МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«ПРОЕКТУВАННЯ ТА РОЗРОБКА ГРИ З ВИКОРИСТАННЯМ UNREAL ENGINE 5 ТА C++»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
М.О. Масюкевич
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Мильцев О.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Масюкевичу Микиті Олексійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Проектування та розробка гри з використанням Unreal Engine 5 та C++

керівник роботи Мильцев Олександр Михайлович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка та впровадження ігрового проєкту за допомогою сучасних технологій.
4. Тестування ігрового застосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	25.03.2024	
2.	Збір вихідних даних.	15.04.2024	
3.	Обробка методичних та теоретичних	19.06.2024	
	джерел.		
4.	Розробка першого та другого розділу.	09.09.2024	
5.	Розробка третього та четвертого розділу.	15.11.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	12.12.2024	

Студент _____ М.О. Масюкевич
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.М. Мильцев
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Проектування та розробка гри з використанням Unreal Engine 5 та C++»: 60 с., 21 рис., 7 джерел, 4 додатки.

ГРА, ІГРОВИЙ РУШІЙ, РОЗРОБКА ІНТЕРАКТИВНИХ СИСТЕМ, C++, GAME DESIGN, UNREAL ENGINE 5.

Об’єкт дослідження – процес проєктування та розробки комп’ютерних ігор за допомогою ігрового рушія Unreal Engine 5 та мови програмування C++.

Мета роботи: створення прототипу гри з використанням Unreal Engine 5 та реалізація основних механік за допомогою C++.

Метод дослідження – аналіз сучасних технологій розробки ігор, дослідження можливостей Unreal Engine 5 та C++ для створення інтерактивного ігрового середовища.

У кваліфікаційній роботі досліджено предметну область комп’ютерних ігор, їх класифікацію та основні вимоги до сучасних ігрових проєктів. Проведено аналіз можливостей Unreal Engine 5 як одного з провідних інструментів для створення інтерактивних продуктів. Вивчено переваги використання мови C++ для роботи з ігровим рушієм.

Розроблено концепцію гри, виконано проєктування її основних механік, інтерфейсу користувача та структури рівнів. Реалізовано основні ігрові функції, такі як керування персонажем, обробка фізики та взаємодія об’єктів у 3D-середовищі. Для забезпечення зручності використання створено інтуїтивно зрозумілий інтерфейс користувача.

Розроблений проєкт може бути використаний як основа для подальшого розвитку ігрового продукту або як навчальний приклад для вивчення можливостей Unreal Engine 5 та C++.

SUMMARY

Master’s qualifying paper “Game Design and Development Using Unreal Engine 5 and C++”: 60 pages, 21 figures, 7 references, 4 supplements.

C++, GAME, GAME DESIGN, GAME ENGINE, INTERACTIVE SYSTEMS DEVELOPMENT, UNREAL ENGINE 5.

The object of the study is the process of designing and developing computer games using the Unreal Engine 5 game engine and the C++ programming language.

The aim of the study is to create a game prototype using Unreal Engine 5 and implement core mechanics with C++.

The method of research is the analysis of modern game development technologies, exploration of Unreal Engine 5 capabilities, and the study of C++ for creating interactive game environments.

In the qualification work, the subject area of computer games, their classification, and the key requirements for modern gaming projects were investigated. The capabilities of Unreal Engine 5 as one of the leading tools for creating interactive products were analyzed. The advantages of using C++ for game engine development were explored.

The concept of the game was developed, including the design of its core mechanics, user interface, and level structures. The primary gameplay features were implemented, such as character control, physics processing, and object interaction in a 3D environment. An intuitive and user-friendly interface was created to enhance user experience.

The developed project can serve as a foundation for further game development or as a teaching example for exploring the capabilities of Unreal Engine 5 and C++.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Теоретичні відомості 10
1.1 Поняття комп’ютерної гри..... 10
1.2 Вимоги до сучасних ігор 11
1.3 Популярні ігрові рушії: порівняльний аналіз 12
1.4 Особливості Unreal Engine 5..... 14
1.5 Переваги використання C++ у розробці ігор 16
1.6 Аналіз вимог до проєкту 19
2 Проєктування..... 21
2.1 Цілі та концепція гри..... 21
2.2 Проєктування механік гри..... 22
2.3 Проєктування дизайну користувацького інтерфейсу..... 24
2.4 Створення сценарію рівнів..... 26
2.5 Створення архітектури проєкту в Unreal Engine 5 28
2.6 Створення класів..... 29
3 Реалізація..... 35
3.1 Налаштування Unreal Engine 5 35
3.2 Розробка механіки гри..... 37
3.3 Розробка ігрових рівнів 39
3.3.1 Побудова рівня замку з двома локаціями та мостами 40
3.3.2 Розміщення NPC, та босів..... 40
3.3.3 Налаштування освітлення для створення атмосфери темного середньовіччя 41
3.4 Розробка користувацького інтерфейсу 42

4 Тестування 44
4.1 Тестування ігрових механік 44
4.2 Тестування UI та UX 46
4.3 Оптимізація 47
Висновки 49
Перелік посилань 51
Додаток А Основний клас PlayerCharacter.h 52
Додаток Б Реалізація методів у PlayerCharacter.cpp 54
Додаток В Код класу GameManager.h 56
Додаток Г Реалізація методів у GameManager.cpp 58

ВСТУП

У сучасному світі ігрова індустрія займає важливе місце серед сфер, що швидко розвиваються. Комп'ютерні ігри перестали бути лише розвагою, вони стали потужним інструментом для навчання, тренувань, соціальної взаємодії та навіть терапії. З розвитком технологій створення ігор стало значно доступнішим, а ігрові продукти – складнішими, більш інтерактивними та більш захоплюючими.

Однією з ключових складових успішної гри є її технічна реалізація. Якість програмного забезпечення, графіки, фізики та інтерактивних елементів напряму впливає на досвід користувача, задоволення аудиторії та комерційний успіх проєкту. Сучасні інструменти, такі як Unreal Engine 5, дозволяють розробникам створювати вражаючі ігри, забезпечуючи потужну графіку, реалістичну фізику та високий рівень продуктивності.

C++ є одним із найпопулярніших мов програмування для створення ігор, оскільки забезпечує низький рівень доступу до ресурсів системи та високу продуктивність. Використання Unreal Engine 5 разом із C++ відкриває перед розробниками широкі можливості для створення ігор будь-якої складності та жанру.

Розробка гри вимагає інтеграції багатьох компонентів: дизайну, програмування, проєктування ігрового процесу та тестування. Вибір правильного інструментарію та засобів розробки є вирішальним фактором для успішного втілення ідеї.

Кваліфікаційну роботу присвячено проєктуванню та розробці гри за допомогою Unreal Engine 5 та C++.

Метою дослідження є створення прототипу гри з використанням сучасного ігрового рушія Unreal Engine 5 та реалізація її основних механік мовою програмування C++.

Об'єктом дослідження є ігровий рушій Unreal Engine 5 та мова програмування C++.

Предметом дослідження є процес розробки гри засобами Unreal Engine 5 та C++.

У відповідності з метою дослідження ставляться такі **завдання**:

- дослідити предметну область комп’ютерних ігор та вимоги до сучасних ігрових продуктів;
- проаналізувати можливості Unreal Engine 5 та C++ для створення ігор;
- виконати проєктування концепції гри, її механік, дизайну та архітектури;
- реалізувати прототип гри із базовими механіками та інтерфейсом користувача;
- провести тестування та оцінити результати.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Поняття комп’ютерної гри

Комп’ютерні ігри є однією з найбільш динамічних сфер сучасних інформаційних технологій. Вони створюються для розваги, навчання, соціальної взаємодії та навіть терапії. Комп’ютерна гра – це інтерактивний програмний продукт, що надає користувачеві можливість зануритися у віртуальний світ, де відбувається симуляція різноманітних процесів, взаємодій та ситуацій.

Комп’ютерні ігри класифікують за багатьма ознаками, такими як жанр, платформа, технології розробки, а також тип взаємодії користувача з ігровим середовищем. Успішна розробка гри базується на дотриманні основних принципів, що сприяють досягненню високої якості та залученості гравця [1].

Принцип інтерактивності. Гра повинна забезпечувати безперервну взаємодію між гравцем та ігровим середовищем, що створює ефект занурення.

Принцип гнучкості. Ігровий продукт має адаптуватися до різних потреб користувачів, надаючи можливість вибору рівня складності, налаштувань графіки та управління.

Принцип модульності. Ігрова архітектура повинна бути побудована таким чином, щоб можна було легко додавати або змінювати окремі компоненти гри, наприклад, рівні, персонажів чи механіки.

Комп’ютерні ігри є унікальними продуктами, які суттєво відрізняються від інших типів програмного забезпечення. Їхня основна мета – забезпечити користувачеві розважальний чи навчальний досвід, а також стимулювати його взаємодію з ігровим світом.

Незважаючи на різноманіття жанрів і форматів, комп’ютерні ігри мають декілька спільних характеристик:

– комп’ютерні ігри завжди мають ігрову логіку, яка визначає правила та механіки, що керують взаємодіями в ігровому середовищі;

– ігри повинні бути зручними для кінцевого користувача, незалежно від його технічних знань;
– інтерфейс гри має бути інтуїтивно зрозумілим, забезпечуючи легкий доступ до всіх функцій;
– якість графіки та аудіо відіграє важливу роль, адже саме ці компоненти формують естетичне сприйняття гри.

Таким чином, комп’ютерна гра є складним продуктом, що поєднує в собі елементи програмування, дизайну, мистецтва та інтерактивності, створюючи унікальний досвід для користувача

1.2 Вимоги до сучасних ігор

Інтерактивність. Інтерактивність є одним із ключових факторів сучасних ігор. Гра повинна надавати гравцеві можливість активно впливати на ігрове середовище, приймати рішення, які впливають на розвиток сюжету, та взаємодіяти з об’єктами у реальному часі. Сучасні ігри мають забезпечувати багаторівневу інтерактивність, включаючи як ігрові механіки, так і соціальну взаємодію з іншими гравцями.

Гнучкість. Гра повинна адаптуватися до різних потреб користувачів, надаючи можливість вибору рівня складності, стилю гри, а також параметрів налаштувань, таких як графіка, звук, чи управління. Це дозволяє залучати ширшу аудиторію та створювати позитивний користувацький досвід.

Надійність. Сучасна гра повинна функціонувати стабільно без збоїв та помилок. Це включа є підтримку серверів для багатокористувацьких режимів, резервне копіювання збережень та стабільну роботу на різних платформах. Надійність також залежить від якості тестування та оптимізації ігрового коду.

Ефективність. Ефективність ігор полягає у швидкій обробці дій користувача, мінімізації затримок у багатокористувацькому режимі та оптимізації використання ресурсів системи. Сучасні ігри повинні працювати

плавно навіть на середньому обладнанні, надаючи користувачам якісний графічний та ігровий досвід.

Безпека. Сучасні ігри повинні забезпечувати захист даних гравців, таких як облікові записи, платіжна інформація та персональні дані. Безпека гарантується шифруванням даних, захистом серверів від атак та постійним оновленням системи для усунення вразливостей.

Естетичність. Графіка, дизайн персонажів та середовища, звуковий супровід та загальна атмосфера гри мають відповідати сучасним стандартам і створювати глибокий емоційний зв’язок із гравцем. Якісне візуальне та аудіо оформлення є важливим аспектом залучення користувачів.

Таким чином, для досягнення успіху сучасна гра має відповідати високим стандартам у різних аспектах, враховуючи технічну стабільність, гнучкість, зручність для гравця та безпеку даних.

1.3 Популярні ігрові рушії: порівняльний аналіз

Сучасна ігрова індустрія пропонує широкий вибір ігрових рушіїв, які є ключовими інструментами для створення комп’ютерних ігор. Рушій – це програмна платформа, яка забезпечує базовий функціонал для розробки ігор, таких як відображення графіки, обробка фізики, управління об’єктами, створення анімацій тощо. Кожен рушії має свої переваги та недоліки, які роблять його більш чи менш придатним для певних завдань.

Unreal Engine від Epic Games є одним із найбільш потужних та популярних ігрових рушіїв у світі.

Переваги:

– висока якість графіки завдяки підтримці передових технологій, таких як Lumen і Nanite;
– можливість використовувати як C++, так і Blueprints для програмування;
– потужний редактор рівнів і візуальних ефектів;

– широкий спектр інструментів для створення багатокористувацьких ігор.

Недоліки:

– високі вимоги до обладнання для розробки;

– висока крива навчання для новачків.

Unity – це універсальний рушій, який підходить як для 2D, так і для 3D ігор. Він широко використовується як інді-розробниками, так і великими студіями.

Переваги:

– простота у використанні, зручний інтерфейс;

– велика кількість ресурсів для навчання та активна спільнота;

– підтримка багатьох платформ, включаючи мобільні пристрої та vr.

Недоліки:

– обмеженість у створенні високоякісної графіки порівняно з unreal engine;

– менше інструментів для роботи з великими проєктами.

Godot – це безкоштовний рушій з відкритим вихідним кодом, який ідеально підходить для розробки інді-ігор.

Переваги:

– повністю безкоштовний і з відкритим кодом;

– проста інтеграція для 2d ігор;

– низькі вимоги до обладнання.

Недоліки:

– менш потужний для створення графіки у 3d;

– відсутність деяких функцій, доступних в unreal engine чи unity.

CryEngine відомий своєю високою якістю графіки та часто використовується для створення ігор AAA-класу.

Переваги:

– реалістична графіка та фізика;

– широкий спектр інструментів для створення відкритих світів.

Недоліки:

– високі вимоги до обладнання;
– менша спільнота розробників та обмежена кількість навчальних ресурсів.

GameMaker Studio є ідеальним вибором для створення 2D-ігор завдяки своїй простоті та орієнтованості на новачків.

Переваги:

– інтуїтивно зрозумілий інтерфейс, зручний для початківців;
– швидка розробка простих ігор.

Недоліки:

– обмежені можливості для створення 3d-ігор;
– менше інструментів для складних ігрових проєктів.

Кожен з описаних рушіїв має свої особливості, що роблять його придатним для конкретних завдань. Вибір інструмента залежить від масштабу проєкту, вимог до графіки, цільової платформи та досвіду розробника. Unreal Engine є оптимальним вибором для створення високоякісних 3D-ігор з реалістичною графікою, тоді як Unity більше підходить для універсальних проєктів. Godot і GameMaker Studio орієнтовані на інді-розробників, які створюють прості та середні за складністю ігри.

1.4 Особливості Unreal Engine 5

Unreal Engine 5 (UE5) є однією з найбільш передових платформ для розробки комп'ютерних ігор [2]. Рушії створений компанією Epic Games і пропонує широкий спектр інструментів для створення віртуальних світів, що відповідають сучасним стандартам графіки та продуктивності. Унікальні технології та функціональні можливості UE5 дозволяють розробляти ігри будь-якого рівня складності, від інді-проєктів до ігор AAA-класу.

Реалістична графіка: Lumen і Nanite (див. рис. 1.1).

Однією з головних переваг Unreal Engine 5 є використання інноваційних технологій Lumen і Nanite.

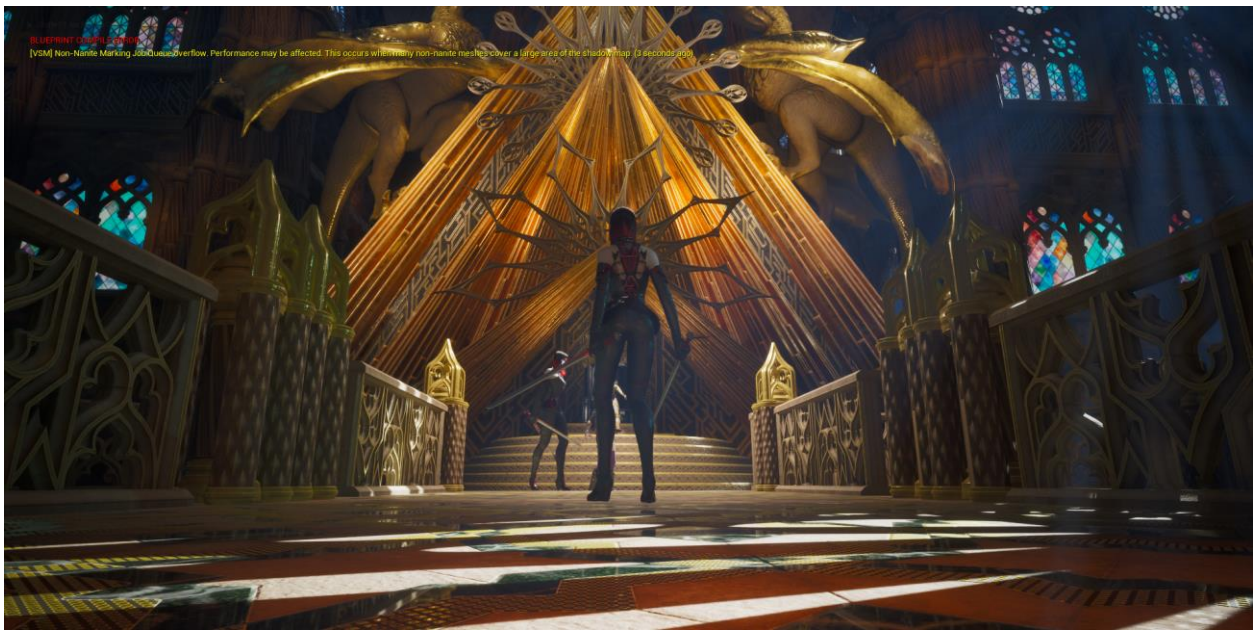


Рисунок 1.1 – Графіка Unreal Engine 5

Lumen – це глобальна система освітлення, яка дозволяє створювати динамічне освітлення з реалістичними тінями та відбиттями. Завдяки цьому гра адаптується до змін у середовищі, таких як зміна часу доби чи освітлення в приміщеннях.

Nanite – це технологія віртуалізованої геометрії, яка дозволяє використовувати детальні 3D-моделі без шкоди для продуктивності. Ця функція дає змогу завантажувати об’єкти з мільйонами полігонів і зменшувати вимоги до оптимізації вручну.

Можливості програмування: Blueprints і C++.

Unreal Engine 5 пропонує два основних підходи до програмування:

– Blueprints – візуальний інструмент, який дозволяє створювати ігрову логіку без написання коду (це зручне рішення для дизайнерів, які можуть працювати над механіками гри без залучення програмістів);
– C++ – мова програмування, яка забезпечує розробникам повний контроль над усіма аспектами гри (використання C++ дозволяє створювати складні системи, оптимізувати код і досягати високої продуктивності).

Методи роботи з відкритими світами. UE5 надає потужні інструменти для створення відкритих світів, які є важливими для багатьох сучасних ігор.

Функція World Partition автоматично розділяє великий ігровий світ на менші сегменти, що спрощує управління ресурсами і прискорює процес розробки.

Методи реалістичної анімації. Unreal Engine 5 має вдосконалені системи анімації, такі як Control Rig і Full-Body IK. Ці інструменти дозволяють створювати складні анімаційні сцени без додаткових програм. Наприклад, розробники можуть додати детальні рухи персонажів або інтегрувати взаємодію з навколишнім середовищем без зайвих зусиль.

Розширена інтеграція для різних платформ. UE5 підтримує розробку для багатьох платформ, включаючи ПК, консолі, мобільні пристрої та VR/AR-системи. Це дозволяє створювати мультиплатформні проєкти без необхідності суттєвих змін у коді або ресурсах гри.

Оптимізація продуктивності. UE5 має інтегровані інструменти для оптимізації гри. Наприклад, розробники можуть використовувати Profiling Tools, щоб відстежувати продуктивність, визначати проблемні місця і швидко їх виправляти.

Активна спільнота і доступні ресурси. Unreal Engine 5 має величезну спільноту розробників, які постійно створюють навчальні матеріали, плагіни та моделі. Epic Games також пропонує безкоштовний доступ до багатьох ресурсів у Unreal Marketplace, що значно спрощує розробку гри.

Таким чином, Unreal Engine 5 є одним із найпотужніших інструментів для створення сучасних ігор. Його функції, такі як Lumen, Nanite, гнучкі підходи до програмування та оптимізація, роблять його ідеальним вибором для розробників, які прагнуть створити високоякісний продукт.

1.5 Переваги використання C++ у розробці ігор

У процесі розробки ігор на Unreal Engine 5 розробникам доступні два основних інструменти для створення ігрової логіки: C++ та Blueprints (рис. 1.2). Кожен із них має свої переваги й недоліки, а вибір між ними залежить від завдань, які необхідно вирішити.

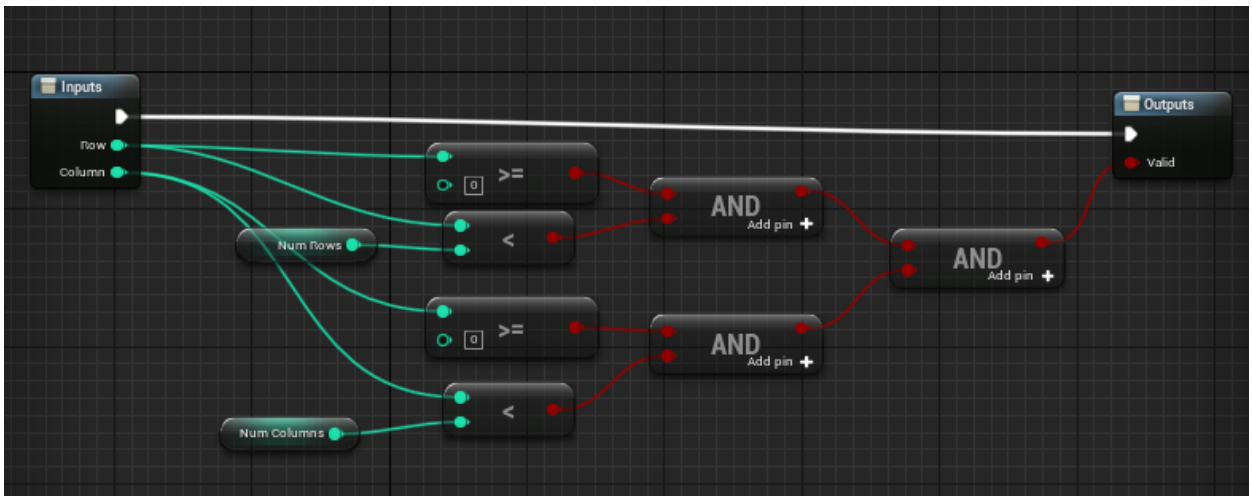


Рисунок 1.2 – Blueprints у Unreal Engine 5

Розглянемо **основні переваги C++**.

Висока продуктивність. Один із ключових аргументів на користь використання C++ – це його продуктивність. Код, написаний на C++, виконується значно швидше, ніж аналогічна логіка, створена у Blueprints. Це особливо важливо для проєктів із великим навантаженням на процесор, наприклад, при обробці фізики, AI або інших складних обчислень.

Контроль над ресурсами. C++ дозволяє розробникам мати повний контроль над пам’яттю та іншими ресурсами системи. Це важливо для оптимізації роботи гри та мінімізації використання ресурсів.

Складні механіки. Деякі функції Unreal Engine доступні тільки при роботі з C++. Наприклад, створення власних класів і компонентів із унікальними властивостями.

Модульність та масштабованість. C++ дозволяє легко підтримувати та розширювати код, що особливо важливо для великих команд і проєктів AAA-класу.

Розглянемо **основні переваги Blueprints**.

Простота використання. Blueprints – це інструмент візуального програмування, що дозволяє створювати ігрову логіку без необхідності писати код. Це ідеально підходить для дизайнерів, які не мають досвіду програмування.

Швидке прототипування. Завдяки простоті та наочності Blueprints дозволяють швидко створювати і тестувати нові ігрові механіки без довгого написання коду.

Зрозумілість для команди. Графічний підхід Blueprints робить логіку зрозумілою для всіх членів команди, включаючи дизайнерів і художників.

Порівняння C++ і Blueprints у практичному застосуванні. Для невеликих ігор або прототипів Blueprints можуть бути достатніми, адже вони швидкі й прості у використанні.

У складних проєктах або в задачах, що потребують високої продуктивності (наприклад, AI, фізика, обчислення), C++ є більш потужним і гнучким інструментом (рис. 1.3).

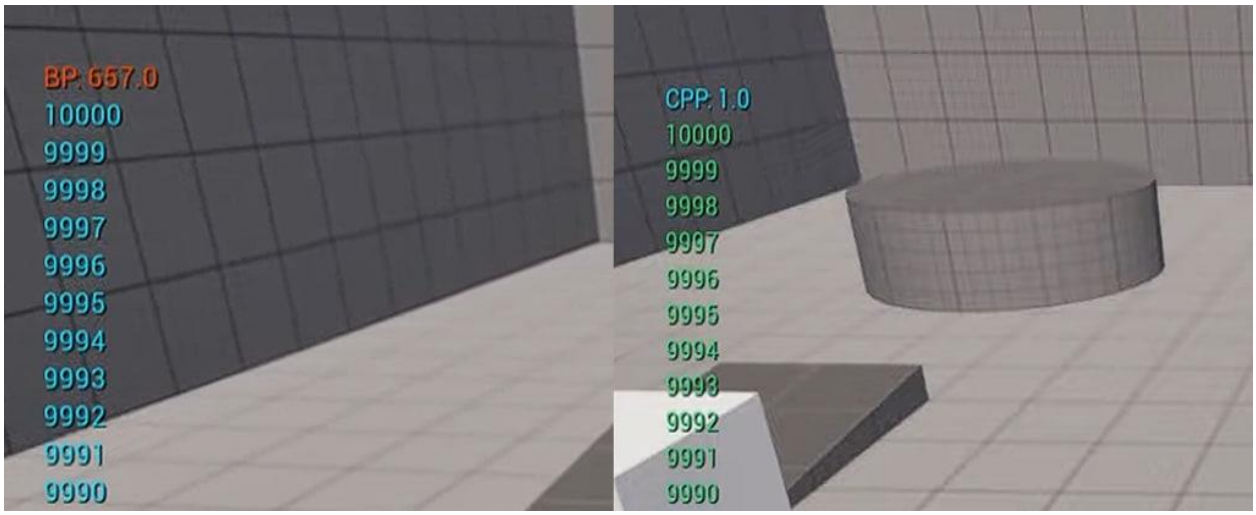


Рисунок 1.3 – Порівняння швидкості Blueprint та C++

Обидва інструменти, C++ та Blueprints, мають свої переваги й знаходять своє місце у розробці ігор.

Оптимальним є комбіноване використання: Blueprints для швидкого створення базової логіки або прототипів, а C++ – для реалізації складних, ресурс затратних механік. Це дозволяє ефективно використовувати всі можливості Unreal Engine 5 та забезпечувати баланс між швидкістю розробки та якістю гри.

1.6 Аналіз вимог до проєкту

Для успішної реалізації ігрового проєкту необхідно провести детальний аналіз вимог, що ставляться до гри. Це дозволяє визначити ключові аспекти розробки, забезпечити узгодженість між етапами створення та досягти поставлених цілей. Вимоги поділяються на функціональні, технічні та візуальні.

Функціональні вимоги визначають, які елементи та механіки повинні бути реалізовані у грі.

Ігровий процес:

– реалізація рівня з чітко визначеним набором завдань для гравця;
– створення взаємодії між персонажем, ворогами та оточенням;
– наявність системи бою, що включає атаку, захист, шкоду і динамічні зміни стану персонажа.

Управління персонажем: плавне управління з можливістю використання різних дій (рух, атака, взаємодія з об’єктами).

Ігровий баланс: пропорційний розподіл складності між різними етапами рівня.

Технічні вимоги. Технічні аспекти гарантують стабільну роботу гри та її сумісність із сучасними платформами.

Продуктивність: гра повинна забезпечувати стабільну роботу на цільових платформах із частотою щонайменше 60 FPS.

Інтеграція ресурсів:

– використання існуючих 3D-моделей для побудови рівня;
– застосування інструментів Unreal Engine 5, таких як Lumen і Nanite, для оптимізації роботи графіки.

Архітектура:

– створення ігрової логіки із застосуванням C++;
– наявність структури, що дозволяє легко додавати нові механіки або об’єкти.

Візуальні вимоги. Візуальна складова гри є важливою частиною для занурення гравця у її світ.

Дизайн рівня: деталізована локація, що включає різноманітні зони: просторі зали, вузькі коридори, відкриті майданчики для бою.

Атмосфера: використання реалістичних текстур, освітлення та візуальних ефектів, які створюють відчуття глибини та деталізації.

Анімація: реалістичні рухи персонажів та взаємодія з об’єктами.

Аналіз вимог є базовим етапом для планування роботи над проєктом. Він дозволяє чітко визначити, які завдання мають бути виконані, забезпечити їхню реалізацію в рамках доступних ресурсів і часу, а також гарантувати високий рівень якості розробки.

2 ПРОЄКТУВАННЯ

2.1 Цілі та концепція гри

Розробка комп’ютерної гри – це складний творчий і технічний процес, що поєднує програмування, дизайн і створення інтерактивного середовища. Метою даного проєкту є створення прототипу гри, яка поєднує захоплюючий ігровий процес, продуману бойову систему та атмосферний дизайн рівня. Гра має демонструвати основні механіки, притаманні сучасним ігровим продуктам, а також відповідати високим стандартам якості.

Ціль проєкту Основною ціллю гри є створення рівня з динамічними боями, у якому гравець стикається з кількома складними противниками та має продемонструвати свою майстерність у бою. Особлива увага приділяється реалізації механік, які підкреслюють складність і тактичність ігрового процесу.

Додаткові цілі:

– забезпечити інтерактивність ігрового середовища, де кожна взаємодія гравця з елементами світу має значення;
– продемонструвати графічні можливості Unreal Engine 5, створивши деталізований і атмосферний рівень;
– реалізувати бойову систему, яка включає управління здоров’ям, витривалістю, а також комбінації атак;
– побудувати фундамент для подальшої розробки повноцінної гри зі складним сюжетом.

Концепція гри. Гра належить до жанру екшн із елементами бойових RPG, де основний акцент зроблено на тактичному бою та складних противниках. Ідея натхненна іграми в жанрі Souls-like, які відомі своїм рівнем складності, детальним дизайном світу та атмосферою.

У грі реалізовано рівень, що представляє собою замок. Замок складається з різноманітних зон, таких як просторі зали для битв, вузькі

коридори для уникнення атак і відкриті арени для боїв із босами. Гравець стикається з кількома послідовними противниками, серед яких є звичайні NPC і боси. Кожен бос має унікальний набір атак і тактик, що змушує гравця адаптуватися до нових викликів.

Ключові особливості концепції:

– атмосферний світ: локація замку створена з використанням реалістичних текстур і динамічного освітлення, що підкреслює напруження ігрового процесу.

Бойова система:

– реалізовані удари, блокування, увороти та комбінаці;
– відображення стану здоров’я та витривалості гравця та босів.

Противники:

– кілька NPC, які створюють додаткові труднощі;
– 2–3 боси, які з’являються по черзі після перемоги над попереднім.

Місце в повноцінній грі. Рівень, створений у рамках цього проєкту, є лише частиною більшого задуму. Гра має включати повноцінний сюжет із глибокою історією, який пояснює події, що відбуваються на рівні. Запропонований прототип слугує демонстрацією механік, візуального стилю та загальної атмосфери, які будуть розширені в майбутньому.

Ціль і концепція гри базуються на створенні атмосферного рівня з акцентом на бойову систему. Реалізація цього рівня дозволяє продемонструвати ключові механіки та потенціал для подальшого розвитку повноцінної гри з багатошаровим сюжетом і більш складними елементами.

2.2 Проєктування механік гри

Проєктування механік гри визначає взаємодію гравця зі світом, ворогами та ключовими елементами ігрового процесу. Основна увага приділена бойовій системі, яка включає атаки, ухилення, блоки та управління витривалістю, забезпечуючи тактичну глибину гри. Гравець може виконувати

атаки, які завдають шкоди противнику, витрачаючи витривалість, що обмежує можливість постійної агресії. Блокування дозволяє захиститися від ударів, але також споживає витривалість, а при її недостатності блок не вдається. Ухилення дає змогу уникати атак, відкриваючи можливість для контратаки, однак теж потребує витривалості.

Витривалість є ключовим ресурсом бойової системи, що забезпечує баланс між атакою, захистом і ухиленням. Вона витрачається під час виконання дій і повільно відновлюється, коли гравець перебуває у стані спокою. Противники також володіють бойовими механіками, здатні атакувати, блокувати удари гравця або ухилятися, що додає бою динамічності та вимагає стратегічного підходу.

Для відображення бойової системи використовуються індикатори здоров’я та витривалості, які дозволяють гравцеві контролювати стан персонажа та приймати обдумані рішення під час бою. Реалістичні анімації рухів персонажів, таких як атака, блокування або ухилення, додають глибини ігровому процесу. Під час проєктування була створена діаграма (рис. 2.1), яка описує взаємозв’язки між гравцем, витривалістю, діями та противниками. Ця структура відображає залежність кожної дії від ресурсу та демонструє можливості ворогів, забезпечуючи цілісність ігрової механіки.



Рисунок 2.1 – Діаграма взаємодії бойових механік у грі

Проєктування механік гри дозволило створити основу для їх подальшої реалізації. Завдяки детальному опрацюванню бойової системи, витривалості та взаємодії з противниками, ігровий процес стає тактичним, захоплюючим та викликає інтерес у гравця.

2.3 Проєктування дизайну користувацького інтерфейсу

Дизайн користувацького інтерфейсу (UI) є важливою складовою будь-якої гри [6], адже саме через нього гравець взаємодіє з ігровим світом. У даному проєкті інтерфейс має бути інтуїтивно зрозумілим, зручним і водночас таким, що підкреслює атмосферу гри. Основна мета – надати гравцеві необхідну інформацію, не перевантажуючи екран зайвими елементами.

Цілі дизайну користувацького інтерфейсу. Проєктуючи UI, визначено такі ключові цілі:

– інформативність: забезпечити гравця усіма необхідними даними, такими як рівень здоров’я, витривалості, статус противника та інші показники;
– естетичність: інтерфейс має відповідати стилю гри, створюючи відчуття занурення;
– мінімалізм: уникнення надмірного навантаження екрану зайвими елементами, щоб фокус гравця залишався на ігровому процесі.

Основні елементи інтерфейсу:

– здоров’я гравця: відображає рівень життєвих сил персонажа – розташовується у верхній частині екрану, щоб бути помітною;
– витривалість гравця: показує залишок ресурсу для виконання дій (удари, блоки, ухилення) – розташована поруч із шкалою здоров’я;
– шкала здоров’я противника: з’являється в нижній частині екрану при битві з босом – цей елемент дозволяє гравцеві оцінити прогрес бою;

– система сповіщень: короткі текстові підказки або іконки, які повідомляють гравця про важливі події – низький рівень здоров’я, завершення витривалості тощо.

Розташування елементів. Розміщення елементів UI враховує, що більшість гравців орієнтується на центральну частину екрану, тому всі важливі показники повинні бути максимально доступними (рис. 2.2).



Рисунок 2.2 – Приклад зручного користувацького інтерфейсу у грі Elden Ring

Верхня частина екрану: показник здоров’я та витривалості гравця.

Нижня частина екрану: показник здоров’я противника та панель дій.

Центральна частина: динамічні індикатори влучення або системи сповіщень.

Проєктування користувацького інтерфейсу дозволяє забезпечити гравцеві комфортну взаємодію з грою. Інтерфейс підкреслює атмосферу проєкту та створює баланс між естетикою та функціональністю. Наступним етапом є реалізація UI в ігровому середовищі.

2.4 Створення сценарію рівнів

Сценарій рівня – це важлива частина процесу розробки гри, що визначає загальну структуру локації, розташування противників і ключові елементи дизайну, які впливають на ігровий досвід [5]. У даному проєкті рівень представлено у вигляді невеликого замку, що включає дві основні локації, з'єднані вузькими мостами. Рівень спроєктований так, щоб поєднувати динамічний бойовий процес, елемент ризику та тактичну складність.

Опис рівня. Рівень складається з трьох основних зон:

– початкова точка (Start Point);

– мости;

– локації босів (Boss Locations).

Початкова точка (Start Point). Це стартова позиція гравця, яка дозволяє підготуватися до подальших дій. Зона оточена безпечними просторами, але відразу за нею розташовані NPC, які атакують гравця.

Мости. Головна небезпечна зона рівня. Вузькі мости з обмеженим простором для маневру ускладнюють пересування. На мостах розташовані NPC, які атакують гравця, змушуючи ретельно планувати свої дії. Крім того, через обмежений простір існує ризик падіння, наприклад, через невдале ухилення.

Локації босів (Boss Locations). По обидва боки від мостів розташовані дві великі зали. У цих зонах відбуваються битви з босами. Локації босів просторі, що дозволяє гравцеві використовувати всі доступні бойові механіки, включаючи ухилення, блокування та атаки.

Елементи рівня та левел-дизайн. Левел-дизайн рівня спрямований на створення напруженої атмосфери та поступове збільшення складності:

– розташування NPC: вороги розташовані на мостах та в початковій зоні, їхнє розташування ускладнює пересування через рівень, змушуючи гравця використовувати бойові механіки, такі як ухилення та атаки;

– небезпечні зони (Danger Zones): мости є ключовим елементом ризику – через обмежений простір і ворожі атаки гравець повинен уникати падіння (вони змушують гравця бути обережним і планувати свої дії);
– локації босів: просторі арени для босів створені таким чином, щоб гравець міг застосувати тактичний підхід, використовуючи простір для ухилення та комбінованих атак.

Атмосфера рівня. Дизайн рівня спрямований на створення атмосфери занедбаного замку. Оформлення включає:

– темні освітлені смолоскипами мости, що додають відчуття небезпеки;
– прозорі індикатори зон ризику, які показують можливість падіння;
– величні тронні зали, що підкреслюють епічність боїв із босами.

Процес створення сценарію:

– планування: у процесі проєктування було створено схему рівня (рис. 2.3), що включає основні зони, розташування NPC та небезпечні області;
– розташування об’єктів: основні об’єкти, такі як мости, NPC і зали босів, розташовані таким чином, щоб поступово збільшувати складність гри;
– тестування: було проведено тестування рівня для забезпечення балансу між складністю боїв, тактичним підходом і динамікою гри.

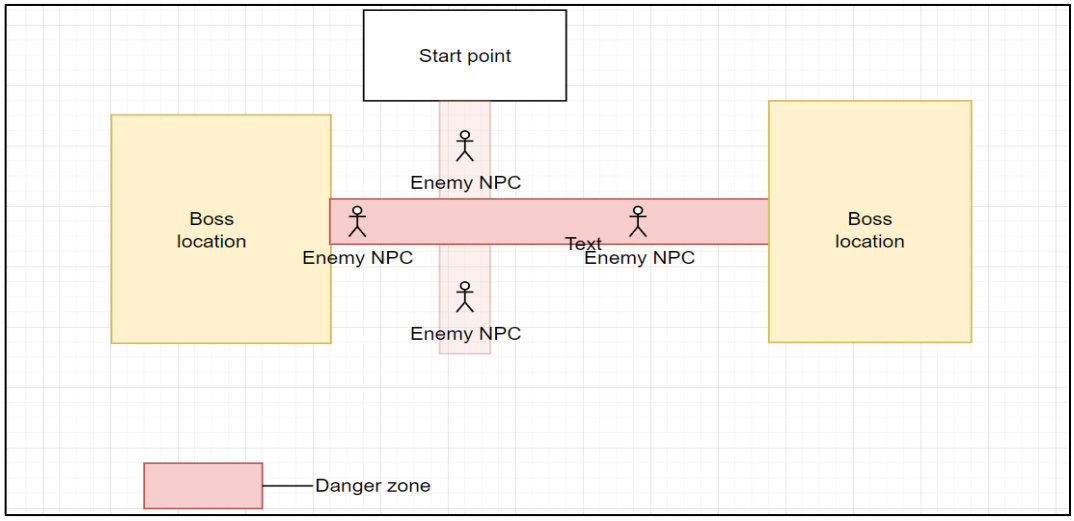


Рисунок 2.3 – Планування рівня

Рівень спроєктований таким чином, щоб забезпечити динамічний ігровий досвід, який поєднує складність, ризик і тактичність. Елементи левел-дизайну, такі як вузькі мости, небезпечні зони та просторі арени, створюють унікальні виклики для гравця та підсилюють атмосферу гри. Наступним кроком є детальна реалізація цього рівня в ігровому середовищі та його оптимізація.

2.5 Створення архітектури проєкту в Unreal Engine 5

Архітектура проєкту є важливим компонентом розробки, який забезпечує структуру та організацію гри. Вона включає ключові компоненти, такі як класи персонажів, противників, механіки та рівні, і дозволяє ефективно реалізувати всі аспекти ігрового процесу.

Мета архітектури. Основна мета архітектури – створити зрозумілу та масштабовану структуру проєкту, яка дозволяє легко вносити зміни, додавати нові механіки або локації. Для реалізації цієї мети використовувалася модульна структура на основі класів C++ та Blueprints в Unreal Engine 5 [3].

Основні компоненти архітектури:

Клас гравця (PlayerCharacter). Відповідає за основну логіку управління персонажем.

Поля:

– health: рівень здоров’я гравця;
– stamina: витривалість для виконання атак та ухилень;
– movementSpeed: швидкість пересування.

Методи:

– takeDamage(float Amount): отримання шкоди;
– useStamina(float Amount): витрачання витривалості;
– performAttack(): виконання атаки.

Базовий клас противників (BaseEnemy): загальний клас для всіх типів ворогів, включаючи NPC і босів.

Поля:

– health: рівень здоров’я ворога;

– attackPower: сила атаки ворога.

Методи:

– takeDamage(float Amount): отримання шкоди;

– performattack(): виконання атаки;

Клас npc. Спеціалізований клас для ворогів, які патрулюють територію. Додатковий метод: patrol(), який забезпечує рух у певному напрямку.

Клас босів (boss). наслідує baseenemy і додає унікальні здібності. Додатковий метод specialability(), який викликає особливу атаку боса.

Клас рівня (level): відповідає за завантаження рівня та спаун об’єктів.

Методи:

– loadlevel(string name): завантаження нового рівня;

– spawnenemies(): створення противників на рівні.

Реалізація структури в Unreal Engine 5. Поєднання C++ та Blueprints:

– основна логіка, що вимагає високої продуктивності (наприклад, бойова система), реалізована на c++;

– інші елементи, такі як тригери подій або інтерфейс, створені через blueprints для швидшого налаштування.

Використання модульного підходу. Кожен елемент (гравець, вороги, рівні) розроблений як окремий модуль – забезпечує простоту розширення гри.

Створення архітектури проєкту в Unreal Engine 5 дозволило побудувати зрозумілу, масштабовану структуру, яка забезпечує легкість у розробці та подальшому розширенні. Така архітектура дозволяє ефективно реалізовувати бойові механіки, поведінку ворогів і структуру рівнів.

2.6 Створення класів

У процесі розробки гри було проведено проєктування класів із використанням об’єктно-орієнтованого підходу, який дозволяє створити

масштабовану, зрозумілу та зручну для підтримки архітектуру коду. Усі класи гри були спроєктовані так, щоб забезпечити модульність, повторне використання та чіткий розподіл функціональності між компонентами.

Основний клас персонажа. Основу для керування персонажем гравця становить клас PlayerCharacter (рис. 2.5), який наслідує функціональність від базового класу ACharacter.

У класі реалізовані такі ключові механіки (рис. 2.4):

– система здоров’я (health) та витривалості (stamina), включаючи методи для їх зміни;
– механіки атак, блокувань і ухилень через інтеграцію анімацій;
– логіка падіння з висоти з використанням методу takefalldamage.

```
APlayerCharacter::APlayerCharacter()
{
    PrimaryActorTick.bCanEverTick = true;

    MaxHealth = 100.0f;
    Health = MaxHealth;

    MaxStamina = 100.0f;
    Stamina = MaxStamina;
}

void APlayerCharacter::BeginPlay()
{
    Super::BeginPlay();
}

void APlayerCharacter::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);
}

void APlayerCharacter::PerformAttack()
{
    if (Stamina > 20.0f)
    {
        Stamina -= 20.0f;
        UE_LOG(LogTemp, Log, TEXT("Attack performed"));
    }
    else
    {
        UE_LOG(LogTemp, Warning, TEXT("Not enough stamina to attack"));
    }
}

void APlayerCharacter::TakeFallDamage()
{
    Health -= 50.0f;
    if (Health <= 0.0f)
    {
        UE_LOG(LogTemp, Warning, TEXT("Player has died!"));
        Destroy();
    }
}
```

Рисунок 2.4 – Реалізація методів у основному класі

```cpp
#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "PlayerCharacter.generated.h"

UCLASS()
class YOURGAME_API APlayerCharacter: public ACharacter
{
    GENERATED_BODY()

public:
    APlayerCharacter();

protected:
    // Здоров'я персонажа
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Stats")
    float Health;

// Максимальне здоров'я
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Stats")
    float MaxHealth;

// Витривалість персонажа
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Stats")
    float Stamina;

// Максимальна витривалість
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Stats")
    float MaxStamina;

// Метод для атаки
void PerformAttack();

// Метод для падіння
void TakeFallDamage();

virtual void Tick(float DeltaTime) override;

protected:
    virtual void BeginPlay() override;
};
```

Рисунок 2.5 – Основний клас персонажа

Клас супротивників. Для NPC створено клас EnemyCharacter (рис. 2.6), який успадковує функціональність від базового ACharacter та включає специфічну поведінку ворогів:

– використання ai behavior tree для реалізації дій, таких як переслідування гравця, атака та відступ;
– обробка анімацій через blueprint, прив'язані до поведінки ші.

Клас босів. Для босів створено окремий клас BossCharacter (рис. 2.7), який успадковує функціональність від EnemyCharacter і додає унікальні механіки:

– поділ бою на фази з переходами між ними залежно від рівня здоров'я боса;
– атаки, що мають розширену логіку та складніші шаблони.

```cpp
#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "EnemyCharacter.generated.h"

UCLASS()
class YOURGAME_API AEnemyCharacter: public ACharacter
{
    GENERATED_BODY()

public:
    AEnemyCharacter();

protected:
    // Здоров'я супротивника
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Stats")
    float Health;

// Метод атаки
void AttackPlayer();

virtual void Tick(float DeltaTime) override;

protected:
    virtual void BeginPlay() override;
};
```

Рисунок 2.6 – Клас супротивників

```cpp
#include "CoreMinimal.h"
#include "EnemyCharacter.h"
#include "BossCharacter.generated.h"

UCLASS()
class YOURGAME_API ABossCharacter: public AEnemyCharacter
{
    GENERATED_BODY()

public:
    ABossCharacter();

protected:
    // Фаза бою
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Boss Mechanics")
    int32 Phase;

// Перехід між фазами
void TransitionToNextPhase();

protected:
    virtual void Tick(float DeltaTime) override;
virtual void BeginPlay() override;
};
```

Рисунок 2.7 – Клас босів

Архітектура керування грою. Створено клас GameManager (рис. 2.8), який відповідає за глобальне керування грою:

– запуск рівня та відслідковування стану гри;

– активування тригерів для боса після завершення бою з npc;

– обробка подій програшу чи перемоги гравця;

– об’єктно-орієнтований підхід.

```
#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "GameManager.generated.h"

UCLASS()
class YOURGAME_API AGameManager: public AActor
{
    GENERATED_BODY()

public:
    AGameManager();

protected:
    // Поточний стан гри
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Game State")
    bool bIsGameOver;

// Активований тригер для боса
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Triggers")
    bool bBossTriggerActivated;

// Лічильник супротивників
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Game State")
    int32 RemainingEnemies;

// Метод для завершення гри
void EndGame(bool bPlayerWon);

// Метод для активації бою з босом
void ActivateBossTrigger();

public:
    virtual void Tick(float DeltaTime) override;

protected:
    virtual void BeginPlay() override;

// Метод для віднімання супротивників
void EnemyDefeated();
};
```

Рисунок 2.8 – Клас GameManager

Використання об’єктно-орієнтованого програмування (ООП) забезпечило:

- інкапсуляцію: усі дані та методи організовані в межах відповідних класів;
- спадкування: базові класи, такі як acharacter, були розширені специфічними класами для гравця, npc і босів;
- поліморфізм: методи, наприклад, attack() чи takedamage(), можуть працювати з об’єктами різних типів, спрощуючи розробку.

Використання C++ і Blueprint. Для створення складної логіки та високої продуктивності ключові механіки реалізовано на C++ (атаки, витривалість, здоров’я).

Для швидкої інтеграції анімацій, тригерів і візуальних ефектів використовувалися Blueprints, які взаємодіють із C++ через інтегровані методи.

Додавання класів на етапі проєктування дозволило створити чітку архітектуру гри. Такий підхід забезпечує легке розширення функціоналу та інтеграцію нових елементів, що дозволяє реалізувати складні ігрові механіки й адаптувати гру під майбутні потреби.

3 РЕАЛІЗАЦІЯ

3.1 Налаштування Unreal Engine 5

Створення нового проєкту. Для початку розробки створюється новий проєкт (рис. 3.1) у Unreal Engine 5. У стартовому меню обирається категорія “Games”, після чого вибирається шаблон “Third Person”. Цей шаблон містить базову логіку персонажа та камери, що дозволяє прискорити початковий етап роботи.

Під час створення проєкту задається назва, наприклад, “CrownFall”, та вказується місце для його збереження. Рекомендується обрати профіль “Maximum Quality” у налаштуваннях графіки, щоб досягти високої якості візуалізації.

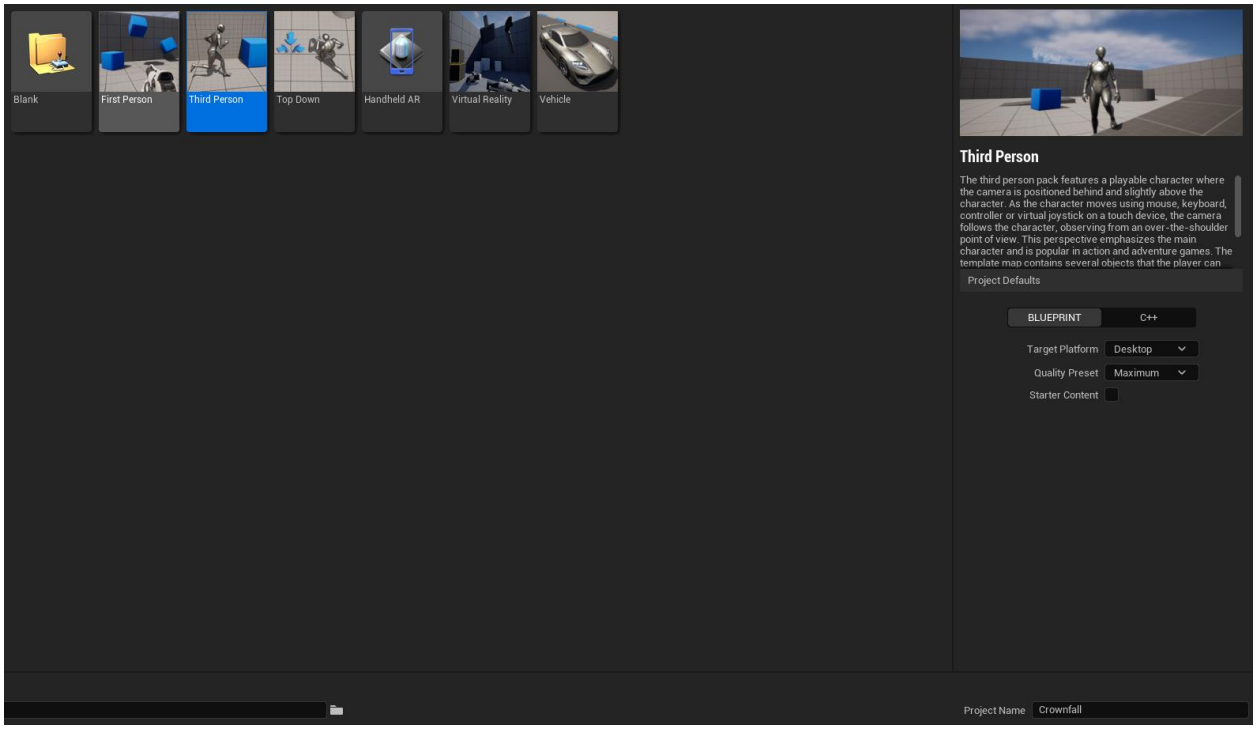


Рисунок 3.1 – Створення проєкту

Налаштування графіки та плагінів. Після створення проєкту необхідно оптимізувати графічні параметри та активувати потрібні плагіни. У меню

“Project Settings” у розділі “Rendering” увімкнено Lumen для реалістичного освітлення та динамічних тіней.

У розділі “Plugins” (рис. 3.2) активуються такі плагіни:

– “niagara” для створення візуальних ефектів, таких як частинки, дим або іскри;
– “animation starter pack” для інтеграції готових анімацій руху персонажа;
– “physics interaction” для забезпечення реалістичної взаємодії з об’єктами.

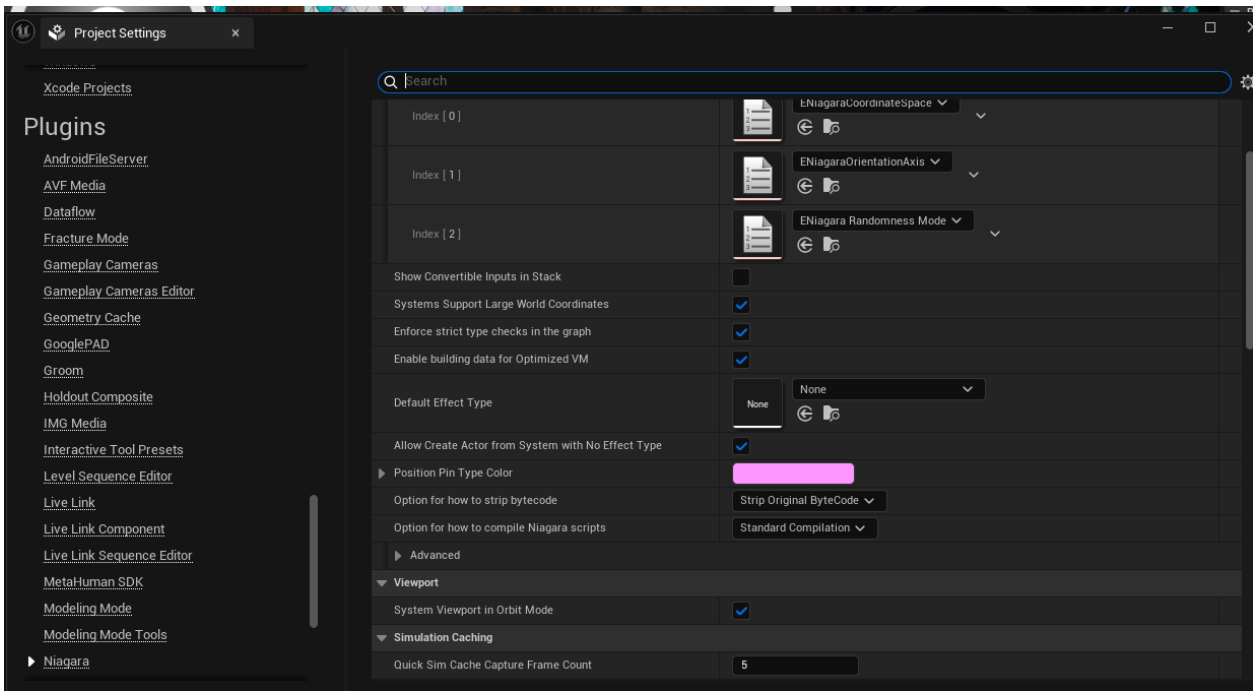


Рисунок 3.2 – Розділ Plugins

Імпорт ресурсів та налаштування початкових елементів. Для створення рівня завантажуються безкоштовні моделі замку з Unreal Marketplace. Моделі розміщуються у папці Levels/Environment, після чого додаються на рівень у редакторі для побудови базової структури замку.

Персонаж гравця імпортується до папки Blueprints/Player. Для нього налаштовуються базові анімації руху, атак і ухилень. Це дозволяє створити базову інтерактивність персонажа вже на початковому етапі.

Оптимізація робочого середовища. Для підвищення зручності роботи активується функція Live Coding, яка дозволяє тестувати зміни в коді без перезапуску редактора. У меню “Editor Preferences” налаштовується автоматичне збереження кожні 5 хвилин, що зменшує ризик втрати прогресу.

Налаштування Unreal Engine 5 – це перший крок до створення гри, який забезпечує готовність проєкту до інтеграції механік та рівнів. Завдяки правильній організації структури папок, активації плагінів і налаштуванню середовища розробки забезпечується ефективність і зручність роботи на наступних етапах.

3.2 Розробка механіки гри

Розробка механіки гри є ключовим етапом, який забезпечує інтерактивність ігрового процесу та створює основний геймплей [4]. У цьому розділі реалізовано основні механіки бою, управління витривалістю та поведінки ворогів. Основна увага приділяється поєднанню Blueprints та C++ для створення продуктивної та масштабованої системи.

Реалізація бойової системи. Основною механікою гри є бойова система, яка включає:

– атаки: виконуються базові удари з витратою витривалості;
– блокування: зменшує шкоду від атак, але також витрачає витривалість;
– ухилення: дозволяє уникати ударів, проте має обмеження через витрати витривалості.

Логіка цих механік реалізована через C++ для забезпечення високої продуктивності. Наприклад, метод PerformAttack() (рис. 3.3) у класі PlayerCharacter відповідає за виконання атак та перевірку доступної витривалості.

```
UCLASS()
    class YOURGAME_API APlayerCharacter : public ACharacter
{
    GENERATED_BODY()

public:
    APlayerCharacter();

protected:
    virtual void BeginPlay() override;

protected:
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Combat")
    float Stamina;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Combat")
    float AttackStaminaCost;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Combat")
    UAnimMontage* AttackMontage;

protected:
    void PerformAttack();

public:
    virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent) override;
};
```

Рисунок 3.3 – Реалізація методу PerfomAttack

Система витривалості. Витривалість є ключовим ресурсом у грі, який впливає на всі дії гравця. Її механіка реалізована через:

– зменшення витривалості: виконується при кожній дії, такій як атака, ухилення або блок;
– відновлення витривалості: починається, коли гравець не виконує жодних активних дій.

Алгоритм відновлення враховує швидкість регенерації, залежно від стану персонажа (наприклад, під час бою або поза ним).

Поведінка NPC і босів. Для ворогів використовуються AI Behavior Tree, що дозволяють створити реалістичну поведінку.

NPC:

– реалізовано патрулювання території та атаку при наближенні гравця;
– вороги на мостах змушують гравця бути обережним через ризик падіння.

боси:

– бос має унікальні фази бою, наприклад, посилення атак або додаткові здібності;

– логіка управління босом також базується на behavior tree для адаптації до дій гравця;

Фізика та взаємодія. Мости в рівні додатково ускладнюють проходження, оскільки при невдалому ухиленні гравець може впасти. Ця механіка реалізована через налаштування коллайдерів у редакторі та обробку подій падіння в C++.

При падінні викликається метод TakeFallDamage() (рис. 3.4), який зменшує здоров’я персонажа або призводить до завершення гри.

```
    protected:
// Мінімальна висота для вважання персонажа таким, що впав у прірву
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Fall Damage")
float DeathHeight;

// Метод для перевірки падіння
void TakeFallDamage();

// Поточна висота персонажа по осі Z
float InitialZ;

virtual void Tick(float DeltaTime) override;

// Прапорець для перевірки, чи персонаж перебуває у повітрі
bool bIsFalling;

protected:
virtual void BeginPlay() override;
```

Рисунок 3.4 – Реалізація механіки падіння у прірву

Розробка механіки гри дозволила створити динамічний ігровий процес, що поєднує тактичний підхід та ризиковані елементи [7]. Поєднання Blueprints та C++ забезпечило високу продуктивність і гнучкість у розробці бойової системи, витривалості та поведінки ворогів. Наступним етапом стане впровадження цих механік у побудову рівнів.

3.3 Розробка ігрових рівнів

Розробка ігрових рівнів – це один із ключових етапів створення гри, що визначає структуру, атмосферу та інтерактивність локацій. У цьому проєкті рівень представлений замком, який складається з двох основних зон, з’єднаних

небезпечними мостами. Рівень побудований таким чином, щоб забезпечити динамічний ігровий процес із поступовим підвищенням складності.

3.3.1 Побудова рівня замку з двома локаціями та мостами

Побудова рівня розпочалася з основного планування структури замку (рис. 3.5). У Unreal Engine 5 були використані безкоштовні моделі замку з FAB (колишній Unreal Marketplace), які інтегрувалися в ігровий рівень:

– перша зона: велика зала для початкового бою з npc, яка служить вступом до рівня;
– мости: небезпечна зона з обмеженим простором, де кожен неправильний рух може призвести до падіння;
– тронна зала: фінальна зона для бою з босом – тут передбачено відкритий простір, що дозволяє використовувати тактичний підхід у бою.



Рисунок 3.5 – Ігрова локація

3.3.2 Розміщення NPC, та босів

Розміщення NPC, тригерів для босів та інтерактивних об’єктів. Рівень включає в себе розташування NPC, тригерів і взаємодій.

NPC (рис. 3.6):

– розміщені на мостах і в початковій зоні для створення напруженості під час пересування;
– використовуються ai behavior tree для реалізації простих атакуючих дій.

Тригери для босів:

– бій із босом активується після перемоги над всіма npc у початковій зоні;
– тригери реалізовані через blueprint, що спрощує їхню інтеграцію в рівень.

Інтерактивні об’єкти: ліки та інші ресурси розташовані в складно доступних місцях, що стимулює дослідження рівня.



Рисунок 3.6 – Розташування NPC

3.3.3 Налаштування освітлення для створення атмосфери темного середньовіччя

Освітлення відіграє ключову роль у створенні атмосфери гри. Смолоскипи та вогнища створюють атмосферу темного середньовіччя. Використано систему Lumen для реалістичного освітлення та тіней.

Також було налаштовано динамічне освітлення для створення відчуття небезпеки та занедбаності. Додано туман і частинки пилу, які підсилюють атмосферу.

3.4 Розробка користувацького інтерфейсу

Користувацький інтерфейс (UI) надає гравцеві інформацію про стан персонажа та перебіг гри, передаючи ключові дані без зайвих відволікань. Шкала здоров’я персонажа показує поточний рівень здоров’я, зменшується під час отримання шкоди та відновлюється за допомогою лікувальних предметів. Шкала витривалості контролює витрати енергії на атаки, ухилення або блокування і автоматично відновлюється за відсутності дій. Під час битви з босом з’являється шкала його здоров’я (рис. 3.7), що відображає поточний стан супротивника.

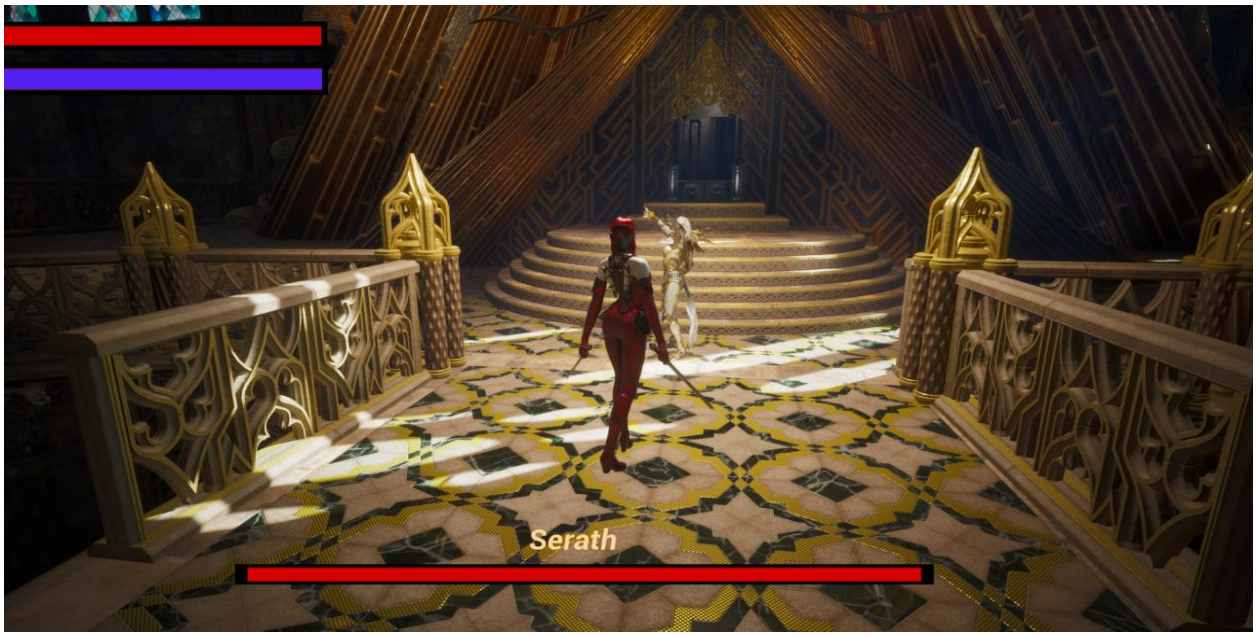


Рисунок 3.7 – Ігровий інтерфейс при проходженні боса

Усі елементи створені через Blueprint Widgets у Unreal Engine 5. Шкали здоров’я та витривалості пов’язані зі змінними персонажа, які динамічно оновлюються під час гри.

Розроблений користувацький інтерфейс дозволяє гравцю зручно контролювати стан персонажа та боса під час гри. Простий та зрозумілий дизайн підтримує загальну атмосферу гри, не перевантажуючи гравця зайвою інформацією.

4 ТЕСТУВАННЯ

4.1 Тестування ігрових механік

Тестування ігрових механік забезпечує перевірку їхньої коректної роботи та відповідності проєктним вимогам. Основний акцент зроблено на перевірці бойової системи, взаємодії з NPC та поведінки штучного інтелекту (ШІ). У цьому розділі представлено результати тестування ключових механік.

Бойова система. Тестування бойової системи включало перевірку наступних елементів:

– атаки: анімація атаки гравця активується після натискання відповідної кнопки (див. рис. 4.1) – витрата витривалості відбувається згідно з кодом, якщо витривалість гравця закінчена, атака не виконується; успішний удар по супротивнику зменшує його шкалу здоров’я;
– блокування: гравець може активувати блок, який зменшує отримувану шкоду (витривалість витрачається під час блокування атак супротивника);
– ухилення: персонаж успішно виконує ухилення після активації відповідної команди – ухилення також витрачає витривалість, і після її закінчення дія стає недоступною.

Було протестовано виведення повідомлень для гравця. Сповіщення про критичний рівень здоров’я або витривалості з’являються у відповідний момент. Повідомлення оновлюються в реальному часі та не відволікають гравця від ігрового процесу.

Поведінка NPC. Перевірка поведінки супротивників базувалася на наступних критеріях:

– реакція на гравця: npc розпізнають гравця, якщо він входить у їхню зону дії, і починають переслідування – супротивники атакують гравця лише тоді, коли він знаходиться в межах досяжності;

– виконання атак: npc успішно виконують анімації атак, завдаючи шкоди персонажу гравця – якщо гравець ухиляється, атака супротивника втрачає ціль;
– розташування: npc коректно розміщені на рівні – вони не застрягають у перешкодах і не сходять з мостів, що забезпечує стабільність геймплею.



Рисунок 4.1 – Анімація атаки при натисканні LMB

Результати тестування.

Механіка бою: усі базові функції працюють стабільно.

Проблем із витратою витривалості, шкалами здоров’я або анімаціями не виявлено.

Поведінка NPC: ШІ супротивників відповідає заданому сценарію. NPC адекватно реагують на дії гравця та виконують передбачені дії.

Тестування ігрових механік підтвердило їхню функціональність і відповідність проєктним вимогам. Бойова система, система витривалості та поведінка NPC працюють стабільно, що дозволяє забезпечити інтерактивність та можливість проходження (див. рис. 4.2). Незважаючи на обмежену кількість скриншотів, ключові аспекти механік продемонстровано у вигляді прикладів роботи системи атаки та взаємодії з NPC.



Рисунок 4.2 – Перемога над босом

4.2 Тестування UI та UX

Тестування користувацького інтерфейсу та взаємодії гравця з елементами гри показало його зручність та інтуїтивність. Шкала здоров’я коректно змінюється у відповідь на отримання шкоди, а при критичному рівні змінює колір на червоний, привертаючи увагу гравця. Шкала витривалості зменшується під час виконання атак, ухилень або блокувань, автоматично відновлюється у стані спокою, а при низькому рівні змінює колір на жовтий або червоний. Шкала здоров’я боса з’являється під час бою, точно відображаючи стан супротивника в реальному часі.

Елементи HUD розташовані так, щоб не заважати огляду ігрового середовища. Текстові сповіщення виводяться у верхній частині екрана, не відволікаючи від бою. Гравці легко орієнтуються в інтерфейсі без додаткових інструкцій, елементи реагують на зміну стану персонажа без затримок. Виконання дій, таких як атаки або ухилення, супроводжується миттєвими змінами в інтерфейсі, а сповіщення типу «Мало здоров’я» чи «Витривалість закінчилася» ефективно інформують про критичні стани.

Елементи UI працюють стабільно, шкали коректно оновлюються в реальному часі (рис. 4.3), а кольорові зміни успішно передають потрібну інформацію. Інтерфейс не перевантажує екран, дозволяючи гравцям приймати тактичні рішення без перешкод, і забезпечує зручність у використанні.



Рисунок 4.3 – Кінець гри при рівні здоров’я 0

Тестування UI та UX підтвердило, що користувацький інтерфейс гри відповідає проєктним вимогам. Елементи HUD та повідомлення успішно виконують свою функцію, забезпечуючи гравця необхідною інформацією без зайвого навантаження на екран. Зручність використання та інтуїтивність інтерфейсу дозволяють гравцю зосередитися на ігровому процесі, що позитивно впливає на загальне враження від гри.

4.3 Оптимізація

Оптимізація гри стала важливим етапом, спрямованим на забезпечення її стабільної роботи на різних системах і зниження навантаження на апаратні ресурси. Було виконано розбиття рівня на зони, які завантажуються лише під

час входу гравця, що зменшило кількість одночасно активних елементів. Великі об’єкти оптимізовано за допомогою LOD, щоб зменшити кількість полігонів залежно від відстані до камери. Непотрібні NPC, що знаходяться поза полем зору або за межами активної зони, було деактивовано. Для текстур використано стискання з оптимальним розширенням, а в ключових зонах динамічне освітлення активоване лише там, де це необхідно, решта освітлюється статичними джерелами.

Оновлення елементів, таких як шкали здоров’я та витривалості, відбувається тільки тоді, коли це потрібно. Поведінка NPC для менш важливих персонажів спрощена, що зменшило навантаження на обчислювальні ресурси. Помилки в системі витривалості, де витрати на атаки іноді не враховувалися, були виправлені. Також усунено проблеми з застряганням персонажа на мостах через неправильно налаштовані коллайдери та виправлено затримки в атаках NPC через конфлікт у черзі дій.

Під час тестування продуктивності визначено, що середній FPS стабільно підтримується на рівні 60 кадрів за секунду на сучасних ПК. Використання інструментів, таких як Unreal Engine Profiler, Stat Unit, Stat GPU і режим “Shader Complexity”, дозволило виявити та усунути «вузькі місця» в продуктивності, оптимізувати ресурси й покращити структуру матеріалів.

Результати оптимізації показали підвищення продуктивності гри на 20-30%, скорочення часу завантаження рівнів завдяки системі стрімінгу та усунення більшості помилок. Загальна стабільність гри значно покращилася, що сприяє створенню позитивного досвіду для гравця.

ВИСНОВКИ

У ході роботи над проєктом було створено ігровий рівень на рушії Unreal Engine 5, який демонструє основні механіки гри в жанрі Souls-like. Рівень складається з двох основних зон: великої зали для боїв із NPC та тронної зали для фінального протистояння з босом. Зони з'єднані небезпечними мостами, які додають елемент ризику та тактичної складності.

Впроваджено бойову систему, яка включає атаки, блокування та ухилення. Ці механіки базуються на витривалості, що додає стратегічного елементу до боїв. Система здоров'я для гравця та супротивників дозволяє гравцеві відслідковувати стан свого персонажа та ворогів у реальному часі. Реалізація цих механік виконана за допомогою C++ для забезпечення високої продуктивності та Blueprints для спрощення інтеграції елементів.

Користувацький інтерфейс було адаптовано до стилістики гри. Шкали здоров'я та витривалості, а також шкала здоров'я боса інтегровані в HUD, забезпечуючи гравця всією необхідною інформацією. Інтерфейс не перевантажує екран і забезпечує зручність під час гри.

Проведено тестування основних механік, таких як система бою, взаємодія з NPC та поведінка штучного інтелекту. Виявлені недоліки були виправлені, а гра оптимізована для стабільної роботи навіть у складних сценах. Для цього використовувались інструменти Unreal Engine, такі як профілювання продуктивності, LOD-система та стрімінг контенту.

Результати роботи демонструють, що Unreal Engine 5 є потужним інструментом для створення високоякісних ігор. Поєднання Blueprints та C++ забезпечило гнучкість і ефективність розробки, дозволивши реалізувати продуктивні ігрові системи.

Подальший розвиток гри має значний потенціал. Зокрема, можна розширити рівні, додати нові локації, супротивників і босів з унікальними механіками. Важливим кроком буде розвиток сюжетної лінії, включаючи кат-

сцени та діалоги, які поглиблять занурення гравця у світ гри. Також можливе впровадження мережевого режиму для кооперативного проходження чи PvP-битв.

Таким чином, виконана робота є лише початковим етапом у створенні гри, що має потенціал для подальшого вдосконалення, доповнення механік та розвитку сюжету. Результат роботи підтверджує перспективність ігрового проєкту як платформи для реалізації складних ідей у жанрі Souls-like.

ПЕРЕЛІК ПОСИЛАНЬ

1. Бендер Р. Основи розробки комп’ютерних ігор. Київ : Видавничий дім «Гельветика», 2020. 312 с.

2. Unreal Engine 5.5 Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-5-documentation (дата звернення: 25.09.2024).

3. Programming with C++. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/programming-with-cplusplus-in-unreal-engine (дата звернення: 15.09.2024).

4. Дяченко І. О., Коваленко А. П. Геймдев: основи програмування ігрових механік. Харків : Видавництво «Освіта», 2019. 256 с.

5. Смирнов М. П. Розробка сценаріїв для комп’ютерних ігор. Львів : «Літера ЛТД», 2021. 198 с.

6. Schell J. The Art of Game Design: A Book of Lenses. 3rd ed. Boca Raton : CRC Press, 2019. 610 p.

7. Salen K., Zimmerman E. Rules of Play: Game Design Fundamentals. Cambridge : MIT Press, 2004. 688 p.

ДОДАТОК А

Основний клас PlayerCharacter.h

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "PlayerCharacter.generated.h"

UCLASS()
class YOURGAME_API APlayerCharacter : public ACharacter
{
   GENERATED_BODY()

public:
   APlayerCharacter();

protected:
   // Здоров’я персонажа
   UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Stats")
   float Health;

   // Максимальне здоров’я
   UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Stats")
   float MaxHealth;

   // Витривалість персонажа
   UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Stats")
   float Stamina;
```

```
    // Максимальна витривалість
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Stats")
    float MaxStamina;

    // Метод для атаки
    void PerformAttack();

    // Метод для падіння
    void TakeFallDamage();

    virtual void Tick(float DeltaTime) override;

protected:
    virtual void BeginPlay() override;
};
```

ДОДАТОК Б

Реалізація методів у PlayerCharacter.cpp

```
#include "PlayerCharacter.h"

APlayerCharacter::APlayerCharacter()
{
   PrimaryActorTick.bCanEverTick = true;

   MaxHealth = 100.0f;
   Health = MaxHealth;

   MaxStamina = 100.0f;
   Stamina = MaxStamina;
}

void APlayerCharacter::BeginPlay()
{
   Super::BeginPlay();
}

void APlayerCharacter::Tick(float DeltaTime)
{
   Super::Tick(DeltaTime);
}

void APlayerCharacter::PerformAttack()
{
   if (Stamina > 20.0f)
```

```
    {
        Stamina -= 20.0f;
        UE_LOG(LogTemp, Log, TEXT("Attack performed"));
    }
    else
    {
        UE_LOG(LogTemp, Warning, TEXT("Not enough stamina to attack"));
    }
}

void APlayerCharacter::TakeFallDamage()
{
    Health -= 50.0f;
    if (Health <= 0.0f)
    {
        UE_LOG(LogTemp, Warning, TEXT("Player has died!"));
        Destroy();
    }
}
```

ДОДАТОК В

Код класу GameManager.h

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "GameManager.generated.h"

UCLASS()
class YOURGAME_API AGameManager : public AActor
{
   GENERATED_BODY()

public:
   AGameManager();

protected:
   // Поточний стан гри
   UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Game State")
   bool bIsGameOver;

   // Активований тригер для боса
   UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Triggers")
   bool bBossTriggerActivated;

   // Лічильник супротивників
   UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Game State")
   int32 RemainingEnemies;
```

```
    // Метод для завершення гри
    void EndGame(bool bPlayerWon);

    // Метод для активації бою з босом
    void ActivateBossTrigger();

public:
    virtual void Tick(float DeltaTime) override;

protected:
    virtual void BeginPlay() override;

    // Метод для віднімання супротивників
    void EnemyDefeated();
};
```

ДОДАТОК Г

Реалізація методів у GameManager.cpp

```
#include "GameManager.h"
#include "Kismet/GameplayStatics.h"

AGameManager::AGameManager()
{
   PrimaryActorTick.bCanEverTick = true;

   // Ініціалізація значень
   bIsGameOver = false;
   bBossTriggerActivated = false;
   RemainingEnemies = 10; // Початкова кількість супротивників
}

void AGameManager::BeginPlay()
{
   Super::BeginPlay();

   UE_LOG(LogTemp, Log, TEXT("GameManager initialized. Remaining enemies: %d"), RemainingEnemies);
}

void AGameManager::Tick(float DeltaTime)
{
   Super::Tick(DeltaTime);

   // Перевірка, чи всі супротивники переможені
```

```
    if (RemainingEnemies <= 0 && !bBossTriggerActivated)
    {
        ActivateBossTrigger();
    }
}

void AGameManager::EnemyDefeated()
{
    if (RemainingEnemies > 0)
    {
        RemainingEnemies--;
        UE_LOG(LogTemp, Log, TEXT("Enemy defeated! Remaining enemies: %d"), RemainingEnemies);

        // Якщо всі супротивники переможені
        if (RemainingEnemies <= 0)
        {
            UE_LOG(LogTemp, Log, TEXT("All enemies defeated. Activating boss trigger."));
        }
    }
}

void AGameManager::ActivateBossTrigger()
{
    bBossTriggerActivated = true;
    UE_LOG(LogTemp, Log, TEXT("Boss trigger activated!"));

    // Логіка активації бою з босом
    // Наприклад, спавн боса
```

```
}

void AGameManager::EndGame(bool bPlayerWon)
{
    bIsGameOver = true;

    if (bPlayerWon)
    {
        UE_LOG(LogTemp, Log, TEXT("Player has won the game!"));
    }
    else
    {
        UE_LOG(LogTemp, Log, TEXT("Player has lost the game."));
    }

    // Завершення гри: наприклад, завантаження головного меню
    UGameplayStatics::OpenLevel(this, FName("MainMenu"));
}
```