

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 комп'ютерної гри у жанрі пригодницького
платформера з елементами dark fantasy»**

Виконав студент групи КН-21-3
ЛАТКО Максим Вікторович

Керівник роботи:
ЛЕВКІВСЬКИЙ Віталій Леонідович

Рецензент:
ФУРІХАТА Денис Вікторович

Житомир – 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук
Марина ГРАФ
«19» березня 2025 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: **ЛАТКО Максим Вікторович**

Керівник роботи: **ЛЕВКІВСЬКИЙ Віталій Леонідович**

Тема роботи: **«Розробка комп'ютерної гри у жанрі пригодницького платформера з елементами dark fantasy»**,

затверджена наказом закладу вищої освіти від **«17» березня 2025 р., №99/с**

Термін здачі закінченої роботи: **«06» червня 2025 р.**

Вихідні дані роботи (зазначається предмет і об'єкт дослідження): об'єкт дослідження – процес повного циклу розробки пригодницької гри на рушії Unity. Предметом дослідження є архітектурні та геймдизанерські рішення, зокрема застосування шаблонів програмування, штучний інтелект ворога та механізм збереження даних, для розробки пригодницької гри на рушії Unity.

Консультанти кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв
I	Олександра СВІНЦИЦЬКА	19.03.2025	19.03.2025
II	Олександра СВІНЦИЦЬКА	19.03.2025	19.03.2025
III	Олександра СВІНЦИЦЬКА	19.03.2025	19.03.2025

Календарний план

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд та аналіз аналогічних розробок. Формулювання завдання. Опрацювання літературних джерел	18 березня 2025 – 1 квітня 2025	Виконано
2	Проектування структури програми	1 квітня 2025 – 12 квітня 2025	Виконано
3	Написання програмного коду	12 квітня 2025 – 12 травня 2025	Виконано
4	Тестування програмного продукту	12 травня 2025 – 20 травня 2025	Виконано
5	Оформлення пояснювальної записки	20 травня 2025 – 1 червня 2025	Виконано

Здобувач вищої освіти **Максим ЛАТКО**
Керівник **Віталій ЛЕВКІВСЬКИЙ**

РЕФЕРАТ

Основний зміст пояснювальної записки кваліфікаційної роботи викладено на 71 сторінці тексту, містить 8 таблиць, 66 ілюстрацій, 4 додатки, бібліографічний список літературних джерел із 23 найменувань. Загальний обсяг роботи – 104 сторінки.

Метою кваліфікаційної роботи є розробка пригодницької 2D-платформерної гри «Final Dawn» з елементами dark fantasy, що демонструє застосування шаблонів проектування й геймдизайну у середовищі Unity.

У рамках роботи створено повноцінну гру з основними механіками, графічним оформленням, інтерфейсом та збереженням прогресу. Робота демонструє практичне застосування знань у сфері програмування, геймдизайну та розробки ігор.

Ключові слова: ГРА, FINAL DAWN, UNITY, C#, DARK FANTASY, ПРИГОДА, ПЛАТФОРМЕР, ШАБЛОН ПРОЕКТУВАННЯ, РІВЕНЬ, ГЕЙМДИЗАЙН, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

					КН.КР.Б – 122 – 25 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка комп'ютерної гри у жанрі пригодницького платформера з елементами dark fantasy Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		М.В. ЛАТКО						
Керівник		В.Л. ЛЕВКІВСЬКИЙ					3	104
Реценз.		Д.В. ФУРІХАТА				Житомирська політехніка, група КН-21-3		
Н. Контр.								
Затверд.		М.С. ГРАФ						

ABSTRACT

The main content of the explanatory note of the final qualification work is presented on 71 pages of text, includes 8 tables, 66 illustrations, 4 appendices, and a list of 23 bibliographic sources. The total volume of the work is 104 pages.

The aim of the qualification work is the development of the adventure 2D platformer game "Final Dawn" with elements of dark fantasy, demonstrating the use of design and game development patterns within the Unity environment.

As a result, a complete game was created, featuring core mechanics, graphical design, user interface, and a save/load system. The project showcases the practical application of knowledge in programming, game design, and game development.

Keywords: GAME, FINAL DAWN, UNITY, C#, DARK FANTASY, ADVENTURE, PLATFORMER, DESIGN PATTERN, LEVEL, GAME DESIGN, SOFTWARE.

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ	6
ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ.....	10
1.1. Постановка задачі.....	10
1.2. Огляд існуючого програмного забезпечення	11
1.3. Концепт-документ гри	14
1.4. Інструментальні засоби та методи дослідження	16
Висновки до першого розділу	17
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	19
2.1. Ресурсна база та побудова ігрової сцени	19
2.2. Моделювання та архітектура гри.....	25
2.3. Реалізація ігрових модулів та функціоналу	33
Висновки до другого розділу.....	44
РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ.....	45
3.1. Вимоги та середовище виконання	45
3.2. Взаємодія користувача з ігровим середовищем.....	46
3.3. Тестування роботи та демонстрація результатів	63
Висновки до третього розділу	66
ВИСНОВКИ	67
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	69
ДОДАТКИ	72

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – англ. Artificial Intelligence; програмна логіка, що керує поведінкою ворогів та NPC у грі.

API – англ. Application Programming Interface; набір зовнішніх функцій і класів, через які код взаємодіє з рушієм або бібліотекою.

Agro-Radius – радіус виявлення, у межах якого ворог переходить у стан переслідування гравця.

Boss – у рамках гри, це особливий противник певного рівня, який має унікальну поведінку, підвищені показники здоров'я та можливу нагороду за його ліквідацію.

FPS – англ. Frames Per Second; кількість кадрів, які рендеряться за 1 секунду, є показником плавності відеоряду.

FSM – англ. Finite State Machine; кінцевий автомат, що переходить між визначеними станами за умовою.

Gem – магічний кристал, сюжетний предмет, що збирається гравцем для відкриття порталу.

Git LFS – Git Large File Storage; розширення Git, яке зберігає великі медіафайли (спрайти, звук) окремо від основного репозиторію.

HP – англ. Hit Points; очки здоров'я персонажа.

HUD – англ. Head-Up Display; екранні елементи, які відображають поточний стан гравця (здоров'я, мана, досвід, гроші тощо).

IDE – англ. Integrated Development Environment; середовище розробки (Visual Studio), у якому пишуться та налагоджуються скрипти.

Level – окрема ігрова сцена, містить певні перешкоди і завдання, які гравець має подолати на своєму шляху.

NPC – англ. Non-Player Character; персонаж, за якого немає можливості пограти, керується зазвичай системою.

Prefab – заздалегідь налаштований шаблон об'єкт Unity, який використовується для створення екземплярів об'єкта, що зберігається на певній сцені.

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

RPG – англ. Role-Playing Game; ігри, у яких гравець розвиває персонажа через характеристики, навички й сюжетні вибори.

SFX – англ. Sound Effects; короткі звукові ефекти дій (удар, стрибок, підбір предмета).

UI – англ. User Interface; користувацький інтерфейс (меню, кнопки).

XP – англ. Experience Points; досвід, який отримує гравець за виконання дій у грі (перемоги над ворогами) і використовується для підвищення ігрового рівня персонажа.

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Світова індустрія відеоігор демонструє стабільне зростання, а сегмент інді-платформерів залишається одним із найдинамічніших. Поєднання механік класичного 2D-платформера з елементами dark fantasy дозволяє створювати ігровий досвід, що одночасно є доступним для широкої аудиторії та має високу естетичну цінність.

За оцінкою дослідницької компанії Newzoo, у 2024 р. світовий ринок відеоігор приніс 187,7 млрд USD, що на 2,1 % більше, ніж торік (прогноз було скориговано з +2,8 %). Попри стримані продажі консолей, ПК-сегмент демонструє найактивніше зростання — +4 % на рік, а кількість геймерів у світі досягне 3,42 млрд (+4,5 %) [1]. Така динаміка підкреслює необхідність ефективних інструментів і шаблонів, що скорочують цикл розробки ігор для ПК-платформи.

Сцена інді-проектів є надзвичайно конкурентною: на Steam під тегом Indie вже представлено понад 126 тис. найменувань, а піджанр «2D Platformer» стабільно входить до списку найпопулярніших категорій магазину [2]. Отже, творчий успіх інді-гри напряду залежить від якісного геймдизайну, оптимізованої архітектури коду та виразного візуального стилю, що вирізняє продукт серед тисячі аналогів.

Попит на естетитку dark fantasy жанру демонструє сталий тренд. Можна зазначити, що у рейтингу Glitchwave «Most popular dark fantasy games 2023» в топ-3 програмних продукти увійшли “Lies of P”, “Final Fantasy XVI” та “Diablo IV” [3]. Це зайвий раз підтверджує обширний інтерес аудиторії до похмурої атмосфери пригодницького сюжету. Комерційний успіх зазначених релізів свідчить, що таке поєднання платформерної механіки та dark fantasy повністю успішне.

Для України тема геймдеву також має вагу. За даними джерела dev.ua у 2024 році одна з українських ігор увійшла до топ-20 продажу на Європейському сегменті, це підтверджує потенціал виробництва у світовому масштабі [4].

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		8

Таким чином, можемо дійти до висновку, що розробка гри “Final Dawn” є актуальною. Вона зачепить своєю приємною естетичною атмосферою dark fantasy, яка перебуває на піку зростання популярності серед аудиторії геймерів.

Об’єкт дослідження – процес повного циклу розробки пригодницької гри на рушії Unity, формування концепції та проектування геймдизайнерського рішення.

Предметом дослідження є архітектурні та геймдизайнерські рішення, зокрема застосування шаблонів програмування, штучний інтелект ворога та механізм збереження даних, для розробки пригодницької гри на рушії Unity.

Мета і завдання дослідження полягає у проектуванні та реалізації платформи Final Dawn з елементами dark fantasy, що демонструє сучасний підхід до розробки програмного продукту.

Практичне значення. Створення завершеного прототипу, який може бути основою для подальшого комерційного розширення проекту.

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1. Постановка задачі

Сучасний ринок інді-платформерів характеризується одночасно високою конкуренцією та зростаючими вимогами до якості ігрового продукту. Щоб успішно конкурувати, розробник має забезпечити не лише привабливу естетику, а й масштабовану архітектуру, стабільну продуктивність, продуманий штучний інтелект ворогів [5]. У переважній більшості навчальних і демонстраційних проєктів ці аспекти розглядаються фрагментарно: або акцент ставиться на геймдизайні без детальної інженерної частини, або навпаки — описується лише технічна реалізація без глибокого дизайнерського обґрунтування.

Метою цієї кваліфікаційної роботи є створення цілісного рішення, яке охоплює всі етапи розробки — від розробки концепції та дизайну рівнів до побудови гнучкої та модульної архітектури програмного забезпечення на базі Unity. Особливу складність становить вибір естетики dark fantasy, що вимагає ретельної роботи над візуальним стилем, музичним супроводом і інтеграцією глибокого, атмосферного сюжету, який підкреслює похмурий настрій світу гри.

Конкретизовані завдання, що впливають із сформульованої проблеми:

- виконати системний огляд наукових джерел і комерційних 2D-платформерів, щоб визначити оптимальні механіки бою, шаблони AI та практики архітектурної модульності;
- сформувати повний пакет попередньої документації гри — concept-document і design-document — із визначенням наративу, структури рівнів, UX-вимог та ключових технічних специфікацій;
- спроектувати програмну архітектуру у Unity 2021, застосувавши шаблони Singleton, State, Observer і Factory для реалізації руху героя, бойової системи, AI ворогів, інвентарю, діалогів і глобальних збережень;

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		10

- створити та інтегрувати візуальні (спрайти, анімації, освітлення) й аудіо-ресурси (SFX, адаптивна музика) так, щоб вони формували узгоджену dark-fantasy атмосферу і не знижували продуктивність;
- провести комплексне тестування, внести коригування та підготувати готову збірку прототипу гри разом із технічною документацією для його подальшого розширення.

В межах виконання кваліфікаційної роботи буде створена компютерна гра у жанрі пригодницького платформера з елементами dark fantasy – Final Dawn.

1.2. Огляд існуючого програмного забезпечення

Ринок 2D-платформерів на сьогодні включає широкий спектр ігор — від простих мобільних проєктів до складних екшн-пригод з елементами RPG. Популярність жанру обумовлена доступністю, універсальністю та можливістю реалізації різноманітних геймплейних ідей у форматі з мінімальними технічними вимогами.

Ігри жанру суттєво відрізняються за складністю, подачею, механіками та візуальним стилем. Враховуючи концепцію Final Dawn — похмурого фентезі платформера з бойовою системою, інвентарем і діалогами — доцільно було дослідити ігри з подібною структурою та наративом. Було проаналізовано чотири проєкти, найбільш релевантні за стилем та реалізацією.

Hollow Knight — атмосферний платформер у стилі метроїдванії з глибокою бойовою системою, дослідженням світу та нелінійною структурою рівнів (рисunek 1.1). Гравець керує безіменним лицарем, який бореться з ворогами, відкриває нові області та взаємодіє з NPC. Функціональні особливості:

- бойова система: з чіткими таймінгами та візуальними індикаторами шкоди;
- інтерактивні NPC з діалогами, які впливають на сюжет;
- вертикальні зони з небезпеками, що вимагають точності;

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		11

- вороги з унікальним AI, які реагують на дії гравця.

Вплив на проект Final Dawn. Фазовість босів та схему AI “Idle Until Agro” для звичайних ворогів; рівневу логіку спрощуємо до каскадної лінійної послідовності, аби зберегти темп.



Рисунок 1.1 – Геймплей гри Hollow Knight [6]

Dead Cells - динамічний rogue-lite платформер з бойовими комбо, швидким геймплеєм і процедурною генерацією рівнів (рисунок 1.2). Основний акцент зроблено на бою, ухиленнях і чітких анімаційних переходах. Функціональні особливості:

- динамічний і швидкий контроль з перекатами;
- високий темп бою підтримується комбінуванням здібностей;
- циклічний геймплей ризик – нагорода: втрата ресурсів після смерті;
- гарне візуальне оформлення (анімації, переходи);

Вплив на гру Final Dawn. Логіка роботи літаючих ворогів. Контрольований бій з візуальним фідбеком. Модульний підхід до AI та структура бою з ворогами.



Рисунок 1.2 – Геймплей гри Dead Cells [7]

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

Salt and Sanctuary - поєднує механіки RPG, платформінг і складні бої в темному фентезі-світі (рисунок 1.3). Відомий своєю хардкорністю, вертикальністю рівнів та сильним атмосферним фоном. Функціональні особливості:

- система стаміни, блоків, що робить бій тактичним;
- похмура візуальна атмосфера, що супроводжується шарами звукових ефектів;
- дерево навичок;
- особливі боси.

Вплив на *Final Dawn*. Відмова від складної стаміни та глибокого дерева скілів, зберігається казуальність в управлінні. Побудова рівня як багаторівневої структури. Логіка босів (активація – бій – нагорода).



Рисунок 1.3 – Геймплей гри Salt and Sanctuary [8]

Prince of Persia: The Lost Crown - сучасна 2.5D-інтерпретація класичного платформера з акцентом на бій, сюжет, діалоги та відкриту карту. Гравець досліджує великі зони, долає перешкоди (рисунок 1.4), бореться з ворогами й босами. Функціональні особливості:

- бойова система з використанням уклонів, парирування та комбо;
- динамічні сцени, перемикання ігрових зон;
- активні вороги, що реагують на дії гравця;
- взаємодія з NPC, розвиток сюжету через діалоги.

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		13

Вплив на Final Dawn. Логіка дзеркального боса як копії гравця, створення двох башт з ігровими подіями.



Рисунок 1.4 – Долання перешкод у грі Prince of Persia: The Lost Crown [9]

1.3. Концепт-документ гри

Концепт-документ для гри — це ключовий документ, який узагальнює основну ідею, механіки, сюжет та візуальний стиль проекту [10]. Він слугує дорожньою картою, допомагає узгодити бачення та забезпечує чітке планування процесу створення гри.

Введення:

Final Dawn – 2D-платформер у пригодницькому форматі та у стилі dark fantasy, де гравець проходить послідовність локацій, від світлого лісу до похмурих небезпечних локацій. Разом з цим він збирає магичні геми, долає ворогів, задля відновлення балансу всесвіту.

Жанр і аудиторія:

Зазначений жанр програмного продукту – adventure-platformer з RPG-елементами. Цільова аудиторія це гравці віку від 14 років, які є прихильниками жанрів metroidvania та rogue, шукають пригод та готові понуритись у темну атмосферу таємничого світу.

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		14

Основні особливості гри:

- поєдинки з босами;
- модульний AI ворогів (Idle, Patrol, Agro, Attack);
- інвентарна частина;
- діалогова частина;
- музикальне супроводження.

Опис гри:

Гравець керує безіменною героїнею, яка проходить через ліси, гори, печери, замки та різні локації. У кожній зоні є певний тип ворогів, перешкод та платформи по яким передбачений рух. Кульмінаційний момент кожного рівня – може різнитись в залежності від рівня, будь то поєдинок з босом, вирішення проблеми, відкриття порталу. Отримання нагороди передбачається у вигляді магічних гемів, які потрібні нам для проходження сюжетної гілки. Можливі інші винагороди на кшталт зілля для покращення характеристик, здоров'я. Також є грошова винагорода і можливість придбати речі в магазині. Інтуїтивно-зрозумілий HUD буде тримати в курсі гравця щодо поточного стану. Завершення гри полягає у перемозі над головним противником, у разі чого герой переміщається у поселення, де панує мир і вдячність за порятунок. У разі невдачі, гравець понурюється у спотворену реальність, де немає майбутнього.

Принцип гри:

2D-платформер із циклом дослідження → контакт → бій/ухилення → винагорода → перехід на іншу локацію/завершення гри

Лінійна послідовність локацій: Forest → Cave → Dark Forest → Castle (Tower) → Light's Dawn/Cataclysm;

Фізична модель персонажа:

Керування відбувається за RigidBody2D + Box / Circle Collider2D. Персонаж рухається зміною velocity, платформи позначені шаром Ground.

Елементи гри:

- платформи;
- вороги (звичайний тип, боси);

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дата		

- портали-перехідники між сценами, двері;
- чек-поінт статуї;
- тригерні зони для діалогів, кат-сцен.

Інтерфейс користувача:

HUD:

- шкала НР, ХР;
- кількість монет, характеристики шкоди і швидкості;
- екран паузи;
- екран після загибелі героя;
- екран перемоги;
- діалогове вікно з NPC;
- інвентар.

Передумови створення:

Ідея виникає при аналізі сучасних трендів інді-ринку, де dark fantasy-естетика й бойові системи залишаються досі популярними. Мета полягає у демонстрації повного циклу розробки продукту, від концепції до робочого прототипу з модульною архітектурою й адаптивним аудіо-супроводженням.

Платформа:

Рушій: Unity 2021.3.45f1.

Цільова платформа: Windows 10 (PC-платформа).

1.4. Інструментальні засоби та методи дослідження

У таблиці 1.1 наведено та обґрунтовано вибір інструментальних засобів, що застосовуються під час розробки гри.

Таблиця 1.1

Інструментальні засоби

Категорія	Інструмент	Обґрунтування вибору та роль у проєкті
Ігровий рушій	Unity 2021.3.45f1 (LTS)	Стабільна Long-Term-Support-версія з повним набором 2D-інструментів, URP-пост-процесом і вже виправленими

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		16

		критичними багами; гарантує оновлення без зміни API протягом усього циклу проєкту.
Мова скриптів	C# 10	Базова мова для Unity; знайома більшості розробників і дозволяє швидко писати зрозумілі скрипти для механік та AI.
IDE	Visual Studio 2022	Глибока інтеграція з Unity (плагін Unity Tools), IntelliSense, профілювання й аналізатор коду; безкоштовна Community-ліцензія для освітніх проєктів.
Контроль версій	Git + GitLab	Локальні git-репозиторії із поширенням через GitLab – підтримка приватних репозиторіїв. Git LFS для великих файлів.
Анімація	Unity Animator / Animation Controller	Дозволяє будувати стейт-машини для персонажів та босів без сторонніх плагінів; підтримує Animation Events, що зручно для синхронізації ударів і SFX.
Звук	Unity Audio Mixer	Вбудована система зручно керує гучністю, ефектами тощо; Зручно для навчальних проєктів та оптимізація збірки.
Графіка	Unity Asset Store, Unity Tilemap, Adobe Photoshop	Стартові спрайти та тайл-сети; Tilemap + Grid забезпечують швидку побудову ігрових рівнів. Adobe Photoshop для створення бекграунд-зображень, елементів меню.

Серед методів дослідження можна виокремити певні категорії:

- системний розбір: розділення гри на підсистеми (рух, AI, звук), щоб кожна було легше робити й тестувати;
- шаблони коду: використовуються готові підходи (Singleton для менеджерів, State для AI), аби код був охайний і зрозумілий;
- FSM-AI: вороги працюють за принципом світлофора, наприклад спокій → патруль → атака → шкода/поранення → смерть;
- тестування геймплею: запуск рівня та аналіз перешкод на шляху.

Висновки до першого розділу

У першому розділі обґрунтовано актуальність створення 2D-platformer гри «Final Dawn» у стилі dark fantasy й сформовано перелік завдань, що

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		17

покривають увесь цикл її розробки. Проведено порівняльний аналіз чотирьох релевантних проєктів — Hollow Knight, Dead Cells, Salt and Sanctuary та Prince of Persia: The Lost Crown; визначено їхні сильні й слабкі сторони та виділено ключові механіки.

Розроблено концепт-документ, що описує жанр, цільову аудиторію, сюжетний синопсис, ігрові механіки, структуру рівнів та унікальні переваги проєкту. Обрано інструментальний стек — Unity 2021.3 LTS, C#, Visual Studio 2022, Git + GitLab, Asset Store / Tilemap, Animator і Audio Mixer — який забезпечує швидке прототипування та стабільну роботу.

Визначено методи розробки: системний аналіз, застосування шаблонів Singleton, State і Observer, FSM-модель AI. Сукупність виконаних етапів задає чіткий напрям подальшого розвитку “Final Dawn” і створює основу для реалізації повноцінного ігрового прототипу.

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Ресурсна база та побудова ігрової сцени

Розглянемо підготовку візуальних ресурсів до безпосереднього компонування ігрового меню та рівнів, загальний вигляд персонажів та їх анімації.

Побудова сцени є важливою частиною гри, адже саме вона формує перше враження гравця, задає атмосферу та забезпечує коректну роботу ігрових механік.

Фон головного меню та усіх меню-панелей (рисунок 2.1) у грі “Final Dawn” – це темне стилізоване зображення, витримане в естетиці dark fantasy. Атмосферний фон передає відчуття загрози, загадковості та древності, задає тон усій грі ще до старту геймплею.

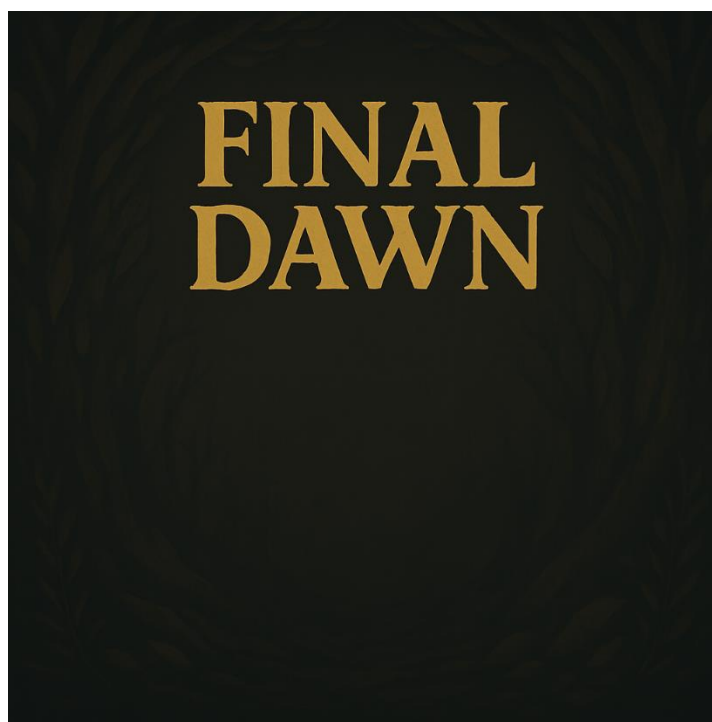


Рисунок 2.1 – Фонове зображення гри

Використовуємо інструмент Unity – Tile Palette, для цього використаємо графічний атлас спрайтів, що містить елементи для побудови архітектурних структур для замку, печери, лісів. Атлас може включати в себе безліч елементів,

наведено у приклад один з таких (рисунок 2.2), що містить: арки, колони, сходи, ґратчасті двері, фрагменти стін.

У інспекторі потрібно встановити певні налаштування (рисунок 2.3):

- Sprite Mode: Multiple;
- Pixels Per Unit: 16 / 32 / 64;
- Filter Mode: Point (no filter).

Далі натискаємо Sprite Editor → Slice (обираємо зручний метод Slice, вказуємо Offset та Pading, після чого Slice → Apply). Далі створюємо Tile Palette (Windows → 2D → Tile Palette), натискаємо Create New Palette – переміщуємо нарізані файли і можемо приступати до створення повноцінного ігрового рівня за допомогою пензеля Default Brush (керується скриптом GridBrush).

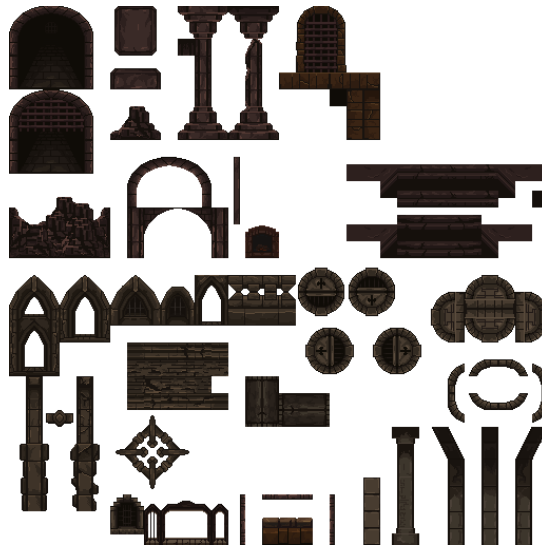


Рисунок 2.2 – Графічний атлас спрайтів [11]

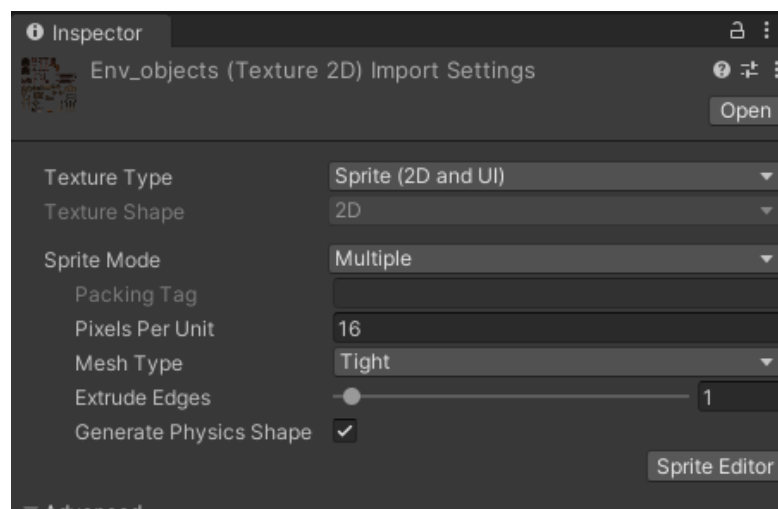


Рисунок 2.3 – Налаштування для атласу в Inspector

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		20

Використовується багат шарова система Tilemap (рисунок 2.4), що дозволяє окремо керувати фоном, декоративними елементами та платформою для взаємодії персонажів [12]. Усі шари розміщені під об'єктом Grid, що забезпечує синхронізацію розташування тайлів. Для створення подібного шару потрібно натиснути правою клавiшею миші на Hierarchy → GameObject → 2D Object → Tilemap → Rectangular. Далі потрібно призначити Sorting Layer задля призначення шару (фонове зображення, дерева, стіни, каміння). Для шару землі ми призначаємо Layer = Ground, додаємо компонент Tilemap Collider 2D (Used by Composite = true), Rigidbody2D (Body type = Static).

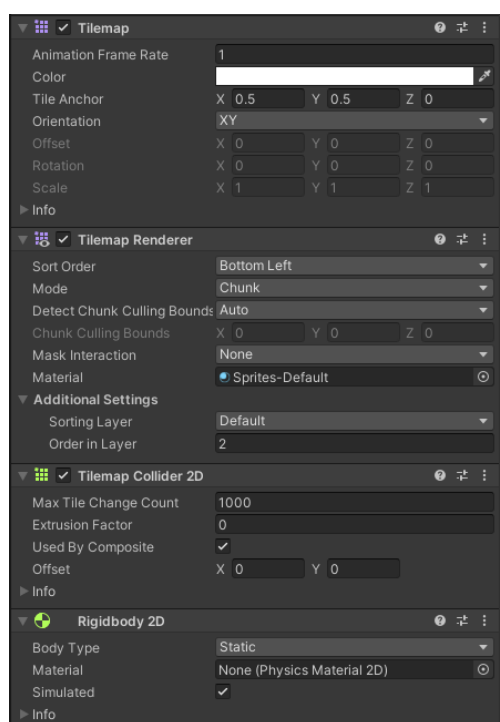


Рисунок 2.4 – Налаштування Tilemap в Inspector

Головна героїня гри – смілива та рішуча, вона не має імені, що робить її символом кожного гравця (рисунок 2.5). Вона проходить через різноманітні локації – ліси, гори, печери та замки, долаючи небезпеки і ворогів у світі, сповненому темної магії і загадок. Її мета – розкрити таємниці спотвореної реальності і врятувати світ від загрози, що насувається. Характер героїні поєднує мужність, витривалість і глибоку внутрішню силу, що допомагає їй долати найскладніші випробування.



Рисунок 2.5 – Вигляд головного персонажа гри [13]

Розібравши основні елементи технічної реалізації ігрового рівня, представлена не менш важлива складова – художня. А саме елементи HUD (рисунок 2.6), предмети магазину (рисунок 2.7), загальний вигляд ігрових рівнів.



Рисунок 2.6 – Елементи HUD [14]



Рисунок 2.7 – Предмети магазину [14]

Перший ігровий рівень має назву Forest (рисунок 2.8) – загадковий ліс, що є відправною точкою для гравця. Стилiстика витримана у класичному піксель-арті з глибокою кольоровою палітрою та чітко структурованими шарами середовища. Освітлення переважно природне, без надмірної похмурості як для початку шляху.

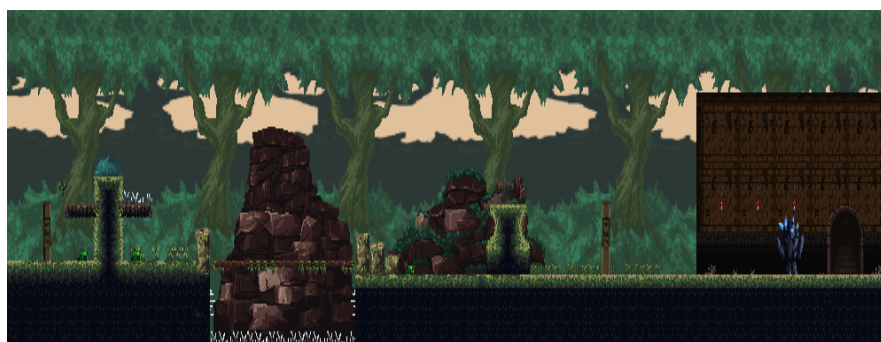


Рисунок 2.8 – Загальний вигляд першого рівня Forest

Другий ігровий рівень має назву Cave (рисунок 2.9, рисунок 2.10). Події розгортаються в печері, яка слугує продовженням після стартової локації. Це

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		22

темна та волога зона, має пастки, небезпечних ворогів (кажани, монстри), також є таємний хід, що веде до магазину (рисунок 2.11).

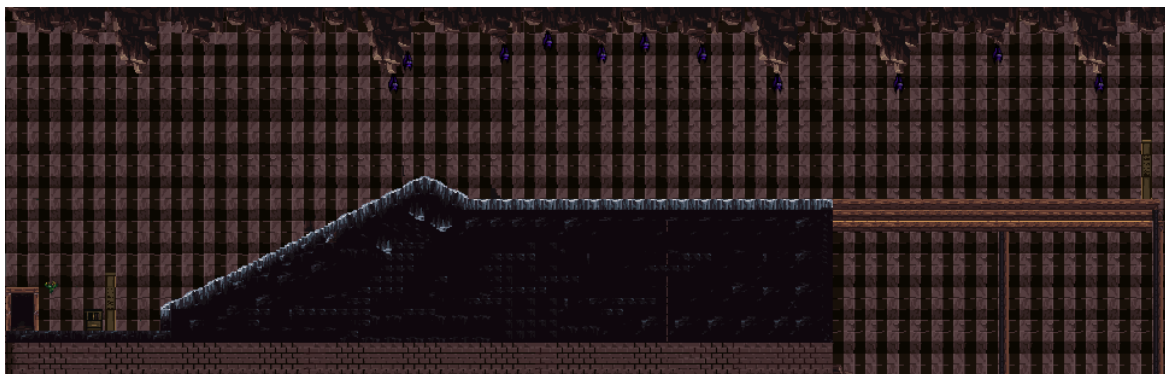


Рисунок 2.9 – Верхня частина локації Cave



Рисунок 2.10 – Нижня частина локації Cave

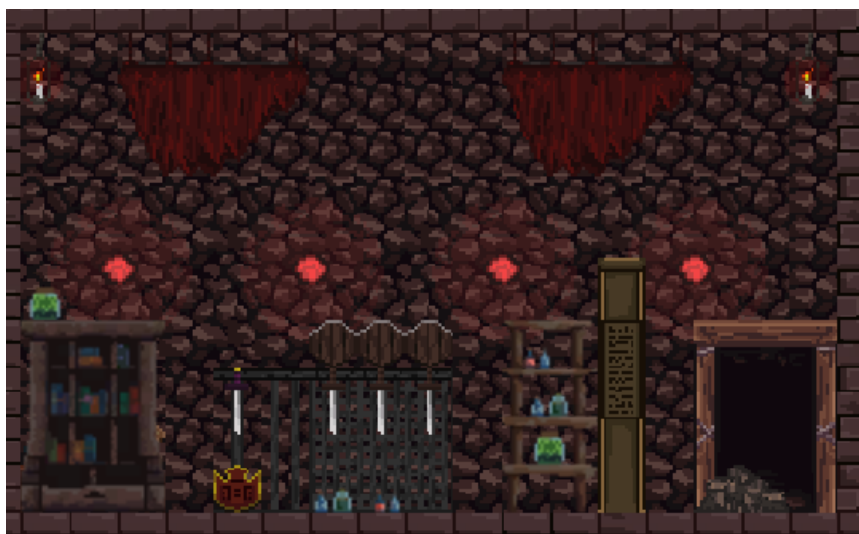


Рисунок 2.11 – Вигляд магазину

Третій ігровий рівень має назву Dark Forest (рисунок 2.12). Це найпохмуріший і найбільш напружений рівень, що служить переходом між підземною печерою та фінальною вежею. Стилiстика рівня натхненна темним фентезі: викривлені дерева, мертвий ліс, старі катакомби й темна магія. Присутня покинута фортеця заповнена мерццями, таємний шлях до магазину у

підземеллі покинутого замку, підземний перехід до монастиря зі спотвореною енергією.



Рисунок 2.12 – Загальний вигляд рівня Dark Forest

Четвертий і останній ігровий рівень має назву Dark Castle (Dark Paradise, Tower). По праву є наймонументальнішим рівнем гри, невеликий коридор зі скелетів що веде до великої багатоповерхової готичної вежі (рисунок 2.13), що є осередком темної сили, і має жертвне місце, яке може перенести гравця у пекельний вимір.



Рисунок 2.13 – Загальний вигляд вежі на рівні Dark Castle

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		24

2.2. Моделювання та архітектура гри

У даному підрозділі розглянемо застосування об'єктно-орієнтованого підходу при проєктуванні гри “Final Dawn”. Основну увагу приділено моделюванню систем гри за допомогою UML-діаграм [15]. Також це стосується застосованих шаблонів проєктування, таких як Singleton, Observer, Factory та інші. Представлено структуру ключових класів і механізм їх взаємодії, що забезпечує гнучкість та масштабованість програмної архітектури гри.

Діаграма варіантів використання (рисунки 2.14) відображає відображає функціональні можливості гри Final Dawn з точки зору користувача-гравця. Основним актором є гравець, який взаємодіє з системою через набір варіантів використання, що охоплюють усі ключові елементи геймплею: запуск, управління персонажем, бій, інвентар, розвиток, взаємодію з NPC, а також інтерфейс та збереження прогресу. Запуск ігрового процесу – центральний варіант використання, що охоплює основні функціональні підсистеми гри. Усі ключові механіки включаються до нього через відношення “include”. Деякі сценарії (смерть, покращення навичок) використовують відношення “extend”, оскільки активуються лише за певних умов.

У таблиці 2.1 наведено детальний опис варіантів використання.

Таблиця 2.1

Основні варіанти використання у Use Case Diagram

Основний сценарій	Розширення
Управління героїнею	Пересуватись ігровим світом, досліджувати локації, взаємодіяти з об'єктами.
Бойова система	Можливість атакувати ворога, отримати від нього шкоду, померти від нанесення смертельного урону.
Рольовий розвиток	Отримання ХР за перемогу над ворогом, підвищити ігровий рівень, отримати на вибір характеристику, що можна покращити.
Інтерфейс користувача	Відображення кількості монет, відображення кількості здоров'я та досвіду, характеристики бігу та атаки.

Збереження прогресу	Можливість в панелі паузи обрати слот для завантаження та вийти з гри, продовжити з обраної точки збереження.
NPC та сюжет	Взаємодія з неігровим персонажем Charlie, отримання квесту або настанови від нього, просування сюжетної гілки.
Предмети та інвентар	Купівля предметів у магазині, керування інвентарем, можливість використати предмет.

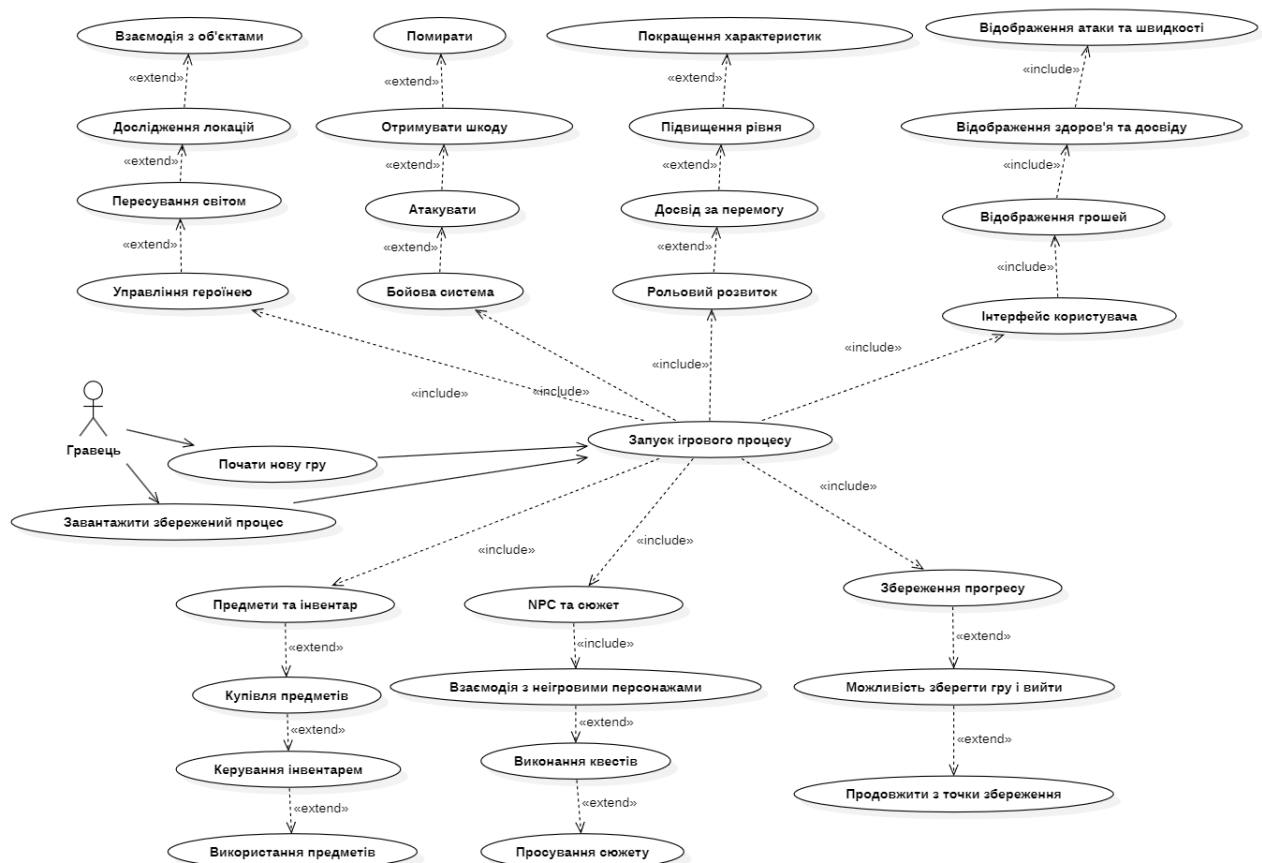


Рисунок 2.14 – Use Case Diagram

Діаграма діяльності моделює послідовність ключових дій гравця та логіку проходження основного сценарію гри. Вона демонструє, як одна дія переходить в іншу в межах загального ігрового процесу. Ця діаграма дозволяє простежити логіку геймплею, від запуску гри до досягнення однієї з кінцівок. Початковий стан діаграми відображає запуск гри через головне меню, де гравець має можливість вибору однієї з дій: почати нову гру або завантажити збереження. Далі відбувається ініціалізація даних та запуск ігрового процесу. Після початку гри далі йде основний цикл геймплею, який представлений як стан піддіяльності, що включає 4 основні гілки активностей:

- дослідження локацій: рух поміж локацій, виявлення об'єктів поблизу (тригерна зона, ворог, портал, гем) і взаємодія з ними;
- взаємодія з NPC: активація діалогів, отримання вказівок або квестів, їх виконання;
- бойова система: зіткнення з ворогами, можливість атакувати або ж використати предмет, можливість втечі від ворога;
- управління інвентарем: відкриття інвентарю, перегляд доступних предметів, можливість їх використання.

Завершується діаграма кінцевими станами, який може викликати поразка у бою з суперником. У такому разі гра викликає меню поразки та дає можливість перейти у головне меню.

Activity Diagram розділена на декілька частин для зручності та зображена на рисунках 2.15 – 2.17.

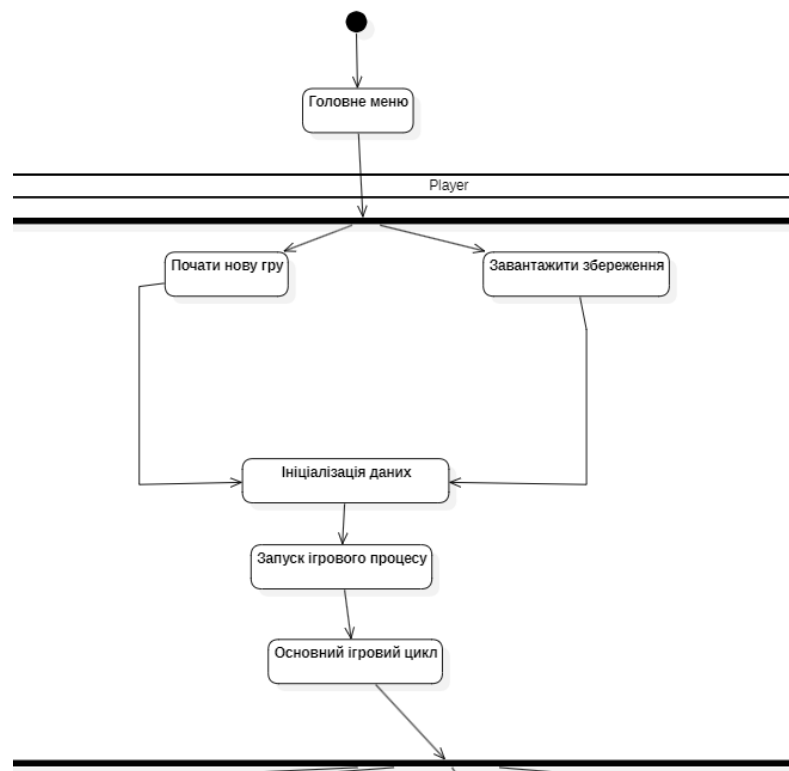


Рисунок 2.15 – Головне меню гри та запуск процесу:

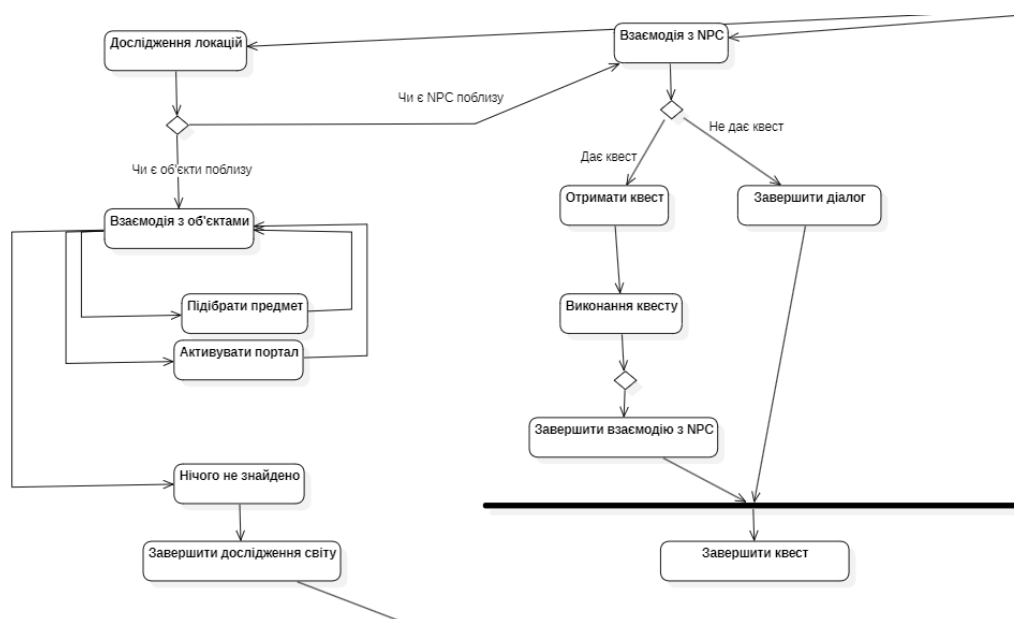


Рисунок 2.16 – Дослідження локацій та взаємодія з NPC

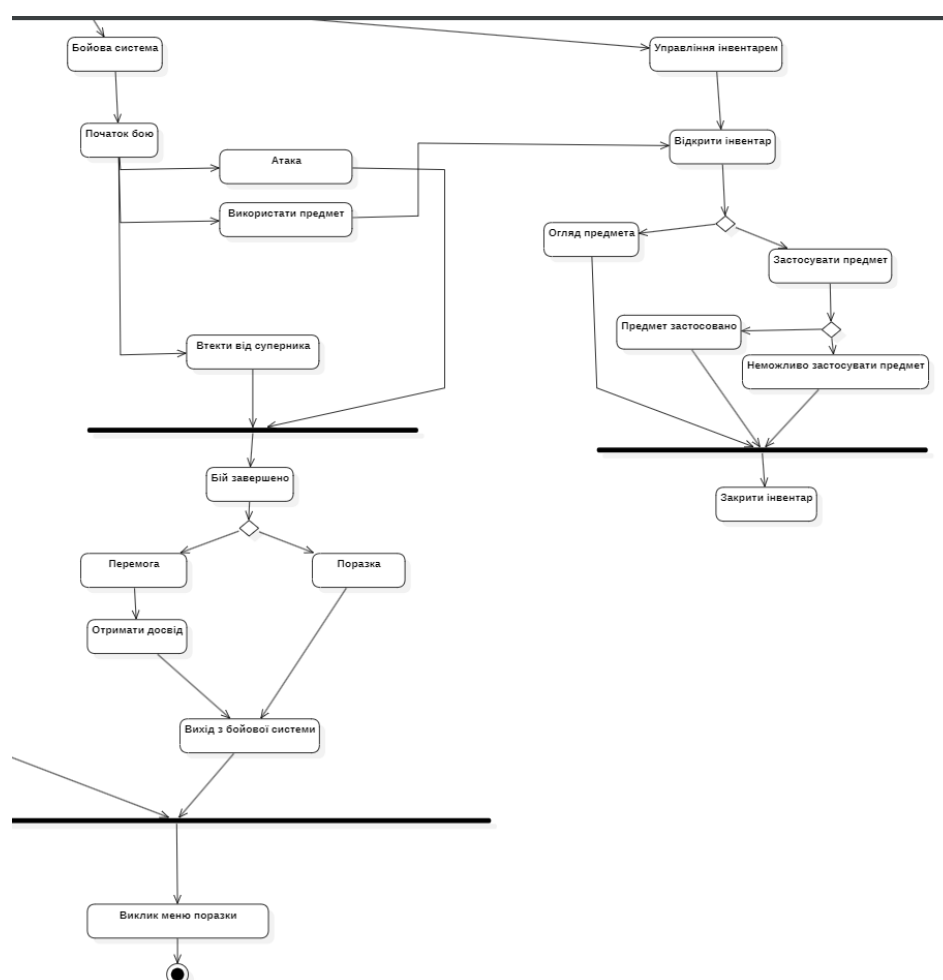


Рисунок 2.17 – Бойова система, управління інвентарем та завершення гри

Діаграма класів (рисунок 2.18) відображає чітку логіку гравця. Основним центром взаємодії виступає клас PlayerStats, який використовується для зберігання усіх ключових характеристик гравця. Інші компоненти, такі як рух,

атака, обробка шкоди та оновлення HUD тісно взаємодіють між собою.

Наведено опис класів:

- PlayerStats (синглтон Instance) – зберігає всі параметри гравця: здоров'я, досвід, рівень, бонуси, гроші. Пов'язаний з PlayerMovement для швидкості, з PlayerHealth для HP, XP, Level, з HUDController для відображення усіх поточних статистик гравця;
- PlayerHealth - обробляє урон, лікування, смерть, набір XP, оновлення HUD;
- PlayerAttack - логіка звичайної та сильної атаки, завдає шкоди через PlayerHealth;
- PlayerMovement - обробка горизонтального руху, стрибків, розвороту;
- HUDController - виводить на екран стан гравця: XP, HP, гроші, рівень, бафи. Пов'язаний з PlayerStats для оновлення характеристик.

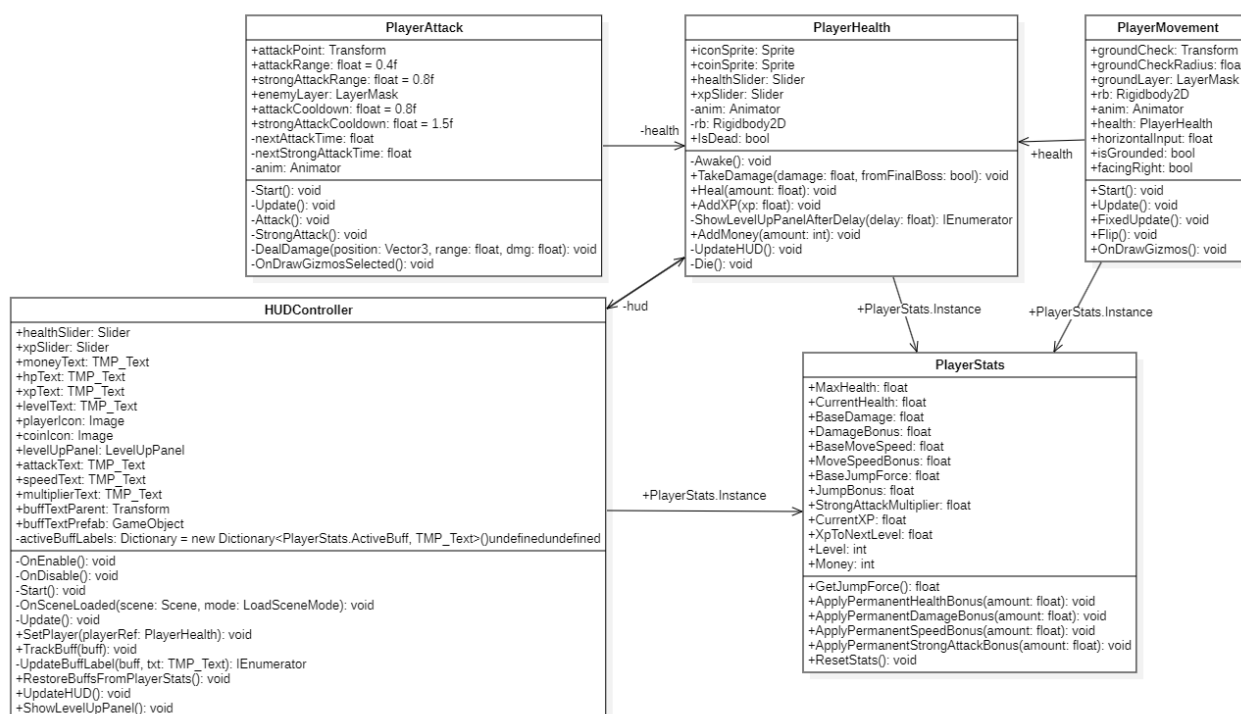


Рисунок 2.18 – Діаграма класів гравця

Наступна діаграма класів (рисунок 2.19) відображає логіку системи інвентарю та магазину. Центром взаємодії виступає клас InventoryManager який відповідає за зберігання, використання та управління предметами. Усі об'єкти інтерфейсу (InventoryItemUI, ShopItemUI) працюють з єдиною структурою

предмета – ItemData. Після придбання предмет передається з магазину до інвентарю, де зберігається й може бути використаний гравцем.

Наведено опис класів:

- InventoryManager (синглтон Instance) – керує списком предметів гравця, підтримує додавання, використання, видалення та завантаження інвентарю. Пов'язаний із ItemData для доступу до інформації про предмети, з InventoryItemUI для відображення в інтерфейсі, та з ShopItemUI для прийому нових предметів після купівлі;
- ItemData – структура ScriptableObject, яка містить всю інформацію про предмет: ID, назву, тип, ефект, іконку, ціну, опис тощо. Пов'язана з усіма класами, які відображають або оперують предметами;
- InventoryItemUI – UI-компонент, який відображає предмет у інвентарі;
- ShopItemUI – UI-компонент магазину, який показує доступний для купівлі предмет. Пов'язаний з PlayerStats для зняття монет, HUDController для оновлення інтерфейсу, та ShopPanelController для виведення повідомлень.

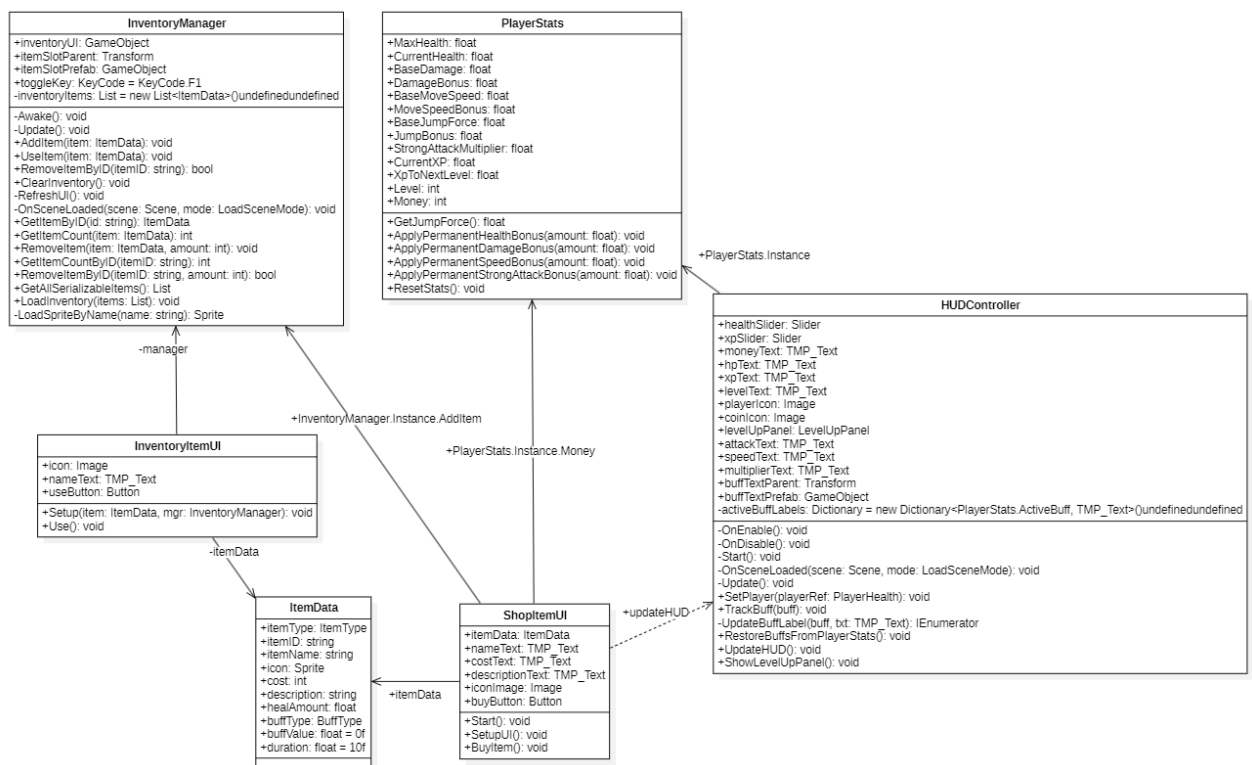


Рисунок 2.19 – Діаграма класів системи інвентарю та торгівлі

Діаграма компонентів (рисунок 2.20) відображає високорівневу архітектуру гри *Final Dawn*. Основні залежності описані у таблиці 2.2. Усі системи гри умовно розділені на три функціональні блоки:

- Game Logic — внутрішня логіка гри: рух гравця, бій, вороги, предмети, статистики та бафи;
- Client Side / UI — інтерфейс користувача: HUD, інвентар, магазин, меню паузи, система підвищення рівня та діалоги;
- Game Engine — основне ядро гри: збереження, аудіо, менеджер сцен і керування потоком гри.

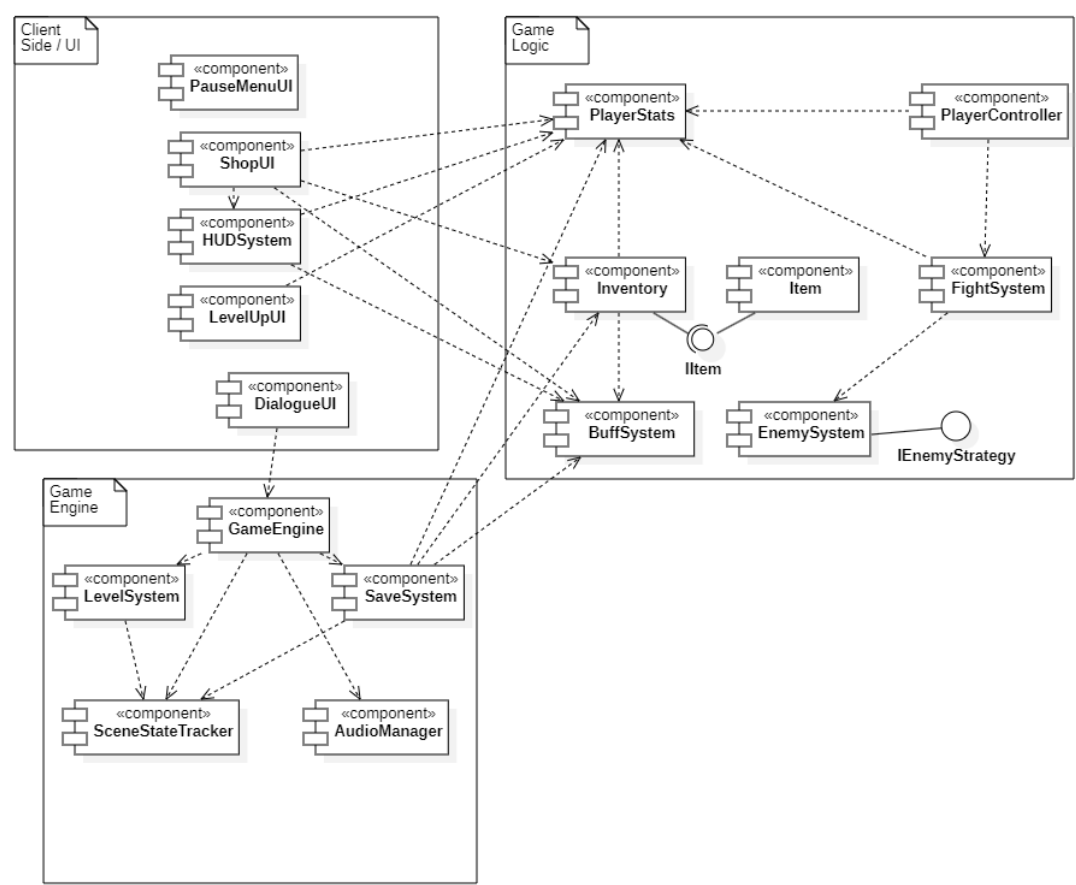


Рисунок 2.20 – Діаграма компонентів

Таблиця 2.2

Основні залежності між компонентами гри Final Dawn

Компонент	До якого компонента	Призначення
ShopUI	Inventory, PlayerStats	Покупка предметів, зняття грошей
InventoryUI	PlayerStats	Використання предметів

LevelUpUI	PlayerStats	Надбавки при підвищенні рівня
HUDSystem	PlayerStats	Відображення здоров'я, досвіду, монет
PlayerController	PlayerStats	Отримання швидкості, стану
FightSystem	PlayerStats, EnemySystem	Логіка бою та завдання шкоди
GameEngine	SaveSystem, LevelSystem, AudioManager	Керування ігровим процесом
SaveSystem	PlayerStats, Inventory	Збереження і відновлення прогресу

Діаграма розгортання (рисунок 2.21) зображує, як виконувані модулі гри Final Dawn взаємодіють із середовищем виконання Unity, локальною файловою системою та пристроєм користувача. Всі основні компоненти, такі як система управління гравцем, HUD, інвентар, діалоги, збереження гри та аудіо, функціонують усередині Unity RunTime. Збереження здійснюється через SaveSystem, який звертається до локальної файлової системи, де зберігаються файли конфігурації та прогресу гри.

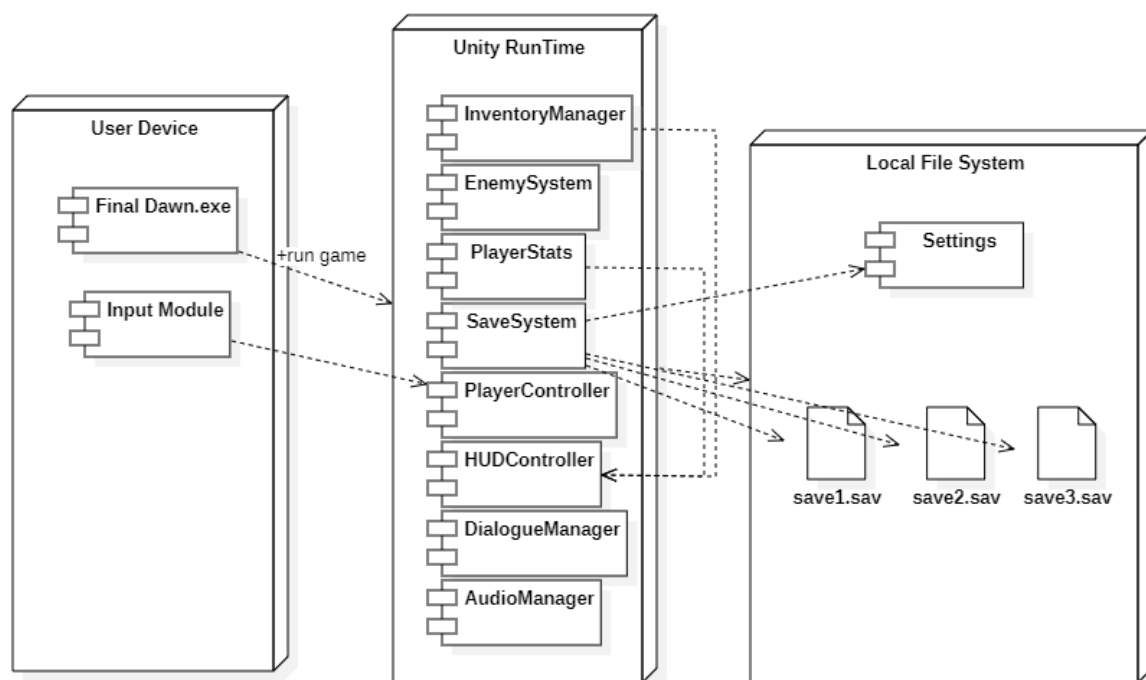


Рисунок 2.21 – Діаграма розгортання

2.3. Реалізація ігрових модулів та функціоналу

У даному підрозділі детально описується реалізація ключових модулів гри “Final Dawn”, зокрема логіка керування персонажем, бойова система, інвентар, система бафів, вороги, фінальний бос, а також модулі збереження гри. Основна мета – показати, як саме програмно реалізовано ігрову логіку, та як окремі системи взаємодіють між собою.

Анімаційний контролер головного героя побудований на базі Unity Animator Controller з використанням станів та параметрів, що активуються з коду відповідно до дій гравця [16]. У таблицях 2.3 та 2.4 наведено інформацію про стани анімацій та параметри самого Animator Controller. Також наведено приклад анімації смерті героїні на рисунку 2.22.

Таблиця 2.3

Основні стани та логіка анімацій

Стан	Умова активації
Idle	Коли швидкість дорівнює нульовому значенню і персонаж не стрибає, не атакує.
Run	Коли швидкість перевищує нульове значення.
JumpFall	Параметр isJumping = true
Attack	При активації тригера Attack
StrongAttack	При активації тригера StrongAttack
Hurt	При активації тригера hurt
Death	При активації тригера die

Таблиця 2.4

Параметри Animator Controller

Параметр	Тип	Призначення
Speed	Float	Значення руху по осі X
isJumping	Bool	Анімація стрибку та приземлення
hurt	Trigger	Активація анімації шкоди
die	Trigger	Активація анімації смерті
Attack	Trigger	Звичайна атака
StrongAttack	trigger	Посилена атака

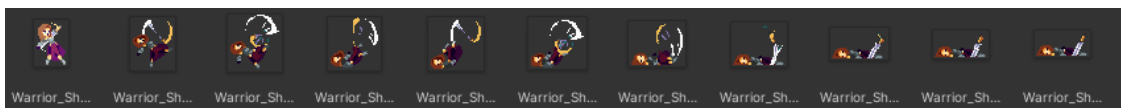


Рисунок 2.22 – Нарізані кадри анімації смерті героїні

Переходи між станами мають чітку логіку – початковий стан Idle (рисунок 2.23). Далі йде взаємодія Idle → Run, та навпаки (за відповідних умов значень Speed). Anystate → Attack/Hurt/StrongAttack/Death виконуються через тригер (тільки Death без повернень до звичайного стану Idle). Також для Attack/StrongAttack/Hurt використано Has Exit Time = true (для завершення анімації перед поверненням у стан Idle).

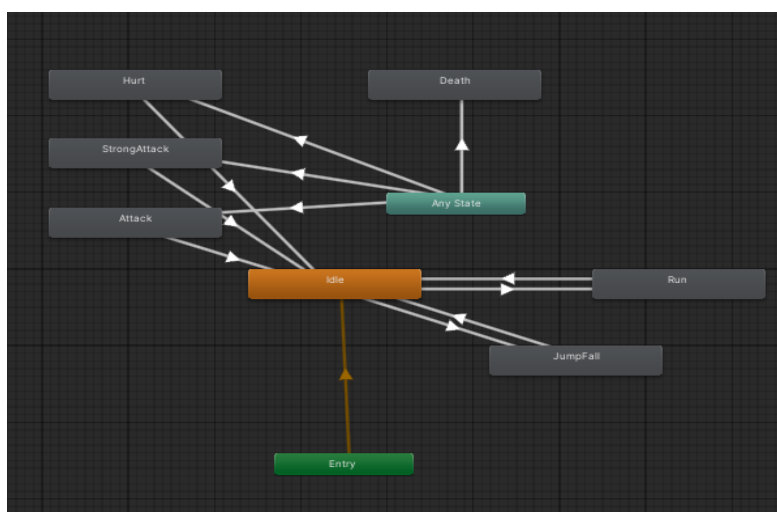


Рисунок 2.23 – Загальний вигляд Animator гравця

Для ворогів (рисунок 2.24) загальною схожий принцип роботи, єдина відмінність може полягати у параметрі spawned (який активує анімацію появи ворога при вході у певну тригер-зону).

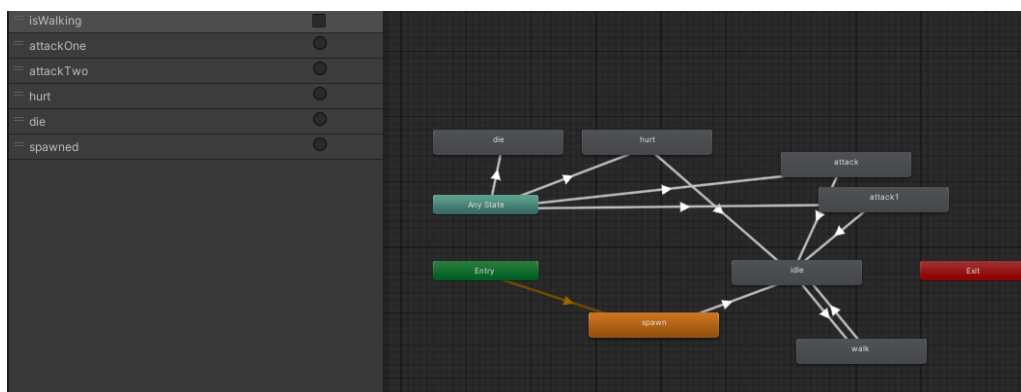


Рисунок 2.24 – Загальний вигляд Animator одного з ворогів

Керування гравцем та взаємодія з HUD. Головна героїня гри керується за допомогою клавіатури та взаємодіє з ігровим середовищем через систему анімацій, атаки, отримання шкоди, збирання досвіду та грошей. Також відбувається динамічне оновлення даних через HUD. Логіка розподілена на декілька скриптів, які відповідають за окремі аспекти. Пересування та стрибки керуються скриптом `PlayerMovement`.

Гравець пересувається по горизонталі та може стрибати, якщо стоїть на землі. Перевірка дотику до землі здійснюється через `OverlapCircle` [17]:

```
isGrounded = Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius, groundLayer);
```

Якщо натиснуто кнопку `Jump`, і гравець знаходиться на землі — виконується стрибок із силою, яка враховує бонус від бафів:

```
if (Input.GetButtonDown("Jump") && isGrounded)
{
    rb.velocity = new Vector2(rb.velocity.x, jumpForce); // jumpForce залежить від PlayerStats
}
```

Атака та бойова взаємодія керується скриптом `PlayerAttack`. Гравець має дві форми атаки: стандартну (F) та сильну (G). Обидві атаки мають затримку через `cooldown` та діапазон дії. Атака виконується через перевірку ворогів у радіусі `OverlapCircleAll`, після чого викликається метод `TakeDamage ()`:

```
private void DealDamage(Vector3 position, float range, float dmg)
{
    Collider2D[] array = Physics2D.OverlapCircleAll(position, range, enemyLayer);
    foreach (Collider2D collider2D in array)
    {
        EnemyHealth eh = collider2D.GetComponentInParent<EnemyHealth>();
        if (eh != null && !eh.IsDead)
        {
            eh.TakeDamage(dmg);
        }
    }
}
```

Шкода розраховується динамічно, залежно від базової атаки та бонусу з синглтона `PlayerStats`:

```
float finalDamage = PlayerStats.Instance.BaseDamage + PlayerStats.Instance.DamageBonus;
```

Скрипт `PlayerHealth` керує усіма можливими маніпуляціями зі здоров'ям гравця та його прокачкою. Отримання шкоди:

```
public void TakeDamage(float damage)
{
    PlayerStats.Instance.CurrentHealth -= damage;
    anim.SetTrigger("hurt");
    UpdateHUD(); // оновлюємо HUD
}
```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		35

При нульовому значення HP запускається анімація смерті, блокується керування та викликається GameOverHandler:

```
if (PlayerStats.Instance.CurrentHealth <= 0f)
{
    Die();
    handler.ShowGameOver(); // панель програшу
}
```

Збирання ХР:

```
public void AddXP(float xp)
{
    PlayerStats.Instance.CurrentXP += xp;
    if (PlayerStats.Instance.CurrentXP >= PlayerStats.Instance.XpToNextLevel)
    {
        PlayerStats.Instance.Level++;
        hud.ShowLevelUpPanel(); // відкривається панель прокачки
    }
}
```

Після підвищення рівня гравцю пропонується вибрати бонус, наприклад, +5 до атаки, ці бонуси мають перманентну дію та одразу оновлюються в HUD:

```
HUDController hud = FindObjectOfType<HUDController>();
hud.UpdateHUD();
```

Статистика гравця керується через PlayerStats. Цей синглтон зберігає всі базові й бонусні характеристики гравця, а також керує тимчасовими бафами, які зникають після певного часу:

```
public class ActiveBuff
{
    public BuffType type;
    public float value;
    public float timeRemaining;
}
```

Оновлення HUD відбувається за допомогою HUDController. Компонент відповідає за оновлення інтерфейсу гравця (також бонусні значення):

```
public void UpdateHUD()
{
    healthSlider.value = stats.CurrentHealth / stats.MaxHealth;
    xpSlider.value = stats.CurrentXP / stats.XpToNextLevel;
    hpText.text = $"{stats.CurrentHealth} / {stats.MaxHealth}";
    moneyText.text = stats.Money.ToString();
    levelText.text = "LVL: " + stats.Level;
}
attackText.text = (stats.BaseDamage + stats.DamageBonus).ToString("F1");
speedText.text = (stats.BaseMoveSpeed + stats.MoveSpeedBonus).ToString("F1");
multiplierText.text = "x" + stats.StrongAttackMultiplier.ToString("F1");
```

Керування ворогами та їх типи. У грі “Final Dawn” реалізовано кілька типів ворогів з різною поведінкою [18]. Класифікуємо їх за принципом звичайних, літаючих та боси.

Звичайні вороги в Final Dawn – це наземні супротивники, які мають просту логіку пересування, виявлення гравця, атаки та смерті.

Патрулювання відбувається за допомогою скрипта EnemyPatrol. Ворог рухається вперед до моменту, поки не побачить край платформи або стіну:

```
bool noGround = !Physics2D.Raycast(groundCheck.position, Vector2.down, checkDistance, groundLayer);  
bool wallAhead = Physics2D.Raycast(wallCheck.position, facingRight ? Vector2.right : Vector2.left, checkDistance, groundLayer);
```

Деякі вороги не рухаються доки в їх зоні не буде гравця, після цього вони починають переслідувати його доки він в зоні їх триггеру:

```
playerInRange = Vector2.Distance(transform.position, player.position) <= agroRange;  
// Напрямок руху  
Vector2 direction = (player.position - transform.position).normalized;  
enemy.SetVelocity(new Vector2(direction.x * enemy.moveSpeed, 0f));
```

Якщо гравець попадає в радіус атаки, запускається анімацію атаки та ворог завдає шкоду:

```
if (Physics2D.OverlapCircle(attackPoint.position, attackRange, playerLayer))  
    StartCoroutine(Attack());  
// Виклик шкоди  
public void DealDamage()  
{  
    Collider2D hit = Physics2D.OverlapCircle(attackPoint.position, attackRange, playerLayer);  
    hit?.GetComponent<PlayerHealth>()?.TakeDamage(damage);  
}
```

Ворог має власне здоров'я, після шкоди активується анімації hurt, а при 0 HP – смерть:

```
public void TakeDamage(float damage)  
{  
    currentHealth -= damage;  
    if (currentHealth <= 0f) Die();  
}
```

Для зручності дебагу використовується спеціальний метод у Unity OnDrawGizmosSelected (рисуюнок 2.25) – для відображення зони атаки та агро:

```
Gizmos.DrawWireSphere(attackPoint.position, attackRange);  
Gizmos.DrawWireSphere(bodyBlockPoint.position, blockRange);
```



Рисуюнок 2.25 – Умовне зображення зони атаки та тригера

Літаючі вороги мають відмінну логіку – вони не прив’язані до землі та не використовують фізику платформінга. Ворог активується, якщо гравець наблизиться в зону detectionRadius:

```
if (!isAwake && Vector2.Distance(transform.position, player.position) <= detectionRadius)
    WakeUp();
```

Переслідування відбувається з анімацією Fly:

```
Vector2 vector = (player.position - transform.position).normalized;
rb.velocity = vector * moveSpeed;
anim.SetBool("isFlying", true);
```

Якщо ворог наближається до гравця на attackRange, запускається атака:

```
if (num <= attackRange && Time.time >= nextAttackTime)
    StartAttack();
```

Босів також можна поділити на 3 підкатегорії: звичайні боси, особливий бос – Dark Player, особливий бос – Demon. Звичайні боси керуються скриптами BossHealth, BossBehaviour. На відміну від звичайних ворогів, боси спавняються тільки після входу гравця в зону тригера:

```
private void OnTriggerEnter2D(Collider2D other)
{
    if (!activated && other.CompareTag("Player"))
    {
        activated = true;
        DialogueUI.StartDialogue(introDialogueLines, OnDialogueComplete);
    }
}
```

Після активації бос починає переслідування гравця:

```
Collider2D hit = Physics2D.OverlapCircle(attackPoint.position, attackRange, playerLayer);
```

Завдання шкоди відбувається через DealDamage (). Після досягнення 0 НР бос окрім смерті, бос може залишити після себе дроп (гем, портал, діалогова вставка):

```
Instantiate(gemPrefab, transform.position, Quaternion.identity);
Instantiate(portalPrefab, portalSpawnPoint.position, Quaternion.identity);
DialogueUI.StartDialogue(deathDialogueLines, () => Destroy(gameObject));
```

Dark Player – це дзеркальне відображення головного героя, має власний AI-контроль руху, атак та дій. Бос має агресивну поведінку переслідування при наближенні гравця та урахування висоти. Якщо гравець знаходиться вище певного порогу, бос виконує стрибок щоб дістатись до гравця:

```
if (isGrounded && player.position.y > transform.position.y + jumpThreshold)
{
    rb.velocity = new Vector2(rb.velocity.x, jumpForce);
}
```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

Demon – головний бос гри і найбільш комплексний противник. Унікальна здатність – телепортація при отриманні шкоди, яка активується з певним шансом:

```
if (Time.time >= lastTeleportTime + teleportCooldown && Random.value < teleportChance)
{
    anim.SetTrigger("teleportOut");
    lastTeleportTime = Time.time;
}
```

Після виходу з анімації телепортації (teleportOut) викликається метод PerformTeleport (), який переміщує боса між двома заданими точками (teleportPoint A/B):

```
transform.position = atPointA ? teleportPointB.position : teleportPointA.position;
```

Бос має дві основні атаки, які обираються випадково. Перша атака викликає метод CastProjectileA, друга – CastProjectileB. Снаряди летять у напрямку позиції гравця:

```
GameObject p = Instantiate(magicProjectileA, magicSpawnPoint.position, Quaternion.identity);
p.GetComponent<Rigidbody2D>().velocity = dir * 7f;
// Позиція гравця
dir = (player.position - spawnPoint).normalized
```

Демон також може атакувати гравця, якщо він занадто близько через метод DealDamage ().

Після знищення боса відбувається діалог та завантаження переможної сцени WinScene.

Система інвентарю та торгівлі. Серцем системи інвентарю є скрипт InventoryManager. Він зберігає список усіх предметів та відповідає за логіку використання, збереження і оновлення UI.

Додавання предмета:

```
public void AddItem(ItemData item)
{
    if (item == null) return;

    inventoryItems.Add(item);
    RefreshUI();
}
```

Після додавання – викликається метод RefreshUI (), який перебудовує список слотів у UI-компоненті. Використання предмета:

```
public void UseItem(ItemData item)
{
    switch (item.itemType)
```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		39

```

{
    case ItemData.ItemType.Heal:
        FindObjectOfType<PlayerHealth>()?.Heal(item.healAmount);
        break;

    case ItemData.ItemType.Buff:
        var stats = FindObjectOfType<PlayerStats>();
        var buff = new PlayerStats.ActiveBuff(
            (PlayerStats.BuffType)item.buffType, item.buffValue, item.duration);
        stats.activeBuffs.Add(buff);
        stats.ApplyBuffCoroutine(buff);
        FindObjectOfType<HUDController>()?.TrackBuff(buff);
        break;
    inventoryItems.Remove(item);
    RefreshUI();
}

```

Інтерфейс реалізовано через InventoryItemUI. Кожен айтем має іконку, назву та кнопку Use. Налаштування елемента:

```

public void Setup(ItemData item, InventoryManager mgr)
{
    itemData = item;
    manager = mgr;

    icon.sprite = item.icon;
    nameText.text = item.itemName;
    useButton.onClick.RemoveAllListeners();
    useButton.onClick.AddListener(Use);
    useButton.interactable = item.itemType != ItemData.ItemType.Gem;
}

```

Магазин реалізований з використанням префабів ShopItemUI, які генеруються скриптом ShopInitializer. Ініціалізація магазину:

```

foreach (ItemData item in availableItems)
{
    GameObject obj = Instantiate(shopItemPrefab, shopItemParent);
    ShopItemUI ui = obj.GetComponent<ShopItemUI>();
    ui.itemData = item;
    ui.SetupUI();
}

```

Придбати предмет (якщо немає грошей – виводиться попередження):

```

public void BuyItem()
{
    if (PlayerStats.Instance.Money >= itemData.cost)
        PlayerStats.Instance.Money -= itemData.cost;
    InventoryManager.Instance.AddItem(itemData);
    HUDController hud = FindObjectOfType<HUDController>();
    hud?.UpdateHUD();
    Destroy(gameObject); // Видалити з вітрини
}

```

Типи предметів ItemData:

```

public enum ItemType { Heal, Buff, Gem }
public enum BuffType { Damage, MoveSpeed, StrongAttack, JumpBoost }

```

Приклад налаштування предмета (ScriptableObject [19]):

```

[CreateAssetMenu(menuName = "Items/ItemData")]
public class ItemData : ScriptableObject
{

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

```

public ItemType itemType;
public BuffType buffType;
public string itemID;
public string itemName;
public Sprite icon;
public int cost;
public float healAmount;
public float buffValue;
public float duration;
}

```

Бафи застосовуються через PlayerStats, вони додають тимчасовий ефект (наприклад урон або швидкість), який триває певну кількість часу:

```

public Coroutine ApplyBuffCoroutine(ActiveBuff buff)
{
    Coroutine coroutine = StartCoroutine(BuffCoroutine(buff));
    activeCoroutines[buff] = coroutine;
    return coroutine;
}

```

Усі активні бафи описані через внутрішню структуру:

```

public class ActiveBuff
{
    public BuffType type; // Тип бафу (наприклад, Damage або MoveSpeed)
    public float value; // Наскільки змінюється характеристика
    public float duration; // Загальна тривалість
    public float timeRemaining; // Час, що залишився (для відновлення після сейву)
}

```

Логіка дії бафу в часі (у корутині відбувається поступовий зворотній відлік дії ефекту):

```

private IEnumerator BuffCoroutine(ActiveBuff buff)
{
    ApplyBuff(buff, true); // Застосувати ефект
    while (buff.timeRemaining > 0f)
    {
        yield return new WaitForSeconds(1f);
        buff.timeRemaining -= 1f;
    }
    ApplyBuff(buff, false); // Зняти ефект
    activeBuffs.Remove(buff);
    activeCoroutines.Remove(buff);
}

```

Метод ApplyBuff динамічно змінює потрібну характеристику:

```

private void ApplyBuff(ActiveBuff buff, bool apply)
{
    float value = apply ? buff.value : -buff.value;
    switch (buff.type)
    {
        case BuffType.Damage:
            tempDamageBonus += value;
            DamageBonus = permanentDamageBonus + tempDamageBonus;
            break;
        // Інші типи: MoveSpeed, StrongAttack, JumpBoost
    }
}

```

Система повністю підтримує збереження/відновлення інвентарю через серіалізацію:

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						41
Змн.	Арк.	№ докум.	Підпис	Дата		

```
public List<SerializableItem> GetAllSerializableItems()
```

Цей метод повертає список предметів у вигляді простих структур (SerializableItem), які зручно зберігати у файлі. Завантаження інвентарю:

```
public void LoadInventory(List<SerializableItem> items)
```

Це дозволяє відновити всі предмети після перезапуску гри.

Система збереження. Система збереження реалізована через серіалізацію даних у файл .sav окремо для кожного слота [20]. Вона підтримує такі речі:

- збереження та відновлення поточної сцени;
- повний стан гравця (ХР, НР, рівень, бафи, гроші);
- інвентар;
- позицію гравця;
- стан об'єктів на сцені (активовані/вимкнені/зібрані/переміщені);
- відновлення всього цього після завантаження.

SaveSystem – основна логіка збереження та завантаження:

```
public static void SaveGame(int slot)
{
    SaveData data = new SaveData();
    data.sceneName = SceneManager.GetActiveScene().name;
    data.playerHP = PlayerStats.Instance.CurrentHealth;
    data.playerXP = PlayerStats.Instance.CurrentXP;
    data.playerLevel = PlayerStats.Instance.Level;
    data.money = PlayerStats.Instance.Money;

    data.buffStates = PlayerStats.Instance.activeBuffs
        .Select(b => new SaveData.BuffState
        {
            type = b.type,
            value = b.value,
            duration = b.duration,
            timeRemaining = b.timeRemaining
        }).ToList();

    data.inventoryItems = InventoryManager.Instance.GetAllSerializableItems();
    data.playerPosition = new SerializableVector3(GameObject.FindWithTag("Player").transform.position);

    data.disabledObjectIDs = SceneStateTracker.Instance.GetAllDisabledIDs();
    data.activatedObjectIDs = SceneStateTracker.Instance.GetAllActivatedIDs();
    data.objectPositions = SceneStateTracker.Instance.GetObjectPositions()
        .ToDictionary(kvp => kvp.Key, kvp => new SerializableVector3(kvp.Value));

    File.WriteAllBytes(GetSavePath(slot), BinarySerialize(data));
}
```

Завантаження гри:

```
public static SaveData LoadGame(int slot)
{

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

```

string path = GetSavePath(slot);
return File.Exists(path) ? BinaryDeserialize<SaveData>(File.ReadAllBytes(path)) : null;
}

```

SaveData – структура збереження даних:

```

public class SaveData
{
    public string sceneName;
    public float playerHP, playerXP;
    public int playerLevel, money;
    public SerializableVector3 playerPosition;
    public List<SerializableItem> inventoryItems;
    public List<BuffState> buffStates;
    public List<string> disabledObjectIDs, activatedObjectIDs;
    public Dictionary<string, SerializableVector3> objectPositions;
    [Serializable]
    public class BuffState
    {
        public PlayerStats.BuffType type;
        public float value, duration, timeRemaining;
    }
}

```

GameInitializer — відновлення після завантаження:

```

void Start()
{
    if (PlayerPrefs.HasKey("LoadSlot"))
    {
        SaveData data = SaveSystem.LoadGame(PlayerPrefs.GetInt("LoadSlot"));

        PlayerStats.Instance.LoadPlayer(data);
        InventoryManager.Instance.LoadInventory(data.inventoryItems);
        GameObject.FindWithTag("Player").transform.position = data.playerPosition.ToVector3();

        SceneStateTracker.Instance.LoadStateFromSave(data);
        PlayerPrefs.DeleteKey("LoadSlot");
    }
}

```

SceneStateTracker – контроль активних та вимкнених об'єктів на ігровій сцені перед завантаженням:

```

public void LoadStateFromSave(SaveData data)
{
    disabledObjects = new HashSet<string>(data.disabledObjectIDs);
    activatedObjects = new HashSet<string>(data.activatedObjectIDs);
    objectPositions = data.objectPositions.ToDictionary(kvp => kvp.Key, kvp => kvp.Value.ToVector3());
}

```

SceneObjectID – унікальний ідентифікатор об'єкта сцени:

```

uniqueID = gameObject.name + "_" + transform.position.ToString("F2");

```

PauseMenu / MainMenu – виклик функцій збереження / завантаження:

```

public void SaveToSlot1() => SaveSystem.SaveGame(1);
public void LoadSlot(int slot)
{
    SaveData data = SaveSystem.LoadGame(slot);
    if (data != null)
    {
        PlayerPrefs.SetInt("LoadSlot", slot);
        SceneManager.LoadScene(data.sceneName);
    }
}

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		43

}

Ця система дозволяє повноцінно зберігати та відновлювати стан гри у будь-якому з трьох слотів, враховуючи НР, ХР, бафи, інвентар, позиції та логіку сцени.

Висновки до другого розділу

У даному розділі було побудовано повноцінну архітектуру гри *Final Dawn*. Визначено ресурсну базу, розроблено структуру сцени з використанням Tilemap, налаштовано фізику, колайдери та анімаційні компоненти.

Розглянуто компонентну архітектуру проєкту, побудовано UML-діаграми взаємодії ключових скриптів. Було реалізовано основні модулі гри: управління гравцем, поведінку ворогів, логіку босів, інвентар, магазин, систему збереження, діалоги та HUD. Усі системи інтегруються між собою та забезпечують цілісний ігровий процес.

Таким чином, сформовано технічну базу для повноцінної роботи рівня та підготовлено гру до фінального тестування.

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		44

РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ

3.1. Вимоги та середовище виконання

Для розробки гри “*Final Dawn*” було обрано програмне середовище, яке забезпечує кросплатформність, високу продуктивність та зручну модульну структуру. Для запуску гри необхідно дотриматись певних апаратних та програмних умов, що гарантують стабільну роботу та коректне відображення графіки [21]. Гра орієнтована на персональні комп’ютери з операційною системою Windows.

Нижче наведено таблиці 3.1 та 3.2 з вимогами для запуску гри.

Таблиця 3.1

Мінімальні системні вимоги до гри

Компонент	Характеристики
Операційна система	Windows 10 (64-bit)
Процесор	Intel Core i3 4-го покоління / AMD аналог
Оперативна пам’ять	4 GB RAM
Відеокарта	Вбудована Intel HD / AMD Vega / GTX 650
Дисплей	800 x 600 або вище
Вільне місце	300 MB
Інше	Підтримка DirectX 11, клавіатура + миша

Таблиця 3.2

Рекомендовані системні вимоги до гри

Компонент	Характеристики
Операційна система	Windows 10/11 (64-bit)
Процесор	Intel Core i5 або AMD Ryzen 5
Оперативна пам’ять	8 GB RAM
Відеокарта	NVIDIA GTX 1050 / AMD RX 560
Дисплей	1920 x 1080, підтримка fullscreen режиму
Вільне місце	До 500 MB з урахуванням файлів збереження
Інше	Вбудована звукова карта, підтримка AudioMixer

Умови експлуатації та використання:

- гра не потребує інсталяції, запускається з каталогу через .exe файл в результаті Unity Build Settings;
- при запуску гра автоматично налаштовує роздільну здатність, але дозволяє змінити її вручну через меню опцій;
- підтримуються два режими: повноекранний та віконний, перемикаються за допомогою налаштувань гри;
- користувач має можливість вибору графічних параметрів (звук, гучність, роздільність) із авто-збереженням налаштувань у наступних ігрових сесіях.

Основне тестування роботи гри проводилось на ПК з ОС Windows 10 (64-bit), відеокарта: NVIDIA GeForce RTX 3060 (12 GB).

Під час тестування гра запустилась без збоїв у Full HD форматі, все працювало стабільно, також перевірена робота на різних налаштуваннях роздільної здатності та зі зміною віконного режиму.

3.2. Взаємодія користувача з ігровим середовищем

Після запуску exe-файлу гри – користувач бачить одразу головне меню (рисунок 3.1), яке має наступні кнопки: New Game, Load Game, Options, Exit. Меню гри оформлено у стилі dark fantasy з елементами ретро-естетики. Фон меню темний, з ледь помітною текстурою та містить великими літерами назву програмного продукту.

Усі кнопки мають білий прямокутний дизайн з чорним піксельним шрифтом. Інтерфейс залишається лаконічним та простим у розумінні, легко читається на основних розширеннях. Також гравець може чути мелодію, яка має повільний, атмосферний темп з темною фентезійною інтонацією, що одразу задає настрій гри.

Музичний супровід у повній мірі занурює користувача у світ гри, поступово вводячи в атмосферу загадковості та небезпеки, які будуть чекати його попереду.

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						46
Змн.	Арк.	№ докум.	Підпис	Дата		



Рисунок 3.1 – Головне меню

Надалі розглянемо доступний функціонал після натискання на кнопку Options. Користувач бачить перед собою нову панель опцій (рисунок 3.2), яка містить новий набір функціоналу: кнопка Music, кнопка About Game, кнопку Back, випадаючий список роздільної здатності програмного вікна, чекбокс з підписом “Fullscreen on/off”.



Рисунок 3.2 – Меню опцій

Натиснувши на кнопку Music, користувач має змогу бачити перед собою чергову панель (рисунок 3.3), яка має чекбокс “Music on/off” (задає загальні налаштування, музика вмикається або вимикається саме через цю панель), слайдер гучності, та кнопку Back – яка поверне до попередньої сторінки.



Рисунок 3.3 – Меню музичних налаштувань

Далі кнопка About Game. Коли користувач натисне на неї, буде нова панель (рисунок 3.4) яка дає список клікабельних кнопок з іменем того чи іншого персонажа. При натисненні на будь яку з кнопок, відкриється інформація про героя, його зображення та короткий опис. Також є кнопка, щоб повернутись назад. Візуальний вигляд інформації буде зображено на рисунках 3.5, 3.6.

Кнопка Back, як зазначено раніше, має повернути користувача у попередню панель, у наступних панелях кнопка не буде описана, так як функціонал є однаковим для будь-якої панелі.

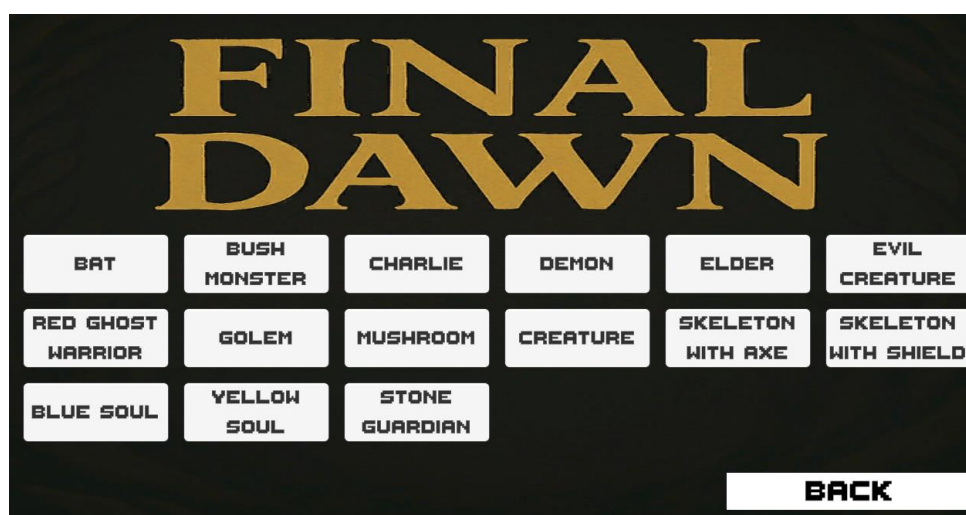


Рисунок 3.4 – Панель “About Game”

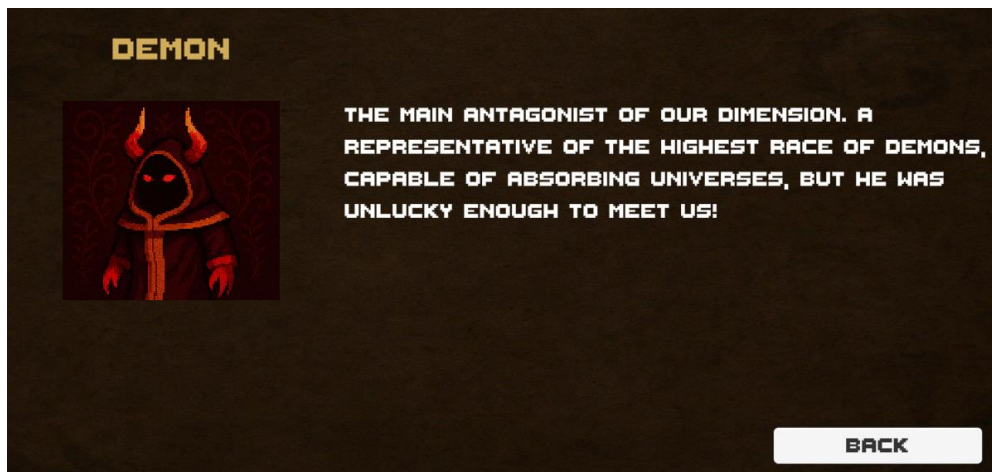


Рисунок 3.5 – Опис головного антагоніста гри “Final Dawn”

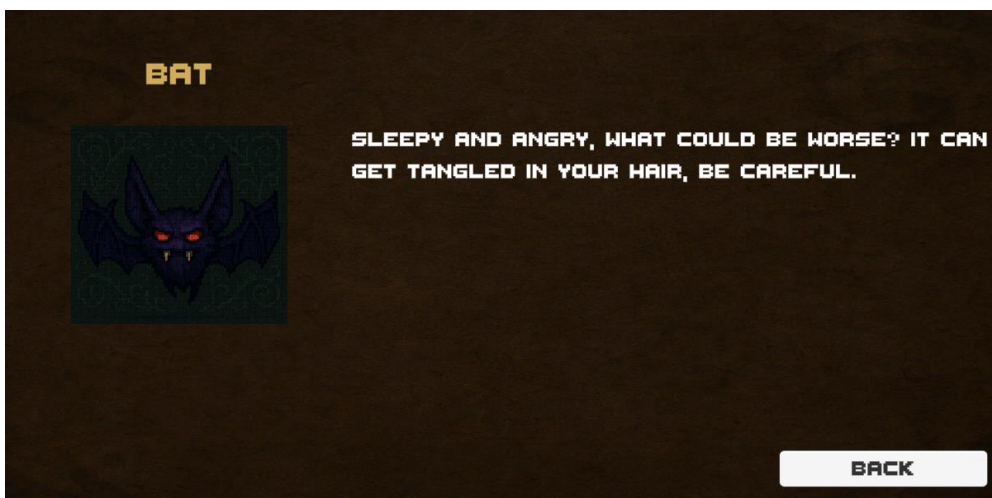


Рисунок 3.6 – Опис рядового ворога однієї з локацій

Також присутня можливість обрати роздільну здатність через випадаючий список, та обрати режим (повноекранний або неповне вікно додатку) за допомогою чекбоксу “FullScreen on/off”.

Повернемось до огляду функціоналу кнопок, що зазначені на рисунку 3.1. Кнопка LoadGame – викликає панель (рисунок 3.7), у який присутні 3 слоти для збереження гри. При натисненні кнопок Load 1, Load 2, Load 3 – завантажиться те збереження, яке гравець зробив під час ігрової сесії. Кожне завантаження починається з моменту, на якому було застосоване – з відповідними налаштуваннями, які заздалегідь були встановлені в ігровому меню.



Рисунок 3.7 – Меню завантажених ігрових сесій

Кнопки New Game та Exit – починають нову гру і виконують завершення роботи програмного забезпечення відповідно.

Запускаємо перший ігровий рівень та бачимо перед собою загальну картинку вступного рівня (рисунок 3.8): головну героїню – безіменну войовницю, поруч знаходиться статуя, що є інформаційним об’єктом, який має тригер, що активує діалогову панель (рисунок 3.9). Також присутній супутник Charlie (зелене створіння, що літає біля статуї) та першого ворога (зелений куц, що патрулює свої володіння). Зверху можна помітити HUD гравця, який містить дані про здоров’я та досвід персонажа (у лівому верхньому кутку), та дані про кількість монет, швидкість пересування, шкода від звичайної та посиленої атаки (у правому верхньому кутку) [22].

Для того щоб рухатись – можна використати клавіші A D (рух у ліву та праву сторони відповідно) або ж стрілки. Стрибок активується на пробіл, при наближенні ворога рекомендовано застосувати атаку – натиснувши клавішу F, це є звичайною атакою персонажа.



Рисунок 3.8 – Вигляд початкової локації та елементів HUD



Рисунок 3.9 – Активоване діалогове вікно з Charlie

Також при активації ігрового рівня гравця зустрічає мелодія, яка створює атмосферу пригоди на початку великого шляху. Вона легка, мелодійна та вже з першими нотками тривоги – вісник того, що цей світ не такий вже й безпечний.

Варто вступити в перший бій, застосувавши свою зброю – і можна буде помітити зміни в HUD (просування experience-шкали), також виводиться повідомлення (рисунок 3.10) про те, що гравець при досягненні нового рівня отримає можливість обрати бонус (збільшення здоров'я, шкоди, монет тощо) – що виводиться на окремій панелі (рисунок 3.11) при досягненні нового рівня шляхом знищення ворогів, які дають очки досвіду. Обираючи будь-яку з характеристик, дані динамічно оновлюються в HUD.

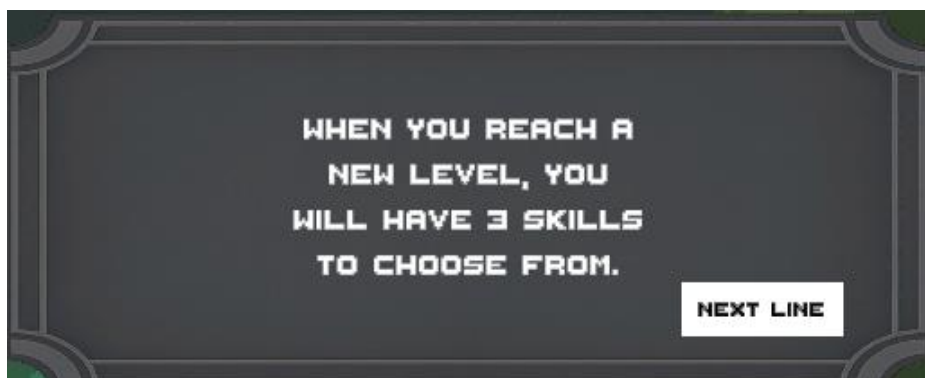


Рисунок 3.10 – Повідомлення після смерті ворога



Рисунок 3.11. Панель вибору умінь після досягнення нового рівня

Під кінець рівня гравець зустрічається з головним суперником першої локації – Stone Guardian (рисунок 3.12), який не дає нам можливості увійти в замок, для того щоб увійти в замок, спочатку потрібно його здолати – застосувавши звичайну атаку на клавішу F, або ж сильну атаку – G.



Рисунок 3.12 – Зустріч з вартовим

Після перемоги над першим серйозним ворогом – знову активується діалогове вікно, яке повідомляє що отримано гроші, які в майбутньому є можливість використати у магазині, також після смерті Stone Guardian з нього

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		52

випадає магічний гем, який є одним з ключів до відкриття порталу у майбутньому, тож варто його підібрати та перевірити його наявність (рисунок 3.13) у інвентарі (клавіша – F1).

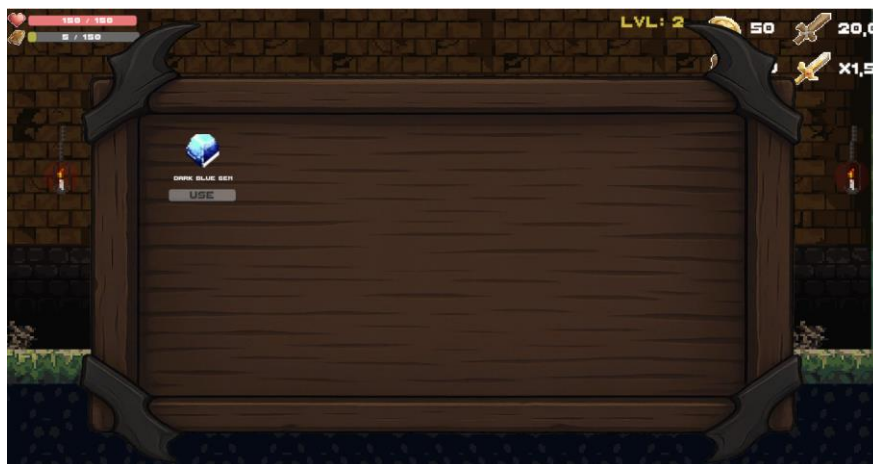


Рисунок 3.13 – Загальний вигляд ігрового інвентарю

Наступна ігрова локація – печера (рисунок 3.14), яка супроводжується відповідною мелодією, що передає відчуття напруженості з повільним темпом та темною атмосферою. Гравця починають атакувати кажани, є можливість втекти від них, або атакувати їх, проте вони можуть наздогнати у будь-якій точці локації, тому варто знищити їх та отримати при цьому грошову нагороду та очки досвіду.



Рисунок 3.14 – Другий ігровий рівень та його рядові вороги

Стрибаючи вниз локації, гравець зустрічає залишок кажанів та має змогу їх подолати, також по дорозі до магазину активується один із рядових ворогів локації – Ghost Warrior, його вбивство принесе гроші. У активованому магазині

(рисунок 3.15) представлено ряд товарів із відповідним описом, ціною та кнопкою Buy, яка дає змогу придбати їх і використати виключно в інвентарі, куди вони автоматично додаються після покупки.. Задля взаємодії з магазином – користувач має використати кнопку F2 (відкрити/закрити).



Рисунок 3.15 – Вигляд магазину та його товари

Гравцю не обов’язково витратити кошти в магазині, на кожному рівні починаючи з другого – присутній магазин, і можна накопити грошей на щось краще у майбутньому. Пройшовши до кінця локації, гравець має змогу активувати головного боса цієї локації – Evil Creature, після перемоги над яким з’явиться червоний гем та портал, який веде до наступної локації. Задля того щоб не виникало проблем з подоланням ворога, застосуємо деякі придбані еліксири, та побачимо їх ефект зверху зліва одразу під HUD (рисунок 3.16)



Рисунок 3.16 – Згенерований портал та активовані бафи у верхній панелі

Третій рівень Dark Forest починається в холі замку, атмосфера різко змінюється і через мелодію передається загрозлива, параноїдальна атмосфера з

відчуттям, що за персонажем стежать з кожного кута. Потрібно пробігти по коридору вперед, долаючи скелетів з абордажною сокирою, отримуючи з них нагороду як і з будь-яких рядових ворогів на кожній локації.



Рисунок 3.17 – Вихід з замку у темний ліс в рамках рівня Dark Forest



Рисунок 3.18 – Магазин під охороною скелетів на третій локації

Знищивши усіх ворогів локації – гравець має можливість активувати одного з босів та провести з ним бійку (рисунок 3.19), бос спритний та має велику шкоду, тому варто перед цим закупити еліксири та їжу, застосовуючи їх перед початком бою задля покращення шансів перемоги над ворогом та поновлення поточного стану здоров'я. Поединок з босом принесе гроші та третій по рахунку гем (рисунок 3.21), який є гарантією проходження сюжетної гілки.

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		55



Рисунок 3.19 – Проведення бою з ворогом, нанесення шкоди нашій героїні

Варто згадати про опцію збереження гри, на випадок якщо бос опиниться сильніше за гравця. Нажимаючи клавішу Esc, активується панель паузи (рисунок 3.20), у якій можна вийти в головне меню, зробити збереження у один з трьох слотів або продовжити гру. У випадку якщо ворог подолав героїню, то з'являється повідомлення про смерть та можливість виходу до головного меню, де представлена можливість завантажити сейв і провести бій ще раз, вже з більш успішним результатом.



Рисунок 3.20 – Меню паузи



Рисунок 3.21 – Заспавнений золотий гем після загибелі боса

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		56

Кульмінаційний момент третього рівня – зустріч з темною версією нашої героїні, з діалогів можна зрозуміти, що спотворена енергія темного лісу створила повноцінну копію гравця – яка має бути знищена і після цього отримати останній гем. Також відкривається портал, який веде до фінальної вежі.

Dark Player (темна версія гравця) на відміну від інших ворогів не тільки переслідує персонажа (рисунки 3.22), а й може стрибати за ним, рекомендовано мати в інвентарі ресурси (рисунки 3.23), що допоможуть у бою з демонічною версією героїні.

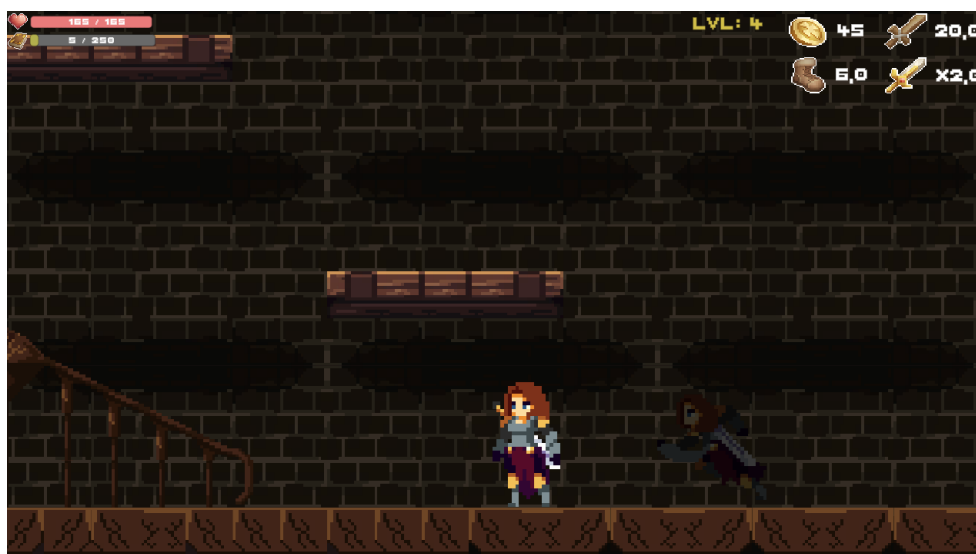


Рисунок 3.22 – Зустріч персонажа з іншою версією самого себе



Рисунок 3.23 – Стан інвентарю перед боєм з останнім ворогом ігрової локації

Четверта і остання локація гри “Final Dawn” – Вежа. Мелодія для фінального рівня – похмура, монументальна та напружена. Вона одразу

викликає відчуття, що гравець наблизився до кінця, але найголовніша небезпека – попереду. На початку рівня гравця зустрічають знайомі з попередніх локацій мерці – скелети (рисунок 3.24).



Рисунок 3.24 – Битва зі скелетами в останній локації

Пройшовши коридор зі скелетами, є можливість входу в саму вежу на перший рівень перешкод. На кожному рівні перешкод є 2 кімнати. У першій кімнаті проходячи повз залізні ланцюги герой активує пастку, яка запускає магічний заряд (рисунок 3.25), що наносить шкоду гравцю. Є можливість її перестрибнути.



Рисунок 3.25 – Магічний заряд, що летить у напрямку гравця

Наступна перешкода – кам'яний голем (рисунок 3.26), який має достатню шкоду, але він дуже повільний і з ним не має виникати проблем, грошова нагорода передбачена у випадку перемоги.

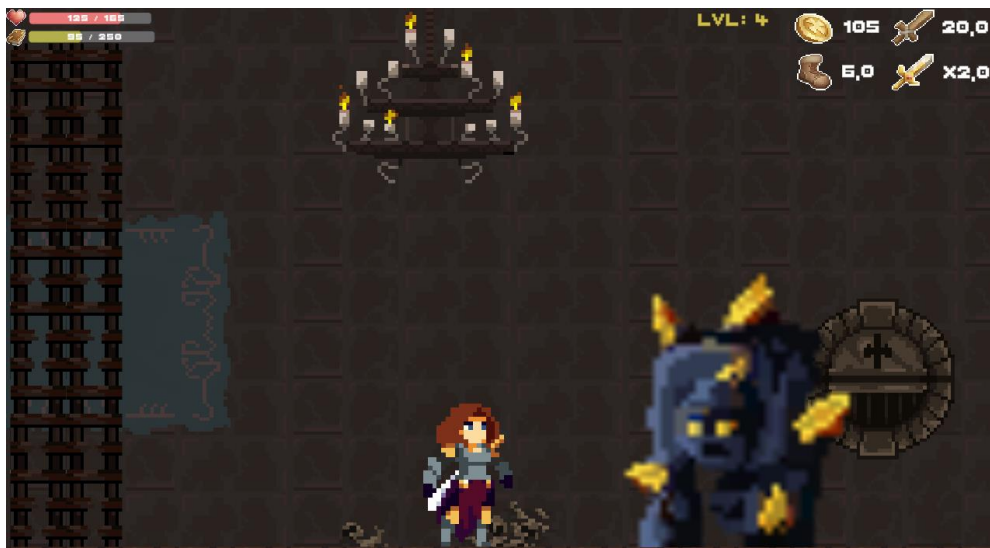


Рисунок 3.26 – Ворог першої лінії перешкод

Другий рівень перешкод майже ідентичний до першого, єдина відмінність полягає у пришвидшенні магічного заряду, та черговий ворог – Ghost Warrior (рисунок 3.27), який є більш небезпечним за попереднього. Бажано мати еліксири, що дають посилення шкоди нашому гравцю.



Рисунок 3.27 – Ghost Warrior атакує гравця мечом

Третій рівень перешкод є останнім у даній вежі, та немає прихованих пасток. Його особливість у надзвичайно спритних ворогах – Yellow/Blue Soul (рисунок 3.28, рисунок 3.29). Якщо протягом гри гравець не прокачував гілку швидкості, то втекти від них буде складною задачею. Після перемоги над ворогами – потрібно піднятися на останній поверх вежі та з діалогового вікна отримати інформацію, що бажано відвідати магазин неподалік.

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		59

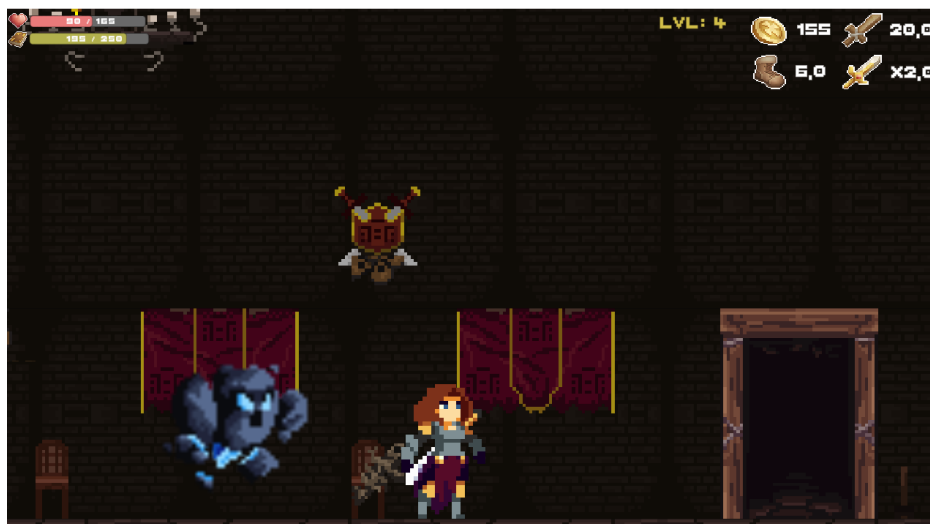


Рисунок 3.28 – Битва з Blue Soul



Рисунок 3.29 – Битва з Yellow Soul

В магазині є можливість придбати особливий предмет, який посилює силу стрибка нашого героя (занадто висока перешкода не дає можливості рухатись далі по рівню, тому потрібно отримати цю річ і застосувати). Далі за сюжетом іде зустріч з компаньйоном Charlie, який повідомляє, що гравець знаходиться у кроці від фіналу.

Користувач бачить перед собою жертвне святилище (рисунок 3.30), у якому має залишити здобуті геми, в результаті активації відкриється портал, який веде до пекла – де і відбудеться зустріч з головним антагоністом гри. Фрагмент битви з босом починається з зустрічі з безліччю скелетів, яких бажано знищити перед активацією головного боса – демона (рисунок 3.31),

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		60

який не атакує гравця звичайними атаками, а має можливість пускати магичні заряди та телепортуватись у різних напрямках при отриманні шкоди.



Рисунок 3.30 – Повідомлення про можливість активувати портал на кнопку V

Результат бою з демоном важливий: у разі перемоги над ним здійснюється перехід у фінальну локацію (рисунок 3.32), де присутній Charlie, різні жителі невідомого поселення, яке було врятовано від демона та відповідно завершується гра у разі заходу в будинок і активації панелі завершення гри (рисунок 3.33).

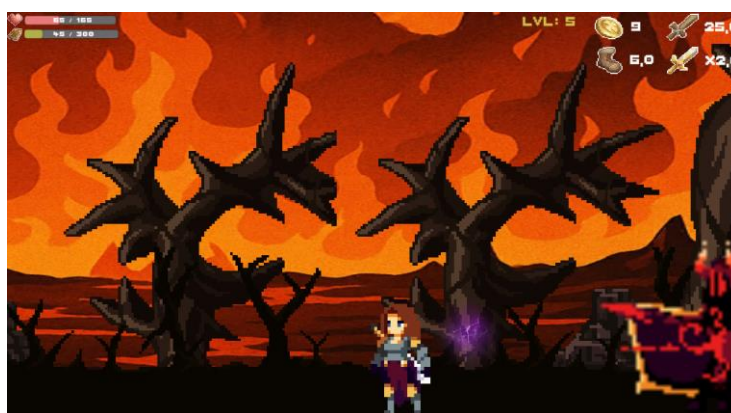


Рисунок 3.31 – Демон запускає магичний імпульс у напрямку героїні



Рисунок 3.32 – NPC у фінальній сцені

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		61



Рисунок 3.33 – Меню перемоги (вихід у головне меню)

Варто згадати про сценарій поразки у битві з демоном: програється невелике відео (рисунки 3.34 – 3.36), у якому визначено майбутнє світу, та в самому кінці гравець опиняється у другій завершальній локації, що є вічним сном. Демони атакують гравця допоки його не буде вбито, а у разі вбивства – панель Game Over (рисунок 3.37) і перехід в головне меню.

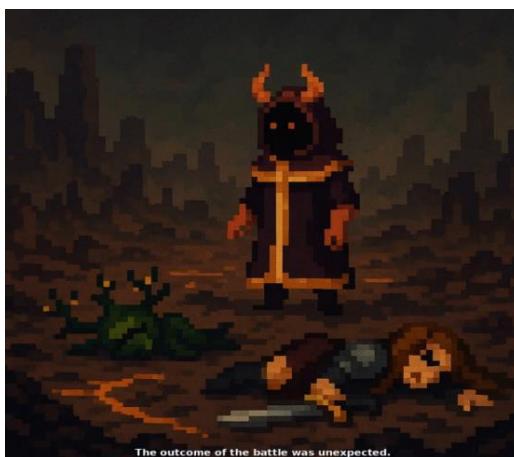


Рисунок 3.34 – Перший кадр відео



Рисунок 3.35 – Другий кадр відео



Рисунок 3.36 – Третій кадр відео



Рисунок 3.37 – Меню Game Over

3.3. Тестування роботи та демонстрація результатів

У даному підрозділі наводяться результати проведеного комплексного тестування ключових функцій у всіх ігрових сценах, де реалізовано активну взаємодію користувача [23]. Таблиця 3.3 містить опис тестових сценаріїв, вхідні умови, очікувану поведінку системи та фактичний результат.

Таблиця 3.3

Результати функціонального тестування гри

Сцена	Тестовий сценарій	Вхідні параметри	Очікуваний результат	Наявний результат / Дефект
MainMenu	Відкриття панелі	Натискається кнопка Options	Активація панелі, закриття	Працює стабільно. Конфліктів

	Options через кнопку		попередньої панелі	між перемиканням панелей немає.
MainMenu	Вибір роздільної здатності через DropDown	Змінюємо значення з 1920 х 1080 на 1280 х 720	Роздільна здатність змінюється, елементи Canvas не їдуть	Працює стабільно
MainMenu	Перехід до гри через кнопку “New Game”	Натискається кнопка New Game	Завантажується сцена Level1_Forest з заданими характеристиками HUD	Працює стабільно
Level1_Forest	Оновлення HUD при дії	Отримання шкоди від Bush Monster, оновлення HP	Динамічне оновлення кількості HP	Оновлюється стабільно
Level1_Forest	Дроп гема з вбитого ворога	Перемога над големом, підбираємо предмет	Анімація смерті голема, дроп предмета	Працює стабільно
Level1_Forest	Активація чекпоінта	Підходимо до статуї та активуємо діалог	При наближенні до статуї активація діалогової панелі	Працює стабільно
Level2_Cave	Активація кажана	Підходимо до кажана, він летить нас атакувати	Кажан переходить зі стану Idle > Fly > Attack	Працює стабільно

Level2_Cave	Активация бафу movespeed/attack	Активуємо магазин, купляємо баф, відкриваємо інвентар і тестуємо	Гравець має баф у вигляді movespeed/attack	Працює стабільно
Level2_Cave	Поява порталу і дроп гема після смерті боса	Вбиваємо боса і очікуємо на спавн портала і гема	Програється анімація смерті, з'являється гем та генерується портал	Стабільно працює.
Level3_DarkForest	Активация скелета	Гравець заходить у зону тригера і активує Attack у скелета	При наближенні до скелета змінюється стан Idle > Walk > Attack	Працює стабільно
Level3_DarkForest	Перевірка роботи сейв системи	Активация панелі паузи, зберегти в слот 1	Завантаження слоту save 1, збережено усі дані і стан поточної сцени	Працює стабільно, без конфліктів
Level4_DarkForest	Активация портала через передачу гемів	Підходимо до жертвової святині та нажимаємо V (в інвентарі маємо 4 гема)	Звук та активация порталу, можливість переходу до файт-сцени	Працює стабільно і активується за умови наявності усіх гемів
Level4_DarkParadise	Результат битви з босом	У разі смерті – завантаження	Герой вмирає – його переносить в LoseScene,	Переходи працюють стабільно,

		LoseScene / У разі перемоги - WinScene	герой вбиває демона – його переносить у WinScene	логіка робоча
LoseScene	Завантаження відео	Перемагає мо останнього боса або помираємо від нього	Перехід на сцену автоматично програє відео відповідно роздільної здатності	Працює стабільно
WinScene	Активація панелі завершення гри	Заходимо в будинок і стаємо в тригер- зону	Мелодія (оплески) та виведення панелі	Працює стабільно

Висновки до третього розділу

У даному розділі було наведено перелік мінімальних та рекомендованих вимог до системи для стабільного запуску гри “Final Dawn”, а також описано середовище розробки та умови експлуатації. Гру створено з урахуванням кросплатформеності та адаптації до широкого спектра обладнання, включаючи машини із середнім рівнем продуктивності.

Проведено візуальний аналіз інтерфейсу користувача на основі показу скріншотів з різних сцен гри. Описано основні UI-компоненти (головне меню, налаштування, HUD). Продемонстровано зручність доступу до функціоналу гри. Проведено аналіз музичного супроводу гри.

Проведено комплексне тестування кожної сцени гри шляхом побудови таблиці сценаріїв. Розгляд кожного кейсу включав вхідні дані, очікуваний результат та фактичний. Виявлені проблеми було усунено. Оцінено стабільність функціонування гри на різних етапах, також зазначена коректна робота механіки збереження, бою, бафів, діалогів та переходів між сценами, елементами меню.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було реалізовано повнофункціональний прототип 2D-гри жанру action-adventure з елементами dark fantasy під назвою *Final Dawn*. Гра поєднує дослідження рівнів, бій із ворогами, взаємодію з неігровими персонажами (NPC), а також поступовий розвиток героя. Розроблений продукт демонструє цілісність ігрового процесу та функціональність основних механік, що відповідає поставленим цілям роботи.

На основі аналізу сучасних підходів до створення 2D-ігор було обґрунтовано вибір рушія Unity як основного інструменту розробки. Вибір мови програмування C# та модульної архітектури дозволив ефективно організувати внутрішню структуру проєкту й забезпечити його масштабованість. У рамках роботи було створено концепцію гри, розроблено концепт-документ, сформовано ресурсну базу та реалізовано ключові компоненти програмного забезпечення, зокрема модулі керування персонажем, інвентар, систему збереження, магазин, HUD, бойову систему та діалогову взаємодію з NPC.

Особливу увагу приділено проектуванню програмної архітектури. Побудовано UML-діаграми, які ілюструють взаємозв'язки між основними компонентами гри. Завдяки поетапній реалізації та модульному підходу вдалося досягти високого рівня надійності та стабільності функціонування. Графічне оформлення інтерфейсу користувача та ігрових сцен було виконано у відповідності до загального стилю гри.

Після завершення розробки було проведено комплексне тестування реалізованого функціоналу в усіх задіяних сценах. Було перевірено такі сценарії, як рух гравця, атака, отримання шкоди, збирання ресурсів, збереження прогресу, використання предметів, перемога над фінальним босом та завершення гри. Результати тестування підтвердили правильність реалізації ігрової логіки та відповідність очікуваній поведінці системи.

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		67

Таким чином, поставлена мета кваліфікаційної роботи була повністю досягнута. Розроблений програмний продукт має практичну цінність як приклад повноцінного ігрового застосунку з інтегрованими базовими механіками. Перспективи подальшого вдосконалення проєкту включають розширення геймплейних можливостей та графічну оптимізацію.

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		68

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Newzoo. Newzoo tempers global game market forecast [Електронний ресурс]. – 2024. – Режим доступу: <https://www.reuters.com/technology/newzoo-tempers-global-game-market-forecast-amid-sluggish-console-sales-2024-08-13/>
2. Valve Corporation. *Steam — Indie category* [Електронний ресурс]. – 2025. – Режим доступу: <https://store.steampowered.com/search/?genre=Indie>
3. Glitchwave. Popular dark fantasy games [Електронний ресурс]. – 2023. – Режим доступу: <https://glitchwave.com/charts/popular/game/2023/g:dark-fantasy/>
4. dev.ua. Ukrainian game entered the top 20 in sales in Europe in 2024 [Електронний ресурс]. – 2024. – Режим доступу: <https://dev.ua/en/news/ukrainska-hra-v-top-20-za-prodazhamy-v-ievropi-1737028926>
5. Марчук Г., Коротун О., Левківський В., Українець М. Дослідження методів штучного інтелекту для створення інтелектуальних ігрових агентів [Електронний ресурс]. – 2024. – Технічні науки та технології, № 3 (37), с. 122-131. – Режим доступу: [https://doi.org/10.25140/2411-5363-2024-3\(37\)-122-131](https://doi.org/10.25140/2411-5363-2024-3(37)-122-131)
6. Team Cherry. *Hollow Knight* [Електронний ресурс]. – 2017. – Режим доступу: https://store.steampowered.com/app/367520/Hollow_Knight/
7. Motion Twin. *Dead Cells* [Електронний ресурс]. – 2018. – Режим доступу: https://store.steampowered.com/app/588650/Dead_Cells/
8. Ska Studios. *Salt and Sanctuary* [Електронний ресурс]. – 2016. – Режим доступу: https://store.steampowered.com/app/283640/Salt_and_Sanctuary/
9. Ubisoft. *Prince of Persia: The Lost Crown* [Електронний ресурс]. – 2024. – Режим доступу: https://store.steampowered.com/app/2751000/Prince_of_Persia_The_Lost_Crown/
10. Житомирська політехніка. Курс «Основи геймдизайну» [Електронний ресурс]. – 2025. – Режим доступу: <https://learn.ztu.edu.ua/course/view.php?id=3810>

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		69

11. Unity Technologies. Unity Asset Store [Електронний ресурс]. – 2025.
– Режим доступу: <https://assetstore.unity.com/publishers/40094>
12. Wisner, J. Unity2D Tilemapping Tutorial [Електронний ресурс]. – 2017.
– Режим доступу: https://madwomb.com/tutorials/GameDesign_Unity2DTilemap.html
13. Unity Asset Store. Warrior Free Asset [Електронний ресурс]. – 2021.
– Режим доступу: <https://assetstore.unity.com/packages/2d/characters/warrior-free-asset-195707>
14. Unity Asset Store. Pixel Art Icon Pack – RPG [Електронний ресурс]. – 2021.
– Режим доступу: <https://assetstore.unity.com/packages/2d/gui/icons/pixel-art-icon-pack-rpg-158343>
15. Creately. UML Diagram Types Guide: Learn About All Types of UML Diagrams with Examples [Електронний ресурс]. – 2025. – Режим доступу: <https://creately.com/blog/diagrams/uml-diagram-types-examples/>
16. Marsh P. Animator Controller as a Finite State Machine in Unity [Електронний ресурс]. – 2022. – Режим доступу: <https://pauliom.medium.com/animator-controller-as-a-finite-state-machine-in-unity-9e1f06742c4b>
17. Unity Technologies. Physics2D.OverlapCircle documentation [Електронний ресурс]. – 2025. – Режим доступу: <https://docs.unity3d.com/ScriptReference/Physics2D.OverlapCircle.html>
18. Левківський В., Марчук Г., Москалик О. Модель ігрового штучного інтелекту для некерованого гравцем персонажу [Електронний ресурс]. – 2024. – Технічні науки та технології, № 4 (38), с. 163–171. – Режим доступу: [https://doi.org/10.25140/2411-5363-2024-4\(38\)-163-171](https://doi.org/10.25140/2411-5363-2024-4(38)-163-171)
19. Unity Technologies. ScriptableObject reference [Електронний ресурс]. – 2025. – Режим доступу: <https://docs.unity3d.com/ScriptReference/ScriptableObject.html>

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		70

20. Unity Technologies. Saving and loading data: JSON and Binary Serialization [Електронний ресурс]. – 2019. – Режим доступу: <https://learn.unity.com/tutorial/persistence-saving-and-loading-data>

21. Unity Technologies. System requirements for Unity games and builds [Електронний ресурс]. – 2025. – Режим доступу: <https://docs.unity3d.com/Manual/system-requirements.html>

22. Pixune Studios. 5 Golden Rules of Game HUD Design – Show, Don't Tell! [Електронний ресурс]. – 2025. – Режим доступу: <https://pixune.com/blog/game-hud-design/>

23. TestDevLab. The Basics of Game Testing [Електронний ресурс]. – 2025. – Режим доступу: <https://www.testdevlab.com/blog/basics-of-game-testing>

ДОДАТКИ

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						72
Змн.	Арк.	№ докум.	Підпис	Дата		

Керування гравцем та взаємодія з HUD

PlayerMovement.cs

```

using UnityEngine;
public class PlayerMovement : MonoBehaviour
{
    private float jumpForce => PlayerStats.Instance != null ? PlayerStats.Instance.GetJumpForce() : 20f;

    public Transform groundCheck;
    public float groundCheckRadius = 0.1f;
    public LayerMask groundLayer;

    private Rigidbody2D rb;
    private Animator anim;
    private PlayerHealth health;

    private float horizontalInput;
    private bool isGrounded;
    private bool facingRight = true;

    private void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponentInChildren<Animator>();
        health = GetComponent<PlayerHealth>();
    }

    private void Update()
    {
        if (!(health != null) || !health.IsDead)
        {
            horizontalInput = Input.GetAxisRaw("Horizontal");
            anim.SetFloat("Speed", Mathf.Abs(horizontalInput));

            if (horizontalInput < 0f && facingRight)
                Flip();
            else if (horizontalInput > 0f && !facingRight)
                Flip();

            isGrounded = Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius, groundLayer);
            anim.SetBool("isJumping", !isGrounded);

            if (Input.GetButtonDown("Jump") && isGrounded)
            {
                rb.velocity = new Vector2(rb.velocity.x, jumpForce);
            }
        }
    }

    private void FixedUpdate()
    {
        if (!(health != null) || !health.IsDead)
        {
            float totalSpeed = PlayerStats.Instance.BaseMoveSpeed + PlayerStats.Instance.MoveSpeedBonus;
            rb.velocity = new Vector2(horizontalInput * totalSpeed, rb.velocity.y);
        }
    }

    private void Flip()
    {

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						73
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        facingRight = !facingRight;
        Vector3 localScale = transform.localScale;
        localScale.x *= -1f;
        transform.localScale = localScale;
    }

    private void OnDrawGizmos()
    {
        if (groundCheck != null)
        {
            Gizmos.color = Color.red;
            Gizmos.DrawWireSphere(groundCheck.position, groundCheckRadius);
        }
    }
}

```

PlayerHealth.cs

```

using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using UnityEngine.Events;

public class PlayerHealth : MonoBehaviour
{
    [Header("Sprites")]
    public Sprite iconSprite;
    public Sprite coinSprite;

    [Header("HUD")]
    public Slider healthSlider;
    public Slider xpSlider;

    private Animator anim;
    private Rigidbody2D rb;

    public bool IsDead { get; private set; }

    private HUDController hud;

    private void Awake()
    {
        anim = GetComponentInChildren<Animator>();
        rb = GetComponent<Rigidbody2D>();

        hud = FindObjectOfType<HUDController>();
        if (hud != null)
        {
            hud.SetPlayer(this);
            healthSlider = hud.healthSlider;
            xpSlider = hud.xpSlider;
        }

        UpdateHUD();
    }

    public void TakeDamage(float damage, bool fromFinalBoss = false)
    {
        if (!IsDead && PlayerStats.Instance != null)
        {
            PlayerStats.Instance.CurrentHealth -= damage;
        }
    }
}

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						74
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        PlayerStats.Instance.CurrentHealth = Mathf.Clamp(PlayerStats.Instance.CurrentHealth, 0f,
PlayerStats.Instance.MaxHealth);

        UpdateHUD();
        anim.SetTrigger("hurt");

        if (PlayerStats.Instance.CurrentHealth <= 0f)
        {
            Die();

            GameOverHandler handler = FindObjectOfType<GameOverHandler>();
            if (handler != null)
            {
                handler.killedByFinalBoss = fromFinalBoss;
                handler.ShowGameOver();
            }
        }
    }

    public void Heal(float amount)
    {
        if (!IsDead && PlayerStats.Instance != null)
        {
            PlayerStats.Instance.CurrentHealth += amount;
            PlayerStats.Instance.CurrentHealth = Mathf.Clamp(PlayerStats.Instance.CurrentHealth, 0f,
PlayerStats.Instance.MaxHealth);
            UpdateHUD();
        }
    }

    public void AddXP(float xp)
    {
        if (PlayerStats.Instance == null) return;

        PlayerStats.Instance.CurrentXP += xp;

        while (PlayerStats.Instance.CurrentXP >= PlayerStats.Instance.XpToNextLevel)
        {
            PlayerStats.Instance.CurrentXP -= PlayerStats.Instance.XpToNextLevel;
            PlayerStats.Instance.XpToNextLevel += 50f;
            PlayerStats.Instance.Level++;

            if (hud != null)
                StartCoroutine(ShowLevelUpPanelAfterDelay(1.5f));
        }

        UpdateHUD();
    }

    private IEnumerator ShowLevelUpPanelAfterDelay(float delay)
    {
        yield return new WaitForSecondsRealtime(delay);
        hud.ShowLevelUpPanel();
    }

    public void AddMoney(int amount)
    {
        if (PlayerStats.Instance == null) return;
        PlayerStats.Instance.Money += amount;
        UpdateHUD();
    }

```

```

private void UpdateHUD()
{
    if (hud != null)
        hud.UpdateHUD();
}

private void Die()
{
    IsDead = true;
    anim.SetTrigger("die");
    rb.velocity = Vector2.zero;
    rb.bodyType = RigidbodyType2D.Static;

    MonoBehaviour[] components = GetComponents<MonoBehaviour>();
    foreach (MonoBehaviour monoBehaviour in components)
    {
        if (monoBehaviour != this)
            monoBehaviour.enabled = false;
    }
}
}

```

PlayerAttack.cs

```

using UnityEngine;
public class PlayerAttack : MonoBehaviour
{
    [Header("Attack Settings")]
    public Transform attackPoint;
    public float attackRange = 0.4f;
    public float strongAttackRange = 0.8f;

    public LayerMask enemyLayer;

    public float attackCooldown = 0.8f;
    public float strongAttackCooldown = 1.5f;

    private float nextAttackTime;
    private float nextStrongAttackTime;

    private Animator anim;
    private PlayerHealth health;

    private void Start()
    {
        anim = GetComponentInChildren<Animator>();
        health = GetComponent<PlayerHealth>();
    }

    private void Update()
    {
        if (!(health != null) || !health.IsDead)
        {
            if (Input.GetKeyDown(KeyCode.F) && Time.time >= nextAttackTime)
            {
                Attack();
                nextAttackTime = Time.time + attackCooldown;
            }

            if (Input.GetKeyDown(KeyCode.G) && Time.time >= nextStrongAttackTime)
            {
                StrongAttack();
                nextStrongAttackTime = Time.time + strongAttackCooldown;
            }
        }
    }
}

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						76
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
}

private void Attack()
{
    if (anim != null)
        anim.SetTrigger("Attack");

    float finalDamage = PlayerStats.Instance.BaseDamage + PlayerStats.Instance.DamageBonus;
    DealDamage(attackPoint.position, attackRange, finalDamage);
}

private void StrongAttack()
{
    if (anim != null)
        anim.SetTrigger("StrongAttack");

    float baseDamage = PlayerStats.Instance.BaseDamage + PlayerStats.Instance.DamageBonus;
    float strong = baseDamage * PlayerStats.Instance.StrongAttackMultiplier;

    DealDamage(attackPoint.position, strongAttackRange, strong);
}

private void DealDamage(Vector3 position, float range, float dmg)
{
    Collider2D[] array = Physics2D.OverlapCircleAll(position, range, enemyLayer);
    foreach (Collider2D collider2D in array)
    {
        EnemyHealth eh = collider2D.GetComponentInParent<EnemyHealth>();
        if (eh != null && !eh.IsDead)
        {
            eh.TakeDamage(dmg);
            continue;
        }

        BossHealth bh = collider2D.GetComponentInParent<BossHealth>();
        if (bh != null && !bh.IsDead)
        {
            bh.TakeDamage(dmg);
        }
    }
}

private void OnDrawGizmosSelected()
{
    if (attackPoint != null)
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(attackPoint.position, attackRange);
        Gizmos.color = Color.yellow;
        Gizmos.DrawWireSphere(attackPoint.position, strongAttackRange);
    }
}
}

```

PlayerStats.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerStats : MonoBehaviour
{
    public static PlayerStats Instance;

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						77
Змн.	Арк.	№ докум.	Підпис	Дата		

```

[Header("Health")]
public float MaxHealth = 150f;
public float CurrentHealth = 150f;
private float permanentMaxHealthBonus = 0f;

[Header("Experience")]
public float CurrentXP = 0f;
public float XpToNextLevel = 100f;
public int Level = 1;

[Header("Money")]
public int Money = 0;

[Header("Damage")]
public float BaseDamage = 10f;
public float DamageBonus = 0f;
private float permanentDamageBonus = 0f;
private float tempDamageBonus = 0f;

[Header("Speed")]
public float BaseMoveSpeed = 5f;
public float MoveSpeedBonus = 0f;
private float permanentMoveSpeedBonus = 0f;
private float tempSpeedBonus = 0f;

[Header("Strong Attack")]
public float StrongAttackMultiplier = 1.5f;
private float permanentStrongAttackBonus = 0f;
private float tempStrongAttackBonus = 0f;

[Header("Jump")]
public float BaseJumpForce = 20f;
public float JumpBonus = 0f;
private float tempJumpBonus = 0f;

public float GetJumpForce() => BaseJumpForce + JumpBonus;

[System.Serializable]
public class ActiveBuff
{
    public BuffType type;
    public float value;
    public float duration;
    public float timeRemaining;

    public ActiveBuff(BuffType type, float value, float duration)
    {
        this.type = type;
        this.value = value;
        this.duration = duration;
        this.timeRemaining = duration;
    }
}

public enum BuffType { Damage, MoveSpeed, StrongAttack, JumpBoost }

public List<ActiveBuff> activeBuffs = new List<ActiveBuff>();
private Dictionary<ActiveBuff, Coroutine> activeCoroutines = new();

private void Awake()
{
    if (Instance != null && Instance != this)

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						78
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    {
        Destroy(gameObject);
        return;
    }

    Instance = this;
    DontDestroyOnLoad(gameObject);
}

public Coroutine ApplyBuffCoroutine(ActiveBuff buff)
{
    Coroutine coroutine = StartCoroutine(BuffCoroutine(buff));
    activeCoroutines[buff] = coroutine;
    return coroutine;
}

private IEnumerator BuffCoroutine(ActiveBuff buff)
{
    ApplyBuff(buff, true);

    while (buff.timeRemaining > 0f)
    {
        yield return new WaitForSeconds(1f);
        buff.timeRemaining -= 1f;
    }

    ApplyBuff(buff, false);
    activeBuffs.Remove(buff);
    activeCoroutines.Remove(buff);
}

private void ApplyBuff(ActiveBuff buff, bool apply)
{
    float value = apply ? buff.value : -buff.value;

    switch (buff.type)
    {
        case BuffType.Damage:
            tempDamageBonus += value;
            DamageBonus = permanentDamageBonus + tempDamageBonus;
            break;

        case BuffType.MoveSpeed:
            tempSpeedBonus += value;
            MoveSpeedBonus = permanentMoveSpeedBonus + tempSpeedBonus;
            break;

        case BuffType.StrongAttack:
            tempStrongAttackBonus += value;
            StrongAttackMultiplier = 1.5f + permanentStrongAttackBonus + tempStrongAttackBonus;
            break;

        case BuffType.JumpBoost:
            tempJumpBonus += value;
            JumpBonus = tempJumpBonus;
            break;
    }
}

public void ApplyPermanentDamageBonus(float amount)
{
    permanentDamageBonus += amount;
    DamageBonus = permanentDamageBonus + tempDamageBonus;
}

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						79
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }

    public void ApplyPermanentSpeedBonus(float amount)
    {
        permanentMoveSpeedBonus += amount;
        MoveSpeedBonus = permanentMoveSpeedBonus + tempSpeedBonus;
    }

    public void ApplyPermanentStrongAttackBonus(float amount)
    {
        permanentStrongAttackBonus += amount;
        StrongAttackMultiplier = 1.5f + permanentStrongAttackBonus + tempStrongAttackBonus;
    }

    public void ApplyPermanentHealthBonus(float amount)
    {
        permanentMaxHealthBonus += amount;
        MaxHealth = 150f + permanentMaxHealthBonus;
        CurrentHealth += amount;
    }

    public void LoadPlayer(SaveData data)
    {
        CurrentHealth = data.playerHP;
        CurrentXP = data.playerXP;
        Level = data.playerLevel;
        Money = data.money;

        DamageBonus = permanentDamageBonus;
        MoveSpeedBonus = permanentMoveSpeedBonus;
        StrongAttackMultiplier = 1.5f + permanentStrongAttackBonus;
        MaxHealth = 150f + permanentMaxHealthBonus;

        // Скидаємо тимчасові значення
        tempDamageBonus = 0f;
        tempSpeedBonus = 0f;
        tempStrongAttackBonus = 0f;
        tempJumpBonus = 0f;
        JumpBonus = 0f;

        foreach (var coroutine in activeCoroutines.Values)
            StopCoroutine(coroutine);

        activeBufs.Clear();
        activeCoroutines.Clear();

        if (data.buffStates != null)
        {
            foreach (var buffData in data.buffStates)
            {
                var buff = new ActiveBuff(buffData.type, buffData.value, buffData.duration);
                buff.timeRemaining = buffData.timeRemaining;
                activeBufs.Add(buff);
                ApplyBuffCoroutine(buff);
            }
        }
    }

    public void ResetStats()
    {
        tempDamageBonus = 0f;
        tempSpeedBonus = 0f;
        tempStrongAttackBonus = 0f;
    }

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						80
Змн.	Арк.	№ докум.	Підпис	Дата		

```

tempJumpBonus = 0f;
JumpBonus = 0f;

MaxHealth = 150f + permanentMaxHealthBonus;
CurrentHealth = MaxHealth;
CurrentXP = 0f;
XpToNextLevel = 100f;
Level = 1;
Money = 0;

DamageBonus = permanentDamageBonus;
MoveSpeedBonus = permanentMoveSpeedBonus;
StrongAttackMultiplier = 1.5f + permanentStrongAttackBonus;

foreach (var coroutine in activeCoroutines.Values)
    StopCoroutine(coroutine);

activeBufs.Clear();
activeCoroutines.Clear();

InventoryManager.Instance?.ClearInventory();
}
}

```

HUDController.cs

```

using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.SceneManagement;

public class HUDController : MonoBehaviour
{
    [Header("Sliders")]
    public Slider healthSlider;
    public Slider xpSlider;

    [Header("Text Fields")]
    public TMP_Text moneyText;
    public TMP_Text hpText;
    public TMP_Text xpText;
    public TMP_Text levelText;

    [Header("Images")]
    public Image playerIcon;
    public Image coinIcon;

    [Header("Level Up")]
    public LevelUpPanel levelUpPanel;

    [Header("Stat Texts")]
    public TMP_Text attackText;
    public TMP_Text speedText;
    public TMP_Text multiplierText;

    [Header("Buff Display")]
    public Transform buffTextParent;
    public GameObject buffTextPrefab;

    private Dictionary<PlayerStats.ActiveBuff, TMP_Text> activeBuffLabels = new
    Dictionary<PlayerStats.ActiveBuff, TMP_Text>();
}

```

```

private void OnEnable()
{
    SceneManager.sceneLoaded += OnSceneLoaded;
}

private void OnDisable()
{
    SceneManager.sceneLoaded -= OnSceneLoaded;
}

private void Start()
{
    RestoreBuffsFromPlayerStats();
}

private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
{
    RestoreBuffsFromPlayerStats();
}

private void Update()
{
    UpdateHUD();
}

public void SetPlayer(PlayerHealth playerRef)
{
    if (playerRef == null) return;

    if (playerIcon != null && playerRef.iconSprite != null)
        playerIcon.sprite = playerRef.iconSprite;

    if (coinIcon != null && playerRef.coinSprite != null)
        coinIcon.sprite = playerRef.coinSprite;

    UpdateHUD();
}

public void TrackBuff(PlayerStats.ActiveBuff buff)
{
    if (buffTextPrefab == null || buffTextParent == null || activeBuffLabels.ContainsKey(buff))
        return;

    GameObject go = Instantiate(buffTextPrefab, buffTextParent);
    TMP_Text txt = go.GetComponent<TMP_Text>();
    if (txt != null)
    {
        activeBuffLabels[buff] = txt;
        StartCoroutine(UpdateBuffLabel(buff, txt));
    }
}

private IEnumerator UpdateBuffLabel(PlayerStats.ActiveBuff buff, TMP_Text txt)
{
    while (buff.timeRemaining > 0f)
    {
        txt.text = $"{buff.type} {buff.value} ({buff.timeRemaining:F0}s)";
        yield return new WaitForSeconds(1f);
    }

    Destroy(txt.gameObject);
    activeBuffLabels.Remove(buff);
}

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						82
Змн.	Арк.	№ докум.	Підпис	Дата		

```

public void RestoreBuffsFromPlayerStats()
{
    var stats = PlayerStats.Instance;
    if (stats == null || buffTextPrefab == null || buffTextParent == null) return;

    // Очищаємо всі попередні баф-надписи
    foreach (var label in activeBuffLabels.Values)
    {
        if (label != null)
            Destroy(label.gameObject);
    }
    activeBuffLabels.Clear();

    // Створюємо нові
    foreach (var buff in stats.activeBuffs)
    {
        TrackBuff(buff);
    }
}

public void UpdateHUD()
{
    var stats = PlayerStats.Instance;
    if (stats == null) return;

    if (healthSlider != null)
        healthSlider.value = stats.CurrentHealth / stats.MaxHealth;

    if (xpSlider != null)
        xpSlider.value = stats.CurrentXP / stats.XpToNextLevel;

    if (hpText != null)
        hpText.text = Mathf.RoundToInt(stats.CurrentHealth) + " / " + Mathf.RoundToInt(stats.MaxHealth);

    if (xpText != null)
        xpText.text = Mathf.RoundToInt(stats.CurrentXP) + " / " + Mathf.RoundToInt(stats.XpToNextLevel);

    if (moneyText != null)
        moneyText.text = stats.Money.ToString();

    if (levelText != null)
        levelText.text = "LVL: " + stats.Level;

    if (attackText != null)
    {
        float atk = stats.BaseDamage + stats.DamageBonus;
        attackText.text = atk.ToString("F1");
    }

    if (speedText != null)
    {
        float spd = stats.BaseMoveSpeed + stats.MoveSpeedBonus;
        speedText.text = spd.ToString("F1");
    }

    if (multiplierText != null)
    {
        multiplierText.text = "x" + stats.StrongAttackMultiplier.ToString("F1");
    }
}

public void ShowLevelUpPanel()

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						83
Змн.	Арк.	№ докум.	Підпис	Дата		

```
{
    if (levelUpPanel != null)
    {
        levelUpPanel.Show();
    }
}
```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						84
Змн.	Арк.	№ докум.	Підпис	Дата		

Керування ворогами та їх типи

Enemy.cs

```

using System.Collections;
using UnityEngine;

public class Enemy : MonoBehaviour
{
    [Header("Attack Settings")]
    public Transform attackPoint;
    public Transform spriteTransform;

    [Header("Block Detection")]
    public Transform bodyBlockPoint; // точка перед тілом ворога
    public float blockRange = 0.4f; // радіус для детекції гравця

    public float moveSpeed = 2f;
    public float attackRange = 0.5f;
    public LayerMask playerLayer;
    public float damage = 20f;
    public float attackCooldown = 1f;

    private float nextAttackTime;
    private bool isAttacking;

    private EnemyHealth health;
    private Rigidbody2D rb;
    private Animator anim;

    private void Start()
    {
        health = GetComponent<EnemyHealth>();
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponentInChildren<Animator>();
    }

    private void Update()
    {
        if (health != null && !health.IsDead)
        {
            DetectAndAttack();
        }
    }

    private void DetectAndAttack()
    {
        if (Physics2D.OverlapCircle(attackPoint.position, attackRange, playerLayer) != null &&
            Time.time >= nextAttackTime && !isAttacking)
        {
            nextAttackTime = Time.time + attackCooldown;
            StartCoroutine(Attack());
        }
    }

    private IEnumerator Attack()
    {
        isAttacking = true;
        rb.velocity = Vector2.zero;
        if (anim != null)
        {

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						85
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        anim.SetBool("isWalking", false);
        anim.SetTrigger("attack");
    }
    yield return new WaitForSeconds(attackCooldown);
    isAttacking = false;
}

public void DealDamage()
{
    if (health != null && health.IsDead) return;

    Collider2D hit = Physics2D.OverlapCircle(attackPoint.position, attackRange, playerLayer);
    if (hit != null)
    {
        PlayerHealth player = hit.GetComponent<PlayerHealth>();
        if (player != null)
        {
            player.TakeDamage(damage);
        }
    }
}

public void SetVelocity(Vector2 vel)
{
    if (rb == null)
        rb = GetComponent<Rigidbody2D>();

    Vector2 targetPos = rb.position + vel * Time.fixedDeltaTime;
    rb.MovePosition(targetPos);
}

public void SetIsWalking(bool isWalking)
{
    if (anim == null)
        anim = GetComponentInChildren<Animator>();

    anim.SetBool("isWalking", isWalking);
}

public void CallFlip()
{
    if (spriteTransform != null)
    {
        Vector3 scale = spriteTransform.localScale;
        scale.x *= -1f;
        spriteTransform.localScale = scale;
    }
}

public bool GetFacingRight()
{
    return spriteTransform != null && spriteTransform.localScale.x > 0f;
}

public bool IsPlayerInFront()
{
    return Physics2D.OverlapCircle(bodyBlockPoint.position, blockRange, playerLayer);
}

private void OnDrawGizmosSelected()
{
    if (attackPoint != null)
    {

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						86
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(attackPoint.position, attackRange);
    }

    if (bodyBlockPoint != null)
    {
        Gizmos.color = Color.cyan;
        Gizmos.DrawWireSphere(bodyBlockPoint.position, blockRange);
    }
}

```

EnemyHealth.cs

```

using System.Collections;
using UnityEngine;
using UnityEngine.Events;

public class EnemyHealth : MonoBehaviour
{
    public float maxHealth = 100f;
    private float currentHealth;

    private Animator anim;
    private Rigidbody2D rb;
    private Collider2D col;

    [Header("Діалог після смерті")]
    public string[] deathDialogueLines;

    public MonoBehaviour movementScript;

    [Header("Портал після смерті")]
    public GameObject portalPrefab;
    public Transform portalSpawnPoint;

    [Header("Дроп після смерті")]
    public GameObject gemPrefab;

    public bool IsDead { get; private set; }

    public UnityEvent onDeath = new UnityEvent();

    private void Awake()
    {
        currentHealth = maxHealth;
        anim = GetComponentInChildren<Animator>();
        rb = GetComponent<Rigidbody2D>();
        col = GetComponent<Collider2D>();

        // Перевірка чи ворог вже був мертвий
        var idComponent = GetComponent<SceneObjectID>();
        if (idComponent != null && SceneStateTracker.Instance != null &&
            SceneStateTracker.Instance.IsObjectDisabled(idComponent.uniqueID))
        {
            HandleAlreadyDead();
        }
    }

    private void HandleAlreadyDead()
    {
        IsDead = true;

        if (anim != null)

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						87
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{
    anim.ResetTrigger("hurt");
    anim.SetTrigger("die");
}

if (rb != null)
{
    rb.velocity = Vector2.zero;
    rb.bodyType = RigidbodyType2D.Kinematic;
    rb.simulated = false;
}

if (col != null)
    col.enabled = false;

if (movementScript != null)
    movementScript.enabled = false;

MonoBehaviour[] scripts = GetComponents<MonoBehaviour>();
foreach (var script in scripts)
{
    if (script != this && script.enabled)
        script.enabled = false;
}

Destroy(gameObject, 0.5f); // очищення без дропа
}

public void TakeDamage(float damage)
{
    if (IsDead) return;

    currentHealth -= damage;
    if (currentHealth < 0f) currentHealth = 0f;

    if (currentHealth > 0f && anim != null)
        anim.SetTrigger("hurt");

    if (currentHealth <= 0f)
    {
        IsDead = true;
        onDeath.Invoke();
        Die();
    }
}

public void ForceDie()
{
    if (!IsDead)
    {
        IsDead = true;
        Die();
    }
}

public void Die()
{
    if (rb != null)
        rb.velocity = Vector2.zero;

    if (col != null)
        col.enabled = false;
}

```

```

if (anim != null)
{
    anim.ResetTrigger("hurt");
    anim.SetTrigger("die");
}

Enemy component = GetComponent<Enemy>();
if (component != null)
    component.enabled = false;

var idComponent = GetComponent<SceneObjectID>();
if (idComponent != null && SceneStateTracker.Instance != null)
{
    SceneStateTracker.Instance.MarkObjectAsDisabled(idComponent.uniqueID);
}

StartCoroutine(DieAnimationRoutine());
}

private IEnumerator DieAnimationRoutine()
{
    yield return null;

    if (rb != null)
    {
        rb.velocity = Vector2.zero;
        rb.bodyType = RigidbodyType2D.Kinematic;
    }

    if (movementScript != null)
        movementScript.enabled = false;

    yield return new WaitForSeconds(1f);

    if (gemPrefab != null)
    {
        GameObject gem = Instantiate(gemPrefab, transform.position, Quaternion.identity);
        var gemId = gem.GetComponent<SceneObjectID>();
        if (gemId != null)
            gemId.uniqueID = "gem_from_" + gameObject.name + "_" + transform.position.ToString("F2");
    }

    if (portalPrefab != null)
    {
        Vector3 position = (portalSpawnPoint != null) ? portalSpawnPoint.position : transform.position;
        GameObject portal = Instantiate(portalPrefab, position, Quaternion.identity);

        var portalID = portal.GetComponent<SceneObjectID>();
        if (portalID != null)
            SceneStateTracker.Instance.MarkObjectAsActivated(portalID.uniqueID);
    }

    if (deathDialogueLines != null && deathDialogueLines.Length != 0)
    {
        DialogueUI dialogueUI = FindObjectOfType<DialogueUI>();
        if (dialogueUI != null)
        {
            dialogueUI.StartDialogue(deathDialogueLines, () =>
            {
                Destroy(gameObject);
            });
        }

        yield break;
    }
}

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						89
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
}

Destroy(gameObject);
}
}

```

FinalBossHealth.cs

```

using UnityEngine;
using System.Collections;

public class FinalBossHealth : BossHealth
{
    [Header("Teleport Logic")]
    public Transform teleportPointA;
    public Transform teleportPointB;
    public float teleportCooldown = 5f;
    public float teleportChance = 0.5f;

    [Header("Victory Dialogue")]
    public string[] victoryDialogueLines;
    public string winSceneName = "WinScene";

    private float lastTeleportTime = -10f;
    private bool atPointA = true;

    public override void TakeDamage(float damage)
    {
        if (IsDead) return;

        currentHealth -= damage;
        if (currentHealth <= 0f)
        {
            currentHealth = 0f;
            IsDead = true;
            onDeath.Invoke();
            Die();
            return;
        }

        if (Time.time >= lastTeleportTime + teleportCooldown && Random.value < teleportChance)
        {
            if (anim != null)
            {
                anim.SetBool("isTeleporting", true);
                anim.SetTrigger("teleportOut");
                lastTeleportTime = Time.time;
            }
        }
        else if (anim != null)
        {
            anim.SetTrigger("hurt");
        }
    }

    public void PerformTeleport()
    {
        if (IsDead || teleportPointA == null || teleportPointB == null || anim == null) return;

        Transform target = atPointA ? teleportPointB : teleportPointA;
        transform.position = target.position;
        atPointA = !atPointA;
    }
}

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        anim.SetTrigger("teleportIn");
        Invoke(nameof(FinishTeleport), 0.5f);
    }

    private void FinishTeleport()
    {
        if (IsDead || anim == null) return;
        anim.SetBool("isTeleporting", false);
    }

    protected override IEnumerator FinalizeDeathWithDelay()
    {
        yield return new WaitForSeconds(1.2f);

        DialogueUI dialogueUI = FindObjectOfType<DialogueUI>();
        if (dialogueUI != null && victoryDialogueLines.Length > 0)
        {
            dialogueUI.StartDialogue(victoryDialogueLines, () =>
            {
                StartCoroutine(LoadVictorySceneAfterDelay(1f));
            });
        }
        else
        {
            StartCoroutine(LoadVictorySceneAfterDelay(1f));
        }
    }

    private IEnumerator LoadVictorySceneAfterDelay(float delay)
    {
        Time.timeScale = 1f;
        yield return new WaitForSecondsRealtime(delay);

        if (SaveSystem.CurrentSlot >= 0)
            PlayerPrefs.SetInt("LoadSlot", SaveSystem.CurrentSlot);

        if (MusicManager.Instance != null)
        {
            MusicManager.Instance.StopMusic();
        }

        UnityEngine.SceneManagement.SceneManager.LoadScene(winSceneName);
    }

    public override void Die()
    {
        var idComponent = GetComponent<SceneObjectID>();
        if (idComponent != null && SceneStateTracker.Instance != null)
        {
            SceneStateTracker.Instance.MarkObjectAsDisabled(idComponent.uniqueID);
        }

        if (rb != null)
        {
            rb.velocity = Vector2.zero;
            rb.bodyType = RigidbodyType2D.Kinematic;
            rb.simulated = false;
        }

        if (col != null)
            col.enabled = false;
    }

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						91
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if (anim != null)
        {
            anim.ResetTrigger("hurt");
            anim.SetTrigger("die");
        }

        MonoBehaviour[] scripts = GetComponents<MonoBehaviour>();
        foreach (var script in scripts)
        {
            if (script != this && script.enabled)
                script.enabled = false;
        }

        StartCoroutine(FinalizeDeathWithDelay());
    }

    public override void TriggerPostDeathEvents()
    {
        StartCoroutine(FinalizeDeathWithDelay());
    }
}

```

FinalBossBehaviour.cs

```

using UnityEngine;
using System.Collections;

public class FinalBossBehaviour : BossBehaviour
{
    [Header("Magic Projectile Variants")]
    public GameObject magicProjectileA; // Снаряд для attackOne
    public GameObject magicProjectileB; // Снаряд для attackTwo
    public Transform magicSpawnPoint;

    private new void Update()
    {
        if ((bossHealth == null || bossHealth.IsDead) || !hasSpawned || isAttacking)
            return;

        FlipSprite(player.position - transform.position);

        if (Time.time >= nextAttackTime)
        {
            nextAttackTime = Time.time + attackCooldown;
            StartCoroutine(RandomAttack());
        }
    }

    private IEnumerator RandomAttack()
    {
        isAttacking = true;
        anim.SetBool("isWalking", false);
        rb.velocity = Vector2.zero;

        // Рандомно обираємо одну з двох магичних атак
        bool useFirstProjectile = Random.value > 0.5f;

        if (useFirstProjectile)
        {
            anim.SetTrigger("attackOne"); // викличе CastProjectileA()
        }
        else
        {
            anim.SetTrigger("attackTwo"); // викличе CastProjectileB()
        }
    }
}

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						92
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }

    yield return new WaitForSeconds(0.6f);
    isAttacking = false;
}

public void CastProjectileA()
{
    if (magicProjectileA != null && magicSpawnPoint != null && player != null)
    {
        GameObject p = Instantiate(magicProjectileA, magicSpawnPoint.position, Quaternion.identity);
        Vector2 dir = (player.position - magicSpawnPoint.position).normalized;
        p.GetComponent<Rigidbody2D>().velocity = dir * 7f;
    }
}

public void CastProjectileB()
{
    if (magicProjectileB != null && magicSpawnPoint != null && player != null)
    {
        GameObject p = Instantiate(magicProjectileB, magicSpawnPoint.position, Quaternion.identity);
        Vector2 dir = (player.position - magicSpawnPoint.position).normalized;
        p.GetComponent<Rigidbody2D>().velocity = dir * 7f;
    }
}

private void OnDrawGizmosSelected()
{
    if (attackPoint != null)
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(attackPoint.position, attackRange);
    }

    if (magicSpawnPoint != null)
    {
        Gizmos.color = Color.blue;
        Gizmos.DrawWireSphere(magicSpawnPoint.position, 0.25f);
    }
}

public new void DealDamage()
{
    if (bossHealth != null && bossHealth.IsDead) return;

    Collider2D hit = Physics2D.OverlapCircle(attackPoint.position, attackRange, playerLayer);
    if (hit != null)
    {
        PlayerHealth ph = hit.GetComponent<PlayerHealth>();
        if (ph != null)
        {
            float currentHP = PlayerStats.Instance.CurrentHealth;

            if (currentHP <= damage)
            {
                ph.TakeDamage(damage, true); // смерть від боса
            }
            else
            {
                ph.TakeDamage(damage, false); // просто вдарив
            }
        }
    }
}

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						93
Змн.	Арк.	№ докум.	Підпис	Дата		

}
}

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						94
Змн.	Арк.	№ докум.	Підпис	Дата		

Система інвентарю та торгівлі

InventoryManager.cs

```

using System.Collections.Generic;
using UnityEngine;
using System.Linq;
using UnityEngine.SceneManagement;

public class InventoryManager : MonoBehaviour
{
    public static InventoryManager Instance;

    [Header("Inventory UI")]
    public GameObject inventoryUI;
    public Transform itemSlotParent;
    public GameObject itemSlotPrefab;
    public KeyCode toggleKey = KeyCode.F1;

    private List<ItemData> inventoryItems = new List<ItemData>();

    private void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
            inventoryUI.SetActive(false);
            SceneManager.sceneLoaded += OnSceneLoaded;
        }
        else
        {
            Destroy(gameObject);
        }
    }

    private void Update()
    {
        if (Input.GetKeyDown(toggleKey))
            inventoryUI.SetActive(!inventoryUI.activeSelf);
    }

    public void AddItem(ItemData item)
    {
        if (item == null) return;

        inventoryItems.Add(item);
        RefreshUI();
    }

    public void UseItem(ItemData item)
    {
        switch (item.itemType)
        {
            case ItemData.ItemType.Heal:
                var playerHealth = FindObjectOfType<PlayerHealth>();
                if (playerHealth != null)
                    playerHealth.Heal(item.healAmount);
                break;

            case ItemData.ItemType.Buff:

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						95
Змн.	Арк.	№ докум.	Підпис	Дата		

```

var stats = FindObjectOfType<PlayerStats>();
PlayerStats.ActiveBuff appliedBuff = null;

if (stats != null)
{
    appliedBuff = new PlayerStats.ActiveBuff(
        (PlayerStats.BuffType)item.buffType,
        item.buffValue,
        item.duration
    );

    stats.activeBuffs.Add(appliedBuff);
    stats.ApplyBuffCoroutine(appliedBuff);
}

var hud = FindObjectOfType<HUDController>();
if (hud != null && appliedBuff != null)
{
    hud.TrackBuff(appliedBuff);
}

break;

case ItemData.ItemType.Gem:
    return;
}

inventoryItems.Remove(item);
RefreshUI();
}

public bool RemoveItemByID(string itemID)
{
    var item = inventoryItems.Find(i => i.itemID == itemID);
    if (item != null)
    {
        inventoryItems.Remove(item);
        RefreshUI();
        return true;
    }
    return false;
}

public void ClearInventory()
{
    inventoryItems.Clear();
    RefreshUI();
}

private void RefreshUI()
{
    foreach (Transform child in itemSlotParent)
    {
        Destroy(child.gameObject);
    }

    foreach (var item in inventoryItems)
    {
        var go = Instantiate(itemSlotPrefab, itemSlotParent);
        var slot = go.GetComponent<InventoryItemUI>();
        if (slot != null)
            slot.Setup(item, this);
    }
}

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						96
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }

    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        var connector = FindObjectOfType<InventoryUIConnector>();
        if (connector == null) return;

        inventoryUI = connector.inventoryUI;
        itemSlotParent = connector.itemSlotParent;

        RefreshUI();
    }

    public ItemData GetItemByID(string id)
    {
        ItemData[] allItems = Resources.LoadAll<ItemData>("Items");

        foreach (var item in allItems)
        {
            if (item != null && item.itemID == id)
                return item;
        }

        Debug.LogWarning($"ItemData з ID '{id}' не знайдено в Resources/Items/");
        return null;
    }

    public int GetItemCount(ItemData item)
    {
        int count = 0;
        foreach (var i in inventoryItems)
        {
            if (i == item) count++;
        }
        return count;
    }

    public void RemoveItem(ItemData item, int amount)
    {
        int removed = 0;
        for (int i = inventoryItems.Count - 1; i >= 0 && removed < amount; i--)
        {
            if (inventoryItems[i] == item)
            {
                inventoryItems.RemoveAt(i);
                removed++;
            }
        }
        RefreshUI();
    }

    public int GetItemCountByID(string itemID)
    {
        int count = 0;
        foreach (var item in inventoryItems)
        {
            if (item.itemID == itemID)
                count++;
        }
        return count;
    }

    public bool RemoveItemByID(string itemID, int amount)

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						97
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{
    int removed = 0;
    for (int i = inventoryItems.Count - 1; i >= 0 && removed < amount; i--)
    {
        if (inventoryItems[i].itemID == itemID)
        {
            inventoryItems.RemoveAt(i);
            removed++;
        }
    }

    if (removed > 0)
        RefreshUI();

    return removed == amount;
}

```

```

//отримати список усіх itemID для збереження
public List<SerializableItem> GetAllSerializableItems()
{
    var list = new List<SerializableItem>();

    foreach (var item in inventoryItems)
    {
        list.Add(new SerializableItem
        {
            itemID = item.itemID,
            itemName = item.itemName,
            cost = item.cost,
            description = item.description,
            itemType = item.itemType,
            buffType = item.buffType,
            healAmount = item.healAmount,
            buffValue = item.buffValue,
            duration = item.duration,
            iconName = item.icon != null ? item.icon.name : ""
        });
    }

    return list;
}

```

```

public void LoadInventory(List<SerializableItem> items)
{
    inventoryItems.Clear();

    foreach (var data in items)
    {
        var clone = ScriptableObject.CreateInstance<ItemData>();
        clone.itemID = data.itemID;
        clone.itemName = data.itemName;
        clone.cost = data.cost;
        clone.description = data.description;
        clone.itemType = data.itemType;
        clone.buffType = data.buffType;
        clone.healAmount = data.healAmount;
        clone.buffValue = data.buffValue;
        clone.duration = data.duration;
        clone.icon = LoadSpriteByName(data.iconName);

        inventoryItems.Add(clone);
    }
}

```

```

        RefreshUI();
    }

    private Sprite LoadSpriteByName(string name)
    {
        if (string.IsNullOrEmpty(name)) return null;

        Sprite[] allIcons = Resources.LoadAll<Sprite>("ItemIcons");
        foreach (var s in allIcons)
        {
            if (s.name == name)
                return s;
        }

        Debug.LogWarning($"Icon '{name}' not found in Resources/ItemIcons/");
        return null;
    }
}

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		99

Система збереження

SaveSystem.cs

```

using System.IO;
using UnityEngine;
using System.Runtime.Serialization.Formatters.Binary;
using System.Collections.Generic;

public static class SaveSystem
{
    public static int CurrentSlot = -1;

    private static string GetSavePath(int slot)
    {
#if UNITY_EDITOR
        return Application.dataPath + $"/Saves/save_{slot}.sav";
#else
        return Application.persistentDataPath + $"/save_{slot}.sav";
#endif
    }

    public static void SaveGame(int slot)
    {
        CurrentSlot = slot; // оновлюємо поточний слот

        SaveData data = new SaveData();

        data.sceneName = UnityEngine.SceneManagement.SceneManager.GetActiveScene().name;

        if (PlayerStats.Instance != null)
        {
            data.playerHP = PlayerStats.Instance.CurrentHealth;
            data.playerXP = PlayerStats.Instance.CurrentXP;
            data.playerLevel = PlayerStats.Instance.Level;
            data.money = PlayerStats.Instance.Money;
            data.damageBonus = PlayerStats.Instance.DamageBonus;
            data.moveSpeedBonus = PlayerStats.Instance.MoveSpeedBonus;
            data.strongAttackMultiplier = PlayerStats.Instance.StrongAttackMultiplier;

            data.buffStates.Clear();
            foreach (var buff in PlayerStats.Instance.activeBuffs)
            {
                data.buffStates.Add(new SaveData.BuffState
                {
                    type = buff.type,
                    value = buff.value,
                    duration = buff.duration,
                    timeRemaining = buff.timeRemaining
                });
            }
        }

        if (InventoryManager.Instance != null)
            data.inventoryItems = InventoryManager.Instance.GetAllSerializableItems();

        GameObject player = GameObject.FindGameObjectWithTag("Player");
        if (player != null)
            data.playerPosition = new SerializableVector3(player.transform.position);

        if (SceneStateTracker.Instance != null)

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						100
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{
    data.disabledObjectIDs = SceneStateTracker.Instance.GetAllDisabledIDs();
    data.activatedObjectIDs = SceneStateTracker.Instance.GetAllActivatedIDs();
    data.givenRewardIDs = SceneStateTracker.Instance.GetAllGivenRewards();

    data.objectPositions = new Dictionary<string, SerializableVector3>();
    foreach (var kvp in SceneStateTracker.Instance.GetObjectPositions())
    {
        data.objectPositions[kvp.Key] = new SerializableVector3(kvp.Value);
    }
}

BinaryFormatter formatter = new BinaryFormatter();
using (FileStream stream = new FileStream(GetSavePath(slot), FileMode.Create))
{
    formatter.Serialize(stream, data);
}

PlayerPrefs.SetString($"Slot{slot}_Time", System.DateTime.Now.ToString("yyyy-MM-dd HH:mm"));
PlayerPrefs.Save();
}

public static SaveData LoadGame(int slot)
{
    string path = GetSavePath(slot);
    if (File.Exists(path))
    {
        CurrentSlot = slot;
        BinaryFormatter formatter = new BinaryFormatter();
        using (FileStream stream = new FileStream(path, FileMode.Open))
        {
            SaveData data = (SaveData)formatter.Deserialize(stream);

            SceneStateTracker.Instance?.LoadStateFromSave(data);
            return data;
        }
    }

    return null;
}
}

```

GameInitializer.cs

```

using UnityEngine;
public class GameInitializer : MonoBehaviour
{
    void Start()
    {
        if (PlayerPrefs.HasKey("LoadSlot"))
        {
            int slot = PlayerPrefs.GetInt("LoadSlot");
            SaveData data = SaveSystem.LoadGame(slot);

            if (data != null)
            {
                if (PlayerStats.Instance != null)
                    PlayerStats.Instance.LoadPlayer(data);

                if (InventoryManager.Instance != null)
                {
                    InventoryManager.Instance.ClearInventory();
                    InventoryManager.Instance.LoadInventory(data.inventoryItems);
                }
            }
        }
    }
}

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						101
Змн.	Арк.	№ докум.	Підпис	Дата		

```

GameObject player = GameObject.FindGameObjectWithTag("Player");
if (player != null)
    player.transform.position = data.playerPosition.ToVector3();

if (SceneStateTracker.Instance != null)
{
    SceneStateTracker.Instance.LoadDisabledIDs(data.disabledObjectIDs);
    SceneStateTracker.Instance.LoadActivatedIDs(data.activatedObjectIDs);
    SceneStateTracker.Instance.LoadObjectPositions(data.objectPositions);

    foreach (var obj in GameObject.FindObjectsOfType<SceneObjectID>())
    {
        string id = obj.uniqueID;

        if (SceneStateTracker.Instance.IsObjectDisabled(id))
        {
            if (obj.TryGetComponent(out EnemyHealth enemyHealth))
            {
                if (!enemyHealth.IsDead)
                    enemyHealth.ForceDie();
            }
            else if (obj.TryGetComponent(out BossHealth bossHealth))
            {
                if (!bossHealth.IsDead)
                    bossHealth.ForceDie();
            }
            else
            {
                obj.gameObject.SetActive(false);
            }
        }
        else if (SceneStateTracker.Instance.IsObjectActivated(id))
        {
            obj.gameObject.SetActive(true);
        }

        if (data.objectPositions.ContainsKey(id))
        {
            obj.transform.position = data.objectPositions[id].ToVector3();
        }
    }
}

PlayerPrefs.DeleteKey("LoadSlot");
}
}
}

```

SceneStateTracker.cs

```

using System.Collections.Generic;
using UnityEngine;

public class SceneStateTracker : MonoBehaviour
{
    public static SceneStateTracker Instance;

    private HashSet<string> disabledObjects = new HashSet<string>();
    private HashSet<string> activatedObjects = new HashSet<string>();
    private Dictionary<string, Vector3> objectPositions = new Dictionary<string, Vector3>();
    private HashSet<string> givenRewards = new HashSet<string>();

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						102
Змн.	Арк.	№ докум.	Підпис	Дата		

```

private void Awake()
{
    if (Instance == null)
    {
        Instance = this;
        DontDestroyOnLoad(gameObject);
    }
    else Destroy(gameObject);
}

public void MarkObjectAsDisabled(string id)
{
    if (!string.IsNullOrEmpty(id))
        disabledObjects.Add(id);
}

public bool IsObjectDisabled(string id) => disabledObjects.Contains(id);
public List<string> GetAllDisabledIDs() => new List<string>(disabledObjects);
public void LoadDisabledIDs(List<string> list) => disabledObjects = new HashSet<string>(list);

public void MarkObjectAsActivated(string id)
{
    if (!string.IsNullOrEmpty(id))
        activatedObjects.Add(id);
}

public bool IsObjectActivated(string id) => activatedObjects.Contains(id);
public List<string> GetAllActivatedIDs() => new List<string>(activatedObjects);
public void LoadActivatedIDs(List<string> list) => activatedObjects = new HashSet<string>(list);

public void SetObjectPosition(string id, Vector3 position)
{
    if (!string.IsNullOrEmpty(id))
        objectPositions[id] = position;
}

public Dictionary<string, Vector3> GetObjectPositions() => objectPositions;

public void LoadObjectPositions(Dictionary<string, SerializableVector3> positions)
{
    objectPositions.Clear();
    foreach (var kvp in positions)
    {
        objectPositions[kvp.Key] = kvp.Value.ToVector3();
    }
}

public void MarkRewardAsGiven(string id)
{
    if (!string.IsNullOrEmpty(id))
        givenRewards.Add(id);
}

public bool IsRewardGiven(string id) => givenRewards.Contains(id);
public List<string> GetAllGivenRewards() => new List<string>(givenRewards);
public void LoadGivenRewards(List<string> list) => givenRewards = new HashSet<string>(list);

public void ResetState()
{
    disabledObjects.Clear();
    activatedObjects.Clear();
    objectPositions.Clear();
}

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
						103
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        givenRewards.Clear();
        Debug.Log("SceneStateTracker очищено.");
    }

    public void LoadStateFromSave(SaveData data)
    {
        ResetState(); // очищає глобальний стан перед завантаженням

        if (data == null) return;

        LoadDisabledIDs(data.disabledObjectIDs);
        LoadActivatedIDs(data.activatedObjectIDs);
        LoadGivenRewards(data.givenRewardIDs);
        LoadObjectPositions(data.objectPositions);
    }
}

```

					КН.КР.Б – 122 – 25 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		104