НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних Наук
(назва кафедри)

_____ Голуб Б. Л.
(підпис) (ПІБ)

"___" _____ 20 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«Програмна система моделювання віртуального бою на прикладі гри в жанрі Action RPG»

Спеціальність 122 – «Комп'ютерні науки»

Гарант освітньої програми
д. пед. н. _____ Глазунова О. Г.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
д. ф. _____ Назаренко В. А.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Бігун Р.Л.
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
____Комп'ютерних Наук_____
_____ _____
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"____" _____20_____р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту

_____Бігун Роман Леонідович_____
(прізвище, ім'я, по батькові)

Спеціальність 122 – «Комп'ютерні науки»
Тема бакалаврської кваліфікаційної роботи __Програмна система моделювання віртуального бою на прикладі гри в жанрі Action RPG
Затверджена наказом ректора НУБіП України від 16.12.2024 № 2246 "С"
Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи_____
Розробка та реалізація бойової системи для ігрового застосунку на базі ігрового рушія Unreal Engine 5
_____
_____

Перелік питань, які потрібно розробити:

1. Аналіз предметної області та вимог.
2. Проектування інформаційної бази та архітектури.
3. Реалізація програмних модулів.
4. Рекомендації щодо впровадження та експлуатації.

Дата видачі завдання "_____"_____20___ р.

Керівник бакалаврської кваліфікаційної роботи _____Назаренко В.А.____
_____
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ _____
(підпис) (прізвище та ініціали студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання бакалаврської роботи	Строк виконання етапів бакалаврської роботи	Примітка
1	Видача завдання	16.12.2024	
2	Аналіз предметної області	Лютий 2025	
3	Моделювання предметної області	Лютий 2025	
4	Поставновка завдання	Лютий 2025	
5	Проєктування системи	Лютий 2025	
6	Розробка системи	Березень 2025	
7	Тестування системи	Травень 2025	
8	Оформлення записки	Травень 2025	
9	Проходження нормо контролю	24-25 травня 2025	
10	Перевірка на плагіат	26-27 травня 2025	
11	Попередній захист	2-3 червня 2025	
12	Захист	9-12 червня 2025	

Студент _____ ___Бігун Р.Л._____
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ Назаренко В.А.
(підпис) (прізвище таініціали)

АНОТАЦІЯ

У бакалаврській кваліфікаційній роботі реалізовано комп’ютерну систему бойової взаємодії для однокористувацької гри в жанрі ActionRPG. Система керує станами персонажа, обробляє логіку таргетування, виконання атак та активних здібностей із урахуванням умов, таких як кулдауни, дистанція та стан супротивника.

Мета — створення гнучкого та модульного компонента бойової системи з інтеграцією у загальну архітектуру гри без зайвої складності для дизайнера. Основу проєкту становить рушій Unreal Engine 5, з використанням Blueprint-сценаріїв, делегатів та компонентної архітектури.

У роботі описано структуру класів, механіку взаємодії між елементами системи, способи оптимізації обробки подій та можливості масштабування. Наведено діаграми, що ілюструють логіку дій персонажа, цільову систему та приклади модифікації поведінки без втручання в базовий код. Система може бути адаптована під інші проєкти, які вимагають простого у використанні бойового інтерфейсу.

ABSTRACT

This bachelor’s thesis presents the development of a combat interaction system for a single-player ActionRPG game. The system manages character states, handles targeting logic, attack execution, and active abilities based on conditions such as cooldowns, range, and enemy status.

The goal is to design a flexible, modular combat component that integrates smoothly into the game’s overall architecture without overcomplicating the designer’s workflow. The project is built using Unreal Engine 5, leveraging Blueprint scripting, event dispatchers, and component-based architecture.

The document outlines the class structure, mechanics of interaction between system elements, event-handling optimizations, and scalability potential. It includes diagrams illustrating character actions, target selection logic, and examples of modifying

behavior without altering core code. The system is suitable for reuse in other projects that require an accessible and extensible combat interface.

Зміст

ВСТУП

Відеоігри жанру Action Role-Playing Game (ActionRPG) займають особливе місце в індустрії інтерактивних розваг, поєднуючи динамічний геймплей екшн-ігор з глибокими RPG-механіками розвитку персонажа. Центральним елементом цих ігор є бойова система, яка визначає не лише

якість ігрового досвіду, але й комерційну успішність проекту. Якісна бойова система може самостійно забезпечити успішність гри, а погано виконана - стати причиною комерційного провалу, оскільки саме бойова система становить основний геймплейний цикл у ActionRPG відеоіграх.

Актуальність теми дослідження обумовлена стрімким розвитком індустрії відеоігор та зростаючими вимогами гравців до якості та інноваційності бойових систем.

За останні десятиліття ActionRPG еволюціонували від простих аркадних бійок до складних систем з реалістичною фізикою, штучним інтелектом та багатошаровими механіками взаємодії.

Сучасні бойові системи в ActionRPG повинні забезпечувати баланс між доступністю для новачків та глибиною для досвідчених гравців, інтегруючи елементи прогресії персонажа, тактичного планування та рефлексів реального часу. Це створює унікальні виклики для розробників, які мають враховувати технічні обмеження, користувацький досвід та ринкові тенденції.

Мета роботи полягає в комплексному аналізі бойових систем у відеоіграх жанру ActionRPG, дослідженні їх структурних компонентів, принципів проектування та методів імплементації.

Сучасні технології розробки ігор, зокрема игровий рушій Unreal Engine 5, надають розробникам потужні інструменти для створення складних та інтерактивних бойових систем. Однак проектування та реалізація ефективної бойової системи потребує глибокого розуміння як технічних аспектів програмування, так і геймдизайну.

Апробація. Теза до бакалаврської роботи, доповідь на тему «КОНЦЕПЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ СИМУЛЯЦІЇ БОЮ НА ПРИКЛАДІ ГРИ В ЖАНРІ ACTIONRPG» була опублікована на науково-практичній конференції «Теоретичні та прикладні аспекти розробки комп’ютерних систем 2025» (23 квітня 2025 року)

РОЗДІЛ 1

СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка задачі

Створення ефективної бойової системи для Action RPG є комплексним завданням, що вимагає глибокого розуміння специфіки жанру та його ключових механік. Для успішної реалізації такої системи необхідно спочатку чітко визначити сам жанр та його ключові особливості, а потім перейти до детального розгляду компонентів, що формують бойовий досвід.

Визначення Action RPG та її ключові елементи:

Action RPG (ARPG) – це піджанр рольових відеоігор, який поєднує елементи традиційних рольових ігор (розвиток персонажа, сюжет, вибір, інвентар) з елементами ігор жанру "екшн" (бій у реальному часі, орієнтований на рефлекси та навички гравця). На відміну від класичних RPG, де бій часто відбувається покроково або з активною паузою, в ARPG гравцеві необхідно активно керувати персонажем під час сутичок, ухилятися від атак, блокувати, використовувати здібності та комбінації ударів у динамічному середовищі.

Розробка бойової системи для такого жанру потребує врахування специфічних вимог до взаємодії гравця з ігровим світом, що створює унікальні технічні та дизайнерські виклики. Необхідно забезпечити плавність анімацій, точність хітбоксів, збалансованість механік та інтуїтивність управління.

Ключові елементи Action RPG, які безпосередньо впливають на бойову систему та визначають вимоги до її реалізації:

● Бій у реальному часі: Це найбільш визначальна характеристика, що створює потребу в оптимізованій системі обробки подій, точному таймінгу та швидкодії алгоритмів. Усі дії гравця та ворогів відбуваються одночасно, вимагаючи швидкої реакції та тактичного мислення в динаміці.

● Розвиток персонажа: Система прогресії повинна бути тісно інтегрована з бойовими механіками, забезпечуючи відчутний вплив покращень на ефективність у бою. Гравці покращують свого персонажа через отримання досвіду, підвищення рівня, розподіл очок характеристик, вивчення нових здібностей та екіпірування кращого спорядження.

● Система інвентарю та спорядження: Реалізація вимагає гнучкої архітектури для обробки різноманітних предметів з унікальними властивостями. Зброя, обладунки, аксесуари та витратні предмети відіграють критичну роль у визначенні бойових можливостей персонажа, надаючи унікальні здібності, бонуси до характеристик або змінюючи стиль бою. Система інвентраю заслуговує окрему дослідницьку роботу через комплексніть цієї теми, у цій роботі ми не будемо зусереджувати на ній увагу.

Класифікація бойових систем та технічні вимоги:

Бойові системи в Action RPG можуть бути класифіковані за різними критеріями, що визначають архітектурні рішення та технічні підходи до їх реалізації.

Темпоральні характеристики бойових систем:

Хоча Action RPG за визначенням орієнтовані на бій у реальному часі, розуміння альтернативних підходів допомагає краще усвідомити специфічні вимоги до реалізації.

● Бій у реальному часі (Real-time Combat): Стандарт для Action RPG, що вимагає безперервної обробки вводу, оновлення стану гри та рендерингу. Гравці постійно контролюють свого персонажа, виконуючи атаки, рух, ухилення та використання здібностей без перерв на "ходи". Успіх залежить від рефлексів, позиціонування, таймінгу та вміння швидко приймати рішення. Технічна реалізація потребує оптимізованих алгоритмів колізій, системи станів персонажа та ефективної обробки множинних об'єктів. Приклади: Diablo, Dark Souls, Monster Hunter.

● Бій з активною паузою (Real-time with Pause): Гібридний підхід, що дозволяє тимчасово призупиняти дію для тактичного планування. Реалізація

вимагає збереження стану всіх об'єктів та можливості їх модифікації в режимі паузи. Хоча основний бій відбувається в реальному часі, елемент тактичної паузи додає глибини та знижує вимоги до реакції гравця.

● Покроковий бій (Turn-based Combat): Протилежність реального часу, що служить контрастом для розуміння специфіки ARPG. Дії відбуваються по черзі, що спрощує логіку обробки, але не відповідає концепції Action RPG.

Баланс між навичками гравця та характеристиками персонажа:

Цей критерій визначає архітектурні рішення щодо обробки дій гравця та розрахунку результатів бою.

● Орієнтовані на навички гравця (Player Skill-Oriented): Системи, де технічна реалізація повинна забезпечувати максимальну точність та відзивність. Успіх значною мірою залежить від вміння гравця, що вимагає точних хітбоксів, мінімальної затримки вводу та складних систем анімацій. Навіть персонаж з низькими характеристиками може перемогти за умови майстерної гри. Приклади: Dark Souls, Sekiro, Monster Hunter.

● Орієнтовані на статистику персонажа (Character Stat-Oriented): Системи з акцентом на числових розрахунках та процедурній генерації контенту. Реалізація фокусується на складних формулах шкоди, системах модифікаторів та генерації предметів. Успіх більше залежить від правильного розвитку персонажа та якості спорядження. Приклади: Diablo, Path of Exile, Borderlands.

За перспективою камери:

- Ізометричні ActionRPG (Diablo series, Path of Exile) - камера розташована під кутом, забезпечуючи огляд великої області.
- ActionRPG від третьої особи (The Witcher 3, Dark Souls) - камера слідує за персонажем ззаду.
- ActionRPG від першої особи (The Elder Scrolls series) - гравець бачить світ очима персонажа.

За складністю бойової системи:

- Спрощені системи - базові атаки, блокування та ухилення.

- Середньої складності - комбо-атаки, різні типи зброї, базова магія.
- Складні системи - розширені комбо, численні типи атак, складні магічні системи.

За фокусом геймплея:

- Hack and slash - акцент на масовому знищенні ворогів
- Souls-like - високий рівень складності, точність таймінгу
- Гібридні - баланс між екшеном та RPG-елементами

Компонентна архітектура бойової системи:

Для ефективної реалізації бойова система повинна бути розбита на взаємопов'язані модулі, кожен з яких відповідає за специфічний аспект бойової взаємодії.

Модуль атак та комбо-систем:

● Базові атаки: Фундаментальні дії, що вимагають системи обробки вводу, анімаційних переходів та розрахунку шкоди. Розділяються на легкі (швидкі, менша шкода) та важкі (повільніші, більша шкода, пробивання блоку) з різними характеристиками таймінгу та ефектів.

● Комбо-системи: Складні послідовності дій, що потребують системи відстеження стану, таймерів та умов переходу. Реалізація включає валідацію послідовностей, розрахунок бонусів та обробку анімаційних ланцюжків. Комбо можуть наносити підвищену шкоду, мати унікальні ефекти або змінювати позицію ворога.

● Спеціальні атаки: Механіки накопичення енергії, зарядових атак та атак по області, що вимагають додаткових систем ресурсів та просторових розрахунків.

Модуль захисту та уникнення:

● Система ухилення: Реалізація кадрів невразливості, розрахунку дистанції переміщення та інтеграції з системою витривалості. Вимагає точного таймінгу та координації з анімаційною системою.

● Система блокування та парірування: Механіки зменшення шкоди, розрахунку кутів блокування та створення вікон для контратак. Парірування

потребує додаткової логіки для визначення ідеального таймінгу та обробки результуючих ефектів.

Модуль руху та позиціонування:

● Система пересування: Базовий рух, спринт, стрибки з інтеграцією до системи фізики та навігаційної сітки.

● Тактичне позиціонування: Алгоритми для визначення оптимальних позицій, обробки зіткнень та взаємодії з навколишнім середовищем.

Модуль здібностей та ресурсів:

● Активні здібності: Система перезарядки, споживання ресурсів та обробки ефектів здібностей.

● Система предметів: Інвентар, використання витратних предметів та тимчасові ефекти.

Модуль обробки шкоди та станів:

● Розрахунок шкоди: Формули для різних типів шкоди, критичних ударів та опору.

● Система станів: Обробка тимчасових ефектів, їх взаємодії та візуалізації.

● Управління ресурсами: Здоров'я, витривалість та інші ресурси з системами регенерації та витрати.

1.1.2 Покращення бойової системи згідно зі стандартами ігрової індустрії та досягнення "Feels Good" ефекту

Розробка бойової системи є одним із найскладніших та найвідповідальніших аспектів створення ігрового застосунку, оскільки саме вона значною мірою визначає залученість гравця, його задоволення від ігрового процесу та загальне "відчуття" гри. Успішна бойова система не лише функціонально коректна, але й інтуїтивно зрозуміла, захоплююча та надає гравцеві відчуття контролю та впливу. Цей підрозділ детально розглядає потенційні напрямки покращення розробленої бойової системи відповідно до сучасних стандартів ігрової індустрії, а також методи досягнення ефекту "Feels Good" – суб'єктивного, але критично важливого відчуття задоволення від взаємодії з ігровими механіками.

1.1.3 Аналіз поточного стану та стандарти індустрії

Розроблена бойова система на Unreal Engine 5 вже містить базові та функціональні механізми, такі як управління персонажами та ворогами, система здібностей, розрахунок шкоди та управління станами. Вона забезпечує логічне виконання бойових дій та взаємодію між основними елементами. Однак, для перетворення функціональної системи на винятковий ігровий досвід, необхідно вийти за рамки базової коректності та зосебитись на аспектах, що формують "відчуття" гри.

Сучасні стандарти ігрової індустрії для бойових систем значно еволюціонували. Якщо раніше достатньо було лише функціональності, то сьогодні від ігрових механік вимагається високий рівень відгуку, глибини, балансу та чіткості. Ефект "Feels Good" є інтегральною сукупністю цих елементів, що створює позитивні емоції у гравця та заохочує його до подальшої взаємодії. Цей ефект досягається через:

Миттєвий та чіткий відгук на дії гравця: Кожна дія гравця, від натискання кнопки до активації здібності, повинна мати негайний та зрозумілий візуальний, аудіальний та, за можливості, тактильний відгук. Це створює відчуття прямого контролю та причинно-наслідкового зв'язку.

Відчуття впливу: Дії гравця повинні відчуватися значущими та мати видимі, відчутні наслідки в ігровому світі. Це стосується як нанесення шкоди ворогам, так і зміни стану ігрового середовища.

Баланс та справедливість: Система має бути справедливою, передбачуваною та надавати гравцеві чіткі правила та можливості для навчання та вдосконалення своїх навичок. Невдачі повинні сприйматися як результат власних помилок, а не як несправедливість системи.

Глибина та різноманітність: Наявність достатньої кількості механік, що дозволяють гравцеві експериментувати з різними стратегіями, комбінаціями здібностей та знаходити власні ефективні підходи до бою. Це забезпечує довготривалий інтерес та реграбельність.

Полірування (Polish): Дрібні деталі, що не є критичними для функціональності, але значно покращують загальне сприйняття гри. Це може бути якість анімацій, плавність переходів, дрібні візуальні ефекти, що підкреслюють дії, та загальна увага до деталей, яка робить ігровий процес приємним та професійним.

1.1.4 Глибина та розширення механік

Для підвищення залученості гравця та забезпечення довготривалого інтересу до бойової системи, необхідно розширити її механічну глибину. Це дозволить гравцям експериментувати з різними стратегіями, адаптуватися до нових викликів та відчувати прогрес у своїх навичках та розумінні системи.

Розширення системи здібностей:

Дерева талантів та прогресії(Рис. 1.1.0): Замість фіксованого набору здібностей, можна впровадити розгалужену систему дерева талантів або систему прогресії здібностей. Це дозволить гравцеві вибирати та покращувати здібності в міру підвищення рівня або виконання певних ігрових досягнень. Кожна здібність може мати кілька рангів покращення, що збільшують її базову ефективність (наприклад, збільшення шкоди, зменшення вартості ресурсів, скорочення кулдауну) або додають нові, унікальні властивості (наприклад, "Вогняна куля" починає підпалювати цілі, або "Льодяний щит" тепер також заморожує ворогів, що атакують). Це надає гравцеві відчуття власності над розвитком свого персонажа та заохочує до стратегічного планування.

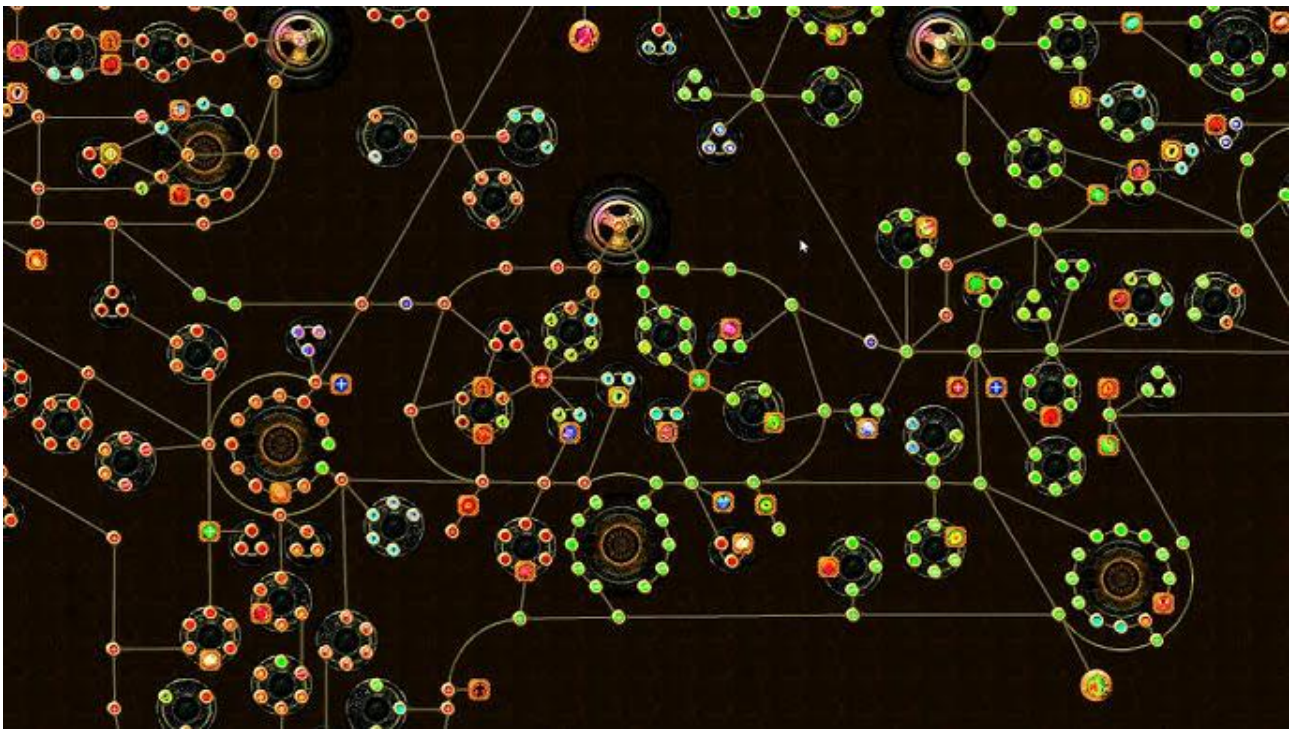

Рис. 1.1.0 Дерево талантів Path Of Exile

Синергії між здібностями: Розробка здібностей, які взаємодіють одна з одною, створюючи потужніші та візуально ефектніші комбінації. Це може бути реалізовано через систему статусних ефектів: одна здібність може накладати певний ефект (наприклад, "мокрий" або "вразливий до магії"), а інша здібність – завдавати додаткової шкоди або мати посилений ефект по цілях, які перебувають під дією цього статусу. Це заохочує гравців до креативного мислення, пошуку оптимальних послідовностей дій та розуміння механік взаємодії.

Модифікатори здібностей: Впровадження механізмів, що дозволяють гравцям модифікувати свої здібності за допомогою рун, гемів, артефактів або інших ігрових предметів. Це може змінювати тип шкоди, додавати вторинні ефекти стану, змінювати область дії здібності (наприклад, перетворювати одиночну ціль на AOE), або навіть змінювати візуальне представлення здібності. Це додає ще один шар кастомізації та реграбельності.

Впровадження розширеної системи статусних ефектів:

Поточна система управління станами може бути значно розширена для створення більш динамічних та тактичних бойових сценаріїв. Окрім базових ефектів (отрута, оглушення), можна додати широкий спектр:

- Баффів: Тимчасове збільшення характеристик (наприклад, "Прискорення", "Підвищення захисту", "Збільшення шкоди").

- Дебаффів: Тимчасове зменшення характеристик (наприклад, "Сповільнення", "Ослаблення", "Кровотеча", "Горіння", "Заморозка", "Паніка").

Кожен ефект повинен мати чітко визначені параметри: тривалість, періодичність дії (якщо ефект є періодичним), сила впливу, можливість диспелу (зняття ефекту), а також умови накладання та зняття. Візуальне та аудіальне представлення ефектів (іконки на UI, частинки над головою персонажа, зміна кольору моделі, специфічні звуки) є критично важливим для їхньої чіткості та розуміння гравцем.

Система екіпіровки та прогресії:

Інтеграція розгалуженої системи інвентарю та екіпіровки, яка безпосередньо впливає на бойові характеристики персонажа. Різні типи зброї (мечі, луки, посохи) та броні (легка, середня, важка) повинні надавати унікальні бонуси до шкоди, захисту, швидкості атаки, критичного шансу, або навіть відкривати нові здібності та пасивні ефекти.

Предмети можуть мати різні рівні рідкості (звичайні, рідкісні, епічні, легендарні), що впливає на їхні характеристики та цінність.

Прогресія персонажа через підвищення рівня та отримання нового, більш потужного спорядження повинна відчутно впливати на його бойову ефективність, надаючи гравцеві відчуття розвитку та посилення. Це створює постійний цикл "грай-отримуй нагороду-ставай сильнішим".

Комбо-системи та послідовності атак:

Для боїв у реальному часі, впровадження комбо-систем, де послідовне та/або таймоване виконання певних атак або здібностей призводить до потужніших, візуально ефектніших або з унікальними властивостями комбінацій. Це може бути реалізовано через:

- Таймований ввід: Натискання кнопок у певний часовий проміжок після попередньої дії.
- Послідовний ввід: Виконання дій у заданій послідовності.
- Ресурсні комбо: Використання певних ресурсів або станів для активації комбо.

Це підвищує вимоги до навичок гравця, заохочує до вивчення оптимальних послідовностей дій та надає більше можливостей для вираження його бойового стилю.

Система уразливостей та опорів:

Впровадження різних типів шкоди (наприклад, фізична, магічна, вогонь, лід, блискавка, отрута, світло, темрява) та відповідних уразливостей/опорів у ворогів. Це заохочує гравців до стратегічного вибору здібностей та зброї проти різних типів противників, створюючи тактичну глибину бою.

- Візуальне представлення уразливостей (наприклад, вогняні ефекти на крижаному монстрі, що отримує підвищену шкоду) та опорів (наприклад, іскри від удару по броньованому ворогу, що отримує зменшену шкоду) є важливим для чіткості та інтуїтивного розуміння механік гравцем.

1.1.5 Покращення відгуку та "Feels Good" ефекту (Player Feedback & Polish)

"Feels Good" ефект є сукупністю дрібних, але критично важливих деталей, які роблять взаємодію з системою приємною, інтуїтивною та задовільною. Це досягається через комплексний та багатошаровий відгук на дії гравця.

Візуальний відгук (Visual Feedback):

Хітмаркери та ефекти попадання: Чіткі візуальні індикатори, що показують, коли атака гравця потрапила в ціль. Це можуть бути спалахи, іскри, бризки крові, відколи від броні, або спеціальні, більш виразні ефекти при критичному ударі чи пробитті захисту. Ці ефекти повинні бути миттєвими та добре помітними.

- Анімації: Плавні, реалістичні та добре таймовані анімації атак, здібностей, отримання шкоди та смерті. Анімації повинні мати "вагу" та відчуття впливу, передаючи силу дії. Використання анімаційних блендів для плавних переходів між анімаціями, інверсної кінематики (IK) для адаптації

кінцівок персонажа до оточення (наприклад, при стоянні на нерівній поверхні) та анімаційних шарів для одночасного відтворення різних анімацій (наприклад, ходьба і атака рукою).

- Камера: Динамічні зміни поведінки камери під час бою. Це може включати легкий шейк камери (Camera Shake) при потужних ударах або вибухах, короткочасний зум на ціль при активації ультимативної здібності, або автоматичний фокус на гравцеві/ворогу в ключові моменти бою. Це додає кінематографічності та відчуття епічності.

- Пост-процесинг: Застосування ефектів пост-процесингу (наприклад, короткочасний блур екрану, зміна кольорової корекції, віньєтування, ефект хроматичної аберації) при отриманні або нанесенні значної шкоди. Це створює додатковий візуальний акцент на критичних подіях.

Аудіальний відгук (Audio Feedback):

Звуки ударів та заклинань: Високоякісні та різноманітні звукові ефекти для кожної атаки, здібності, попадання та отримання шкоди. Звуки повинні мати "вагу" та чітко передавати тип дії (наприклад, дзвінкий звук меча, глухий удар булави, шиплячий звук вогняного заклинання). Використання багатошарових звуків (layering) для створення більш насиченого аудіо-досвіду.

- Динамічна музика: Зміна музичного супроводу залежно від інтенсивності бою. Це може включати перехід від фонової музики до більш напруженої бойової теми при виявленні ворога, або зміну інструментів/ритму при переході боса в нову фазу.

- Голосові репліки: Короткі голосові репліки персонажів або ворогів при активації здібностей, отриманні значної шкоди, перемозі або поразці. Це додає персоналізації та емоційного забарвлення.

Тактильний відгук (Haptic Feedback):

- Вібрація геймпада: Використання вібрації геймпада для посилення відчуття ударів, активації потужних здібностей, отримання шкоди або вибухів. Різні типи вібрації (різна інтенсивність, частота, тривалість) можуть передавати різні події, посилюючи занурення гравця.

- Таймінг та "Game Feel":

"Hit Stop" (Короткочасна пауза): Дуже короткочасна (зазвичай 0.05-0.1 секунди) пауза в ігровому часі при нанесенні удару. Це створює ілюзію більшого впливу та "ваги" атаки, підкреслюючи момент контакту.

- "Screen Shake" (Тремтіння екрану): Легке, але відчутне тремтіння екрану при потужних ударах, вибухах або активації ультимативних здібностей, що підкреслює їхню силу та вплив на оточення.

- "Juice": Загальний термін, що охоплює всі ці дрібні елементи відгуку (анімації, звуки, ефекти, таймінг), які роблять гру більш динамічною, інтерактивною та приємною. Досягнення "Juice" вимагає ітеративного полірування, уваги до деталей та тонкого налаштування всіх елементів відгуку.

- "Coyote Time" та "Input Buffering": Це механізми, що покращують контроль гравця, роблячи його більш "пробачаючим" та інтуїтивним. "Coyote Time" дозволяє гравцеві виконати дію (наприклад, стрибок) протягом дуже короткого часу після того, як він покинув платформу, що

запобігає відчуттю "несправедливості". "Input Buffering" дозволяє системі запам'ятати ввід гравця на короткий час, щоб дія спрацювала, навіть якщо гравець натиснув кнопку трохи раніше, ніж це стало можливим, що підвищує відчуття відгуку.

Чіткість (Clarity):

- Відображення шкоди (Floating Combat Text): Чітке відображення чисел шкоди (та її типу), що вилітають над головами цілей. Це може включати різні кольори для різних типів шкоди (наприклад, червоний для фізичної, синій для магічної, жовтий для критичної), а також анімації вильоту чисел, що підкреслюють їхню важливість.

- Індикатори кулдаунів та ресурсів: Чіткі та зрозумілі візуальні індикатори на UI, що показують поточний рівень ресурсів (HP, Mana, витривалість) та час до перезарядки здібностей. Це можуть бути прогрес-бари, іконки, що затемнюються, або таймери зворотного відліку.

- Візуалізація зон дії здібностей: Візуальне підсвічування області, на яку вплине здібність, перед її активацією (наприклад, коло на землі для AOE-заклинання, конус для конусної атаки). Це допомагає гравцеві точно прицілюватися та розуміти наслідки своїх дій.

- Статусні іконки: Чіткі та інтуїтивно зрозумілі іконки над головами персонажів або на UI, що відображають активні баффи та дебаффи, їхню тривалість та силу.

Висновок до розділу 1.1

Досягнення "Feels Good" ефекту та відповідність стандартам ігрової індустрії вимагає комплексного підходу, що поєднує глибоке розуміння

ігрового дизайну, психології гравця та технічної реалізації. Розширення механічної глибини бойової системи, ретельне полірування відгуку (візуального, аудіального, тактильного), збалансована прогресія та вдосконалений штучний інтелект ворогів є ключовими напрямками для подальшого розвитку. Впровадження цих рекомендацій дозволить трансформувати функціональну бойову систему у захоплюючий та пам'ятний ігровий досвід, що є кінцевою метою будь-якої успішної гри.

1.2 Огляд інформаційних джерел та існуючих рішень

Для розуміння сучасних трендів та найкращих практик у розробці бойових систем, проаналізуємо декілька знакових ActionRPG ігор:

The Witcher 3: Wild Hunt

Рис.1.1 The Witcher 3 бій

"Відьмак 3" демонструє складну бойову систему, що поєднує фехтування на мечах з магічними знаками та алхімією.

Ключові особливості бойової системи:

- Система знаків: П'ять магічних знаків (Aard, Igni, Quen, Yrden, Axii) з унікальними ефектами.
- Механіка паррування та ухилення: Точний тайминг для ефективного захисту.
- Система мутацій: Глибока кастомізація здібностей персонажа.
- Підготовка до бою: Використання олій та бомб для підвищення ефективності в бою.

Технічна реалізація:

- Використання власного движка REDengine 3.
- Складна система анімацій з плавними переходами.
- Реалістична фізика взаємодії об'єктів.

Переваги:

- Глибока тактична складова.
- Різноманітність підходів до бою.
- Високий рівень імерсії.

Недоліки:

- Складність для нових гравців.
- Необхідність постійної підготовки.

Dark Souls серія

Рис.1.2 Dark Souls 3 бій

Серія Dark Souls відома своєю унікальною філософією бойової системи, що базується на точності, терпінні та вивченні патернів ворогів.

Ключові особливості:

- Система витривалості: Кожна дія (атака, блок, ухилення) споживає витривалість.
- Важливість тайминга: Успіх залежить від точного виконання дій у правильний момент.
- Різноманітність зброї: Кожен тип зброї має унікальні характеристики та стиль бою.
- Система босів: Складні противники з унікальними механіками.

Технічні аспекти:

- Точна система колізій та хітбоксів
- Складна система анімацій з мінімальним input lag
- Ефективне використання кадрів невразливості

Вплив на індустрію:

- Популяризація "souls-like" піджанру
- Демонстрація важливості системи responsive controls
- Встановлення нових стандартів складності ігор

Diablo серія

Рис.1.3 Diablo 4 бій

Серія Diablo представляє класичний підхід до ActionRPG з ізометричної перспективи, фокусуючись на швидкому геймплеї та системі лута.

Особливості бойової системи:

- Click-to-attack механіка: Інтуїтивне керування через миші
- Система класів: Різні стилі гри для кожного класу персонажа
- Масові бої: Одночасна боротьба з великою кількістю ворогів

Технічна реалізація:

- Ефективні алгоритми pathfinding для великої кількості юнітів
- Система процедурної генерації подземель
- Оптимізована обробка великої кількості об'єктів на екрані

1.3 Моделювання предметної області

1.3.1 Загальні відомості про етапи розробки системи бою

Процес розробки бойової системи в Action RPG — це багатоетапний шлях, що вимагає ретельного планування та ітеративного вдосконалення для створення захоплюючого та збалансованого ігрового досвіду.

Визначення Концепції та Технічне Завдання (ТЗ): Цей початковий етап передбачає чітке формулювання бачення бойової системи. Розробляється ТЗ, яке детально описує ключові механіки (наприклад, типи атак, системи захисту, використання здібностей), цілі (наприклад, швидкість, тактична глибина), інтерфейс користувача для бойових дій (наприклад, відображення здоров'я, ресурсів, підказок), структуру даних для бойових параметрів (шкода, опір, час перезарядки) та правила взаємодії між усіма компонентами. ТЗ закладає основу для всіх подальших етапів розробки.

Проектування Архітектури та Прототипування: На цьому етапі створюються перші схеми та моделі бойової системи. Розробляються базові прототипи механік (наприклад, можливість атакувати, ухилятися, використовувати одну-дві здібності). Це дозволяє швидко перевірити основні концепції, такі як відчуття від бою, швидкість реакції, інтуїтивність управління,

а також виявити потенційні проблеми та покращити зручність користування. Прототипування є ключовим для раннього тестування гіпотез.

Створення Мінімально Життєздатного Продукту (MVP): MVP бойової системи включає її базові функціональні можливості. Це може бути реалізація основних атак, простих здібностей, механіки ухилення або блокування та взаємодія з базовими ворогами. MVP дозволяє провести перші внутрішні тестування та, можливо, залучити невелику групу користувачів для збору зворотного зв'язку, що є критично важливим для раннього коригування напрямку розробки.

Альфа-версія: На етапі альфа-версії бойова система отримує повну функціональність. Реалізуються складніші комбо-системи, розширений арсенал здібностей, різноманітні механіки захисту (парірування, контратаки), система станів, інтеграція з системою інвентарю та прогресії персонажа. Проводиться інтенсивне внутрішнє тестування для виявлення та виправлення багів, балансування числових показників (шкоди, здоров'я, таймінгів) та оптимізації продуктивності.

Бета-версія: Бета-версія бойової системи проходить розширене тестування вже на значно ширшій аудиторії. Метою є збір детального зворотного зв'язку від реальних гравців щодо балансу, відчуття від бою, можливих експлойтів, а також виявлення та виправлення останніх критичних помилок. Цей етап є вирішальним для полірування та стабілізації системи перед її релізом.

Реліз та Подальший Супровід: Після випуску гри з готовою бойовою системою розробка не зупиняється. Здійснюється постійний моніторинг зворотного зв'язку від спільноти, випускаються оновлення для виправлення виявлених помилок, коригування балансу та, можливо, додавання нових бойових механік або здібностей. Це забезпечує довгострокову життєздатність та актуальність бойової системи.

1.3.2 Загальні відомості про процеси моделювання предметної області

Ефективне проектування будь якої системи починається зі створення її моделі. Це дозволяє відокремити деталі і зосередитися на ключових компонентах, їхніх властивостях та взаємодіях. Такий підхід допомагає провести ретельний аналіз, уникнути типових помилок та забезпечити логічну структуру майбутньої системи.

Основним інструментом для цього моделювання є UML (Unified Modeling Language). Це стандартизована мова, що дозволяє візуально представити різні аспекти системи через набір діаграм. Завдяки UML, розробники можуть чітко бачити, як взаємодіють елементи бою, хто є їхніми користувачами, та як відбуваються ті чи інші процеси.

Розглянемо ключові типи UML-діаграм, що використовуються для моделювання бойової системи:

Діаграма Прецедентів (Use Case Diagram)

Ця діаграма є початковою точкою для розуміння системи. Вона визначає основні функціональні взаємодії з бойовою системою. Це як список "що можна зробити" в бою та хто (або що) ініціює ці дії.

Діаграми Діяльності (Activity Diagram)

Діаграми діяльності відображають логіку та послідовність дій у бойовій системі. Вони показують "потік" виконання певного процесу, наприклад, як відбувається обчислення шкоди при атаці або як система реагує на активацію здібності.

Діаграми Послідовності (Sequence Diagram)

Ці діаграми ілюструють порядок взаємодії об'єктів у часі для певного сценарію. Вони показують, які повідомлення (виклики методів) надсилаються між об'єктами та в якій послідовності. Це допомагає зрозуміти динаміку комунікації між різними частинами бойової системи під час виконання конкретних дій.

1.3.3 Діаграма прецендентів(Рис 2.1)

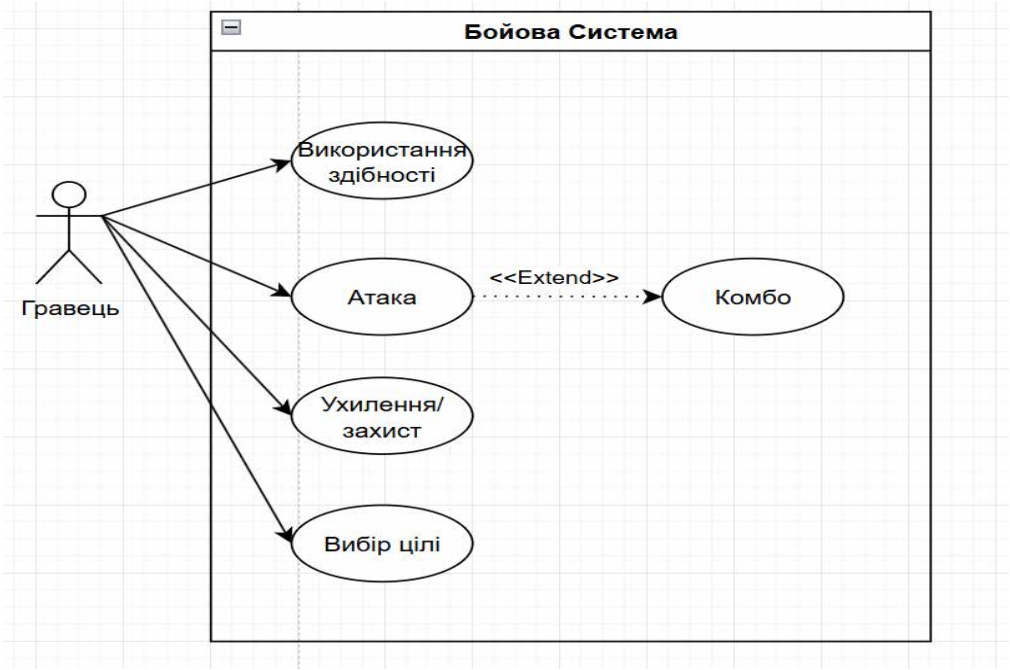

Рис 2.1

Актори:

Гравець

Прцеденти:

Використання здібності

Атака може бути розширена у Комбо

Ухилення/Захист

Вибір цілі

1.3.4 Діаграма діяльності(Рис 2.2**)**

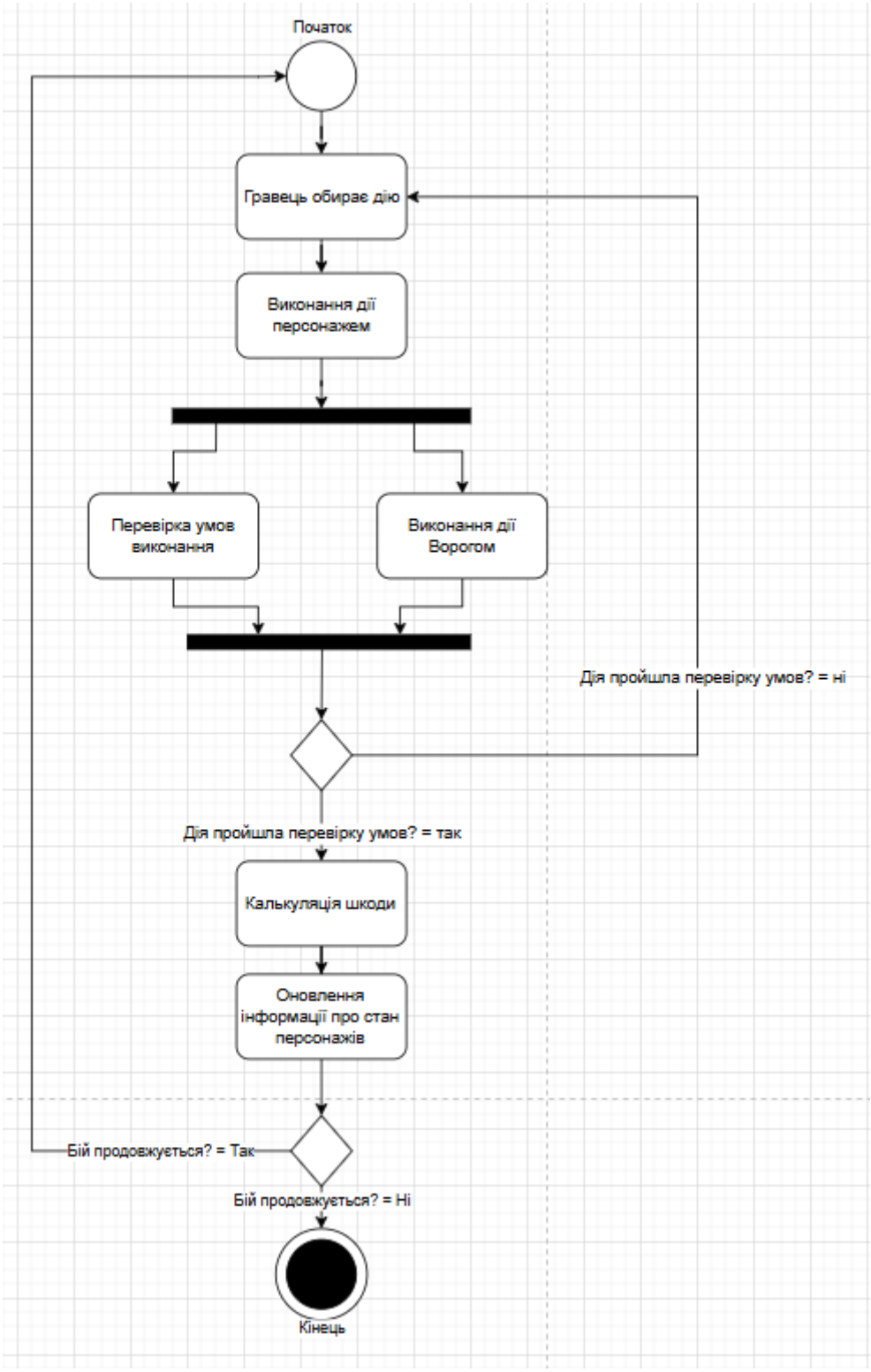

Рис 2.2

Діаграма діяльності(Рис 2.2) ілюструє основний цикл обробки бойових дій, що виконується системою безперервно протягом фази активного бою в реальному часі.

Кожна ітерація цього циклу включає обробку вводу гравця, виконання дій персонажами та ворогами, перевірку умов, розрахунок шкоди та оновлення ігрового стану. Цикл повторюється до моменту завершення бою.

1.3.4 Діаграма послідовності (Рис 2.3**)**

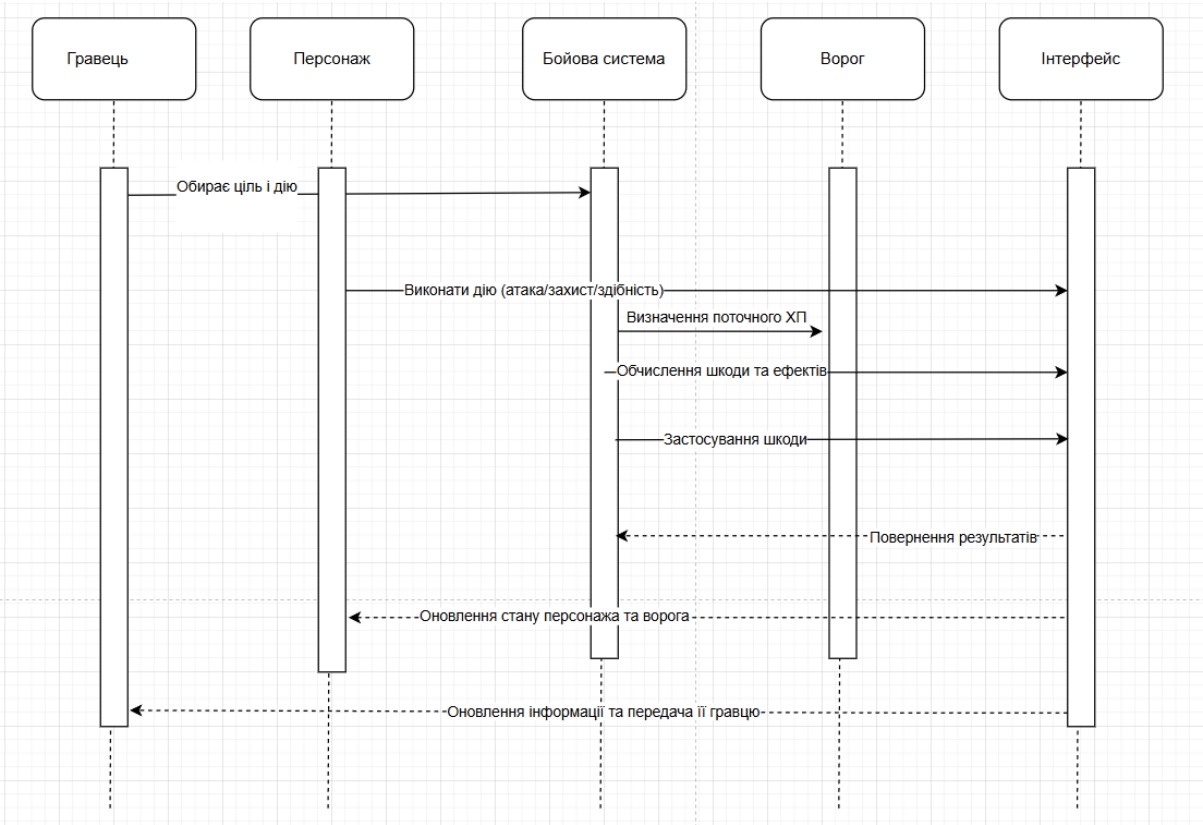

Рис 2.3

Діаграма послідовності ілюструє динамічну взаємодію між ключовими об'єктами системи під час виконання однієї бойової дії (наприклад, атаки, захисту або використання здібності). Вона деталізує послідовність повідомлень, які передаються між акторами та компонентами для реалізації функціоналу, визначеного прецедентами.

Процес розпочинається з Гравця, який ініціює вибір цілі та дії через Персонажа. Далі Персонаж передає запит на виконання дії до Бойової системи.

Бойова система є центральним координатором, що:

Отримує інформацію про поточний стан Ворога (наприклад, для розрахунку шкоди).

Виконує Обчислення шкоди та ефектів згідно з ігровою логікою.

Передає команду Ворогу на Застосування шкоди.

Отримує Повернення результатів від Ворога після застосування шкоди.

Після всіх розрахунків та взаємодій, Бойова система ініціює Оновлення стану Персонажа та Ворога, відображаючи зміни (наприклад, у рівні здоров'я, статусах). Завершальним етапом є Оновлення інформації та її передача Інтерфейсу, який відповідає за відображення актуальних даних Гравцю, забезпечуючи зворотний зв'язок про результати бойової дії. Ця діаграма показує типовий потік подій для одного акту взаємодії в системі бою в реальному часі.

РОЗДІЛ 2

ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Побудова логічної моделі

Проектування архітектури програмної системи вимагає системного підходу до моделювання даних, що забезпечує їхню структуру, цілісність та ефективність обробки. В контексті розробки бойової системи, де динамічні взаємодії між ігровими елементами відіграють ключову роль, ретельне проектування логічної моделі даних є фундаментальним етапом. Вона дозволяє

абстрагуватися від специфіки фізичної реалізації бази даних, зосередившись на сутнісних об'єктах предметної області, їхніх характеристиках та правилах взаємодії.

Для візуалізації логічної моделі даних було застосовано Діаграму "Сутність-Зв'язок" (Entity-Relationship Diagram – ERD), представлену на рисунку 3.1 . ERD є стандартизованим графічним засобом, що дозволяє наочно відобразити сутності – важливі об'єкти чи концепції системи, їхні атрибути – детальні характеристики, та зв'язки – логічні асоціації між сутностями. Такий підхід не лише спрощує розуміння складної архітектури даних, а й сприяє їхній консистентності та ефективній організації.

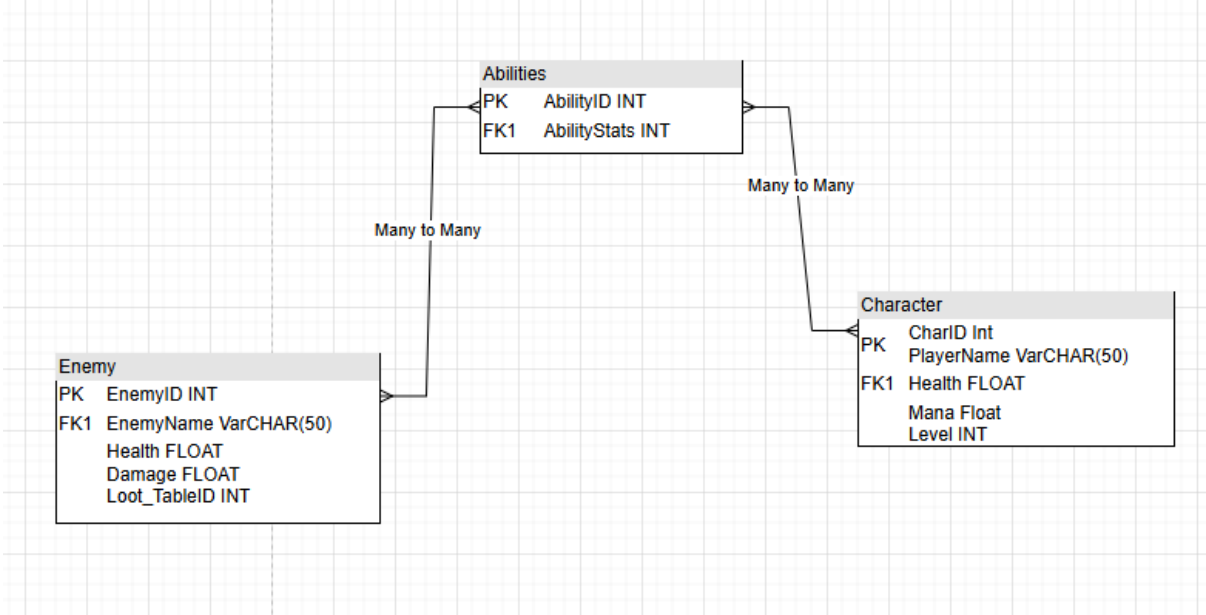

Рис.3.1

Аналіз ключових сутностей та їх атрибутів:

Дана логічна модель даних фокусується на основних ігрових елементах, які безпосередньо впливають на процес бойових дій та керування здібностями. Вона представляє ядро функціоналу, необхідного для обробки бойової логіки.

Сутність "Abilities" (Здібності)

Ця сутність призначена для зберігання всіх унікальних здібностей, які можуть бути використані як ігровими персонажами, так і їхніми опонентами.

Кожна здібність є самостійною одиницею зі своїми параметрами, що визначають її поведінку в бою.

AbilityID (PK): Первинний ключ, що гарантує унікальну ідентифікацію кожної здібності в системі. Це забезпечує можливість швидкого доступу та посилань на конкретні здібності.

AbilityStats (INT): Цей атрибут представляє загальний показник або ідентифікатор, що характеризує статистичні параметри здібності (наприклад, її базову ефективність, вартість активації, тривалість перезарядки). У більш деталізованій моделі або при програмній реалізації, він може посилатися на окрему сутність, яка містить розгорнутий набір статистичних даних, або бути декомпозиваний на множину окремих атрибутів.

Сутність "Enemy" (Ворог)

Сутність "Enemy" інкапсулює дані про всіх неігрових персонажів, що виступають противниками гравця у бойовій системі. Різноманітність ворогів є ключовим аспектом створення цікавого та збалансованого ігрового процесу.

EnemyID (PK): Унікальний ідентифікатор для кожного типу ворога, що забезпечує його розпізнавання в базі даних.

EnemyName (VARCHAR(50)): Текстове поле для зберігання назви ворога. Тип VARCHAR з обмеженням у 50 символів є достатнім для більшості іменованих сутностей.

Health (FLOAT): Відображає поточний рівень здоров'я ворога. Використання типу FLOAT дозволяє гнучко обробляти дробові значення, що важливо для точних розрахунків шкоди та ефектів.

Damage (FLOAT): Визначає базове значення шкоди, яку даний тип ворога може нанести. Також використовується тип FLOAT для забезпечення точності розрахунків.

Loot_TableID (INT): Зовнішній ключ (FK1), що посилається на ідентифікатор таблиці луту. Це дозволяє гнучко асоціювати з кожним типом

ворога певний набір потенційних предметів, що можуть випасти після його знищення, що є критичним для ігрової економіки та прогресії.

Сутність "Character" (Персонаж)

Сутність "Character" є центральною для представлення ігрового персонажа, яким керує Гравець. Вона містить всі ключові атрибути, що характеризують його стан та можливості в бойовій системі.

CharID (PK): Унікальний ідентифікатор для кожного ігрового персонажа.

PlayerName (VARCHAR(50)): Ім'я, яке гравець обирає для свого персонажа. Це поле є ідентифікатором персонажа, а не облікового запису гравця.

Health (FLOAT): Поточний рівень здоров'я персонажа, що визначає його здатність продовжувати бій.

Mana (FLOAT): Поточний рівень ресурсу (мани або аналогічного), необхідного для активації спеціальних здібностей.

Level (INT): Рівень розвитку персонажа, що, як правило, впливає на його базові характеристики, ефективність здібностей та доступ до нового контенту.

Аналіз зв'язків між сутностями:

Зв'язки в ERD визначають логічні асоціації та кардинальність взаємодії між сутностями, що є ключовим для правильної організації даних.

Enemy (Many to Many) Abilities:

Цей зв'язок "Багато до Багатьох" відображає концепцію, згідно з якою один тип Ворога може володіти багатьма Здібностями, і одна Здібність може бути притаманна багатьом типам Ворогів. Це забезпечує гнучкість у створенні різноманітних противників з унікальними наборами бойових навичок.

У реляційних базах даних для реалізації зв'язків "Багато до Багатьох" необхідно використовувати проміжну асоціативну таблицю (наприклад, Enemy_Abilities). Ця таблиця міститиме зовнішні ключі до EnemyID та AbilityID, забезпечуючи ефективне відображення всіх зв'язків.

Character (Many to Many) Abilities:

Аналогічно попередньому, цей зв'язок "Багато до Багатьох" показує, що багато ігрових Персонажів можуть вивчати та використовувати багато Здібностей, і одна Здібність може бути вивчена багатьма Персонажами. Це є фундаментальним для системи розвитку персонажа, дозволяючи гравцям налаштовувати набори своїх навичок.

Для реалізації цього зв'язку також необхідна проміжна таблиця (наприклад, Character_Abilities або LearnedAbilities). Ця таблиця міститиме зовнішні ключі до CharID та AbilityID, дозволяючи системі відстежувати, які здібності належать якому персонажу.

Висновок та архітектурні міркування:

Представлена ERD формує логічну модель даних для ядра бойової системи, фокусуючись на безпосередніх учасниках бою та їхніх ключових характеристиках і взаємодіях. Вона забезпечує чітку та структуровану основу для зберігання даних про персонажів, ворогів та їхні здібності, що є критичним для реалізації бойової логіки.

Важливо відзначити, що в контексті комплексної ігрової системи, існують також інші модулі та підсистеми, які взаємодіють з бойовою системою. Зокрема, система інвентаризації (управління предметами, їх зберіганням та використанням) є невід'ємною частиною ігрового процесу та має власну, окрему логічну модель даних. Хоча детальний розгляд цієї моделі виходить за рамки поточної діаграми, її інтеграція з бойовою системою (наприклад, через застосування витратних предметів, що впливають на здоров'я/ману, або через вплив екіпірованих предметів на характеристики персонажа) є архітектурно передбаченою та реалізується через відповідні програмні інтерфейси та виклики між модулями.

2.2 Вибір та обґрунтування вибору системи управління базою даних

В основі будь-якої складної програмної системи, особливо інтерактивної, такої як ігрова, лежить ефективна та надійна система управління інформаційною базою. Це забезпечує персистентність даних, їхню цілісність та оперативний доступ, що є критично важливим для динамічного ігрового процесу. Вибір відповідної системи управління інформаційною базою данних (СУБД) є одним з ключових архітектурних рішень, що впливає на продуктивність, масштабованість, складність розробки та супроводу системи.

У контексті розробки ігрових застосунків, зокрема на сучасному ігровому рушії Unreal Engine 5, концепція "СУБД" набуває дещо іншої інтерпретації, ніж у традиційних корпоративних системах з реляційними базами даних. Для розробки гри, де швидкість доступу до даних, простота інтеграції з ігровою логікою та гнучкість конфігурації є пріоритетними, вибір СУБД обґрунтовується на основі можливостей, що надаються самим ігровим рушієм.

Для реалізації бойової системи було обрано Unreal Engine 5, який надає широкий спектр вбудованих механізмів для ефективного управління ігровими даними. Такий вибір мотивований кількома ключовими факторами:

Вбудовані рішення для ігрових даних: Unreal Engine 5 інтегрує потужні інструменти для роботи з даними, які спеціально оптимізовані для ігрової розробки. Це включає системи таблиць даних (Data Tables), асетів, структур даних (Structs), та механізми серіалізації/десеріалізації для збереження ігрового прогресу. Ці інструменти дозволяють зберігати як статичні (конфігураційні), так і динамічні (стан ігрового світу) дані.

Продуктивність та оптимізація: Механізми управління даними в Unreal Engine розроблені з урахуванням високих вимог до продуктивності, типових для ігор у реальному часі. Дані завантажуються в пам'ять ігрового процесу

ефективно, забезпечуючи мінімальні затримки при доступі та обробці, що є критично важливим для безперебійної роботи бойової системи.

Простота інтеграції з ігровою логікою: Використання нативних для рушія способів зберігання даних спрощує їх інтеграцію з ігровою логікою, реалізованою на C++ та Blueprints. Це зменшує складність коду, усуває потребу у написанні додаткових шарів абстракції та забезпечує швидкий доступ до необхідних параметрів під час обчислень шкоди, перевірок здібностей тощо.

Гнучкість конфігурації та ітерації: Data Tables, зокрема, дозволяють ігровим дизайнерам та розробникам гнучко конфігурувати ігрові параметри (характеристики здібностей, ворогів) без прямої модифікації коду. Це прискорює процес балансування гри та дозволяє швидко вносити ітераційні зміни.

Відсутність зовнішніх залежностей: Для однокористувацької (або локальної) гри, використання вбудованих механізмів Unreal Engine усуває потребу у розгортанні, адмініструванні та підтримці зовнішніх систем управління базами даних (наприклад, SQL-серверів). Це значно спрощує архітектуру проекту, зменшує витрати ресурсів на розробку та супровід, а також підвищує мобільність розгортання гри.

Таким чином, у даній бакалаврській роботі під "системою управління інформаційною базою" розуміється комплексний інструментарій та підходи, надані ігровим рушієм Unreal Engine 5, що дозволяють ефективно зберігати, організовувати та управляти всіма необхідними даними бойової системи. Це рішення повністю відповідає специфічним вимогам ігрової розробки та оптимізоване для досягнення високої продуктивності та гнучкості.

Порівняння з альтернативними підходами:

Хоча Unreal Engine 5 надає власні рішення для збереження та обробки даних, важливо розглянути альтернативні підходи до управління інформацією в ігрових проектах:

- Зовнішні реляційні СУБД (MySQL, PostgreSQL): використовуються переважно у багатокористувацьких іграх з серверною архітектурою. Забезпечують потужну систему запитів, але потребують складної інтеграції з рушієм, додаткової авторизації, адміністрування серверів, а також серйозного захисту.

- NoSQL-рішення (MongoDB, Redis): ефективні для зберігання великих обсягів слабо структурованих даних. Проте, як і в попередньому випадку, потребують додаткового програмного забезпечення та не є оптимальними для локальних чи синглплеєрних проєктів.

У контексті цієї роботи, ці альтернативи не обрано через невиправдану складність та надлишковість для цілей проєкту.

Для повноти аналізу доцільно порівняти вбудовану систему управління даними Unreal Engine 5 з іншими можливими підходами, які теоретично могли б бути застосовані для зберігання інформації у грі:

Критерій	Unreal Engine 5 (DataTables, Assets, Structs)	Реляційна СУБД (MySQL, PostgreSQL)	NoSQL СУБД (MongoDB, Redis)
Продуктивність у реальному часі	Висока, оптимізована під ігрові задачі	Середня, залежить від архітектури доступу	Висока (Redis), середня (MongoDB)

Критерій	Unreal Engine 5 (DataTables, Assets, Structs)	Реляційна СУБД (MySQL, PostgreSQL)	NoSQL СУБД (MongoDB, Redis)
Інтеграція з рушієм	Нативна, без додаткового коду	Потребує розробки API, драйверів	Потребує окремої логіки синхронізації
Складність налаштування	Мінімальна	Висока: установка, конфігурація сервера	Висока: встановлення сервера, залежності
Гнучкість конфігурації	Висока (таблиці, Struct’и)	Висока, але потребує SQL-запитів	Висока, з використанням JSON
Цілісність даних	Контролюється рушієм	Гарантується транзакціями	Частково гарантується
Підтримка автономності	Повна, не потребує інтернету	Потребує запуску сервера	Часто прив’язано до хмари або сервера
Придатність для синглплеєр гри	Ідеальна	Надлишкова складність	Сумнівна доцільність
Масштабованість	Обмежена	Висока	Висока

Аналіз альтернатив

1. Реляційні СУБД (MySQL, PostgreSQL)

Підходять для клієнт-серверних ігор, де збереження даних користувача має відбуватися централізовано. Однак у контексті локальної бойової системи вони створюють непотрібні залежності:

- Потребують серверної інфраструктури
- Ускладнюють деплой гри
- Збільшують час розробки
- Вимагають написання додаткової логіки авторизації та обробки помилок

2. NoSQL СУБД (MongoDB, Redis)

Ці системи ефективні для неструктурованих або тимчасових даних. Redis — чудовий вибір для кешу, а MongoDB — для об’єктів у JSON-стилі. Але:

- Потребують запуску окремого сервера або залежності від хмари
- Погано інтегруються з Blueprint логікою
- Ускладнюють локальну архітектуру

Перспективи масштабування архітектури

Попри локальний характер бойової системи, обраний підхід дозволяє у майбутньому розширити гру наступним чином:

- Мережева синхронізація збережень — збереження в локальних SaveGame об’єктах можуть бути перетворені у формат, придатний для передачі через REST API або WebSockets.

- Інтеграція з хмарними службами — при потребі додати багатокористувацький режим, реалізація клієнт-серверної взаємодії можлива шляхом перетворення існуючих структур у об'єктно-серіалізовані JSON-пакети.

- Змішане зберігання — використання як локальних, так і серверних джерел даних (наприклад, локальна кешована конфігурація + серверна авторизація).

Висновки

Обраний інструментарій Unreal Engine 5 виконує роль повноцінної СУБД у специфічному контексті ігрової розробки, де основні вимоги — це продуктивність, простота, інтеграція з логікою та гнучкість. Альтернативи у вигляді традиційних баз даних є надлишковими й недоцільними в умовах локального застосунку. Таким чином, використання вбудованих засобів рушія — це не тільки технічно виправдане, а й стратегічно ефективне рішення для даного проєкту.

2.3 Створення інформаційної бази

Створення інформаційної бази є логічним продовженням етапу побудови логічної моделі даних та вибору системи управління.

Цей розділ описує практичні аспекти реалізації структури даних у вибраному середовищі – ігровому рушії Unreal Engine 5. Процес створення інформаційної бази включає визначення конкретних механізмів зберігання, наповнення їх початковими даними та налагодження взаємодії з ігровою логікою.

Відповідно до логічної моделі, представленої в розділі 2.1, інформаційна база бойової системи була організована з використанням гібридного підходу, що поєднує статичне зберігання конфігураційних даних та динамічне управління станом об'єктів у реальному часі.

Рис. 3.2 Модульний файл Actor component

Згідно з методологією розробки, увесь код, який використовує рушій для логіки гри, є модульним й зберігається в окремих Actor Component файлах самого рушія.

Рис. 3.3 Мастер-файл

В данному випадку файл BPC_Stats виконує функцію мастер-файла в якому будуть зберігатися:

- Функції, за допомогою яких будуть змінюватись змінні
- Виклики самих змінних з окремого файлу E_Stats(Див.Рис. 3.5) - Це зроблено для легшого зберігання та редагування персистентних даних
- Логіка подій, пов`язаних зі зміною змінних

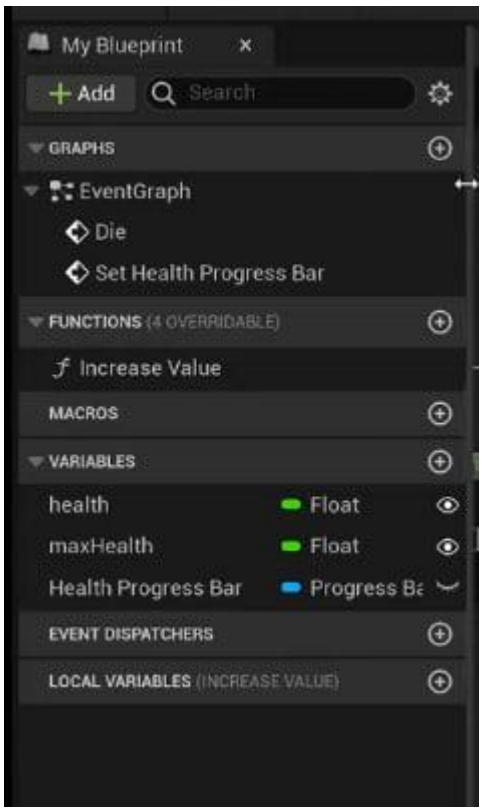

Рис. 3.4 Змінні, функції, події

Рис. 3.5 Enumeration файл

У процесі розробки ігрових систем, зокрема бойової механіки, виникає потреба у зберіганні та обробці чітко визначених множин значень. Це можуть бути стани персонажа, типи атак, категорії ворогів тощо. Для вирішення таких задач в Unreal Engine 5 доцільно застосовуються Enumeration-файли (або ж Enum-и), які представляють собою зручний механізм організації строго обмежених множин значень у вигляді перерахувань.

Загальна характеристика Enumeration-файлів

Enumeration (або перелік) — це користувацький тип даних, який дозволяє оголосити набір іменованих констант, кожна з яких асоціюється з унікальним цілим значенням. У контексті Unreal Engine, енумерації реалізуються як

Blueprint Enumeration (у редакторі) або C++ enum (у коді), що дозволяє інтегрувати їх у візуальну логіку або програмний код відповідно.

Переваги використання Enum у грі:

- Чіткість і структура: забезпечує декларативне оголошення всіх можливих значень певної сутності (наприклад, типу атаки: Melee, Ranged, Magic), що унеможливлює випадкове використання неправильного значення.

- Зручність у Blueprint-середовищі: UE5 дозволяє легко створювати та використовувати Enum без написання коду — через візуальний редактор, з автоматичною інтеграцією у випадаючі списки, Select-ноди, порівняння тощо.

- Інтеграція зі структурами даних: Enum-и можуть бути використані як поля в Struct-ах або таблицях даних, що дозволяє організовувати складні конфігурації ігрових об'єктів у зрозумілому вигляді.

- Гнучкість розширення: у разі зміни дизайну гри можна додати нові значення до Enum без порушення існуючої логіки, якщо структура умов та обробки побудована грамотно.

- Зменшення помилок: використання перерахувань значно знижує ймовірність типографічних чи логічних помилок при порівнянні строкових або числових значень.

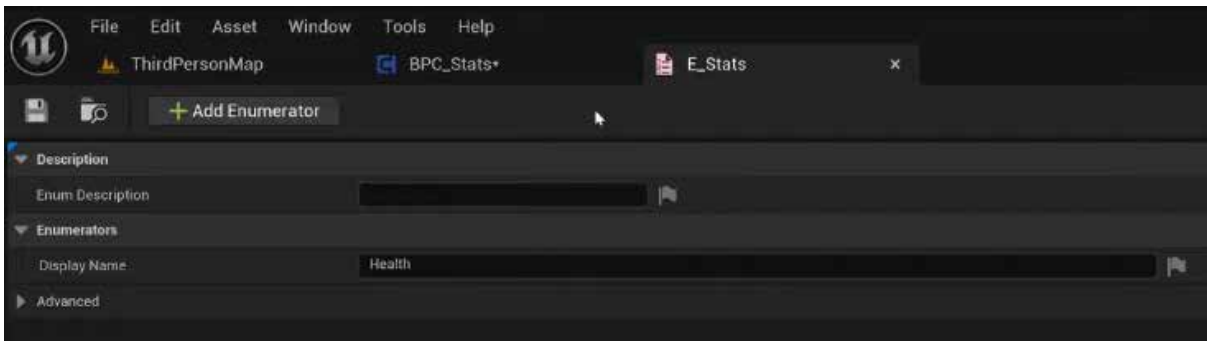

Рис. 3.6 Інтерфейс Enum файла

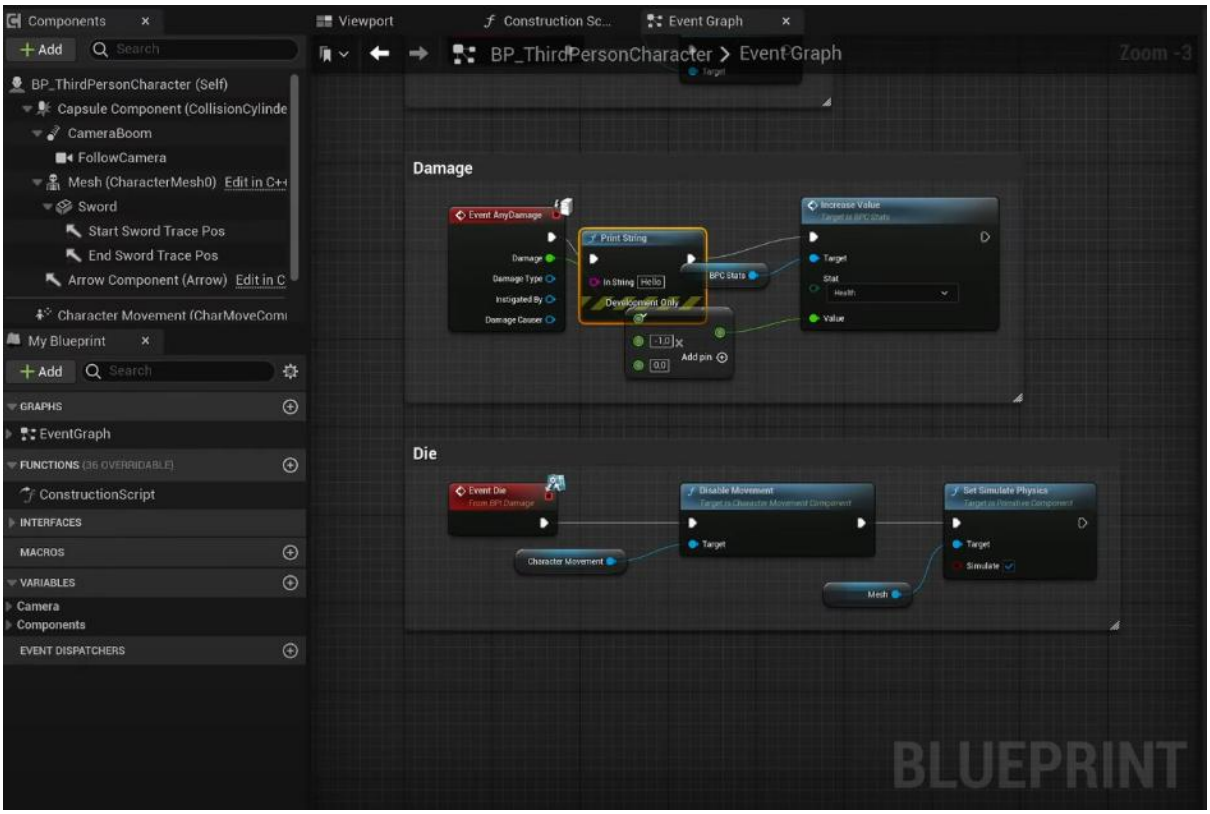

Рис. 3.7 Внутрішня логіка мастер-файлу

Система інвентаря є нерозривно пов’язаною з бойовою системою, але, як й було сказано раніше, інвентарна система потребує окремого дослідженя, тому огляд імплементації буде на поверхневому рівні й тільки те, що напряму стосується бойового процесу.

Основні аспекти цієї інтеграції включають:

- Активна зброя: гравець обирає зброю з інвентаря, після чого вона прив’язується до бойових функцій персонажа (анімації, шкода, дальність тощо)

- Споживані предмети: зілля, бомби, скролли — активуються через інвентар, що змінює стани персонажа (HP, MP, стани)

- Модифікатори бою: спорядження може містити бонуси (до атаки, захисту, швидкості), які враховуються під час обчислення шкоди

Ключова перевага — у динамічності: гравець може в реальному часі змінювати спорядження, обирати стратегію бою відповідно до доступних предметів. Це створює глибший геймплей і потребу у тактичному мисленні.

Інвентар як частина UI/UX досвіду:

Інвентар також є елементом інтерфейсу користувача, тому важливою є його візуальна реалізація. У UE5 для цього зазвичай використовуються:

- Widget Blueprints (UMG): створення сіток предметів, описів, кнопок взаємодії

- Drag & Drop системи: підтримка перетягування предметів, сортування, комбінування

- Індикатори: відображення кількості, рідкості, стану предметів

Оскільки інвентар тісно взаємодіє з гравцем, правильна реалізація UI підвищує загальний рівень зручності, а також впливає на сприйняття якості гри.

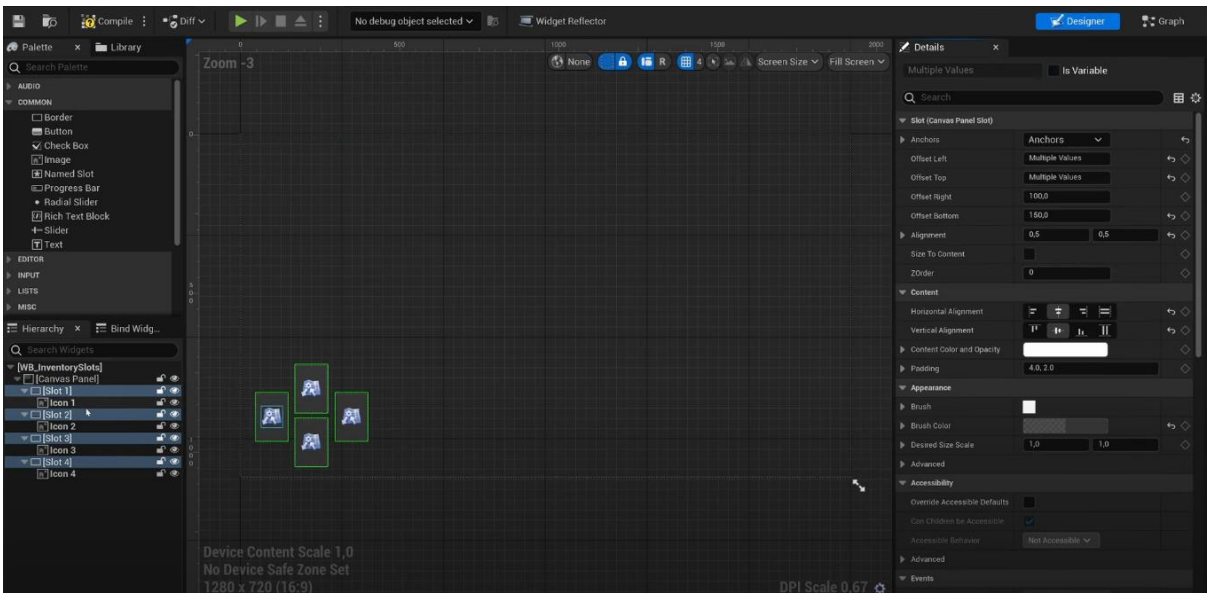

Рис. 3.8 Приклад імплементації віджета інвентарю

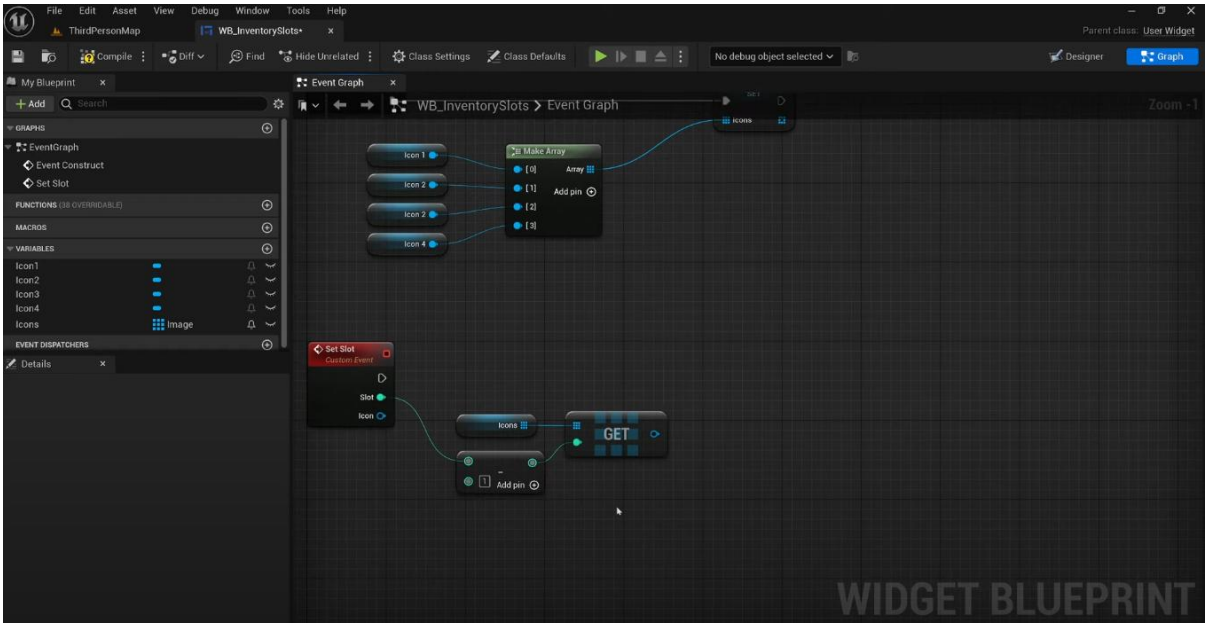

Рис. 3.9 Внутрішня логіка слотів, зав`язана на системі Blueprints.

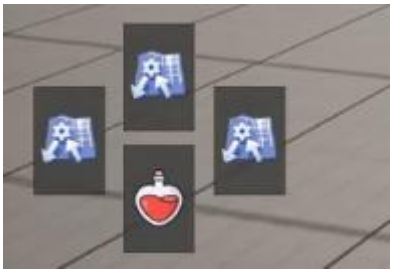

Рис. 3.10 Фінальний результат слотів інвентарю. Спрощена версія.

РОЗДІЛ 3

ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Організаційна структура програмного забезпечення

Розробка прикладного програмного забезпечення (ППЗ) є кульмінаційним етапом життєвого циклу проекту, де теоретичні моделі та вимоги трансформуються у функціональну систему.

У контексті даної бакалаврської роботи, цей розділ детально висвітлює архітектурні рішення, обґрунтування вибору технологічного стеку та методологію реалізації бойової системи в середовищі ігрового рушія Unreal Engine 5. Метою є демонстрація практичного застосування досліджень у галузі програмної інженерії для створення складного, динамічного та інтерактивного програмного продукту, що відповідає сучасним вимогам до ігрових систем.

Ефективна організаційна структура програмного забезпечення є фундаментальним аспектом, що визначає успіх проекту, його масштабованість, підтримуваність та гнучкість до майбутніх змін. У розробці ігор, де системи часто є високоінтерактивними та вимагають оптимальної продуктивності, архітектурні рішення відіграють критичну роль.

Даний підрозділ описує принципи та реалізацію архітектурної організації бойової системи, розробленої на Unreal Engine 5, демонструючи, як логічні моделі трансформуються у функціональні програмні модулі.

3.1.1 Загальні принципи архітектури

Проектування бойової системи базувалося на загальноприйнятих принципах програмної інженерії, що забезпечують високу якість коду та архітектури. Серед них ключовими є:

Модульність: Система була розділена на логічно незалежні, але взаємодіючі модулі. Кожен модуль відповідає за певний функціональний аспект (наприклад, управління персонажами, обробка здібностей, штучний інтелект ворогів, інтерфейс користувача). Такий поділ дозволяє розробляти, тестувати та підтримувати частини системи незалежно, зменшуючи загальну складність проекту та мінімізуючи ризики при внесенні змін. Модульність також сприяє паралельній розробці, що є перевагою для командних проектів.

Компонентний підхід: Unreal Engine 5 активно використовує компонентну архітектуру, яка була прийнята як основна парадигма для побудови ігрових об'єктів. Замість глибоких ієрархій успадкування, функціональність інкапсулюється в переуспадковувані компоненти(Див. Рис. 4.1). Це дозволяє уникнути проблем "смертельної діамантової ієрархії"(deadly diamond of death), сприяє повторному використанню коду, забезпечує гнучке конфігурування ігрових об'єктів та полегшує розширення їхніх можливостей без модифікації базових класів.

Слабка зв'язність (Loose Coupling): Взаємодія між модулями та компонентами реалізується переважно через інтерфейси, події (Event Dispatchers) та делегати

(Delegates), а не через прямі жорсткі залежності. Це означає, що зміни у внутрішній реалізації одного модуля мінімально впливають на інші модулі, що з ним взаємодіють. Такий підхід значно спрощує модифікацію, тестування та розширення системи, оскільки компоненти не знають про внутрішню структуру один одного, а лише про їхній публічний інтерфейс або події, на які вони реагують.

Принцип єдиної відповідальності (Single Responsibility Principle - SRP): Кожен клас або модуль має відповідати лише за одну чітко визначену функцію. Наприклад, BP_Stats відповідає виключно за управління характеристиками, а BPC_CombatSystem – за логіку атаки, здібностей, тощо. Це підвищує читабельність коду/блюпринтів, спрощує налагодження та зменшує ймовірність виникнення помилок.

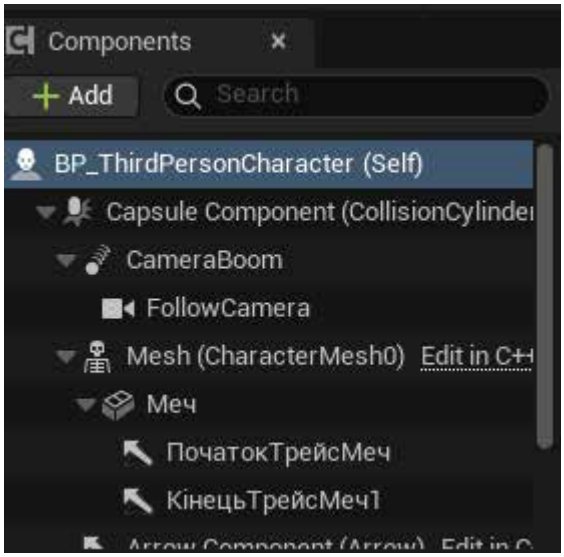

Рис. 4.1 До mesh’а додан компонент «меч»

3.1.2 Декомпозиція бойової системи на функціональні модулі

На основі вищезгаданих принципів, бойова система була декомпозована на такі ключові функціональні модулі у самому рушії, кожен з яких виконує свою специфічну роль у загальному бойовому процесі:

Модуль управління боєм (Combat Manager Module):

- Призначення: Цей модуль є архітектурним ядром бойової системи, що відповідає за координацію всіх бойових дій та управління загальним станом бою. Він виступає як центральний диспетчер, який ініціює, контролює та завершує бойові зіткнення.
- Реалізація: Реалізується як окремий Actor-component (ACombatManager), або як компонент ігрового стану (AGameStateBase або AGameModeBase), що дозволяє йому мати глобальний доступ до ігрового світу та керувати станом бою. Він оперує колекціями посилань на всі активні бойові одиниці (персонажів, ворогів), що беруть участь у поточному зіткненні.
- Ключові функції: Ініціація бойового режиму, реєстрація та дереєстрація учасників бою, управління бойовим циклом (обробка ігрових "тіків" або "ходів" у реальному часі), делегування обчислень шкоди та ефектів, перевірка умов завершення бою та ініціація логіки післябою (наприклад, видача досвіду, луту).

Варто зазначити, що більшу частину всієї роботи бере на себе саме рушій, тому багато логіки оптимізовано та скрито від очей середньостатистичного розробника.

Модуль персонажів та ворогів (Character & Enemy Core Module):

- Призначення: Цей модуль інкапсулює базові атрибути, функціональність та поведінку всіх бойових одиниць в ігровому світі. Він є основою для створення як ігрового персонажа, так і різноманітних типів ворогів.
- Реалізація: Представлений базовими C++ класами (ACharacterBase для ігрового персонажа, що успадковується від ACharacter, та AEnemyBase для ворогів, що успадковується від APawn або ACharacter). Ці базові класи містять спільні атрибути (здоров'я, мана, рівень, базовий захист) та функції (отримання шкоди, використання здібностей, переміщення).
- Взаємодія: Ці модулі взаємодіють з Combat Manager для виконання дій та отримання шкоди, а також з Ability System для активації та управління здібностями.

Модуль штучного інтелекту (AI Module):

- Призначення: Відповідає за автономну та реалістичну поведінку неігрових персонажів (ворогів) у бойових сценаріях. Він визначає логіку вибору цілей, прийняття рішень щодо використання здібностей, переміщення та реакції на ігрові події.
- Реалізація: В Unreal Engine AI часто реалізується за допомогою Behavior Trees та AI Controllers. AIController є "мозком" ворога, який керує його актором і виконує Behavior Tree. Behavior Tree дозволяє візуально створювати складну логіку прийняття рішень (наприклад, "Якщо гравець близько, атакуй; якщо HP низьке, відступай; якщо союзник у небезпеці, допоможи"). Додатково можуть використовуватися сервіси (Services) та таски (Tasks) для деталізації поведінки.
- Взаємодія: AI Module взаємодіє з Enemy Core для отримання інформації про стан ворога та його оточення, а також з Ability System для

активації здібностей ворога відповідно до прийнятих рішень. Також LootTables для визначення лута, який падає з ворогів при їх смерті(Якщо такий є).

3.1.3 Взаємодія між модулями та принципи комунікації

Взаємодія між описаними модулями є ключовим аспектом функціонування бойової системи. Вона реалізується за допомогою чітко визначених механізмів комунікації, що забезпечують гнучкість, розширюваність та ефективність.

Прямі виклики функцій: Використовуються для тісно пов'язаних компонентів або для запитів, що вимагають негайної синхронної відповіді (наприклад, Combat Manager викликає функцію TakeDamage на AttributeComponent цілі). Це переважно внутрішньомодульна комунікація або виклики від координатора до керованого компонента.

Делегати та події (Event Dispatchers): Це механізм, що дозволяє об’єктам взаємодіяти між собою без прямого зв’язку. Тобто один об’єкт може «викликати» подію, не знаючи, хто її буде слухати. Це суть патерна Observer (спостерігач), реалізованого в стилі UE5.

Інтерфейси: Використовуються для визначення контрактів між класами. Клас може реалізовувати інтерфейс, гарантуючи наявність певних публічних функцій, які можуть бути викликані іншими модулями, не знаючи конкретного типу об'єкта. Це забезпечує поліморфізм та підвищує гнучкість системи.

Така організаційна структура програмного забезпечення, що базується на принципах модульності, компонентного підходу та слабкої зв'язності, забезпечує високий рівень гнучкості та розширюваності. Це є критично

важливим для великих ігрових проектів, дозволяючи команді розробників працювати паралельно над різними частинами системи, а також спрощуючи тестування, налагодження та подальше розширення функціоналу бойової системи.

3.2 Вибір інструментарію для створення ППЗ

Вибір інструментарію є одним з найважливіших стратегічних рішень на початкових етапах розробки програмного забезпечення, оскільки він безпосередньо впливає на продуктивність розробки, якість кінцевого продукту, можливості масштабування та супроводу. Для створення бойової системи було обрано комплексний набір інструментів, центральним елементом якого є ігровий рушій Unreal Engine 5. Обґрунтування цього вибору базується на аналізі вимог до проекту, сучасних тенденцій у геймдеві та переваг, які надає даний технологічний стек.

3.2.1 Ігровий рушій Unreal Engine 5

Unreal Engine 5 (UE5) є одним з найпотужніших та найсучасніших ігрових рушіїв у світі, розробленим компанією Epic Games. Його вибір для даного проекту обґрунтований низкою ключових переваг, що роблять його ідеальним для розробки складних та високопродуктивних ігрових систем, таких як бойова:

Передова візуальна якість та продуктивність: UE5 відомий своїми революційними графічними можливостями, такими як Nanite (система віртуалізованої геометрії, що дозволяє імпортувати мільярди полігонів без втрати продуктивності) та Lumen (система глобального освітлення в реальному часі). Хоча основний фокус бакалаврської роботи — бойова логіка, ці технології забезпечують неперевершене фотореалістичне візуальне середовище, що є важливим для занурення гравця в ігровий світ. Рушій оптимізований для роботи

з великими обсягами даних та складними обчисленнями, що критично для динамічних бойових систем, де кожна мілісекунда затримки може вплинути на ігровий досвід.

Комплексний та інтегрований набір інструментів: UE5 надає розробникам повністю інтегроване середовище розробки (Unreal Editor), що включає широкий спектр інструментів для всіх аспектів створення гри, що значно прискорює робочий процес:

Редактор рівнів: Інтуїтивно зрозумілий інструмент для створення та налаштування ігрового простору, розміщення акторів, налаштування освітлення та оточення.

Система анімації: Потужні інструменти для імпорту, ретаргетингу, змішування анімацій (Animation Blueprints, Montages, Control Rig, IK), що дозволяє створювати реалістичні та плавні анімації бойових дій, критично важливі для візуального відгуку.

Система візуальних ефектів (Niagara): Передова система для створення складних ефектів частинок (вибухи, спалахи, магічні ефекти, сліди від ударів), які супроводжують бойові здібності, підвищуючи їхню виразність та вплив.

Система звуку (MetaSounds): Гнучка та потужна система для інтеграції, управління та динамічного генерування звукових ефектів, що підвищує занурення в ігровий процес та надає важливий аудіальний відгук на бойові події.

Фізичний рушій (Chaos): Вбудована система для реалістичної симуляції фізики об'єктів, що може бути використано для інтерактивного руйнування оточення або реалістичної поведінки об'єктів під час бойових зіткнень.

Unreal Motion Graphics (UMG): Інтуїтивно зрозуміла та потужна система для створення інтерфейсу користувача (UI/HUD), що дозволяє легко відображати інформацію про бій (HP бари, лог подій, іконки здібностей, повідомлення про шкоду) та обробляти ввід гравця.

Гнучка архітектура та модульність: Рушій побудований на компонентній архітектурі, що сприяє високому рівню модульності та повторному використанню коду. Це дозволяє розробляти бойову систему як набір взаємодіючих компонентів, які можна додавати до різних акторів, забезпечуючи гнучкість та розширюваність дизайну.

Відкритий вихідний код: Доступ до повного вихідного коду рушія (на C++) дозволяє розробникам глибоко розуміти його функціонування, оптимізувати або розширювати функціонал під специфічні потреби проекту, а також налагоджувати проблеми на низькому рівні.

Активна спільнота та документація: Велика та активна спільнота розробників по всьому світу, а також детальна та регулярно оновлювана офіційна документація, забезпечують широку підтримку та доступ до численних ресурсів, навчальних матеріалів та прикладів для вирішення будь-яких технічних завдань.

3.2.2 Мови програмування

Розробка бойової системи в Unreal Engine 5 здійснювалася з використанням гібридного підходу, що поєднує можливості двох основних мов програмування, які є нативними для рушія: C++ та Blueprints. Такий підхід дозволяє максимально ефективно використовувати переваги кожної мови.

C++:

Призначення: C++ є основною мовою програмування для розробки в Unreal Engine. Він використовується для створення базових класів, реалізації складних алгоритмів, оптимізованих обчислень та будь-якої логіки, що вимагає високої продуктивності та низькорівневого контролю над системними ресурсами.

Переваги:

- Висока продуктивність: C++ забезпечує максимальну продуктивність, що є критично важливим для обчислень у реальному часі, таких як розрахунок шкоди, перевірка колізій, управління великою кількістю ігрових об'єктів та обробка складних ігрових механік.
- Повний контроль: Надає розробнику повний контроль над пам'яттю та апаратними ресурсами системи, дозволяючи створювати високоефективний та оптимізований код.
- Доступ до рушія: Дозволяє безпосередньо взаємодіяти з API рушія, розширювати його функціонал, створювати власні класи та оптимізувати внутрішні механізми.
- Масштабованість: Ідеально підходить для створення великих та складних систем, які можуть бути легко розширені та підтримувані в майбутньому.

Blueprints:

Призначення: Blueprints є візуальною мовою скриптингу в Unreal Engine, що дозволяє створювати ігрову логіку шляхом з'єднання вузлів у графічному інтерфейсі редактора. Вона ідеально підходить для швидкого прототипування, реалізації логіки, що не вимагає екстремальної продуктивності, та для роботи ігрових дизайнерів.

Переваги:

- Швидкість розробки: Значно прискорює процес створення логіки, оскільки не вимагає компіляції коду та надає миттєвий візуальний зворотний зв'язок.
- Візуалізація: Графічне представлення логіки робить її більш зрозумілою, легкою для читання, налагодження та спільної роботи в команді.
- Доступність: Дозволяє не-програмістам (гейм-дизайнерам, художникам) створювати інтерактивні елементи та ігрову логіку, розширюючи можливості команди.
- Інтеграція: Безшовно інтегрується з C++ кодом, дозволяючи розширювати C++ класи за допомогою Blueprints, додавати специфічну логіку або налаштовувати поведінку.
- Застосування в проекті: Blueprints використовувались для реалізації фактично усієї логіки в данному проекті. Бій, анімації, VFX/SFX, тощо, усе зроблено за допомогою blueprints.

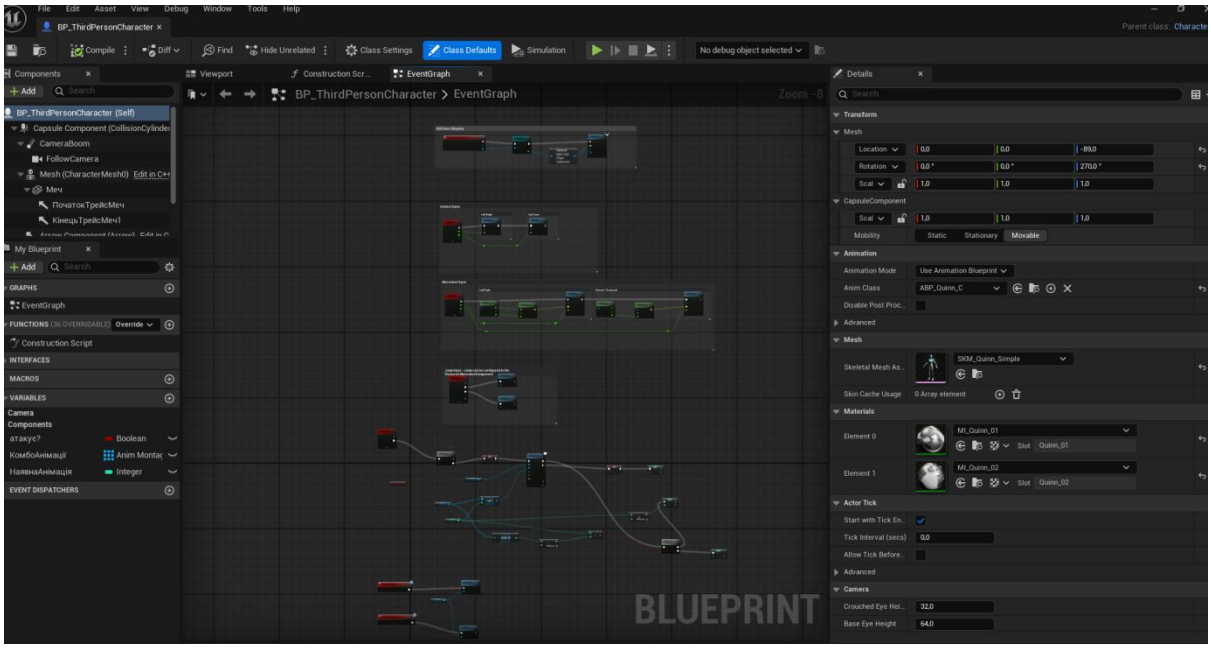

Рис. 4.2 Приклад blueprint’а BP_ThirdPersonCharacter.

РОЗДІЛ 4

РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Цей розділ присвячений ключовим аспектам, пов'язаним із впровадженням та подальшою експлуатацією розробленої бойової системи. Він охоплює методології тестування, необхідні для забезпечення якості та стабільності програмного забезпечення, а також деталізує вимоги до апаратного та програмного забезпечення для успішного запуску та функціонування системи. Ефективне впровадження та експлуатація є невід'ємною частиною життєвого циклу будь-якого програмного продукту, гарантуючи його надійність та доступність для кінцевого користувача.

Тестування є фундаментальним етапом у життєвому циклі розробки програмного забезпечення, що забезпечує його якість, стабільність та

відповідність початковим вимогам. Для такої складної та динамічної системи, як бойова, тестування має бути всеосяжним, охоплюючи різні рівні абстракції – від окремих функціональних блоків до загальної функціональності. Розроблена бойова система може бути протестована за допомогою різних методів, що дозволяють виявити та усунути потенційні дефекти на ранніх стадіях розробки та підтримувати її надійність протягом усього періоду експлуатації.

4.1.1 Методи тестування

Для забезпечення всебічної перевірки функціоналу та стабільності бойової системи застосовуються такі основні методи тестування:

Юніт-тести (Unit Tests):

Призначення: Юніт-тести зосереджені на перевірці найменших, ізольованих одиниць коду – окремих функцій, методів, класів або компонентів. Їхня мета полягає в тому, щоб переконатися, що кожен окремий програмний елемент поводиться коректно відповідно до своїх специфікацій, незалежно від інших частин системи.

Застосування в проекті: У контексті бойової системи, юніт-тести можуть бути розроблені для перевірки:

- Коректності математичних формул розрахунку шкоди в UAttributeComponent (наприклад, перевірка, чи правильно обчислюється кінцева шкода з урахуванням броні та модифікаторів).
- Логіки управління кулдаунами здібностей у UAbilityComponent (чи правильно встановлюється та зменшується час перезарядки).
- Функціоналу управління здоров'ям та маною (чи правильно зменшуються/збільшуються показники, чи спрацьовує подія при досягненні нуля).

- Окремих функцій AI, що відповідають за вибір цілі або прийняття рішення про використання здібності.

Переваги: Раннє виявлення помилок, спрощення налагодження, підвищення якості коду, забезпечення регресійного тестування (гарантія, що нові зміни не порушують існуючий функціонал). В Unreal Engine, юніт-тести для C++ коду можуть бути інтегровані через Automation Framework, що дозволяє автоматизувати їх запуск.

Інтеграційні тести (Integration Tests):

Призначення: Інтеграційні тести перевіряють коректність взаємодії між різними компонентами та модулями системи. Вони допомагають переконатися, що окремо протестовані юніти працюють разом правильно, що дані коректно передаються між ними, і що залежності між модулями функціонують належним чином.

Застосування в проекті: Для бойової системи інтеграційні тести можуть включати:

- Перевірку взаємодії між UAbilityComponent та Combat Manager при активації здібності та подальшому розрахунку шкоди.
- Тестування коректності передачі даних від UAttributeComponent до UI Module для відображення HP-барів.
- Перевірку взаємодії AI ворога з Combat Manager та Ability System при виконанні атаки.
- Тестування життєвого циклу ефекту стану: від його застосування здібністю до його періодичної дії та зняття.

Переваги: Виявлення помилок інтерфейсів та протоколів взаємодії між модулями, забезпечення цілісності системи.

Функціональні тести (Functional Tests) / Системні тести (System Tests):

Призначення: Функціональні тести перевіряють функціональність системи в цілому з точки зору кінцевого користувача та відповідність її бізнес-вимогам. Вони симулюють дії користувача та перевіряють, чи працюють різні функції та сценарії використання системи відповідно до очікувань. Системні тести додатково перевіряють нефункціональні вимоги, такі як продуктивність, стабільність, безпека.

Застосування в проекті: Для бойової системи функціональні тести включають:

- Повний сценарій бою: ініціація бою, використання гравцем здібностей, реакція AI ворогів, розрахунок шкоди, візуальні та звукові ефекти, оновлення UI, завершення бою та видача нагород.
- Перевірка різних бойових сценаріїв: бій проти одного ворога, проти групи, бій з різними типами здібностей.
- Тестування граничних умов: що відбувається при критично низькому HP, при спробі використати здібність без ресурсів або на кулдауні.

4.1.2 Інструменти та підходи до тестування в Unreal Engine 5

Для реалізації тестування в середовищі Unreal Engine 5 використовується комбінація вбудованих інструментів рушія та загальноприйнятих підходів:

Вбудований Automation Framework: Unreal Engine надає власний Automation Framework для написання та запуску автоматизованих тестів для C++ коду. Це дозволяє створювати юніт- та інтеграційні тести, які виконуються

безпосередньо в рушії, що забезпечує точність перевірок у реальному ігровому середовищі.

Інструменти для налагодження Blueprints: Редактор Blueprints має потужні вбудовані інструменти для налагодження, що дозволяють покроково відстежувати виконання логіки, перевіряти значення змінних та виявляти помилки у візуальному скриптингу.

Play In Editor (PIE) та Standalone Game: Найбільш поширеним методом інтеграційного та функціонального тестування є запуск гри безпосередньо в редакторі (PIE) або як окремий виконуваний файл (Standalone Game). Це дозволяє розробникам та тестувальникам вручну взаємодіяти з бойовою системою, симулюючи дії гравця та спостерігаючи за поведінкою системи.

Unreal Insights: Це потужний інструмент профілювання, що дозволяє аналізувати продуктивність системи в реальному часі. Він використовується для виявлення "вузьких місць" у бойовій логіці, оптимізації алгоритмів розрахунку шкоди, AI та інших ресурсоємних операцій.

Система логування: Вбудована система логування Unreal Engine (UE_LOG) дозволяє виводити діагностичну інформацію про хід бойових дій, значення змінних та спрацьовування подій, що є незамінним для налагодження та аналізу поведінки системи.

Мануальне тестування (Playtesting): Для ігрових систем мануальне тестування, або "плейтестинг", є незамінним. Воно дозволяє оцінити ігровий досвід, баланс бою, інтуїтивність управління та загальне "відчуття" системи, що неможливо перевірити автоматизованими тестами.

Загальною метою тестування є не лише виявлення та усунення помилок, а й підтвердження того, що розроблена бойова система функціонує відповідно до очікувань, є стабільною, продуктивною та забезпечує якісний ігровий досвід. Регулярне тестування на всіх етапах розробки, від юнітів до повної системи, є гарантією успішного впровадження та експлуатації.

4.2 Вимоги до запуску ПЗ

Для успішного розгортання та функціонування розробленої бойової системи, що є частиною ігрового застосунку на Unreal Engine 5, необхідно дотримуватися певних вимог до апаратного та програмного забезпечення. Ці вимоги визначають мінімальні та рекомендовані конфігурації, при яких система може бути запущена та забезпечувати прийнятний рівень продуктивності.

4.2.1 Апаратне забезпечення

Вимоги до апаратного забезпечення для ігрових застосунків на Unreal Engine 5 є значно вищими, ніж для типових веб-додатків, що обумовлено складністю графіки, фізичних симуляцій та обчислювальної логіки.

У Таблиці 1.1 представлені основні вимоги до апаратного забезпечення для запуску розробленої бойової системи.

Компонент	Мінімальні вимоги	Рекомендовані вимоги
Процесор	Intel Core i5-8400 / AMD Ryzen 5 2600 або еквівалентний	Intel Core i7-12700K / AMD Ryzen 7 5800X3D або новіший (для максимальної продуктивності)
Оперативна пам'ять	Мінімум 16 ГБ	32 ГБ або більше (рекомендовано DDR4-3200 МГц або DDR5-5200 МГц+)
Жорсткий диск	Мінімум 50 ГБ вільного простору на SSD (NVMe рекомендовано)	100 ГБ+ вільного простору на NVMe SSD (для максимальної швидкості завантаження)

Графічна карта	NVIDIA GeForce RTX 2060 / AMD Radeon RX 5600 XT (6 ГБ VRAM)	NVIDIA GeForce RTX 3070 / AMD Radeon RX 6700 XT або краще (8 ГБ+ VRAM)
Монітор	Роздільна здатність 1920x1080 (Full HD)	Роздільна здатність 2560x1440 (QHD) з високою частотою оновлення (144 Гц+) або 3840x2160 (4K)
Звукова карта	Сумісна з DirectX	Сумісна з DirectX, з підтримкою об'ємного звуку 7.1

Таблиця 1.1. Основні вимоги до апаратного забезпечення

4.2.2 Програмне забезпечення

Окрім апаратних вимог, для коректного функціонування ППЗ необхідне відповідне програмне середовище.

Операційна система:

- Windows 10 (64-bit) або новіша версія.
- Підтримка DirectX 11 або DirectX 12 (для Windows).
- Підтримка Vulkan (для Windows, Linux).
- Драйвери графічної карти: Актуальні версії драйверів для встановленої графічної карти (NVIDIA GeForce Game Ready Driver, AMD Radeon Software). Це критично важливо для забезпечення стабільності та продуктивності графіки.

Додаткові компоненти:

- Microsoft Visual C++ Redistributable (зазвичай поставляється разом з інсталяційним пакетом гри).

4.2.3 Інсталяційний пакет та процес запуску

На відміну від веб-додатків, які вимагають встановлення додаткових середовищ виконання (наприклад, Node.js, Angular CLI), ігровий застосунок на Unreal Engine 5 розповсюджується як самодостатній виконуваний пакет.

Створення інсталяційного пакету (Packaging Project):

Процес підготовки до розповсюдження починається з "пакування" проекту в редакторі Unreal Engine. Ця функція (доступна через меню File -> Package Project) компілює весь код (C++ та Blueprints), збирає всі необхідні асети (моделі, текстури, анімації, звуки, карти) та конфігураційні файли в єдиний, оптимізований для розповсюдження пакет.

Під час пакування можна вибрати різні конфігурації збірки (наприклад, Development для налагодження або Shipping для фінального релізу), що впливає на розмір файлів та рівень оптимізації.

Результатом пакування є самодостатня папка з виконуваним файлом (.exe для Windows) та всіма необхідними залежностями.

Розповсюдження:

Отриманий інсталяційний пакет може бути розповсюджений безпосередньо (наприклад, через хмарне сховище) або через цифрові платформи дистрибуції (наприклад, Steam, Epic Games Store), які надають власні інструменти для встановлення та оновлення.

Процес запуску для кінцевого користувача:

Завантаження та розпакування/встановлення: Користувач завантажує архівний файл проекту або встановлює його через платформу дистрибуції. Якщо це просто архів, його необхідно розпакувати у бажану папку на комп'ютері.

Запуск виконуваного файлу: Для запуску гри користувачу достатньо запустити головний виконуваний файл (.exe) з розпакованої папки проекту. Жодних додаткових інструментів (таких як Node.js, Angular CLI) або складних командних рядків не потрібно. Усі необхідні бібліотеки та ресурси вже включені до пакету або автоматично встановлюються інсталятором.

Оновлення: У випадку випуску оновлень, вони зазвичай поширюються через патчі, які автоматично завантажуються та встановлюються через обрану платформу дистрибуції, або вручну, якщо проект розповсюджується безпосередньо.

Цей підхід значно спрощує процес впровадження та експлуатації для кінцевого користувача, роблячи систему максимально доступною та зручною у використанні.

Висновки до розділу 4

Розділ 4 детально розглянув критично важливі аспекти впровадження та експлуатації розробленої бойової системи. Було обґрунтовано необхідність комплексного тестування, що включає юніт-, інтеграційні та функціональні методи, для забезпечення високої якості та стабільності програмного забезпечення. Застосування вбудованих інструментів Unreal Engine 5 для тестування та налагодження підкреслює ефективність обраного інструментарію.

Крім того, було визначено чіткі вимоги до апаратного та програмного забезпечення, що забезпечують коректний та продуктивний запуск системи. Розроблена бойова система на Unreal Engine 5 вимагає сучасного апаратного забезпечення, включаючи багатоядерний процесор, достатній обсяг оперативної пам'яті та потужну графічну карту для забезпечення оптимального ігрового досвіду. Процес встановлення та запуску для кінцевого користувача є

максимально спрощеним, оскільки система розповсюджується як самодостатній виконуваний пакет, що не потребує додаткових середовищ виконання.

Загалом, рекомендації, викладені в цьому розділі, спрямовані на забезпечення надійного впровадження та безперебійної експлуатації розробленої бойової системи, що є запорукою її успішного функціонування та позитивного сприйняття користувачами.

ВИСНОВОК

У рамках даної дипломної роботи було успішно розроблено та реалізовано комплексну бойову систему для ігрового застосунку, використовуючи сучасний ігровий рушій Unreal Engine 5. Проект охопив усі ключові етапи розробки програмного забезпечення, починаючи від аналізу вимог та моделювання, і закінчуючи практичною реалізацією та рекомендаціями щодо впровадження.

На початкових етапах було проведено ретельний аналіз предметної області та визначено функціональні вимоги до бойової системи, що знайшло своє відображення у діаграмі прецедентів. Логічна модель даних була спроектована з використанням діаграми "Сутність-Зв'язок" (ERD), що дозволило чітко структурувати інформацію про персонажів, ворогів та їхні здібності, заклавши міцний фундамент для інформаційної бази. Динаміка взаємодії в бою була деталізована за допомогою діаграм активності та послідовності, що забезпечило глибоке розуміння логічних потоків та комунікації між компонентами системи.

Вибір Unreal Engine 5 як основного інструментарію розробки був обґрунтований його високою продуктивністю, гнучкістю, широким набором вбудованих інструментів та підтримкою гібридного програмування на C++ та Blueprints. Це дозволило ефективно реалізувати складні алгоритми бойової логіки, штучного інтелекту та взаємодії з користувацьким інтерфейсом. Організаційна структура програмного забезпечення базувалася на принципах

модульності та компонентного підходу, що забезпечило легкість розширення та підтримки системи. Ключові алгоритми, такі як розрахунок шкоди, управління здібностями та станами, були детально розроблені та реалізовані, демонструючи здатність системи до динамічної та реалістичної поведінки в бою.

Успішна реалізація бойової системи підтверджує ефективність застосованих підходів до проектування та розробки програмного забезпечення. Проект демонструє можливість вирішення складних інженерних завдань та впровадження передових технологій і методологій у створенні високопродуктивних інтерактивних систем.

В цілому, дана дипломна робота засвідчує комплексний підхід до проектування та розробки програмного забезпечення, надаючи міцну основу для подальших досліджень та вдосконалення бойових систем у комп'ютерних іграх.

Подальший розвиток системи може включати розширення функціоналу здібностей, впровадження багатокористувацького режиму, оптимізацію продуктивності для ширшого спектру апаратних конфігурацій та інтеграцію з більш складними ігровими механіками.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. **Epic Games.** *Event Dispatchers / Delegates Quick Start Guide*. https://dev.epicgames.com/documentation/en-us/unreal-engine/event-dispatchers-/-delegates-quick-start-guide

2. **Epic Games.** *Event Dispatchers (Delegates)*. https://dev.epicgames.com/community/learning/tutorials/ZdaB/event-dispatchers-delegates

3. **Epic Games.** *Epic C++ Coding Standard for Unreal Engine*. https://dev.epicgames.com/documentation/en-us/unreal-engine/epic-cplusplus-coding-standard-for-unreal-engine

4. **Epic Games.** *Unreal Engine Modules*. https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-modules

5. **Unreal Engine Forums.** *Event Dispatchers explained - Finally!*. https://forums.unrealengine.com/t/event-dispatchers-explained-finally/55570

6. **Unreal Engine Forums.** *Why use event dispatchers vs interface when ED's need a hard reference?*. https://forums.unrealengine.com/t/why-use-event-dispatchers-vs-interface-when-eds-need-a-hard-reference/1651829

7. **Unreal Engine Forums.** *Good and best practices in UE Architecture*. https://forums.unrealengine.com/t/good-and-best-practices-in-ue-architecture/1642211

8. **Unreal Engine Forums.** *Modular Design Principles - Actors and Components*. https://forums.unrealengine.com/t/modular-design-principles-actors-and-components/1807121

9. **Reddit.** *My personal 'Best Practices' for Unreal Engine*. https://www.reddit.com/r/unrealengine/comments/jx77xj/my_personal_best_practices_for_unreal_engine/

10. **YouTube.** *Event Dispatchers (Delegates) 101 in Unreal Engine 5* [Video]. https://www.youtube.com/watch?v=n5MC9EsbV7M

11. **YouTube.** *Event Dispatchers | Unreal Engine 5 Tutorial* [Video]. https://www.youtube.com/watch?v=r20VEPH_e0o

12. **YouTube.** *Event Dispatchers in UE5 are EASY! Simple STEP-BY-STEP Tutorial* [Video]. https://www.youtube.com/watch?v=uBl9kIdOT-k

13. **YouTube.** *How To Create A Melee Combat System in Unreal Engine* [Video]. https://www.youtube.com/watch?v=DC7XkWXAKoE

14. **YouTube.** *Unreal Engine 5 | Basic Melee Combat Tutorial Series* [Playlist]. https://www.youtube.com/playlist?list=PLlswSOADCx3fG8ATHm3GDmmdGGCfPhl0y

15. **YouTube.** *Learn Modular Programming - Unreal Engine 5's Blueprint* [Video]. https://www.youtube.com/watch?v=urSG0X4-r6s

ДОДАТОК А

Діаграма пакетів

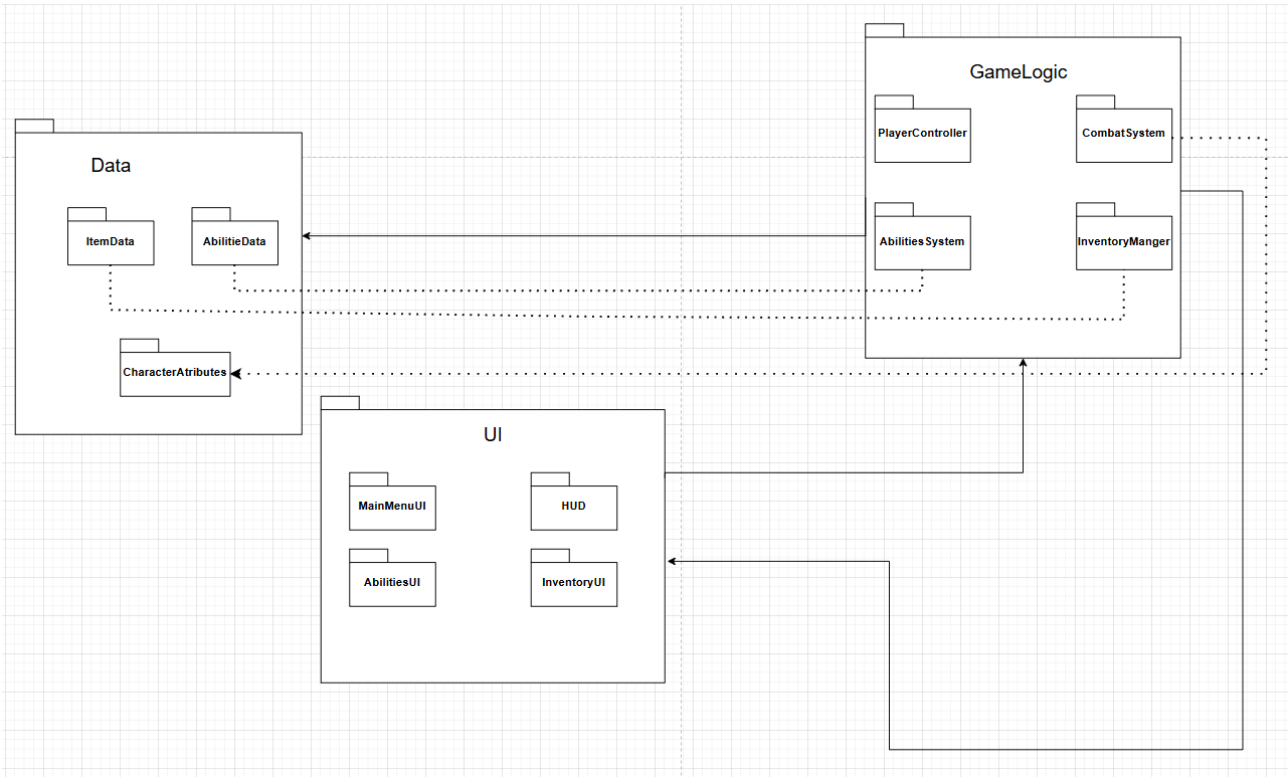

Рис. А.1 Діаграма пакетів програми

ДОДАТОК Б

Діаграма Розгортання

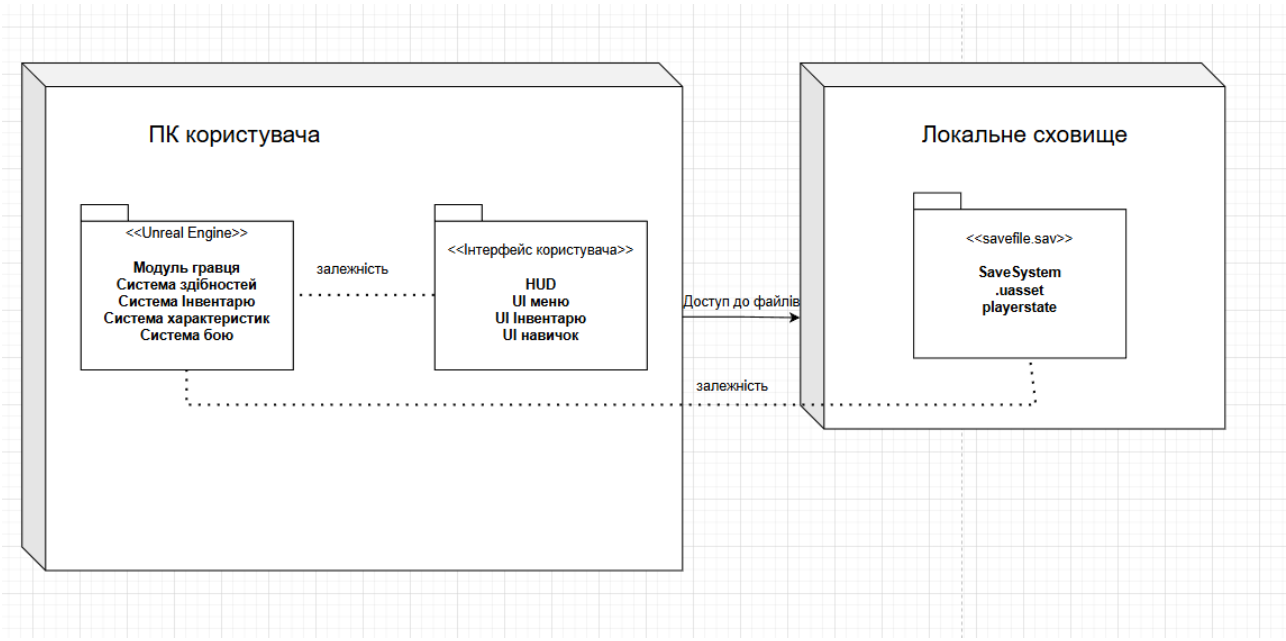

Рис. Б.1 Діаграма Розгортання

ДОДАТОК В

Блюпринт для трейсу персонажа гравця для сисетми таргетЛок

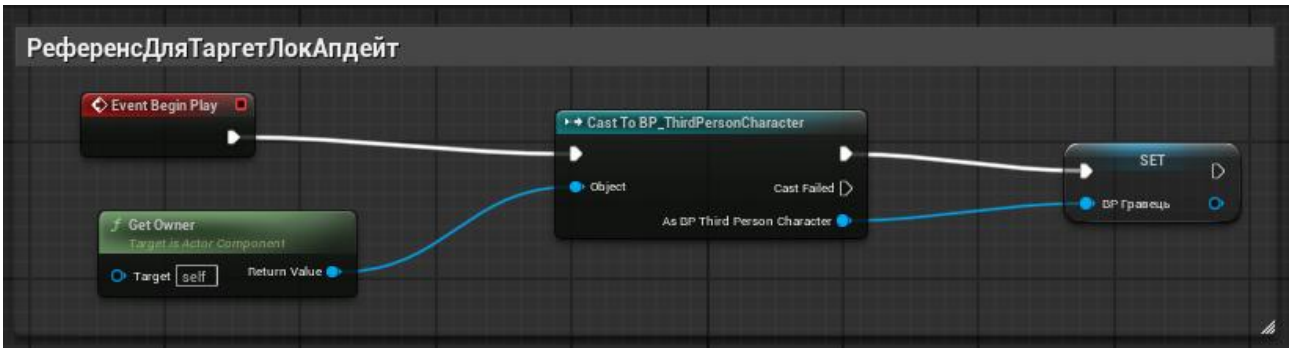

Рис. В.1 ТаргетЛокАпдейт

ДОДАТОК Г

Блюпринт всієї логіки системи ТаргетЛок

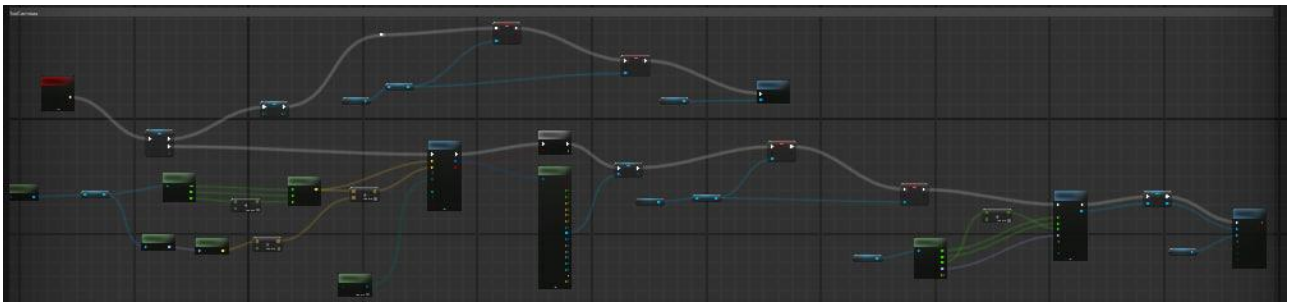

Рис. Г.1 ТаргетЛок система

ДОДАТОК Ґ

C++ Header код для блюпринта ТаргетЛок

```
/** Please add a class description */
UCLASS(Blueprintable, BlueprintType)
class UBPC_Бойова_Система : public UActorComponent
{
GENERATED_BODY()

public:

/** Please add a variable description */
UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
```

```
TObjectPtr<AActor> КамераЛок_Таргет;

/** Please add a variable description */
UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
TObjectPtr<ABP_ThirdPersonCharacter_C> BP_Гравець;

/** Please add a variable description */
UPROPERTY(BlueprintReadWrite, EditDefaultsOnly, Category="Default")
TObjectPtr<ABP_ТаргетЛок_Віджет_C> ТаргетЛок_Віджет;
};
```

ДОДАТОК Д

Блюпринт логіки постійного апдейту ТаргетЛоку

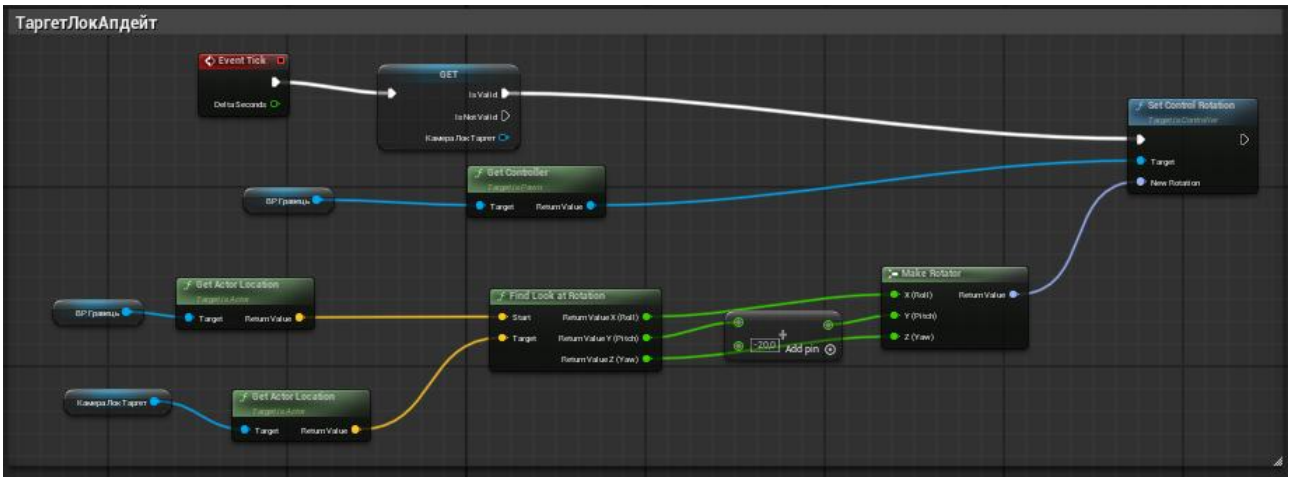

Рис. Д.1 ТаргетЛокАпдейт логіка

ДОДАТОК Е

Блюпринт логіки хітбоксів

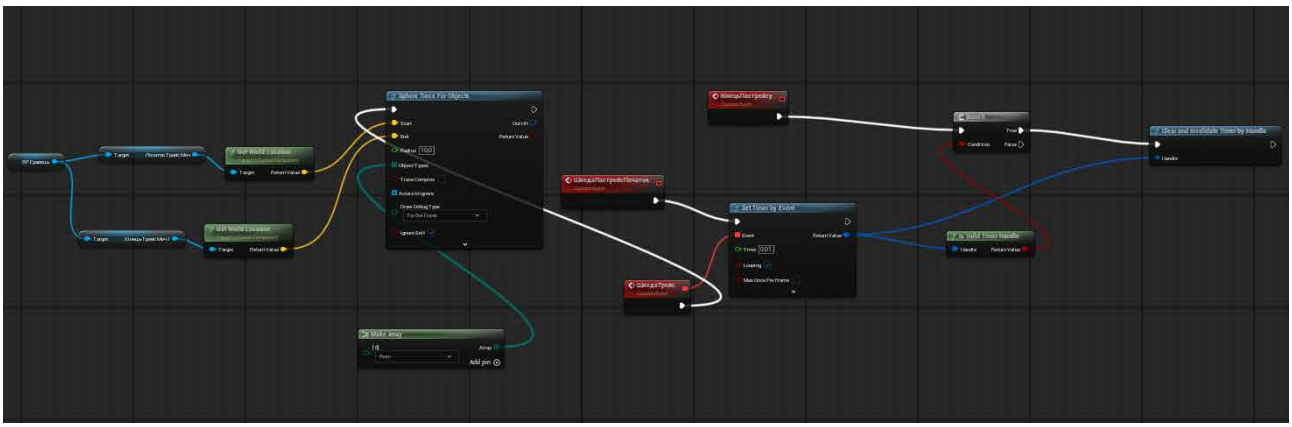

Рис. Е.1 Пастрейс Хітбоксів для меча

ДОДАТОК Є

Логіка програшу анімацій та системи комбо

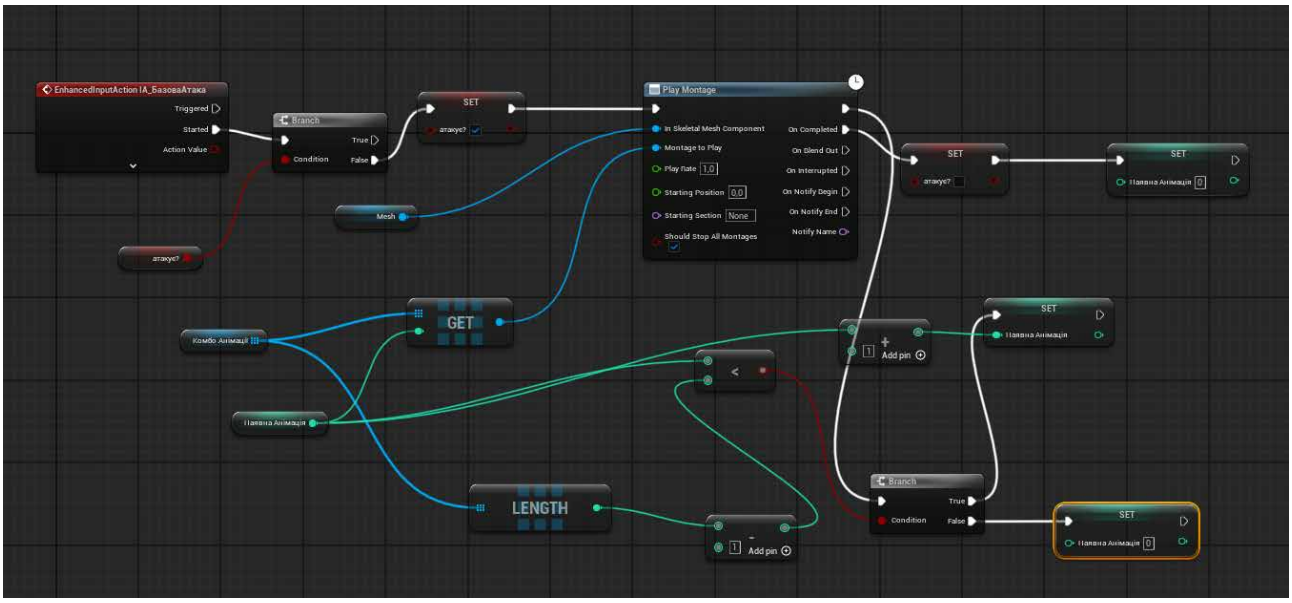

Рис. Є.1 Комбо та анімації

ДОДАТОК Ж

Блюпринт системи “Локомоушен”

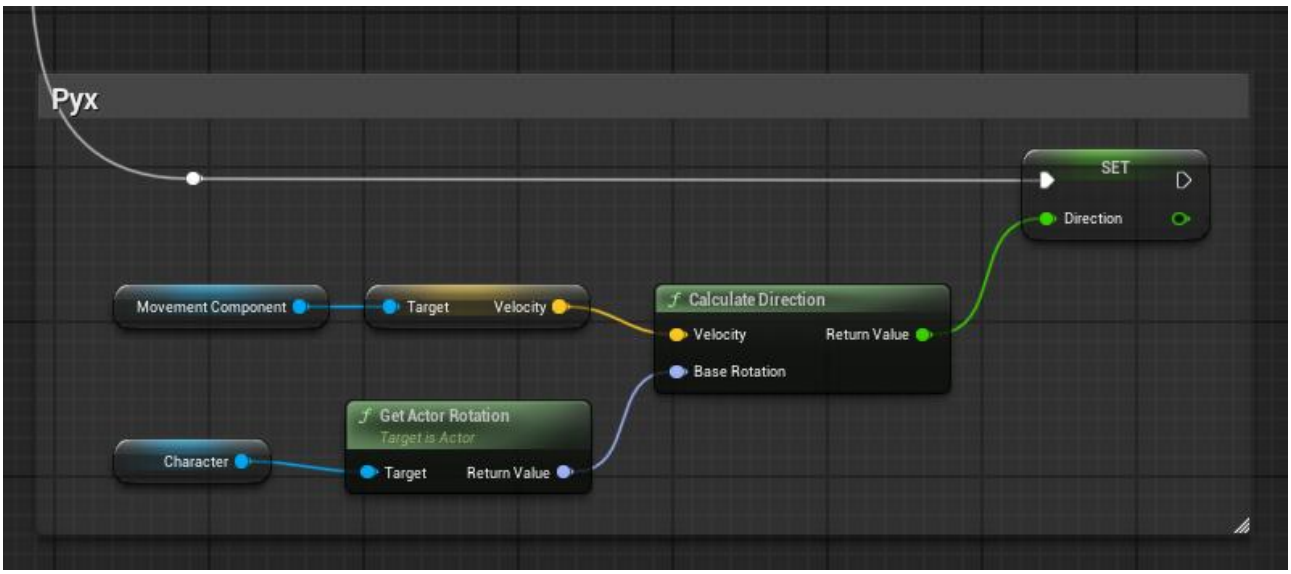

Рис. Ж.1 Система руху

ДОДАТОК З

Блюпринт системи “Локомоушен” частина 2

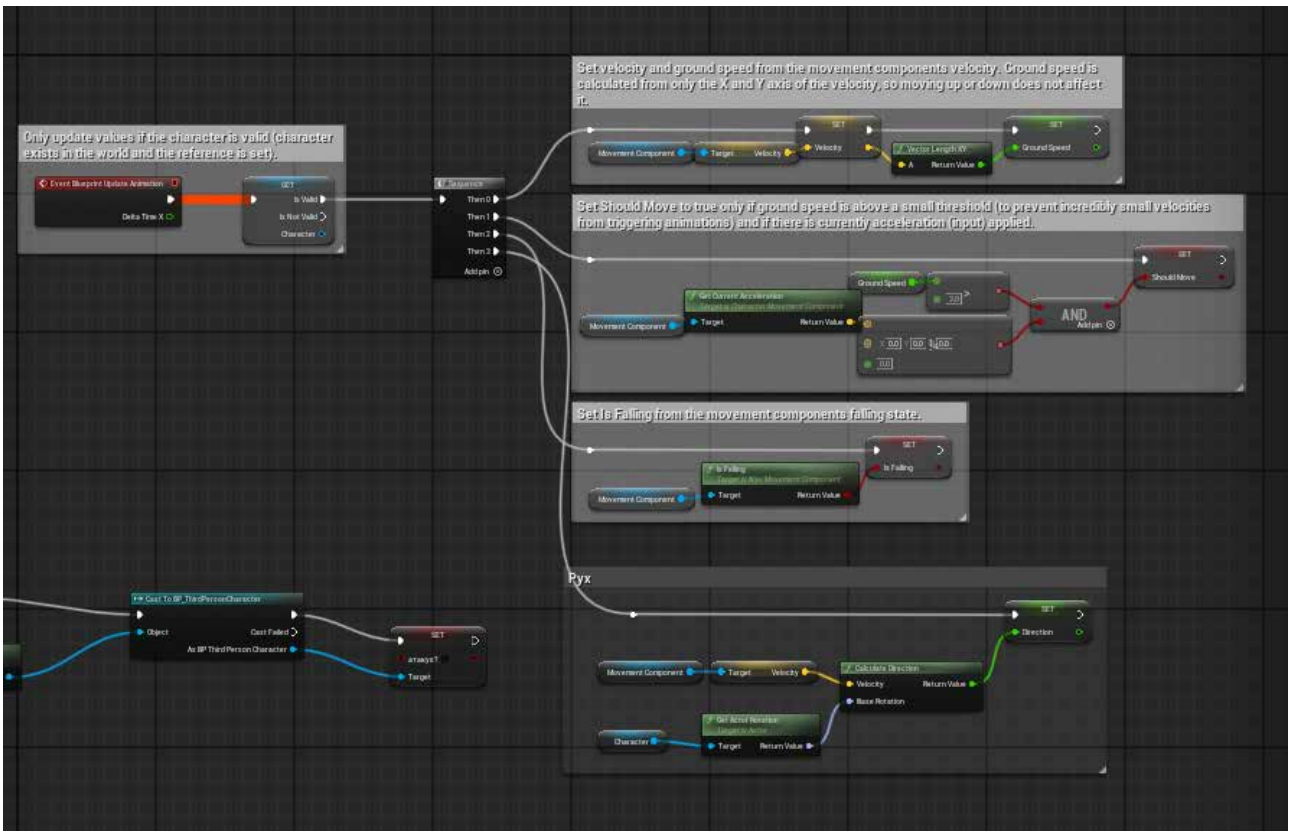

Рис. 3.1 Розширений погляд на логіку руху