні (Functional testing, GUI Testing) та нефункціональні (тестування установки та зручності користування).

Для тестової оцінки необхідно розбити проєкт на менші завдання та виділити час і зусилля, що необхідні для кожного. Це можна зробити за допомогою загального плану тестування. Документованого набору дій, що використовується для перевірки певного аспекту роботи програми. У ньому описується, що саме підлягає тестуванню та який результат очікується. Отримані результати зведені до таблиці 3.1.

Таблиця 3.1

Загальний план тестування роботи графічного інтерфейсу користувача

<i>Функція програмного забезпечення та тип тестування</i>	<i>Тест</i>	<i>Очікувані результати</i>	<i>Отримані результати</i>
<i>Основна функціональність головного меню. <u>Functional testing</u></i>	Запуск гри, перевірка завантаження збереження, зміна налаштувань, перегляд відкритих досягнень.	Коректний запуск сцен у відповідності зі збереженням, робота та автоматичне підтягування заданих значень налаштувань. Можливість перегляду відкритих досягнень.	Як очікувалось.

<i>Функція програмного забезпечення та тип тестування</i>	<i>Тест</i>	<i>Очікувані результати</i>	<i>Отримані результати</i>
<i>Графічний інтерфейс користувача, HUD <u>Functional testing</u></i>	Перевірка на відповідність числовим значенням, та коректну появу елементів.	Поява та зникнення елементів під час роботи із внутрішньоігровим меню та навичками. Відповідність візуальних характеристик числовим.	Як очікувалосьь.
<i>Основна функціональність внутрішньоігрового меню <u>Functional testing</u></i>	Екіпіювання навичок та перегляд інформації, зміна налаштувань та прив'язки клавіш.	Коректна робота усіх сторінок меню, автоматичне збереження обраних елементів, відображення та робота прив'язаних клавіш керування.	Як очікувалосьь.
<i>Відображення графічних елементів <u>GUI testing</u></i>	Перевірка відображення графіки та інтерфейсу.	Висока якість графіки та анімацій. Коректне відображення та розміщення ігрових елементів на екрані, зокрема на розширеннях 1920x1080 та вище. Зручне відображення інформації на екрані гравця.	Як очікувалосьь.
<i>Відповідь гри на різних пристроях <u>GUI testing</u></i>	Тест відображення на різних пристроях та конфігураціях.	Правильна робота гри на різних типах пристроїв та конфігураціях. Нормальна оптимізація інтерфейсу для різних пристроїв, включаючи ПК та ноутбуки.	Як очікувалосьь.

Під час тестування, як і під час розробки було виявлено багато помилок. Наприклад: некоректне відображення елементів, неправильна робота та послідовність перевірок, відсутність необхідних елементів тощо. Більшість із них була виправлена за допомогою регресійного тестування та тестування складання, що проводилось для кожної версії гри або після внесення змін у код.

Висновки до третього розділу

В третьому розділі було вказано порядок встановлення додатку для запуску, а також мінімальні характеристики для його коректної роботи. Показано фінальний результат графічного інтерфейсу, і детально описано порядок роботи із кожною частиною, а саме: головним меню, внутрішньоігровим меню, взаємодія з мапою або предметами.

Для спрощення процесу перевірки роботи фгрового додатку було наведено загальний план тестування з очікуваними та отриманими результатами, вказано типи проведених тестувань та їх ефективність.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

ВИСНОВКИ

В результаті роботи над кваліфікаційною роботою було досягнуто поставленої мети та розроблено ігровий програмний продукт «Dethrone» з наступними складовими: головне та внутрішньо-ігрове меню, інтерфейс для взаємодії з користувачем та передачею необхідної інформації, технологіями поліпшення візуальної частини та власними графічними елементами.

В першому розділі було обрано та проаналізовано три популярних додатки у жанрах Metroidvania та sous-like. На їх прикладі було визначено основні особливості та потреби гравців, які обов'язково мають бути включені у створюваний додаток. Створено концепт-документ в якому зазначено жанр, цільову аудиторію, платформу та загальний опис продукту. Також було проведено аналіз існуючого інструментарію для роботи з піксельною графікою, і як результат обрано програму, що найбільше задовольняє потреби проєкту.

У другому розділі за допомогою діаграм варіантів використання, активності, класів, компонентів та розгортання було створено і описано архітектуру додатку. Реалізовано та розписано основні етапи роботи над інтерфейсом та програмною частиною у Unity. Також наведено найважливіші методи.

В третьому розділі було детально описано процес завантаження ігрового додатку на ПК, та продемонстровано фінальний результат користувацького інтерфейсу. Був пояснений порядок роботи з інтерфейсом та меню, описані основні можливості користувача. Після цього були підведені підсумки тестування у вигляді загального плану. В додатках наведено лістинг найважливіших скриптів.

У результаті отримано робочий програмний продукт, що справно виконує усі необхідні дії і коректно відповідає на запити гравця.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What Does Metroidvania Mean? [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://www.thegamer.com/what-does-metroidvania-mean/>.
2. Феномен Soulslike жанру [Електронний ресурс] – Режим доступу до ресурсу: <https://migdb.com/uk/statti/phenomen-souls-like-zhanru>.
3. What is HUD [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: <https://www.webopedia.com/definitions/hud/>.
4. What is UI in games? [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://retrostylegames.com/blog/ui-game-meaning/>.
5. «Blasphemous» Home - The Game Kitchen [Електронний ресурс] – Режим доступу до ресурсу: <https://thegamekitchen.com/blasphemous/>.
6. «Hollow knight» Games - Team Cherry [Електронний ресурс] – Режим доступу до ресурсу: <https://www.teamcherry.com.au/games>.
7. «Dark Souls» [Електронний ресурс] – Режим доступу до ресурсу: <https://www.darksouls.jp>.
8. Марчук Г.В., Дяченко В.В. Важливість візуальної частини ігор. Комп'ютерні технології: інновації, проблеми, рішення: Тези VI Всеукраїнської науково-технічної конференції, 29-30 листопада 2023 року. Житомир: «Житомирська політехніка», 2023. С.153-155.
9. Aseprite - Animated sprite editor & pixel art tool [Електронний ресурс] – Режим доступу до ресурсу: <https://www.aseprite.org>.
10. Aseprite Animation Tutorial [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=B0enS9BJne4>.
11. How to infinite scrolling background - Unity Parallax [Електронний ресурс] – Режим доступу до ресурсу: youtube.com/watch?v=4xzj3imRd20.
12. Creating a 2D Parallax Effect in Unity [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://blog.yarsalabs.com/parallax-effect-in-unity-2d/>.

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

13. Поведінкові патерни - шаблонний метод [Електронний ресурс] – Режим доступу до ресурсу: <https://refactoring.guru/uk/design-patterns/template-method>.
14. Подорожувальні патерни - одинак [Електронний ресурс] – Режим доступу до ресурсу: <https://refactoring.guru/uk/design-patterns/singleton>.
15. Unity manual: Canvas 1.0.0 [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/Packages/com.unity.ugui@2.0/manual/class-Canvas.html>
16. Creating a UI Canvas in Unity [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: <https://frankgwarman.medium.com/creating-a-ui-canvas-in-unity-1a04ca9eef9e>.
17. Dialogue Editor [Електронний ресурс] – Режим доступу до ресурсу: <https://assetstore.unity.com/packages/tools/utilities/dialogue-editor-168329>.
18. Unity manual: particle system [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/Manual/class-ParticleSystem.html>.
19. Learn everything about particles in Unity [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=0HKSvT2gcuk>.
20. Universal Render Pipeline [Електронний ресурс] – Режим доступу до ресурсу: <https://unity.com/srp/universal-render-pipeline>.
21. Manual: Tilemap component reference [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/Manual/class-Tilemap.html>.
22. Introduction to tilemap [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: <https://medium.com/nerd-for-tech/introduction-to-tilemap-unity-394ec63f6e18>.

ДОДАТКИ

		Дяченко В.В.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Марчук Г.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

Логіка роботи головного меню

Скрипт MainMenuController.cs:

```
[System.Serializable]
public struct Achivement
{
    public Image image;
    public bool isOwned;
    public string description;
    public string name;
}

public class MainMenu : MonoBehaviour
{
    [SerializeField] private TMP_Dropdown resolutionDropdown;
    private string[] resolutions = new string[] { "1920x1080", "1280x720",
"1600x900", "2560x1440" };
    private bool isFullScreen;
    private int currentQualityLevel;
    private int width;
    private int height;
    private string currentResolution;

    public Achivement[] achivements = new Achivement[14];
    private int selectedAchivementIndex = -1;
    private string saveKey = "mainMenuSettings";
    private string saveKey2 = "playerPosition";
    private int sceneIndex;

    void Start()
    {
        Load();
        //currentResolution = $"{width}x{height}";

        descriptionText.text = "";
        nameText.text = "";
        UpdateAchivementImages();
    }

    public void ShowOptions()
    {
        gameMenu.SetActive(false);
        optionsMenu.SetActive(true);
        resolutionSettings.SetActive(false);
        qualitySettings.SetActive(false);
        soundSettings.SetActive(false);

        currentActiveSettings = null;
    }

    public void ActivateResolutionSettings()
    {
        ActivateSettings(resolutionSettings);
        resolutionDropdown.ClearOptions();
        resolutionDropdown.AddOptions(resolutions.ToList());

        resolutionDropdown.onValueChanged.AddListener(OnResolutionChanged);
    }
}
```

```

}
public void OnResolutionChanged(int resolutionIndex)
{
    string selectedResolution = resolutions[resolutionIndex];
    string[] resolutionParts = selectedResolution.Split('x');

    width = int.Parse(resolutionParts[0]);
    height = int.Parse(resolutionParts[1]);

    Screen.SetResolution(width, height, Screen.fullScreen);
    Debug.Log("Changed resolution to: " + width + "x" + height);
    Save();
}

public void ToggleFullScreen()
{
    isFullScreen = !isFullScreen;
    Screen.fullScreen = isFullScreen;
    if (isFullScreen)
    {
        fullscreenText.text = "Fullscreen";
    }
    else
        fullscreenText.text = "Window";
    Save();
}

public void ActivateQualitySettings()
{
    ActivateSettings(qualitySettings);

    lowQualityToggle.onValueChanged.AddListener(newValue =>
OnQualityToggleValueChanged(newValue, 0));
    mediumQualityToggle.onValueChanged.AddListener(newValue =>
OnQualityToggleValueChanged(newValue, 2));
    highQualityToggle.onValueChanged.AddListener(newValue =>
OnQualityToggleValueChanged(newValue, 5));
}

void UpdateToggles()
{
    lowQualityToggle.isOn = currentQualityLevel == 0;
    mediumQualityToggle.isOn = currentQualityLevel == 2;
    highQualityToggle.isOn = currentQualityLevel == 5;
}

void OnQualityToggleValueChanged(bool newValue, int qualityLevel)
{
    if (newValue)
    {
        currentQualityLevel = qualityLevel;
        QualitySettings.SetQualityLevel(currentQualityLevel);
        UpdateToggles();
        Save();
    }
    else if (!lowQualityToggle.isOn && !mediumQualityToggle.isOn &&
!highQualityToggle.isOn)
    {
        mediumQualityToggle.isOn = true;
        currentQualityLevel = 2;
    }
}

```

```

        QualitySettings.SetQualityLevel(currentQualityLevel);
        Save();
    }

}

public void ShowAchievements()
{
    gameMenu.SetActive(false);
    achievementsMenu.SetActive(true);
}

public void BackToMainMenu()
{
    optionsMenu.SetActive(false);
    achievementsMenu.SetActive(false);
    gameMenu.SetActive(true);

    descriptionText.text = "";
    nameText.text = "";
}

void UpdateDescriptionText(int achivementIndex)
{
    if (achivementIndex >= 0 && achivementIndex < achievements.Length &&
    achievements[achivementIndex].isOwned)
    {
        Debug.Log(achievements[achivementIndex].description);
        descriptionText.text = achievements[achivementIndex].description;
    }
}

void UpdateAchivementImages()
{
    for (int i = 0; i < achievements.Length; i++)
    {
        // Set image color based on whether the charm is purchased or not
        achievements[i].image.color = achievements[i].isOwned ? Color.white
: Color.gray;
    }
}

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

public void Save()
{
    SaveManager.Save(saveKey, GetData());
}

private void SetFull(bool status)
{
    isFullScreen = status;
}

public void StartGame()
{
    PlayerPrefs.DeleteKey("playerSave");
    SceneManager.LoadScene(1);
}

public void ContinueGame()
{
    var data2 = SaveManager.Load<SaveData.PlayerPos>(saveKey2);
    sceneIndex = data2._sceneIndex;
    SceneManager.LoadScene(sceneIndex);
}

```

```

private void Load()
{
    var data = SaveManager.Load<SaveData.MainSettings>(saveKey);
    SetFull(data._isFullScreen);
    Screen.fullScreen = isFullScreen;
    fullscreenText.text = isFullScreen ? "Fullscreen" : "Window";

    currentQualityLevel = data._currentQualityLevel;
    UpdateToggles();
    if (data._width == 0 || data._height == 0)
    {
        string defaultResolution = resolutions[0];
        string[] resolutionParts = defaultResolution.Split('x');
        width = int.Parse(resolutionParts[0]);
        height = int.Parse(resolutionParts[1]);
    }
    else
    {
        width = data._width;
        height = data._height;
    }

    Screen.SetResolution(width, height, Screen.fullScreen);
    volumeSlider.value = data._volumeValue;

    string savedResolution = $"{width}x{height}";

    resolutionDropdown.ClearOptions();
    resolutionDropdown.AddOptions(resolutions.ToList());

    int savedResolutionIndex = Array.IndexOf(resolutions,
savedResolution);
    if (savedResolutionIndex != -1)
    {
        resolutions = resolutions.OrderBy(r => r !=
savedResolution).ToArray();
    }
    else
    {
        resolutionDropdown.value = 0; // Встановлюємо перше значення за
замовчуванням
    }
}

private SaveData.MainSettings GetData()
{
    var data = new SaveData.MainSettings()
    {
        _isFullScreen = isFullScreen,
        _currentQualityLevel = currentQualityLevel,
        _height = height,
        _width = width,
        _volumeValue = volumeSlider.value,
    };
    return data;
}

```

Логіка роботи внутрішньої ігрового меню

Скрипт PauseMenuController.cs:

```

public Button[] navigationButtons;
public Text currentEssences;
private int essenceAmount;
private int increaseRate = 100;
[SerializeField] private Text essenceText;

private void Start()
{
    // Initial state: menu is hidden
    if (pauseMenu != null)
        pauseMenu.SetActive(false);
    gui.SetActive(true);
    postProcessing.SetActive(true);

    // Ensure the arrays are of the same length
    if (menuButtons.Length == menuPanels.Length &&
        navigationButtons.Length == 2)
    {
        // Map buttons to their respective panels
        for (int i = 0; i < menuButtons.Length; i++)
        {
            int index = i; // Capture the current value of i
            menuButtons[i].onClick.AddListener(() =>
                ShowPanel(menuPanels[index]));
        }

        navigationButtons[0].onClick.AddListener(ShowPreviousPanel);
        navigationButtons[1].onClick.AddListener(ShowNextPanel);
    }
    else
    {
        Debug.LogError("The number of buttons does not match the number of
panels or navigation buttons are not set.");
    }
    essenceAmount = PlayerManager.instance.GetEssenceAmount();
    Debug.Log(essenceAmount);
    currentEssences.text = $"{essenceAmount}$";
    essenceText.text = $"{essenceAmount}$";
}

private void Update()
{
    // Check if the Escape key is pressed
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        TogglePauseMenu();
    }

    if (closeButton != null)
    {
        if (closeButton.onClick != null)
            closeButton.onClick.RemoveAllListeners();

        closeButton.onClick = new Button.ButtonClickedEvent();

```

```

        closeButton.onClick.AddListener(TogglePauseMenu);
    }
    essenceAmount = PlayerManager.instance.GetEssenceAmount();
    currentEssences.text = $"{essenceAmount}$";
    essenceText.text = $"{essenceAmount}$";
}

int SetEssenceAmount(int value) {
    int currentEssences = value;
    essenceText.text = currentEssences.ToString();
    return currentEssences;
}

public int GetEssenceAmount(int value)
{
    value = essenceAmount;
    return value;
}

public void UseEssenceAmount(int amount)
{
    essenceAmount -= amount;
    PlayerManager.instance.RemoveEssences(amount);
    SetEssenceAmount(essenceAmount);
    currentEssences.text = $"{essenceAmount}$";
}

// Method to toggle the visibility of the pause menu and control the
game's pause state
void TogglePauseMenu()
{
    if (pauseMenu != null)
    {
        bool isMenuActive = pauseMenu.activeSelf;
        bool isGuiActive = gui.activeSelf;
        bool isPostActive = postProcessing.activeSelf;
        if (map != null)
        {
            bool isMapActive = map.activeSelf;
            map.SetActive(!isMapActive);
        }
        pauseMenu.SetActive(!isMenuActive);
        gui.SetActive(!isGuiActive);
        postProcessing.SetActive(!isPostActive);

        // Pause or resume the game based on the menu's visibility
        Time.timeScale = isMenuActive ? 1f : 0f;
    }
}

// Method to show a specific panel and hide others
void ShowPanel(GameObject panelToShow)
{
    foreach (var panel in menuPanels)
    {
        panel.SetActive(panel == panelToShow);
    }
}

// Method to show the previous panel
void ShowPreviousPanel()
{
    int currentIndex = GetCurrentPanelIndex();
    int previousIndex = (currentIndex - 1 + menuPanels.Length) %
menuPanels.Length;
    ShowPanel(menuPanels[previousIndex]);
}

```

```

}

// Method to show the next panel
void ShowNextPanel()
{
    int currentIndex = GetCurrentPanelIndex();
    int nextIndex = (currentIndex + 1) % menuPanels.Length;
    ShowPanel(menuPanels[nextIndex]);
}

// Helper method to get the index of the currently active panel
int GetCurrentPanelIndex()
{
    for (int i = 0; i < menuPanels.Length; i++)
    {
        if (menuPanels[i].activeSelf)
        {
            return i;
        }
    }

    return -1; // Default value if no panel is active
}

```

Скрипт CharacterPanel.cs:

```

private SphereGUI sphereGUI;
private int lastPicked;
private int currentStyle;
private int selectedSkillIndex = -1;
bool allRequiredSkillsPurchased;
private string saveKey = "PlayerWeaponSkills";
private int isEqiped;

private void Awake()
{
    weaponStyles = FindObjectsOfType<WeaponStyle>();
    sphereGUI = FindAnyObjectByType<SphereGUI>();
}

private void Start()
{
    pauseMenuScript = GetComponent<PauseMenuScript>();

    if (pauseMenuScript == null)
    {
        Debug.LogError("PauseMenuScript not found!");
        return;
    }
    Load();
    currentStyle = 1;
    descriptionText.text = "select weapon";
    Prompt.text = "";
    UpdateSkillImages();
    SetStats(playerStats.currentHealth, playerStats.GetMaxHealthValue());
}

private void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        SetStats(playerStats.currentHealth,
playerStats.GetMaxHealthValue());
        descriptionText.text = "select weapon";
        Prompt.text = "";
    }
}

```

```

    }
}

private void UpdateSkillImages()
{
    foreach (var skill in skills)
    {
        if (skill.image != null)
        {
            if (skill.isPurchased)
            {
                skill.image.color = Color.white;
            }
            else
            {
                skill.image.color = Color.gray;
            }
        }
    }
}

private void EquipSkill(int skillIndex)
{
    if(skillIndex >= 0)
    {
        WeaponSkill selectedSkill = skills[skillIndex];
        if (selectedSkill.isBasicSkill)
        {
            int pickedStyle = skillIndex / 3;
            if (pickedStyle != lastPicked)
            {
                weaponStyles[lastPicked].DeactivateEffect();
                lastPicked = pickedStyle;
                bool isUpgradedOnce = false;
                bool isUpgradedTwice = false;

                if (isUpgradedTwice)
                {
                    SetStyle(skillIndex);
                    Prompt.text = "skill equipped";
                    Debug.Log("Equipped Skill " + skillIndex + " with two
upgrades");

                    weaponStyles[pickedStyle].ActivateFirstUpgrade();
                    weaponStyles[pickedStyle].ActivateSecondUpgrade();
                    weaponStyles[pickedStyle].ActivateThirdUpgrade();
                }
                else if (isUpgradedOnce)
                {
                    SetStyle(skillIndex);
                    Prompt.text = "skill equipped";
                    Debug.Log("Equipped Skill " + skillIndex + " with one
upgrade");

                    weaponStyles[pickedStyle].ActivateFirstUpgrade();
                    weaponStyles[pickedStyle].ActivateSecondUpgrade();
                }
            }
            else
            {
                SetStyle(skillIndex);
                Prompt.text = "skill equipped";
                Debug.Log("Equipped Skill " + skillIndex + " without
upgrades");

                weaponStyles[pickedStyle].ActivateFirstUpgrade();
            }
        }
    }
}

```

```

    }
    else
    {
        int baseSkillIndex = GetBaseSkillIndex(skillIndex);

        Debug.Log("set style color" + baseSkillIndex);
        Debug.Log("Skill Index: " + skillIndex + " is not a basic
skill and cannot be equipped directly.");
        isEquiped = baseSkillIndex;
        EquipSkill(baseSkillIndex);
    }
}
Save();
}

```

```

private bool GetRequiedSkills(WeaponSkill currentSkill)
{
    foreach (int requiredSkillIndex in currentSkill.requiredSkill)
    {
        if (requiredSkillIndex >= 0 && requiredSkillIndex < skills.Length)
        {
            if (!skills[requiredSkillIndex].isPurchased)
            {
                Debug.Log("no");
                return false;
            }
        }
        else
        {
            return false;
        }
    }
    Debug.Log("yes");
    return true;
}

```

```

public void PurchaseSkill()
{
    int skillIndex = selectedSkillIndex;
    if (skillIndex < skills.Length && skillIndex != -1)
    {
        WeaponSkill currentSkill = skills[skillIndex];

        allRequiredSkillsPurchased = GetRequiedSkills(currentSkill);
        if (allRequiredSkillsPurchased)
        {
            if (!currentSkill.isPurchased)
            {
                int skillCost = currentSkill.cost;

                if (pauseMenuScript.GetEssenceAmount(0) >= skillCost)
                {
                    pauseMenuScript.UseEssenceAmount(skillCost);
                    Prompt.text = "skill purshcased";
                    SetSkillPurchased(skillIndex, true);
                    EquipSkill(skillIndex);
                    isEquiped = skillIndex;
                    Debug.Log("Purchased Skill Index: " + skillIndex);
                    confrimText.text = "Confrim";
                    if (!skills[skillIndex].isBasicSkill)
                    {
                        confrim.SetActive(false);
                    }
                }
            }
        }
    }
}

```

```

        }
        else
        {
            descriptionText.text = "Unable to buy, not enough
money";
        }
    }
    else
    {
        Debug.Log("Skill Index: " + skillIndex + " is already
purchased.");
        EquipSkill(skillIndex);
        isEqiped = skillIndex;
        SetStyle(skillIndex);
    }
}
else
{
    descriptionText.text = "Unable to buy, required skill is not
purchased";
}
UpdateSkillImages();
Save();
}
}

public void OnSkillImageClick(int skillIndex)
{
    WeaponSkill currentSkill = skills[skillIndex];
    allRequiredSkillsPurchased = GetRequiedSkills(currentSkill);

    int clickedStyle = skillIndex / 3 + 1;
    if (clickedStyle != currentStyle)
    {
        currentStyle = clickedStyle;
        UpdateDescriptionText(0);
    }

    int descriptionIndex = skillIndex % 3;
    UpdateDescriptionText(descriptionIndex);

    if (selectedSkillIndex != -1)
    {
        UpdateSkillImageScale(selectedSkillIndex, 1f);
    }

    UpdateSkillImageScale(skillIndex, 1.1f);
    selectedSkillIndex = skillIndex;
}

private void UpdateSkillImageScale(int skillIndex, float scale)
{
    WeaponSkill currentSkill = skills[skillIndex];
    for (int i = 0; i < skills.Length; i++)
    {
        // Reduce the scale of all skill images except the clicked one
        if (i != skillIndex)
        {
            skills[i].image.transform.localScale = new Vector3(1f, 1f,
1f);
        }
        else
        {
            if (!currentSkill.isPurchased || currentSkill.isBasicSkill)

```

```

        {
            skills[skillIndex].image.transform.localScale = new
Vector3(scale, scale, scale);
        }
        else
        {
            skills[skillIndex].image.transform.localScale = new
Vector3(1f, 1f, 1f);
        }
    }
}

private void UpdateDescriptionText(int skillIndex)
{
    int descriptionIndex = skillIndex;

    string[] descriptions = null;

    if (currentStyle >= 1 && currentStyle <= 3)
    {
        int styleIndex = currentStyle - 1;
        descriptions = new string[] { skills[styleIndex * 3].description,
skills[styleIndex * 3 + 1].description, skills[styleIndex * 3 + 2].description
};
    }

    if (descriptions != null && descriptionIndex >= 0)
    {
        descriptionText.text = descriptions[descriptionIndex];
    }
    else
    {
        Debug.LogError("Invalid description index: " + descriptionIndex);
    }
}

private void SetStatTextFloat(int index, float currentValue, float maxValue)
{
    if (index >= 0 && index < statTextsFloat.Length)
    {
        statTextsFloat[index].text = string.Format("{0}/{1}",
currentValue, maxValue);
    }
}

public void Save()
{
    SaveManager.Save(saveKey, GetData());
}

private void Load()
{
    var data = SaveManager.Load<SaveData.WeaponSave>(saveKey);

    isEquiped = data._isEquiped;

    for (int i = 0; i < skills.Length; i++)
    {
        skills[i].isPurchased = data._isOwned[i];
    }

    UpdateSkillImages();
}

```

```

        if(isEqiped!= -1)
        {
            EquipSkill(isEqiped);
            SetStyle(isEqiped);
        }
    }

    private SaveData.WeaponSave GetData()
    {

        var data = new SaveData.WeaponSave()
        {
            _isEqiped = isEqiped,
            _isOwned = new bool[skills.Length]
        };

        for (int i = 0; i < skills.Length; i++)
        {
            data._isOwned[i] = skills[i].isPurchased;
        }

        return data;
    }

```

Скрипт AbilitiesPanel.cs:

```

private int selectedAbilityIndex = -1;
private bool equipped = false;
private int index;
private int abilityIndex;
private string saveKey = "PlayerAbilities";
void Start()
{
    descriptionText.text = "select ability";
    nameText.text = "";
    promptText.text = "";
    UpdateAbilityImages();
    Load();
}

// Update is called once per frame
void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        if (selectedAbilityIndex != -1)
        {
            abilities[selectedAbilityIndex].image.transform.localScale =
new Vector3(1f, 1f, 1f);
        }
        descriptionText.text = "select ability";
        nameText.text = "";
        promptText.text = "";
        UpdateAbilityImages();
    }
}

public void SetAbilityOwned(int index)
{
    abilities[index].isOwned = true;
    Save();
}

public void EquipAbility()
{
    if (selectedAbilityIndex != -1)

```

```

    {
        abilityIndex = selectedAbilityIndex;
        index = abilityIndex;
    }

    int equippedAbilityIndex = FindIndexOfEquippedAbility();
    if (abilities[abilityIndex].isOwned && equipped == false)
    {
        abilities[abilityIndex].isEquipped = true;
        equipped = true;
        UpdateDescriptionText(abilityIndex);
        nameText.text = abilities[abilityIndex].name;
        promptText.text = "equiped " + nameText.text;
        equippedText.text = "Equiped: " + nameText.text;
    }
    else {
        if (abilities[abilityIndex].isOwned &&
!abilities[abilityIndex].isEquipped)
        {
            abilities[abilityIndex].isEquipped = true;
            abilities[equippedAbilityIndex].isEquipped = false;
            UpdateDescriptionText(abilityIndex);
            nameText.text = abilities[abilityIndex].name;
            promptText.text = "replaced with " + nameText.text;
            equippedText.text = "Equiped: " + nameText.text;
        }
        else if (abilities[abilityIndex].isOwned &&
abilities[abilityIndex].isEquipped)
        {
            abilities[abilityIndex].isEquipped = false;
            equipped = false;
            promptText.text = "unequiped " + nameText.text;
            equippedText.text = "Equiped: " ;
            index = -1;
        }
        else
        {
            descriptionText.text = "You do not own this ability yet";
        }
    }
    Save();
}

private int FindIndexOfEquippedAbility()
{
    for (int i = 0; i < abilities.Length; i++)
    {
        if (abilities[i].isEquipped)
        {
            return i;
        }
    }
    return -1;
}

```

Скрипт CharmsPanel.cs:

```

private int lastClickedIndex = -1;

private string saveKey = "PlayerCharms";
public static CharmsPanelScript instance;

private void Awake()
{
    charmsArray = FindObjectsOfType<Charm>();
}

```

```

        pauseMenuScript = GetComponent<PauseMenuScript>();

        if (instance != null)
        {
            Destroy(instance.gameObject);
        }
        else
        {
            instance = this;
        }
    }

    public void OnEquippedCharmImageClick(int charmIndex)
    {
        selectedCharmIndex = GetEquippedCharmInList(charmIndex);
        if (selectedCharmIndex != -1)
        {
            Prompt.text = "unequip charm?";
            UpdateDescriptionText(selectedCharmIndex);
        }
    }

    public void OnCharmImageClick(int charmIndex)
    {
        Prompt.text = "";
        confirm.SetActive(true);
        UpdateDescriptionText(charmIndex);

        if (selectedCharmIndex != -1)
        {
            charms[selectedCharmIndex].image.transform.localScale = new
            Vector3(1f, 1f, 1f);
        }

        charms[charmIndex].image.transform.localScale = new Vector3(1.1f,
        1.1f, 1.1f);
        selectedCharmIndex = charmIndex;
    }

    private void UpdateCharmImages()
    {
        for (int i = 0; i < charms.Length; i++)
        {
            // Set image color based on whether the charm is purchased or not
            charms[i].image.color = charms[i].isOwned ? Color.white :
            Color.black;
        }
    }

    public void PurchaseCharm()
    {
        int charmIndex = selectedCharmIndex;
        if (charmIndex >= 0 && charmIndex < charms.Length)
        {
            CharmInfo currentCharm = charms[charmIndex];
            if (currentCharm.isOwned)
            {
                if (!currentCharm.isEquipped)
                {
                    EquipCharm(charmIndex);
                }
                else UnequipCharm(charmIndex);
            }
            else
            {
                if (currentCharm.cost == 0 ||
                pauseMenuScript.GetEssenceAmount(0) < currentCharm.cost)

```

```

        {
            Prompt.text = "Charm unowned";
            descriptionText.text = "This charm is not available for
purchase.";
        }
        else
        {
            pauseMenuScript.UseEssenceAmount(currentCharm.cost);
            Prompt.text = "charm purshcased";
            charms[charmIndex].isOwned = true;
            Debug.Log(currentCharm.isOwned);
            UpdateCharmImages();
        }
    }
}
else
{
    descriptionText.text = "You have not selected a charm.";
}

UpdateCharmImages();
Save();
}

public void SetCharmOwned(int charmIndex)
{
    CharmInfo[] updatedCharms = charms;

    if (charmIndex < updatedCharms.Length)
    {
        updatedCharms[charmIndex].isOwned = true;
        charms = updatedCharms;

        // Add logic to handle the charm discovery
        if (true)
        {
            Debug.Log("You found Charm Index: " + charmIndex);
        }
    }
    Save();
}

private void UnequippedCharm(int charmIndex)
{
    selectedCharmIndex = -1;
    nameText.text = "";
    descriptionText.text = "select skill";
    Prompt.text = "charm unequipped";
    CharmInfo currentCharm = charms[charmIndex];
    charms[charmIndex].isEquipped = false;

    for (int i = 0; i < equippedCharms.Length; i++)
    {
        if (currentCharm.description == equippedCharms[i].description)
        {
            equippedCharms[i].image = null;
            equippedCharms[i].description = "";
            equippedCharms[i].isEquipped = false;
            equippedCharms[i].isUnique = false;
            equippedCharms[i].isOwned = false;
            equippedCharms[i].cost = 0;
            DeleteColorForSlot(i);
        }
    }
    if (!charms[charmIndex].isEquipped && charms != null)

```

```

    {
        Debug.Log("deactivated " + charmsArray[charmIndex]);
        charmsArray[charmIndex].DeactivateEffect();
    }
    Save();
}

private void EquipCharm(int charmIndex)
{
    if (charms[charmIndex].isUnique)
    {
        int equippedUniqueIndex = FindIndexOfEquippedUniqueCharm();
        int previousIndex = FindIndexOfEquippedUniqueCharmInList();
        if (equippedUniqueIndex != -1)
        {
            Debug.Log("found unique ");
            if
(!equippedCharms[equippedUniqueIndex].Equals(charms[charmIndex]))
            {
                charms[previousIndex].isEquiped = false;
                equippedCharms[equippedUniqueIndex] = charms[charmIndex];

                charms[charmIndex].isEquiped = true;
                SetColorForSlot(equippedUniqueIndex);
                UpdateDescription(equippedUniqueIndex);
            }
            return;
        }
    }
    int freeSlot = FindFreeSlot();

    if (freeSlot != -1)
    {
        equippedCharms[freeSlot] = charms[charmIndex];

        charms[charmIndex].isEquiped = true;
        SetColorForSlot(freeSlot);
        UpdateDescription(freeSlot);
    }

    if (charms[charmIndex].isEquiped)
    {
        Prompt.text = "charm equipped";
        Debug.Log("activated" + charmsArray[charmIndex]);
        charmsArray[charmIndex].ActivateEffect();
    }
}

private void SetColorForSlot(int slotIndex)
{
    // check if slot has charm in it
    if (slotIndex >= 0 && slotIndex < equippedCharms.Length)
    {
        switch (slotIndex)
        {
            case 0:
                Image_one.sprite = equippedCharms[slotIndex].image.sprite;
                break;
            case 1:
                Image_two.sprite = equippedCharms[slotIndex].image.sprite;
                break;
            case 2:
                Image_three.sprite =
equippedCharms[slotIndex].image.sprite;
                break;
        }
    }
}

```

```
    }
}
```

Скрипт TalantsPanel.cs:

```
private string saveKey = "PlayerTalants";
public static TalantsPanelScript instance;

private void Awake()
{
    if (instance != null)
    {
        Destroy(instance.gameObject);
    }
    else
    {
        instance = this;
    }
}

public void OnTalantImageClick(int talantIndex)
{
    UpdateDescriptionText(talantIndex);

    if (selectedTalantIndex != -1)
    {
        talants[selectedTalantIndex].image.transform.localScale = new
Vector3(1f, 1f, 1f);
    }
    if (talants[talantIndex].isOwned) {
        talants[talantIndex].image.transform.localScale = new
Vector3(1.1f, 1.1f, 1.1f);
        selectedTalantIndex = talantIndex;
        nameText.text = talants[talantIndex].name;
    }
}

public void SetTalantOwned(int talantIndex)
{
    talants[talantIndex].isOwned = true;
    Debug.Log("You found talant Index: " + talantIndex);
    UpdateTalantImages();
    Save();
}

public void Save()
{
    SaveManager.Save(saveKey, GetData());
}

private void Load()
{
    var data = SaveManager.Load<SaveData.TalantsSave>(saveKey);

    for (int i = 0; i < talants.Length; i++)
    {
        talants[i].isOwned = data._isOwned[i];
    }
    UpdateTalantImages();
}
```

Скрипт InventoryPanel.cs:

```
private void FindItems()
{
    for (int i = 0; i < items.Length; i++)
    {
        if (items[i].isOwned && !ItemExistsInInventory(i))
        {
            GameObject newItemObject = new GameObject("Item_" + i);
            Image newItemImage = newItemObject.AddComponent<Image>();
            Button newItemButton = newItemObject.AddComponent<Button>();

            newItemImage.overrideSprite = items[i].sprite;
            newItemImage.rectTransform.localScale = scale;

            newItemImage.rectTransform.sizeDelta = imageSize;

            newItemObject.transform.SetParent(layoutInventory.transform);
            newItemImage.rectTransform.localScale = scale;
            newItemImage.rectTransform.localPosition = new
Vector3(newItemImage.rectTransform.localPosition.x,
            newItemImage.rectTransform.localPosition.y, 0f);

            int index = i;
            newItemButton.onClick.AddListener(() => OnItemClick(index));
        } else if (!items[i].isOwned)
        {
            RemoveItemFromInventory(i);
        }
    }
}

bool ItemExistsInInventory(int itemId)
{
    for (int i = 0; i < layoutInventory.transform.childCount; i++)
    {
        Transform child = layoutInventory.transform.GetChild(i);
        if (child.name == "Item_" + itemId.ToString())
        {
            return true;
        }
    }

    return false;
}

private void RemoveItemFromInventory(int itemId)
{
    for (int i = 0; i < layoutInventory.transform.childCount; i++)
    {
        Transform child = layoutInventory.transform.GetChild(i);
        if (child.name == "Item_" + itemId.ToString())
        {
            Destroy(child.gameObject);
            return;
        }
    }
}
```

```

private void OnItemClick(int itemsIndex)
{
    Debug.Log(itemsIndex);
    UpdateDescriptionText(itemsIndex);
    nameText.text = items[itemsIndex].name;

    if (selecteditemIndex != -1 && selecteditemIndex <=
layoutInventory.transform.childCount)
    {
        int prevIndex = FindObjectId(selecteditemIndex);
        layoutInventory.transform.GetChild(prevIndex).localScale = new
Vector3(1f, 1f, 1f);
    }

    int currentIndex = FindObjectId(itemsIndex);
    if (currentIndex != -1 && currentIndex <=
layoutInventory.transform.childCount)
    {
        layoutInventory.transform.GetChild(currentIndex).localScale = new
Vector3(1.1f, 1.1f, 1.1f);
    }

    selecteditemIndex = itemsIndex;
}

private int FindObjectId(int index)
{
    string itemName = "Item_" + index;

    for (int i = 0; i < layoutInventory.transform.childCount; i++)
    {
        Transform child = layoutInventory.transform.GetChild(i);
        if (child.name == itemName)
        {
            return i;
        }
    }

    return -1;
}

public void SetItemOwned(int index)
{
    items[index].isOwned = true;
    FindItems();
    Save();
}

```

Скрипт MovementKey.cs:

```

void SetupButton(Button button, string defaultKey)
{
    button.onClick.AddListener(() => StartKeyAssignment(button));
    UpdateButtonText(button, defaultKey);
    buttonKeyMap.Add(button, GetKeyCodeFromString(defaultKey));
}

void StartKeyAssignment(Button button)
{
    StartCoroutine(WaitForKeyPress(button));
}

```

```

IEnumerator WaitForKeyPress(Button button)
{
    KeyCode currentKey = KeyCode.None;

    while (currentKey == KeyCode.None)
    {
        yield return null;

        foreach (KeyCode keyCode in
System.Enum.GetValues(typeof(KeyCode)))
        {
            if (Input.GetKeyDown(keyCode) && !IsKeyAssigned(keyCode))
            {
                currentKey = keyCode;
                UpdateKey(button, currentKey);
                break;
            }
        }
    }
}

bool IsKeyAssigned(KeyCode newKeyCode)
{
    return buttonKeyMap.ContainsValue(newKeyCode);
}

void UpdateKey(Button button, KeyCode newKeyCode)
{
    buttonKeyMap[button] = newKeyCode;
    UpdateButtonText(button, GetKeyText(newKeyCode));
}

void UpdateButtonText(Button button, string keyText)
{
    button.GetComponentInChildren<Text>().text = keyText;
}

string GetKeyText(KeyCode keyCode)
{
    switch (keyCode)
    {
        case KeyCode.Mouse0:
            return "LM";
        case KeyCode.Mouse1:
            return "RM";

        default:
            return keyCode.ToString();
    }
}

KeyCode GetKeyCodeFromString(string keyString)
{
    if (keyString == "LM") return KeyCode.Mouse0;
    if (keyString == "RM") return KeyCode.Mouse1;

    KeyCode keyCode;
    if (System.Enum.TryParse(keyString, out keyCode))
    {
        return keyCode;
    }
    else
    {
        return KeyCode.None
    }
}

```

Логіка роботи тригерів

Скрипт CameraGhangeTrigger.cs:

```

private void Awake()
{
    Load();
    mainCamera.gameObject.SetActive(isMainCameraActive);
    alternativeCamera.gameObject.SetActive(!isMainCameraActive);
}

private void TriggerAction()
{
    isMainCameraActive = !isMainCameraActive;
    mainCamera.gameObject.SetActive(isMainCameraActive);
    alternativeCamera.gameObject.SetActive(!isMainCameraActive);

    if (otherCameraTrigger != null)
    {
        otherCameraTrigger.SetCameraState(!isMainCameraActive);
    }

    Save();
}

public void SetCameraState(bool state)
{
    isMainCameraActive = state;
    mainCamera.gameObject.SetActive(isMainCameraActive);
    alternativeCamera.gameObject.SetActive(!isMainCameraActive);
    Save();
}

private void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Player"))
    {
        TriggerAction();
    }
}

```

Скрипт Collision.cs:

```

private void Update()
{
    if (!status && isPlayerNearby && Input.GetKeyDown(KeyCode.E))
    {
        inventoryPanelScript.SetItemOwned(itemIndex);
        gameObject.SetActive(false);
        message.text = "";
    }
    else if (status)
    {
        gameObject.SetActive(false);
    }
}

private void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Player"))

```

```

        {
            isPlayerNearby = true;
            message.text = text;
        }
    }

    private void OnTriggerExit2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            isPlayerNearby = false;
            message.text = "";
        }
    }

    private void Load()
    {
        var data = SaveManager.Load<SaveData.ItemSave>(saveKey);
        status = data._isOwned[itemIndex];
    }

```

Скрипт DeathController.cs:

```

private void Start()
{
    Load();
    player = FindAnyObjectByType<Player>();
    barsController = FindAnyObjectByType<BarsController>();
    healthFlask = FindAnyObjectByType<HealthFlask>();
    moneyTrigger = FindAnyObjectByType<MoneyTrigger>();
}

private void Update()
{
    if (player.stats.isDead)
    {
        playerDeathPoint = player.transform.position;
        deathScreen.SetActive(true);
    }
}

public void Respawn()
{
    player.stats.isDead = false;
    lostCurrency = PlayerManager.instance.essenceAmount;
    PlayerManager.instance.RemoveEssences(lostCurrency);
    deathScreen.SetActive(false);
    money.transform.position = playerDeathPoint;

    status = true;

    money.SetActive(status);
    Save2();
    currency.GetCurrency(lostCurrency);
    player.stats.currentHealth = player.stats.maxHealth.GetValue();
    healthFlask.currentFlasks = healthFlask.maxFlasks;
    healthFlask.Save();
    healthFlask.Load();
    barsController.SetHealth(player.stats.currentHealth,
player.stats.maxHealth.GetValue());
    index = SceneManager.GetActiveScene().buildIndex;
    Save();
}

```

Скрипт ScreenFade.cs:

```
private void Awake()
{
    if (fadeImage == null)
    {
        fadeImage = GetComponent<Image>();
    }
}

public void FadeOutAndChangeScene(int sceneIndex)
{
    StartCoroutine(FadeOut(sceneIndex));
}

private IEnumerator FadeOut(int sceneIndex)
{
    float elapsedTime = 0f;
    Color color = fadeImage.color;

    while (elapsedTime < fadeDuration)
    {
        elapsedTime += Time.deltaTime;
        color.a = Mathf.Clamp01(elapsedTime / fadeDuration);
        fadeImage.color = color;
        yield return null;
    }

    SceneManager.LoadScene(sceneIndex);
}
```

Скрипт DoorTrigger.cs:

```
private IEnumerator UpdateAnimation()
{
    yield return new WaitForSeconds(1f);
    doorAnimator.SetBool("isFalling", false);
}

private void Update()
{
    if (isOpen)
    {
        doorAnimator.SetBool("isStaying", true);
    }
}

private void OnTriggerEnter2D(Collider2D other)
{
    Debug.Log("Open");
    OpenDoor();
    StartCoroutine(UpdateAnimation());
    isOpen = true;
    Save();
}

private void OpenDoor()
{
    if (doorAnimator != null)
    {
        doorAnimator.SetBool("isFalling", true);
    }
}
```

Логіка роботи паралаксу та мапи

Скрипт ParallaxBehavior.cs:

```

public class ParallaxBehavior : MonoBehaviour
{
    [SerializeField] private Transform followingTarget;
    [SerializeField, Range(-1f, 1f)] private float parallaxStrength = 0.1f;
    [SerializeField] private bool disableVerticalParallax;
    Vector3 targetPreviousPosition;

    void Start()
    {
        if (!followingTarget)
            followingTarget = Camera.main.transform;

        targetPreviousPosition = followingTarget.position;
    }

    void Update()
    {
        UpdateCameraPosition();
    }

    void UpdateCameraPosition()
    {
        var delta = followingTarget.position - targetPreviousPosition;

        if (disableVerticalParallax)
            delta.y = 0;

        targetPreviousPosition = followingTarget.position;
        transform.position += delta * parallaxStrength;
    }
}

```

Скрипт ParallaxBehavior.cs:

```

[SerializeField] private Transform targetObject;
[SerializeField] private float speed = 10.0f;

void Start()
{
    transform.position = targetObject.position;
}

void Update()
{
    Vector2 targetPosition = targetObject.position;
    Vector3 direction = (targetPosition -
(Vector2)transform.position).normalized;

    float distanceToMove = speed * Time.deltaTime;
    Vector3 newPosition = transform.position + direction * distanceToMove;

    newPosition.z = 100;
    transform.position = newPosition;
}

```

Логіка роботи інтерфейсу

Скрипт BarsController.cs:

```

void Start()
{
    currentHP = stats.currentHealth;
    currentMana = stats.currentMana;
    currentUlt = stats.currentUlt;
    UpdateBars();
}

private void FixedUpdate()
{
    UpdateBars();
}

public void UpdateBars()
{
    BarHP.fillAmount = currentHP / stats.GetMaxHealthValue();
    BarMana.fillAmount = currentMana / stats.GetMaxManaValue();
    BarUlt.fillAmount = currentUlt / stats.GetMaxUltValue();
}

public void SetHealth(float health, float maxHealth)
{
    currentHP = Mathf.Clamp(health, 0f, maxHealth);
    UpdateBars();
}

public void SetMana(float mana, float maxMana)
{
    currentMana = Mathf.Clamp(mana, 0f, maxMana);
    UpdateBars();
}

public void SetUlt(float ult, float maxUlt)
{
    currentUlt = Mathf.Clamp(ult, 0f, maxUlt);
    UpdateBars();
}

```

Скрипт BossGUI.cs:

```

void UpdateBar()
{
    BarHP.fillAmount = currentHP / maxHP;
}

public async void FixedUpdate()
{
    if (!isDead)
    {
        currentHP = bossOne.stats.currentHealth;
        UpdateBar();
    }

    if (currentHP <= 0 && HUD.activeSelf)
    {
        UpdateBar();
        isDead = true;
        Save();
        victoryText.SetActive(true);
    }
}

```

```

        walls[0].SetActive(false);
        walls[1].SetActive(false);
        exitTrigger.SetActive(true);
        await FadeOutHUD(displayTime);

        HUD.SetActive(false);
    }
}

private async Task FadeOutHUD(float duration)
{
    Color[] originalColors = new Color[4];
    originalColors[0] = hpBack.GetComponent<Image>().color;
    originalColors[1] = hp.GetComponent<Image>().color;
    originalColors[2] = nameText.GetComponent<Text>().color;
    originalColors[3] = victoryText.GetComponent<Text>().color;

    Color[] fadeColors = new Color[4];
    for (int i = 0; i < 4; i++)
    {
        fadeColors[i] = new Color(originalColors[i].r,
originalColors[i].g, originalColors[i].b, 0f);
    }

    float timer = 0f;
    if(HUD.activeSelf == true && HUD != null) {
        while (timer < duration)
        {
            float progress = timer / duration * 5f;

            hpBack.GetComponent<Image>().color =
Color.Lerp(originalColors[0], fadeColors[0], progress);
            hp.GetComponent<Image>().color = Color.Lerp(originalColors[1],
fadeColors[1], progress);
            nameText.GetComponent<Text>().color =
Color.Lerp(originalColors[2], fadeColors[2], progress);
            victoryText.GetComponent<Text>().color =
Color.Lerp(originalColors[3], fadeColors[3], progress);

            timer += Time.deltaTime;
            await Task.Yield();
        }
    }

}

void ResetHUDColors()
{
    Color hpBackColor = hpBackImage.color;
    hpBackColor.a = 1f;
    hpBackImage.color = hpBackColor;

    Color hpColor = hpImage.color;
    hpColor.a = 1f;
    hpImage.color = hpColor;

    Color nameColor = nameTextComponent.color;
    nameColor.a = 1f;
    nameTextComponent.color = nameColor;

    Color victoryColor = victoryTextComponent.color;

```

```

        victoryColor.a = 1f;
        victoryTextComponent.color = victoryColor;
    }
    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player") && !isDead)
        {
            HUD.SetActive(true);
            walls[0].SetActive(true);
            walls[1].SetActive(true);
            exitTrigger.SetActive(false);

            if (bossOne != null)
            {
                maxHP = bossOne.stats.maxHealth.GetValue();

                currentHP = maxHP;
            }

            isBossOpened = true;
            bossIcon.SetActive(true);
            Save2();
        }
    }
}

```

Скрипт GUIController.cs:

```

public abstract Image[] GetImages();
public void Awake()
{
    images = GetImages();
}
public virtual void Start()
{
    Deactivate();
}

public virtual void Activate()
{
    foreach (Image img in images)
    {
        img.enabled = true;
    }
}

public virtual void Deactivate()
{
    foreach (Image img in images)
    {
        img.enabled = false;
    }
}

public virtual void FillImage(int index)
{
    images[index].sprite = fullImage;
}

public virtual void ClearImage(int index)
{
    Debug.Log(index);
    images[index].sprite = emptyImage;
}

```

Створені графічні елементи (зображення та анімації)

Модель ворога лучника



Модель ворога таємного вбивці



Модель босу локації «Замок»



Модель босу локації «Ліс»



Модель ворога собаки



Модель ворога ока



Модель ворога чаклуна



Модель ворога лісового гобліну



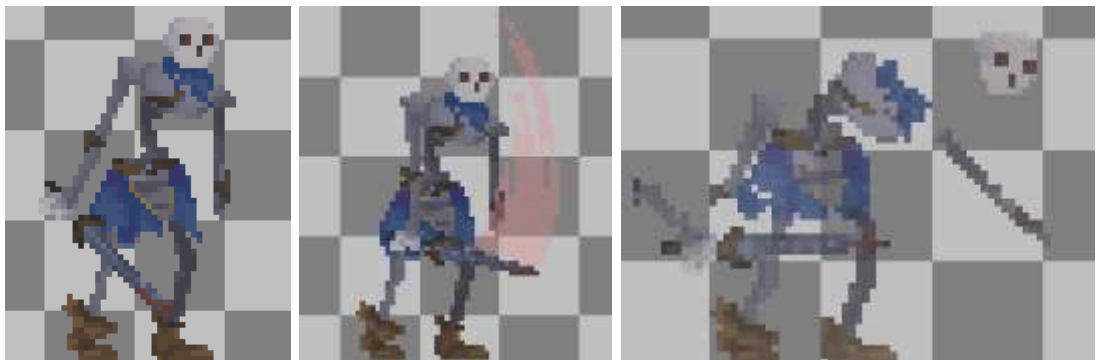
Модель врага паука



Модель врага волка



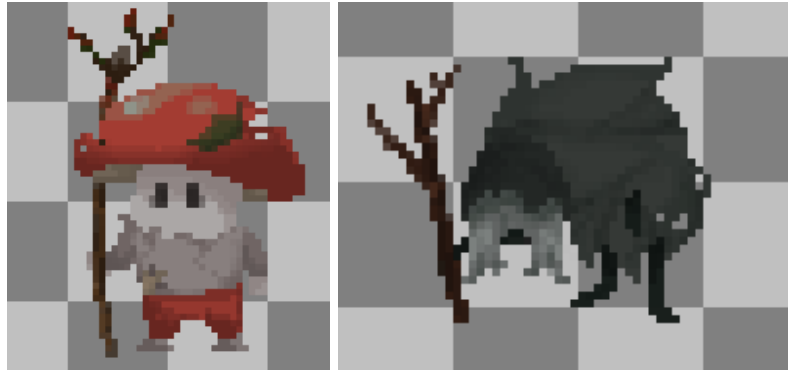
Модель врага скелета



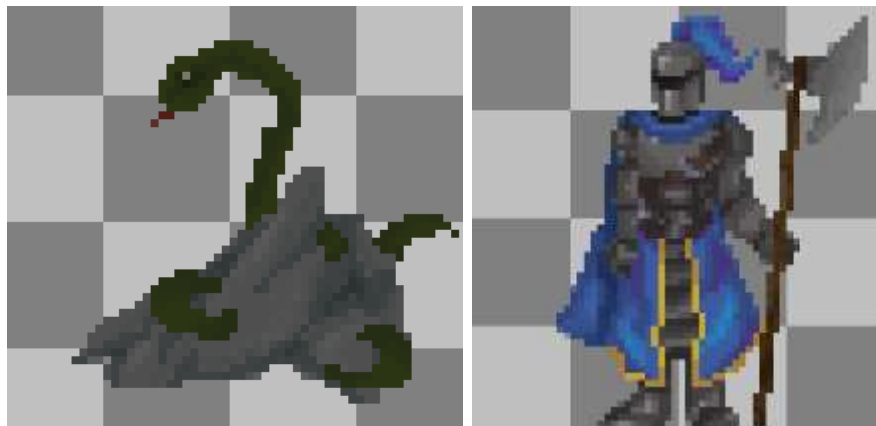
Модель врага привада



Модель NPC Mushroom та Pilgrim



Модель NPC Snake та Knight



Модель головного героя



Іконки досягнень



Фонове зображення головного меню



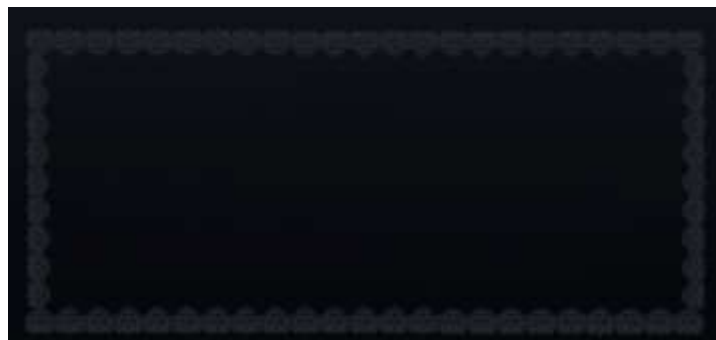
Фонове зображення локації «Ліс»



Фонове зображення локації «Замок»



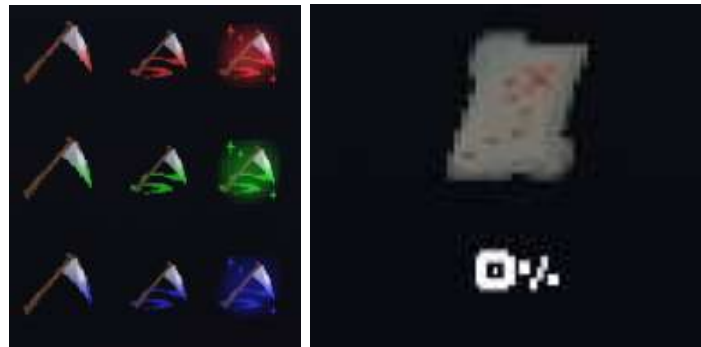
Фонове зображення внутрішньоігровго меню



Кнопки внутрішньоігрового меню



Іконки для сторінки Character



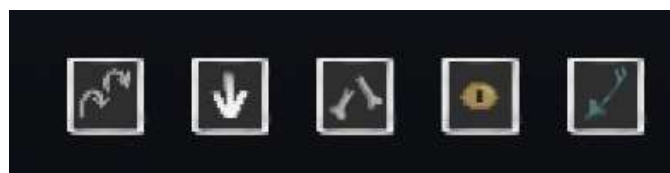
Іконки для сторінки Abilities



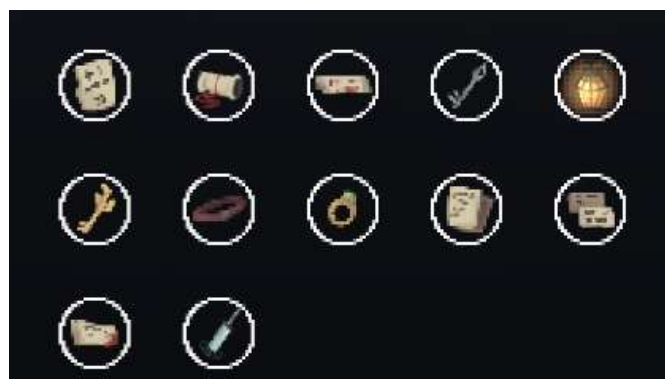
Іконки для сторінки Charms



Іконки для сторінки Talants



Іконки для сторінки Inventory



Туроріал



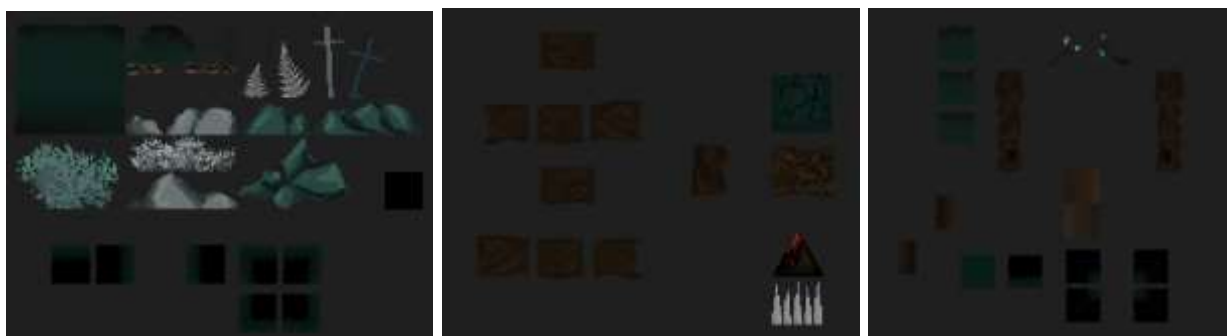
Мапа



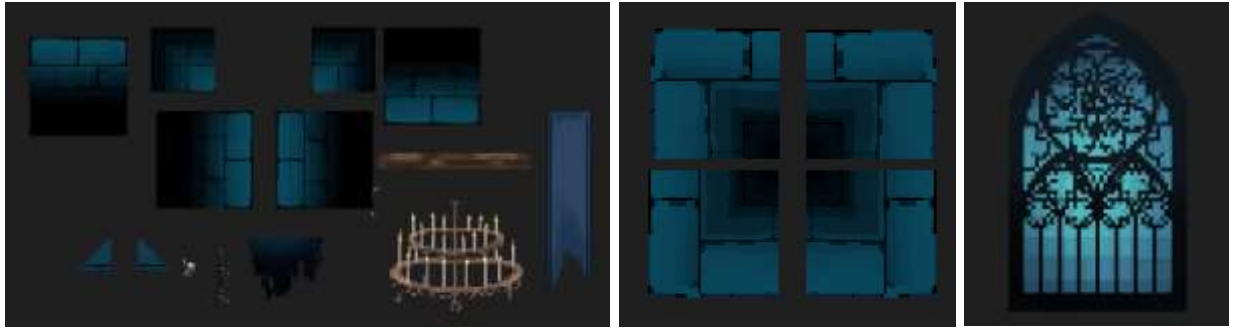
Іконки для мапи



Tilemap локації «Ліс»



Tilemap локації «Замок»



Додаткові елементи декору

