Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

КОМП'ЮТЕРНА ГРА MOONLIGHT SONG З ПОКРАЩЕНИМ СПОСОБОМ КЕРУВАННЯ В СТИЛІ КЛАСИЧНИХ ІГОР

Виконала: студентка 4 курсу, групи 4КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Поліщук А. І.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Малініч І. П.
(прізвище та ініціали)
«04» червня 2025 р.

Рецензент: к.т.н., проф., доцент каф. АІІТ
Паламарчук Є. А.
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри Яровий А. А.
«06» червня 2025 р

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А. А.
«05» травня 2025 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Поліщук Альоні Ігорівні

1. Тема роботи: Комп'ютерна гра Moonlight Song з покращеним способом керування в стилі класичних ігор.
керівник роботи: Малініч Ілля Павлович, асистент каф. КН.
затверджені наказом вищого навчального закладу «20» березня 2025 року № 97

2. Термін подання студенткою роботи 30 травня 2025р.

3. Вихідні дані до роботи: мова програмування – об'єктно-орієнтована; кількість фіксованих камер – 4; інтерфейс користувача має бути інтуїтивно зрозумілий; кількість персонажів не менше 2; кількість способів управління не менше 2.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
вступ; аналіз предметної області; проєктування комп'ютерної гри Moonlight Song з покращеним способом керування в стилі класичних ігор; реалізація комп'ютерної гри Moonlight Song з покращеним способом керування в стилі класичних ігор; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): приклади використання 3д графіки, приклади камер в різних іграх, загальний план гри, UML-діаграма класів для предметів, UML-діаграма класів для NPC-персонажів, схема алгоритму мультимодального управління персонажем, схема опрацювання «танкового» управління, «зони» фіксованих камер, налаштування логіки активації фіксованої камери, приклад роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1–3	Малініч І. П., асистент каф. КН		

7. Дата видачі завдання 06 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Вступ. Аналіз предметної області	05.05.2025	07.05.2025	
2	Проєктування комп'ютерної гри Moonlight Song з покращеним способом керування в стилі класичних ігор	08.05.2025	11.05.2025	
3	Реалізація комп'ютерної гри Moonlight Song з покращеним способом керування в стилі класичних ігор	12.05.2025	15.05.2025	
4	Висновки та аналіз отриманих результатів	16.05.2025	26.05.2025	
5	Оформлення додатків	27.05.2025	29.05.2025	
6	Розробка інструкції користувача	30.05.2025	01.06.2025	
7	Оформлення матеріалів до захисту БКР	02.06.2025	04.06.2025	

Студентка _____ **Поліщук А. І.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Малініч І. П.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 90 сторінок формату А4, які включають 53 рисунків і 1 таблицю. У списку використаних джерел зазначено 28 найменувань.

Метою роботи є розробка комп’ютерної гри «Moonlight Song» з покращеним способом керування в стилі класичних ігор.

У загальній частині роботи на основі аналізу існуючих відеоігор, методів та технологій доведена доцільність розробки комп’ютерної гри Moonlight Song. Спроєктовано алгоритм роботи управління та UML діаграми класів предметів та персонажів комп’ютерної гри Moonlight Song, які спрощують розробку та розуміння структури програмного забезпечення. Програмний продукт реалізований з використанням сучасного ігрового рушія Unreal Engine 5 та Blender.

Реалізовано програмний продукт та проведено його тестування. Результати тестування розробки вказують на те, що ігровий процес комп'ютерної гри Moonlight Song повноцінний та всі поставлені завдання було виконано.

Ключові слова: 3D-гра, класичне управління, Unreal Engine 5, Blender.

ABSTRACT

The bachelor's qualification work consists of 90 pages of A4 format, which include 53 figures and 1 table. The list of used sources includes 28 items.

The purpose of the work is to develop a computer game "Moonlight Song" with an improved control method in the style of classic games.

In the general part of the work, based on the analysis of existing video games, methods and technologies, the feasibility of developing a computer game Moonlight Song is proven. An algorithm for the operation of "tank" control and UML diagrams of classes of objects and characters of the computer game Moonlight Song are designed, which simplify the development and understanding of the software structure. The software product is implemented using the modern game engine Unreal Engine 5 and Blender.

The software product is implemented and tested. The results of the development testing indicate that the gameplay of the computer game Moonlight Song is complete and all the tasks have been completed.

Keywords: 3D game, classic control, Unreal Engine 5, Blender.

ЗМІСТ

ВСТУП

Актуальність дослідження.

В теперішньому часі індустрія відеоігор має великий вплив на покоління, чиє зростання та розвиток припали на момент її розквіту. Також багато людей лише починають відкривати для себе ігровий світ, який з кожним роком стає різноманітнішим та доступнішим, пропонуючи унікальний досвід.

На тлі цього розвитку спостерігається повернення інтересу до класичних ігор які були популярними у 90-х – на початку 2000-х років, та сформували фундамент сучасного геймдизайну. Серед них – ігри з фіксованою камерою та танковим управлінням. Тема бакалаврської роботи «Комп'ютерна гра Moonlight Song з покращеним способом керування в стилі класичних ігор» є актуальною з огляду на декілька ключових аспектів:

- Унікальність: Сучасні ігри здебільшого мають відкритий світ та надають перевагу вільному русі камерою. Саме тому фіксовані камери можуть виділити проект серед безлічі інших, пропонуючи гравцям дещо інше – структуроване середовище, посилене занурення в атмосферу гри та сфокусовану візуальну композицію.
- Unreal Engine 5 – ігровий рушій, який відомий як зручне середовище для розробки 3D проектів та тим що надає широкі можливості для реалізації стилізованої графіки. Завдяки використанню технологій Unreal Engine 5 можна втілити динамічне освітлення та кінематографічну атмосферу, не жертвуючи продуктивністю гри.
- Ностальгія: Фіксовані камери та танкове управління – елементи, що були характерні для багатьох класичних ігор 90-років. Повернення до цих елементів геймплею може стати чудовим прикладом того, як поєднувати сучасні технології з класичними механіками.

Отже, створення комп'ютерної гри Moonlight Song з покращеним способом керування в стилі класичних ігор на рушії Unreal Engine 5 є важливою та актуальною темою в умовах сучасних тенденцій геймдизайну, оскільки буде

незвичайним поєднанням сучасних технологій розробки з класичним підходом до геймплею.

Метою дослідження є покращення способу керування у відеоіграх з танковим управлінням.

Об’єктом дослідження є ігровий процес та ігрова механіка.

Предметом дослідження є програмні засоби для реалізації комп’ютерної гри Moonlight Song з покращеним способом керування в стилі класичних ігор на ігровому рушії Unreal Engine 5.

Задачі дослідження:

1. Провести аналіз існуючих ігор у жанрах 3D та ігор з різними видами управління та обґрунтувати доцільність розробки;
2. Спроєктувати комп’ютерну гру Moonlight Song з покращеним способом керування в стилі класичних ігор;
3. Програмно реалізувати комп’ютерну гру Moonlight Song з покращеним способом керування в стилі класичних ігор за допомогою ігрового рушія Unreal Engine 5;
4. Протестувати та проаналізувати отримані результати;
5. Розробити інструкцію користувача.

Апробація роботи.

Результати досліджень було апробовано на LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) м. Вінниці у 2025 р. [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд ігрової індустрії та жанрів відеоігор

Розвиток ігрової індустрії розпочався приблизно в 90-х та 2000-х роках, коли відеоігри почали переходити з простих піксельних аркад до повноцінних тривимірних світів. Завдяки зростаючому інтересу з боку людей, пізніше появились нові покоління ігрових консолей та графічних процесорів. І лише за останнє десятиліття ігрова індустрія зробила величезний стрибок уперед – з’явилися шоломи віртуальної реальності, хмарні сервіси, можливість грати з різних пристроїв завдяки кросплатформерності та запровадження штучного інтелекту в NPC для покращення ігрового процесу.

Завдяки цьому технологічному прориву відеогеймерство стало доступнішим для багатьох гравців. Якщо раніше для такого задоволення потрібно було мати дорогі консолі та потужні комп’ютери, то на сьогоднішній день кожен може зіграти навіть на смартфоні або бюджетному ноутбуці. Платформи на зразок Steam, Epic Games Store чи Xbox Game Pass надають швидкий доступ до тисячі ігор і навіть час від часу пропонують вигідні знижки або ж безкоштовно отримати копії ігор які є досить популярними серед відеогеймерів.

Також розробники все частіше забезпечують комфорт навіть для людей з обмеженими можливостями – озвучення тексту та активація субтитрів, фільтри для різних видів дальтонізму, адаптивне управління та можливість налаштовувати розмір шрифтів. Крім того, деякі компанії розробили спеціальні контролери з урахуванням потреб людей з інвалідністю, наприклад як Xbox Adaptive Controller (рис. 1.1) [2].

Рисунок 1.1 – Xbox Adaptive Controller для людей з інвалідністю

Загалом ігрова індустрія перетворилася на одну з найвпливовіших та прибуткових сфер розваг. Відеоігри об’єднують мільйони гравців різного віку, культур та вподобань – кожен може знайти для себе свій комфортний стиль геймінгу – будь то сюжетна RPG, змагальні шутери, чи ігри з свободою вибору та їх наслідками [3].

Яскравим прикладом цього стала поява Baldur’s Gate 3 – рольова гра заснована на правилах настільної гри Dungeons and Dragons яка ж одразу завоювала любов та увагу більшості геймерів. В ній нам надають можливість створення власного персонажа (стать, раса і загальна зовнішність) з його навичками та передісторією, що в певних моментах буде впливати на розвиток сюжету, діалогів та ставлення інших персонажів [4].

Геймплей базується на покроковій системі, яка надає нам свободу у виборі тактики. Персонаж може атакувати різними видами зброї, лікувати себе або товаришів по команді, переміщатись на полі бою, або ж навіть втекти подалі від конфлікту взамін на бали дій, кількість яких також варіюється від певних навичок. На рисунку 1.2 зображено ігровий процес Baldur’s Gate 3.

Рисунок 1.2 – Ігровий процес Baldur’s Gate 3

Ключовою особливістю Baldur’s Gate 3 є використання механіки кидання кубика (рис. 1.3) під час прийняття рішень або перевірки навичок. Двадцятигранний кубик з’являтиметься на екрані і визначатиме успіх або невдачу дії – будь то відкриття замка або знешкодження пастки. Така механіка не лише передає напругу, а й передає атмосферу класичних настільних ігор.

Рисунок 1.3 – Ігрова механіка в Baldur’s Gate 3

У грі наявний кооперативний режим – який може вміщати в себе чотирьох гравців, які можуть спільно проходити сюжет, приймати як власні так і колективні рішення та мати різні стосунки з компаньйонами – дружні, романтичні або ворожі.

Великою популярністю серед геймерів користується серія ігор Resident Evil, яка розпочала свій шлях ще з далеких 90-років, коли вийшла перша частина яка поклала початок жанру survival horror. Нагнітаюча атмосфера, обмежені ресурси та зіткнення з монстрами створювало неабияке відчуття тривоги, особливо завдяки обмеженому полі зору – фіксованій камері, яка створювала ефект кінематографічності [5].

На рисунку 1.4 зображено ігровий процес Resident Evil 1.

Рисунок 1.4 – Ігровий процес Resident Evil 1

Гравець не мав повного контролю над оглядом місцевості – камера заздалегідь була розміщена в певних точках, які не дозволяли вільно обертати кут огляду що посилювало відчуття тривоги. І також ця особливість дозволяла розробникам більше попрацювати над атмосферою – композиція кадру, розміщення об’єктів, світла й тіні.

Але саме більше ця серія ігор запам’яталась тим що в ранніх частинах для збереження свого прогресу, потрібно було мати в інвентарі друкарську стрічку та знайти кімнату з друкарською машинкою. Через це гравці мали планувати заздалегідь кожне своє збереження через обмежену кількість стрічок.

Одночасно з іграми Resident Evil, великої популярності набули ігри від Konami, а саме їхня серія ігор Silent Hill [6]. Перша частина вийшла ще в далекому 1999 році, і одною її особливістю стало використання густого туману в місті, який слугував не тільки як художній прийом для створення похмурої атмосфери, а й також був технічним рішенням – туман допомагав оптимізувати гру для апаратного забезпечення першої Playstation. За допомогою нього розробники могли приховати низький рівень деталізації вдалечині, зменшуючи навантаження на систему. На рисунку 1.5 наведено ігровий процес Silent Hill 1.

Рисунок 1.5 – Ігровий процес Silent Hill 1

З розвитком ігрової індустрії, серед класичних одиночних survival horror-ігор, також з’явився жанр багатокористувацьких ігор – де гравці можуть як співпрацювати так і протистояти одне одному. Однією з найвідоміших стала гра Dead by Daylight яка вийшла у 2016 році від студії Behaviour Interactive, в якій

об’єдналися ці два жанри – хоррор та мультиплеєр. Одна сторона має чотирьох гравців які називаються вцілілими, а інша – одного гравця який грає за вбивцю і повинен перешкодити ремонтуванню генераторів та якнайшвидше позбутись всіх гравців перш ніж вони втекли через вихід який відчиняється після відремонтування всіх 5 генераторів [7].

Рисунок 1.6 – Ігровий процес за вцілілого в Dead by Daylight

Але крім цього, гра пропонує для цих двох ролей різний тип камери – для вцілілих використовується third-person view (третя особа) – камера розташована позаду персонажа для кращої орієнтації в просторі та уникання вбивці. Натомість вбивця грає від first-person view (перша особа) – що значно обмежує поле зору але посилює концентрацію та напруження, адже гравець повинен покладатись на звуки, сліди та власну уважність. На рисунку 1.6 наведено ігровий процес за вцілілого, а на рисунку 1.7 – за вбивцю.

Рисунок 1.7 – Ігровий процес за вбивцю в Dead by Daylight

Завдяки своїй популярності яка зростає з кожним роком, розробники отримали можливість укласти співпрацю (collaboration) з відомими франшизами (Resident Evil, Silent Hill, Alien, Stranger Things, Saw та багатьма іншими) що привернуло увагу фанатів так і просто гравців які знайомі з цими світами.

Успішність гри стала також поштовхом до впровадження івентів (event) – тимчасових подій, які додають нові механіки та випробовування. Прикладом може слугувати подія «2&8» яка з недавніх під стала улюбленою серед гравців через свій новий формат – двоє вбивць та вісім вцілілих.

Окрім цього, у 2022 році в світ вийшла досить колоритна гра – ANNO: Mutationem яка одразу привертає увагу своїм 2.5D стилем та піксельною графікою. Гра заворожує кіберпанківською атмосферою та унікальним візуальним стилем. На рисунку 1.8 наведено приклад ігрового процесу ANNO: Mutationem [8].

Рисунок 1.8 – Ігровий процес ANNO: Mutationem

1.2 Огляд графічної частини відеоігор

Графічна складова в іграх є важливим елементом через який розробники не лише передають візуальну привабливість, а й створюють загальну атмосферу й емоційне сприйняття. В залежності від свого жанру, ігри можуть мати різний підхід до візуального оформлення. Від реалістичних шутерів, що прагнуть максимального занурення та деталізації, до яскравих, стилізованих платформерів чи ігор-головоломок з мінімалістичним дизайном.

Тривимірна графіка – це потужний інструмент ігрової індустрії який використовують ще з 90-х років і який з часом зазнав значної еволюції. За допомогою 3D графіки багато розробників отримали змогу створити більш глибші та динамічніші світи які приваблювали своєю новизною – адже зображення більше не були пласкими і гравець отримував змогу вільно переміщатись у просторі [9].

Пізніше це стало популяризуватись серед багатьох проєктів, що навіть згодом призвело до поступового витіснення 2D у багатьох жанрах. Перевагою 3D-моделей стала їх гнучкість – скелетна система (rigging) яка створювалась

всередині моделі. За допомогою набору кісток (bones) які були пов’язані між собою певною ієрархією, розробники легко та ефективно могли анімувати персонажів – їх біг, ходьбу, бій і інші рухи, на відміну від 2D графіки, де кожен рух потрібно було малювати, кадр за кадром [10].

У сучасному геймдизайні 3D- графіка поділяється на два напрями:

1. Малополігональна (low-poly) – це стиль, що характеризується використанням мінімальної кількості полігонів для створення моделей персонажів, об’єктів та середовища. Завдяки спрощеній геометрії такі світи мають характерний «кубічний» або «блоковий» вигляд, який часто виглядає стилізовано і навіть художньо. Також малополігональна графіка популярна серед інді-розробників через її легкість для обробки та швидкості рендерингу, що дозволяє іграм працювати навіть на пристроях із обмеженими технічними ресурсами.

2. Багатополігональна (high-poly), або як її ще називають реалістична – застосовується переважно у великих проєктах, де потрібно максимально точно відтворити деталі та тукстури об’єктів, персонажів і середовища. В ній використовується велика кількість полігонів, що дозволяє створити плавні контури, реалістичні поверхні, складне освітлення та тіні, а також детальну анімацію. Реалістична графіка вимагає потужного апаратного забезпечення та великої кількості ресурсів на розробку, але результат вартий зусиль.

Одним з яскравих прикладів застосування малополігональної графіки та її еволюції у багатополігональну, знову ж таки є гра Resident Evil 1. Оригінальна версія була випущена у 1996 році та мала досить характерну для того часу графіку з обмеженою деталізацією моделей. На рисунку 1.9 наведено приклад використання малополігональної графіки в Resident Evil 1(1996).

Рисунок 1.9 – Приклад використання малополігональної графіки в Resident Evil 1(1996)

І не зважаючи на те що оригінальна версія мала досить примітивну малополігональну графіку, гра змогла створити напружену атмосферу, чим завоювала серця багатьох фанатів і стала класикою жанру survival horror.

Пізніше, завдяки технічному прогресу, гра отримала другий шанс нагадати про себе, отримавши ремастер – версію з оновленою графікою, де моделі тепер мали кращу деталізацію, освітлення було більш реалістичним, а текстури – якіснішими.

У грі збереглись основні механіки – фіксована камера та танкове управління, повертаючи класичну атмосферу оригіналу та все ще присутнє відчуття напруги від обмеженості поля зору. Складне танкове управління запам'яталося як одна з визначальних рис гри, яка змушувала гравців діяти обережно та стратегічно.

На рисунку 1.10 наведено приклад використання багатополігональної графіки в ремастері Resident Evil 1(2002)

Рисунок 1.10 – Приклад використання багатополігональної графіки в ремастері Resident Evil 1(2002)

Гарним прикладом використання малополігональної графіки в сучасній ігровій індустрії є інді-гра Rental, яка вийшла у березні 2024 року. Завдяки своєму унікальному візуальному стилю – графіці з простими але стилізованими моделями, приглушеною кольоровою палітрою та використанням певних ефектів для створення тривожного, а часом сюрреалістичного настрою, гра виділилась серед безлічі інших інді-релізів і здобула популярність, особливо від шанувальників психологічних трилерів [11].

Не зважаючи на свою простоту графіки, розробник досить добре попрацював над емоційним сприйняттям. Гра вміло використовує неочікувані візуальні ефекти та звуки аби створити дискомфорт і напругу у гравця, незважаючи на свій миловидний та безневинний вигляд на початку.

Таке поєднання ще раз підтверджує, що графіка насамперед це інструмент передачі атмосфери, стилю та наративу, а не лише засіб демонстрації технічних можливостей

На рисунку 1.11 наведено приклад використання стилізованої 3D графіки у Rental.

Рисунок 1.11 – Приклад використання стилізованої 3D графіки у Rental

Також цікавим є використання 3D графіки в поєднанні з стилем 2D, де текстури моделі імітують малюнок за допомогою чітких контурів. На рисунку 1.12 наведено приклад з використанням техніки «cel-shading» в грі Bendy and the Ink Machine [12].

Рисунок 1.12 – Приклад з використанням техніки «cel-shading» в грі Bendy and the Ink Machine

Незалежно від того, чи це малополігональна графіка, що зачаровує своєю стилізацією та простотою, чи реалістична, що прагне до максимальної деталізації та занурення, ключовим залишається її призначення. За грамотного застосування, гра яка має гармонійне поєднання між графікою, геймплеєм та історією здатна стати чимось унікальним, що запам’ятається ще на довгі часи.

1.3 Аналіз способів керування в іграх

Система керування – є одним із ключових компонентів ігрового процесу, що впливає на комфорт та ігровий досвід геймера. Від того наскільки інтуїтивно зрозумілим буде управління, залежить здатність гравця швидко адаптуватися до механік гри, отримувати задоволення від процесу та ефективно взаємодіяти з ігровим середовищем.

Основним аспектом є доступність – керування грою повинно бути простим у вивчені та використанні, враховуючи фізичні обмеження людини. Дизайн управління має брати до уваги обмеження наших рук [13].

Існує три основні «групи пальців», які розробники враховують під час розробки елементів керування:

- Першочерговий контроль – великий та вказівний пальці. Досить гнучкі та точні, тому використовуються для основних дій.
- Другорядний контроль – середній палець. Гнучкий, але не такий точний, тому використовується для дій утримання (прицілювання, натискання клавіші W для ходьби тощо.)
- Підтримка – безіменний палець та мізинець. Слабкі та не дуже гнучкі, тому використовуються для додаткових дій.

Такий поділ функцій між пальцями дає розробникам точну уяву про те як розташувати елементи керування для комфорту та ефективності. Продумане управління не викликає фізичного навантаження на руки під час тривалих

ігрових сесій та знижує ризик дискомфорту або травм, таких як тунельний синдром.

На рисунку 1.13 наведено розподіл пальців на групи під час керування.

Рисунок 1.13 – Розподіл пальців на групи під час керування

Крім анатомічного поділу функцій пальців, ще одним головним аспектом є вибір пристрою керування – адже різні пристрої мають свої особливості, переваги та обмеження. Основні види пристроїв керування, що застосовуються в іграх, включають в себе:

- Клавіатура та миша. Найбільш поширений вибір для ПК-геймінгу, що забезпечує велику точність і швидкість реакції. Клавіатура забезпечує велику кількість комбінацій клавіш, а миша – точне позиціонування та швидкий огляд, що особливо важливе у шутерах. Налаштування клавіш під власні вподобання, дозволяє гравцю адаптувати управління гри під себе, проте для новачків ця система може вимагати певної вправності та певного часу для освоєння.
- Геймпад (контролер). Стандартний вибір для консолей, який також часто використовується і на ПК. Комфортна форма геймпада розроблена з

урахуванням анатомії руки, що дозволяє довго грати без дискомфорту. Стіки та кнопки дозволяють плавне управління рухом персонажа або камерою, а також наявна підтримка тактильного зворотного зв’язку(вібрації), що посилює відчуття занурення. Основним обмеженням є менша кількість кнопок у порівнянні з клавіатурою, та відносно нижча точність прицілювання, що є значним недоліком для шутерів.

- Сенсорний екран – головний спосіб керування на мобільних пристроях та планшетах. Дотики та свайпи дозволяють інтуїтивно взаємодіяти з ігровим світом без додаткових пристроїв. Однак, обмежена точність може ускладнювати виконання складних дій. Крім того, тривале тримання пальців на екрані може закривати частину ігрового поля, що ускладнює огляд.
- VR-шолом та контролери. Один з найсучасніших типів керування, що дозволяє максимально зануритися в ігровий світ, але також не дуже розповсюджений порівняно з іншими пристроями через свою вартість. Рухи гравця відслідковуються у віртуальному просторі, дозволяючи йому взаємодіяти з віртуальним світом за допомогою рухів рук, голови та тіла. Але довге використання може викликати дискомфорт, нудоту і запаморочення в голові.
- Ігровий руль з педалями – спеціалізоване обладнання для автосимуляторів та гоночних ігор, яке максимально точно відтворює реалістичні умови водіння. Такий пристрій значно відтворює занурення, але його використання обмежується відповідними жанрами і потребує певного простору для установки.

Кожен з перелічених пристроїв керування має свої переваги та недоліки, які розробники враховують при проєктуванні управління гри. На сьогоднішній день, ігрова індустрія все частіше пропонує мультиплатформерний підхід, адаптуючи систему керування під кілька пристроїв одночасно, будь це мишка з клавіатурою чи контролер, залучаючи за допомогою цього все більшу аудиторію із різних платформ. Така універсальність дозволяє гравцям самим обирати вид управління який найбільше відповідає їхнім уподобанням.

Ще одним зручним доповненням під час розробки управління є можливість біндінгу – тобто переназначення клавіш під потреби гравця. Часто коли розробники ігнорують важливість зручності управління або намагаються експериментувати без врахування потреб користувачів, це призводить до заплутаних або незручних моментів під час геймплею. Прикладом такого невдалого підходу можна назвати гру Resident Evil Village.

Під час проходження гри, багато гравців на клавіатурі стикнулися з проблемою пов’язаною з розміщенням основних дій на непривичних кнопках – клавіша Ctrl, яка в більшості ігор традиційно використовується для присідання, у Resident Evil Village була призначена для дії лікування. Це нестандартне призначення викликало обурення та розчарування коли гравці випадково витрачали цінні лікувальні ресурси замість того щоб присісти.

На рисунку 1.14 наведено помилкове використання клавіші з зміненим призначенням у Resident Evil Village.

Рисунок 1.14 – Помилкове використання клавіші з зміненим призначенням у Resident Evil Village

Наявність можливості змінити призначення клавіш дозволила гравцям налаштувати управління під власний комфорт, усуваючи можливість випадкових помилок під час гри.

Якщо частою проблемою на клавіатурі є велика кількість клавіш, які не завжди мають однакове призначення, що може викликати плутанину – то використання контролеру може бути комфортною заміною: менше кнопок і їх розташування інтуїтивно зрозуміле. На сьогоднішній день для користувачів геймпаду та сенсорних екранів розробники впроваджують допоміжні функції, до прикладу – автоматичне прицілювання (aim assist), яке полегшує наведення на ціль, що значно допомагає користувачам у динамічних іграх, особливо шутерах. Наприклад, у грі Fortnite – така система авто прицілювання дозволяє гравцям консолей та мобільних пристроїв конкурувати з користувачами клавіатури на рівних умовах. Розробники пропонують ряд налаштувань, які гравець може адаптувати під свої потреби:

- Precision Aim Assist Strength – сила притягування прицілу до цілі при точному наведені.
- Tracking Aim Assist Strength – сила допоміжного корегування при слідуванні за рухомою ціллю, що знижує потребу в постійному корегуванні.

На рисунку 1.15 наведено налаштування авто прицілювання у грі Fortnite.

Рисунок 1.15 – Налаштування авто прицілювання у грі Fortnite

Окрім допоміжних функцій та варіативності пристроїв керування, ігрова індустрія також впровадила різні типи управління персонажем. На ранніх етапах розвитку ігрової індустрії керування персонажем у 2D іграх відбувалося на прості кнопки що відповідали за рух вліво, вправо та стрибок – до прикладу, гра Sonic the Hedgehog, випущена у 1991 році, в якій використовувався цей тип управління. На рисунку 1.16 наведено приклад управління у 2D-платформері Sonic the Hedgehog.

Рисунок 1.16 – Приклад управління у 2D-платформері Sonic the Hedgehog

З появою тривимірної графіки, управління персонажем значно ускладнилося, оскільки включало в себе не лише напрямки вліво та вправо, а й вперед, назад і також обертання камери, щоб змінювати кут огляду. На джойстику задля цього використовувались два стіки: лівий – відповідав за управління персонажем у просторі, а правий – за обертання камери. На клавіатурі цей процес відбувався простіше, оскільки обертання камерою відбувалося за допомогою мишки, а керування персонажем на WASD-клавіші. Тогочасною схемою керування яка була характерною в ті часи, було «танкове» управління (tank

controls), в якій персонаж рухався відносно власного положення, незалежно від кута камери. Така система вимагала від гравця адаптації, але дозволяла точніше орієнтуватися в просторі з фіксованими ракурсами.

Певні ігри полегшували процес огляду простору, прикріплюючи камеру вслід за персонажем – камера автоматично розверталась у бік руху гравця, що дозволяло йому не витрачати зайвий час на регулювання вручну. Прикладом такого підходу є гра Silent Hill 3, де камера зберігає персонажа у центрі екрану, плавно повертаючись у напрямку його руху. На рисунку 1.17 наведено приклад слідкуючої камери з танковим управлінням у грі Silent Hill 3 [14].

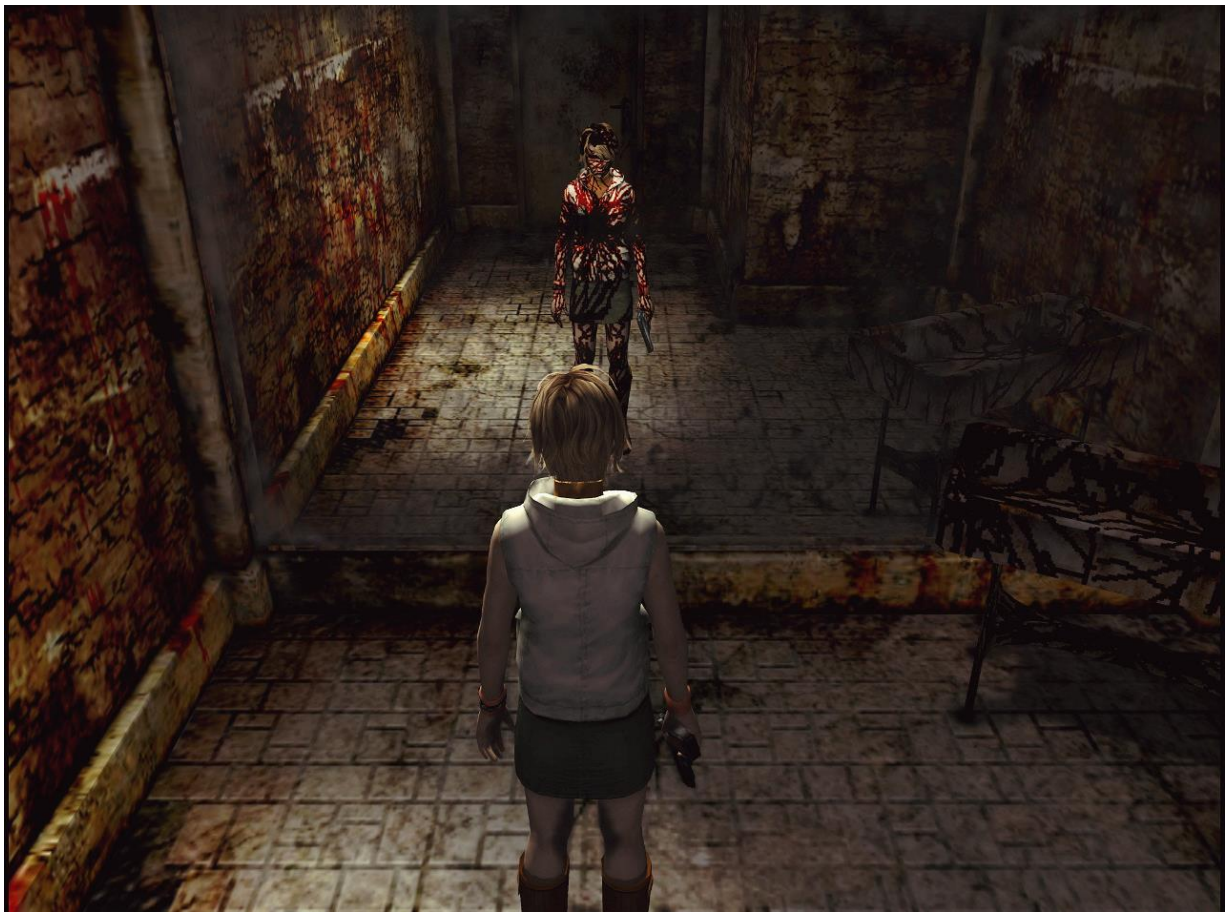

Рисунок 1.17 – Приклад слідкуючої камери з «танковим» управлінням у 3D-грі Silent Hill 3

Натомість у серії ігор Resident Evil, в ранніх частинах використовувалася система фіксованих камер, де камера була закріплена в певних точках локацій, і

кут огляду змінювався лише при переході персонажа на іншу зону. Це не вимагало від гравця лишній раз витрачати час на управління камерою, а розробник мав змогу краще контролювати атмосферу гри. Але таке управління робило гру трохи складнішою, адже гравець мусив звикати до частої зміни перспективи, що викликало складнощі під час боїв. На рисунку 1.18 наведено приклад фіксованої камери з танковим управлінням у грі Resident Evil Code: Veronica.

Рисунок 1.18 – Приклад фіксованої камери з «танковим» управлінням у грі Resident Evil Code: Veronica

В нинішній ігровій індустрії, розробники відійшли від ідеї «танкового» управління, віддаючи перевагу камерам від першої та третьої особи. Яскравим прикладом може слугувати Cyberpunk 2077 та недавній ремейк Resident Evil 4. На рисунку 1.19 наведено приклад з виглядом від першої особи у грі Cyberpunk 2077.

Рисунок 1.19 – Приклад з виглядом від першої особи у грі Cyberpunk 2077

На рисунку 1.20 наведено приклад з виглядом від третьої особи у ремейку гри Resident Evil 4

Рисунок 1.20 – Приклад з виглядом від третьої особи у ремейку гри Resident Evil 4

1.4 Середовища для реалізації програмного продукту

З розвитком технологій, ігрова індустрія розробила потужні ігрові рушії, які значно прискорюють і спрощують процес створення ігор. Це досить комплексні програмні інструменти які надають готові механізми для роботи з фізикою, освітленням, анімацією та іншими аспектами ігрового процесу. В залежності від ігрового рушія, складність роботи з ними може відрізнятися, адже деякі можуть вимагати певного часу на опанування. До прикладу, Unreal Engine 5 та Unity, які розроблені на базі C++ та C#, потребують певних знань у програмуванні.

Unity – це популярний крос-платформний ігровий рушій, що використовує об'єктно-орієнтовану мову C#. Він призначений для створення як 2D, так і 3D ігор і пропонує розширені можливості завдяки численним плагінам. Для створення 3D-проєктів Unity пропонує багатий набір функцій, включаючи підтримку фізики (на базі PhysX), освітлення в реальному часі, а також інструменти для роботи з матеріалами та шейдерами [15].

Unreal Engine 5 – також один з найкращих ігрових рушіїв, розроблений на мові C++. [16] Його особливістю є система візуальних сценаріїв Blueprint, яка є повноцінною системою для створення ігрового процесу. Вона базується на використанні вузлового інтерфейсу в редакторі Unreal Editor для розробки елементів гри та визначення об'єктно-орієнтованих класів. Система Blueprint значно прискорює процес створення і тестування ігрових механік, оскільки не вимагає написання коду вручну [17].

Крім того, Unreal Engine 5 також має потужні інструменти для роботи з 3D-графікою, включаючи системи Nanite – віртуалізована геометрія, яка дозволяє працювати з моделями високої полігональності без втрати продуктивності; та Lumen – реалістичне глобальне освітлення та відбиття світла в реальному часі, що дозволяє створювати реалістичні ігрові світи з високим рівнем деталізації. Саме тому для створення 3D-проєктів, розробники частіше віддають перевагу

використанню Unreal Engine 5 завдяки своїй зручності у роботі з Blueprint та високій кількості можливостей.

Задля створення візуальної частини, розробники 2D та 3D проєктів використовують різні програмні забезпечення. До прикладу, для 2D графіки часто застосовують програми по типу Affinity Designer, Adobe Photoshop, Krita та Clip Studio Paint – вони підходять для створення текстур, концепт-артів або спрайтів.

У випадку з 3D графікою, для її створення часто використовують такі інструменти як 3D Max, Autodesk Maya, Zbrush, Adobe Painter або Blender, який є гарним початком для новачків.

Blender – це безкоштовна програма, яка дозволяє створення 3D-моделей, їх скульптинг, текстурування, анімацію та рендеринг. Для новачків існує безліч безкоштовних ресурсів, включаючи офіційні відеоуроки [18].

Загалом, вибір програмного забезпечення залежить від поставленого завдання, бюджету проекту, рівня досвіду художника, а також технічних вимог до кінцевого продукту. Зараз розробники поєднують кілька програм у робочому процесі – наприклад, створення базової моделі в Blender, скульптинг у Zbrush, текстурування в Substance Painter, рендеринг – у Unreal Engine.

На рисунку 1.21 наведено інтерфейс програми Blender.

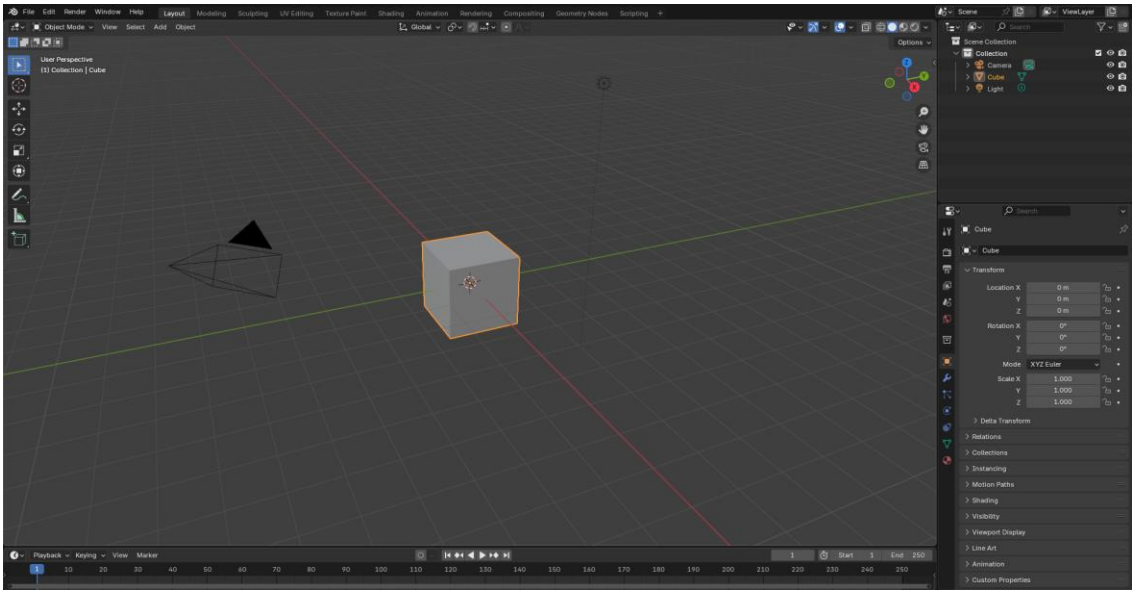

Рисунок 1.21 – Інтерфейс програми Blender

Основною проблемою моделювання завжди було UV-розгортка (UV-unwrapping) – процес, який полягає у «розпакуванні» 3D моделі на 2D площину для подальшого накладання текстур. Це важкий та клопіткий процес, адже якість розгортки напряму впливає на вигляд текстур. Тому у такому випадку, завжди можна скористуватись спеціалізованим інструментами, які спрощують UV-розгортку. Популярною опцією є програма – RizomUV, яка дозволяє швидко створити оптимальні розгортки та керувати їх «швами», що значно економить час [19]. На рисунку 1.22 наведено інтерфейс програми RizomUV.

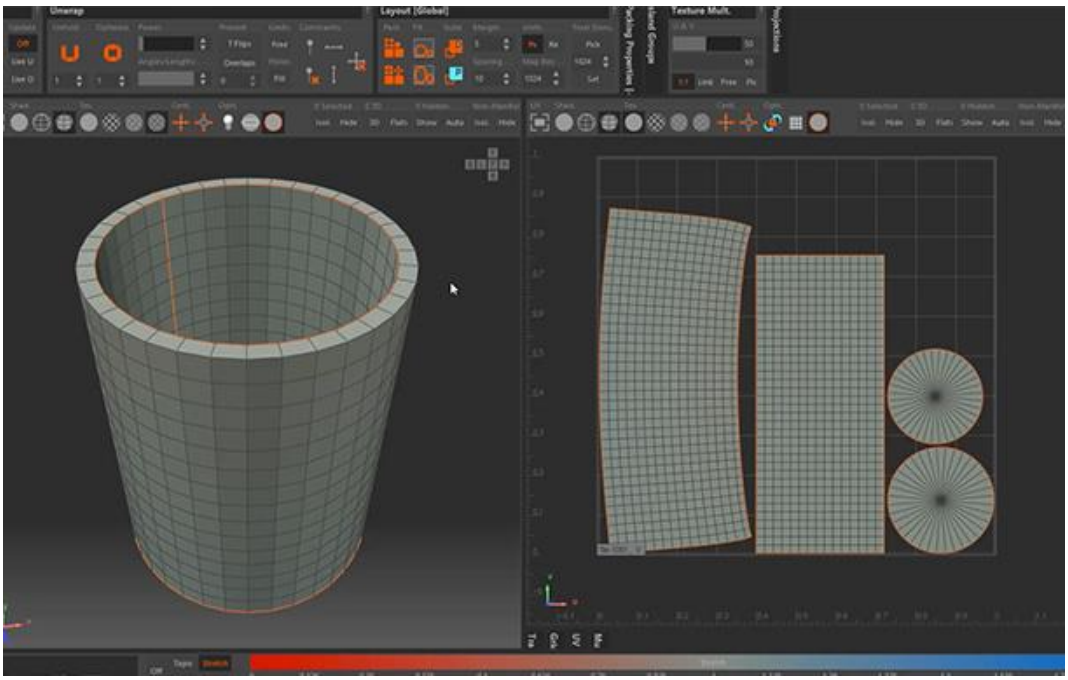

Рисунок 1.22 – Інтерфейс програми RizomUV

1.5 Формулювання вимог та постановка задачі

На сьогоднішній день існує безліч різноманітних ігор з різними типами управління та жанрами, але не завжди на ринку можна зустріти проєкт, який має ностальгічну атмосферу класичних ігор та водночас використовує сучасні технології для розробки ігрового процесу.

Головною ідеєю є створення 3D гри, керування якої буде ностальгічно похоже до класичних ігор – фіксовані камери та «танкове» управління. Для

створення атмосфери старих ігор має використовуватись легка і естетична графіка.

Ще має бути можливість збирати певні предмети в інвентар та розмовляти з неігровими персонажами (NPC).

Основною задачею є реалізація 3D гри з фіксованими камерами та «танковим» управлінням на ігровому рушії Unreal Engine 5. Цей ігровий рушій добре підходить для роботи з 3D проєктами, маючи при собі потужний набір інструментів для роботи з графікою та розробки механік.

Для реалізації гри буде найкращим варіантом застосувати малополігональну 3D графіку. Це оптимальний варіант, адже для її реалізації не потрібно багато часу на обробку деталей та створення реалістичності. Для реалізації буде створено графічні елементи у програмі Blender з допомогою RizomUV, а також використано безкоштовні ассети з платформи Fab, яка інтегрована з Unreal Engine 5. Для розробки обрано ігровий рушій Unreal Engine 5, через його зручність у створенні 3D проєктів.

Для розробки гри, потрібно вирішити такі задачі:

- Налаштувати середовище Unreal Engine 5.
- Створити та імпортувати 3D-моделі. Підготувати графічну частину.
- Розробити рівень з урахуванням фіксованих камер.
- Реалізувати механіку: керування, переходи між камерами, підняття предмету, взаємодія з NPC.
- Розробити інтерфейс діалогів для гри.
- Провести тестування гри.

1.6 Висновок до розділу першого

У цьому розділі було проаналізовано ігрову індустрію та певні жанри відеоігор.

Також було проаналізовано графіку в іграх, та як вона впливає на сприйняття і атмосферу гри. Ще проведено аналіз пристроїв управління та типів керування в іграх, та як вони впливають на ігровий досвід. Проведено ретельний аналіз програм та інструментів для розробки відеоігор.

Для реалізації гри обрано стиль 3D малополігональної графіки та ігровий рушій Unreal Engine 5.

Розробка комп'ютерної гри Moonlight Song з покращеним способом керування в стилі класичних ігор на ігровому рушії Unreal Engine 5 є актуальною, тому що це поєднання нових технологій розробки з ностальгічним підходом до ігрового процесу.

Розробка є актуальною, оскільки багатьом гравцям які знайомі з управлінням в класичних іграх, буде цікавою гра в якій буде відтворено цю ностальгічну особливість.

В цьому розділі також було поставлено задачі для розробки комп'ютерної гри Moonlight Song з покращеним способом керування в стилі класичних ігор.

2 ПРОЄКТУВАННЯ КОМП'ЮТЕРНОЇ ГРИ MOONLIGHT SONG З ПОКРАЩЕНИМ СПОСОБОМ КЕРУВАННЯ В СТИЛІ КЛАСИЧНИХ ІГОР

2.1 Розробка механік 3D-гри Moonlight Song

Ігрова механіка — це ключовий елемент у розробці відеоігор, який формує унікальні особливості проєкту та безпосередньо впливає на його жанр. Щоб урізноманітнити ігровий процес, розробники часто поєднують основну механіку з додатковими, простішими механіками.

Ігрова механіка охоплює правила та системи, які керують усім процесом гри і визначають досвід гравця — від базового пересування персонажа до взаємодії з іншими об'єктами. Саме вона є фундаментом, на якому будується гра. Навіть без надзвичайної графіки чи оригінального стилю, гра може бути успішною, якщо має привабливу механіку.

Основна механіка є ядром геймплею, але вона часто проявляється через різні варіації. Наприклад, у грі Lost Records, до прикладу, реалізована механіка використання камери: гравець може фотографувати важливі об'єкти або миті, що не є обов'язковим для просування по сюжету, проте сприяє глибшому зануренню в історію та формуванню емоційного зв'язку з подіями. Такі механіки додають грі "смаку" та дозволяють відчути більше, ніж просто виконання завдань — вони відкривають нові враження й іноді кидають несподівані виклики [20].

При розробці 3D-гри Moonlight Song головним натхненням є гра Resident Evil 1. Ця гра є гарним прикладом застосування фіксованих камер та взаємодії з об'єктами – обмежений простір, ретельно спроєктовані локації, де кожна деталь чи об'єкт має значення, а також тісне поєднання дослідження, вирішення головоломок і керованих напружених моментів. Для покращення ігрового досвіду гравця поєднання схожих механік буде гарним рішенням.

Основною рисою ігор із дослідницьким геймплеєм є акцент на атмосфері та її емоційному сприйнятті. Гравець занурюється у світ гри, де важливу роль відіграють звуки, обмежене поле зору та освітлення — усе це формує унікальний настрій.

Фіксована камера відіграє ключову роль у посиленні цього ефекту, скеровуючи увагу гравця на оточення. Вона створює кінематографічне враження, дозволяючи спостерігати за персонажем із певних ракурсів, ніби через об'єктив камери. Такий підхід надає змогу режисерськи контролювати композицію кадру, підкреслюючи важливі деталі або посилюючи емоційний вплив сцен — наприклад, створюючи напругу чи несподівані ситуації.

На рисунку 2.1 представлено загальний план гри.

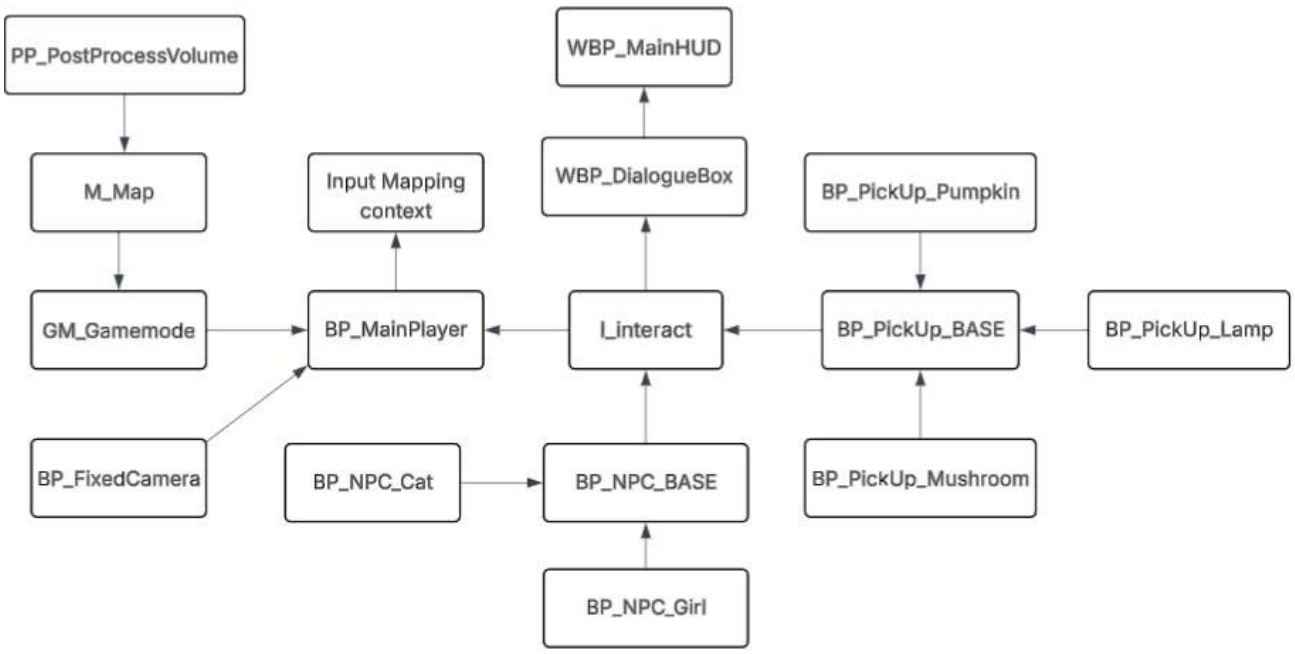

Рисунок 2.1 – Загальний план гри

Було сплановано базову архітектуру гри з підтримкою взаємодії з предметами та персонажами, також створено інтерфейс (діалогове вікно).

Основними компонентами є:

- Сцена: На ній розміщено об’єкти, зокрема постпроцесинг (PP_PostProcessVolume) для візуальних ефектів та фіксовані камери (BP_FixedCamera).
- Гравець (BP_MainPlayer): має налаштування керування («танкове») та камери, може взаємодіяти з предметами та NPC.
- NPC: Усі створені NPC (Cat, Girl) створені на основі базового класу (BP_NPC_BASE) і можуть вести діалог із гравцем.
- Предмети для підняття: Всі об’єкти наслідують базовий клас (BP_PickUp_BASE) і можуть бути підняті гравцем через інтерфейс взаємодії.
- UI: Включає головний HUD та діалогове вікно для відображення тексту під час взаємодії з NPC або предметом.

2.2 Розробка UML-діаграми

Об’єктно-орієнтоване програмування (ООП) є одним із найефективніших підходів у розробці відеоігор, оскільки дає змогу логічно структурувати проєкт за допомогою класів і об’єктів, які відображають елементи реального світу. У грі об’єкти на зразок гравця, ворогів або предметів реалізуються як класи, що поєднують у собі як властивості (рівень здоров’я, рівень сили) так і поведінкові функції. Завдяки механізму успадкування можна створити універсальний базовий клас — наприклад, «Персонаж», — від якого наслідують усі спільні характеристики ворогів і союзників. Це значно знижує рівень дублювання коду та підвищує гнучкість і масштабованість системи.

Для візуалізації структури гри часто використовуються UML-діаграми класів. Вони наочно демонструють архітектуру системи, відображаючи класи, їхні властивості, методи та взаємозв’язки між ними. Саме тому UML-діаграми класів є одним з найзручніших інструментів для представлення структури програмних модулів у грі.

На рисунку 2.2 представлено UML-діаграма класів для предметів.

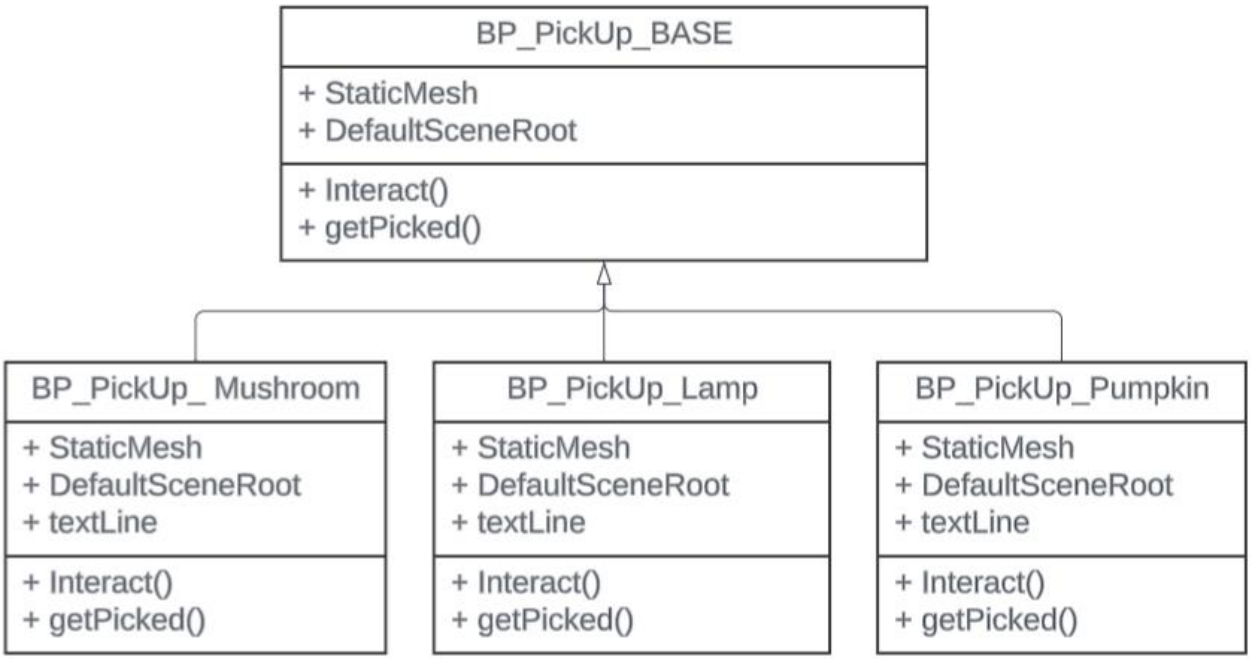

Рисунок 2.2 – UML-діаграма класів для предметів

BP_PickUp_BASE – це батьківський клас, який виконує роль шаблону для всіх об’єктів, з якими гравець може взаємодіяти. Від цього базового класу наслідують кілька похідних класів, зокрема: BP_PickUp_Mushroom, BP_PickUp_Lamp та BP_PickUp_Pumpkin. Кожен із цих класів представляє конкретний тип предмета, який гравець може підібрати під час проходження гри. У цих дочірніх класах додано змінну textLine, яка зберігає текстове повідомлення, що з’являється при взаємодії з відповідним об’єктом. Це дозволяє персоналізувати відображувану інформацію залежно від типу предмета, наприклад: “Ви знайшли гриб”, “Ви підібрали лампу” або “Цей гарбуз виглядає підозріло”.

На рисунку 2.3 представлено ієрархію класів NPC-персонажів

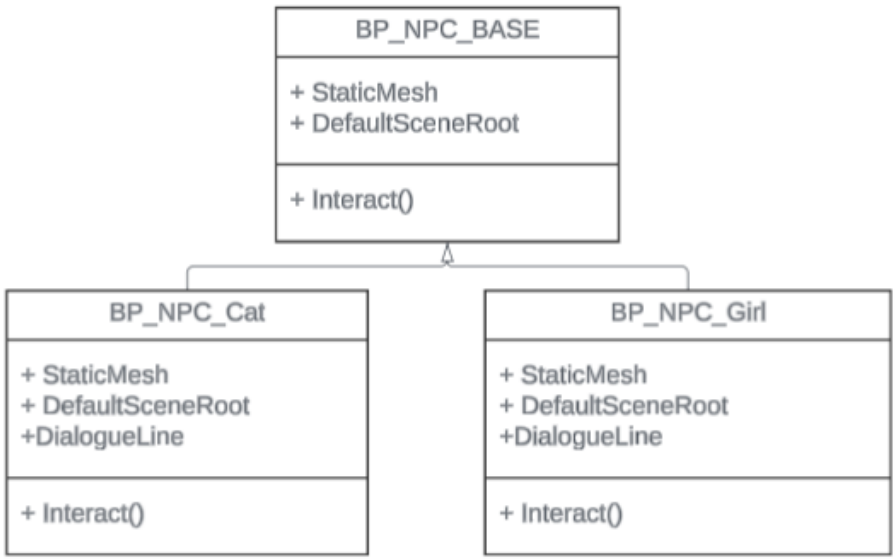

Рисунок 2.3 – UML-діаграма класів для NPC-персонажів

На рисунку зображено BP_NPC_BASE – батьківський клас, який містить базові компоненти та методи, спільні для всіх NPC. Зокрема, до складу цього класу входять компоненти StaticMesh (який визначає зовнішній вигляд об’єкта), DefaultSceneRoot (який виконує роль кореневого компонента сцени) та функція Interact(), що відповідає за логіку взаємодії між NPC і гравцем.

Від BP_NPC_BASE наслідують два дочірні класи — BP_NPC_Cat та BP_NPC_Girl. Вони успадковують усі властивості й методи батьківського класу, зокрема функцію Interact(), що дозволяє їм реагувати на дії гравця так само, як і інші NPC. Крім цього, обидва дочірні класи розширюють свою функціональність за допомогою власної змінної DialogueLine, яка містить унікальні репліки, що NPC може повідомити гравцю при взаємодії. Наприклад, BP_NPC_Cat може видавати коротке жартівливе повідомлення, тоді як BP_NPC_Girl — повноцінну репліку діалогу, або запропонувати невеличке завдання.

На рисунку 2.4 наведено ієрархію класів персонажів.

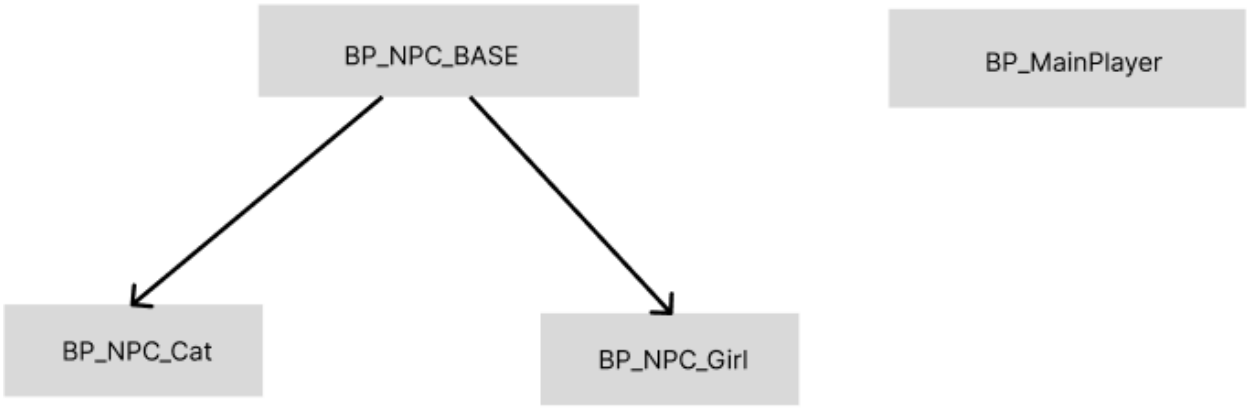

Рисунок 2.4 – Ієрархія класів персонажів

На рисунку зображено ієрархію класів персонажів, і можна помітити що BP_MainPlayer не входить до цієї ієрархії та існує окремо. Це свідчить про те, що головний персонаж, яким керує гравець, має власну структуру класу, незалежну від системи NPC.

На рисунку 2.5 зображено ієрархію предметів для взаємодії.

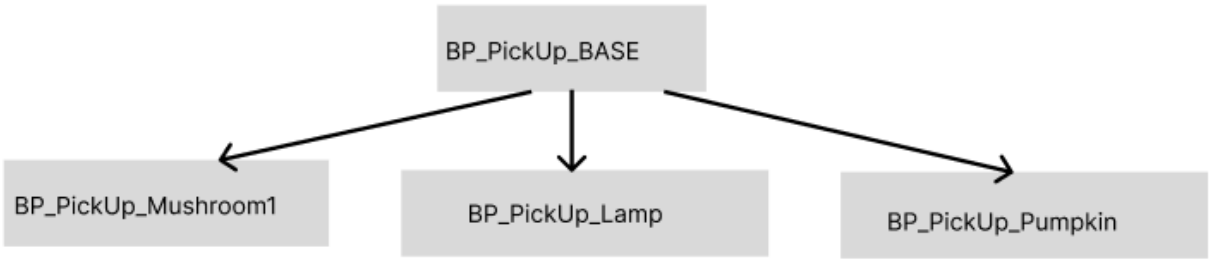

Рисунок 2.5 – Ієрархія класів предметів для взаємодії

2.3 Розробка схеми алгоритму роботи програмного модуля комп'ютерної гри Moonlight Song

Алгоритм — це чітка послідовність дій, призначена для розв'язання конкретної задачі або виконання обчислювального процесу. Він виконується крок за кроком і використовується як в програмному забезпеченні, так і в апаратних системах. Алгоритми мають широке застосування в інформаційних технологіях. У сферах математики, програмування та інформатики алгоритми зазвичай описують стандартні процедури для розв'язання типових завдань. Крім того, вони слугують основою для обробки даних та є ключовим елементом у побудові автоматизованих систем.

Для наочності та зручності розуміння алгоритмів використовуються спеціальні схеми. Такі схеми дозволяють відобразити послідовність дій, структуру логіки та зв'язки між кроками алгоритму. Завдяки цьому процеси можна ефективно аналізувати, оптимізувати й узгоджувати в межах команди.

Алгоритм опрацювання «танкового» управління (рис. 2.6) складається з таких кроків:

Крок 1. Отримання повороту контролера. Визначається поточний оберт контролера у просторі.

Крок 2. Обчислення повертання за віссю Z.

Крок 3. Створення Rotator. На основі обчисленого кута створюється новий об'єкт типу Rotator.

Крок 4. Встановлення повороту контролера для осі X.

Крок 5. Повторне отримання повороту контролера. Після налаштування кутів обертання система ще раз зчитує поворот контролера для коректного оновлення даних.

Крок 6. Отримання вектору напряму вперед за значенням Z. Обчислюється напрямок, у якому гравець повинен рухатися — вектор, що вказує «вперед» відповідно до кута повороту.

Крок 7. Додавання Movement Input. До руху гравця додається вхід (input), що фактично змушує його рухатися у вказаному напрямку.

На рисунку 2.6 показано схему опрацювання «танкового» управління.

Рисунок 2.6 – Схема опрацювання «танкового» управління

Крок 1. Відбувається або натискання клавіш або дотик тачскріну.

Крок 2. Одразу після цього відбувається зупинка автопілоту.

Крок 3. Для клавіш, відбувається перемикання в режим ручного управління, та їх опрацювання головним героєм.

Крок 4. Для дотику тачскріна, відбувається трасування точки кліку за допомогою функції GetHitResult.

Крок 5. Потім йде проектування точки на поверхню.

Крок 6. Йде перевірка, чи можливо прокласти маршрут. Якщо так – відбувається перемикання в режим автопілоту. Якщо ні – виконується завершення оброки дії.

Крок 7. Під час автопілоту, персонаж отримує команду слідування до визначеної точки.

На рисунку 2.7 зображено схему алгоритму мультимодального управління персонажем.

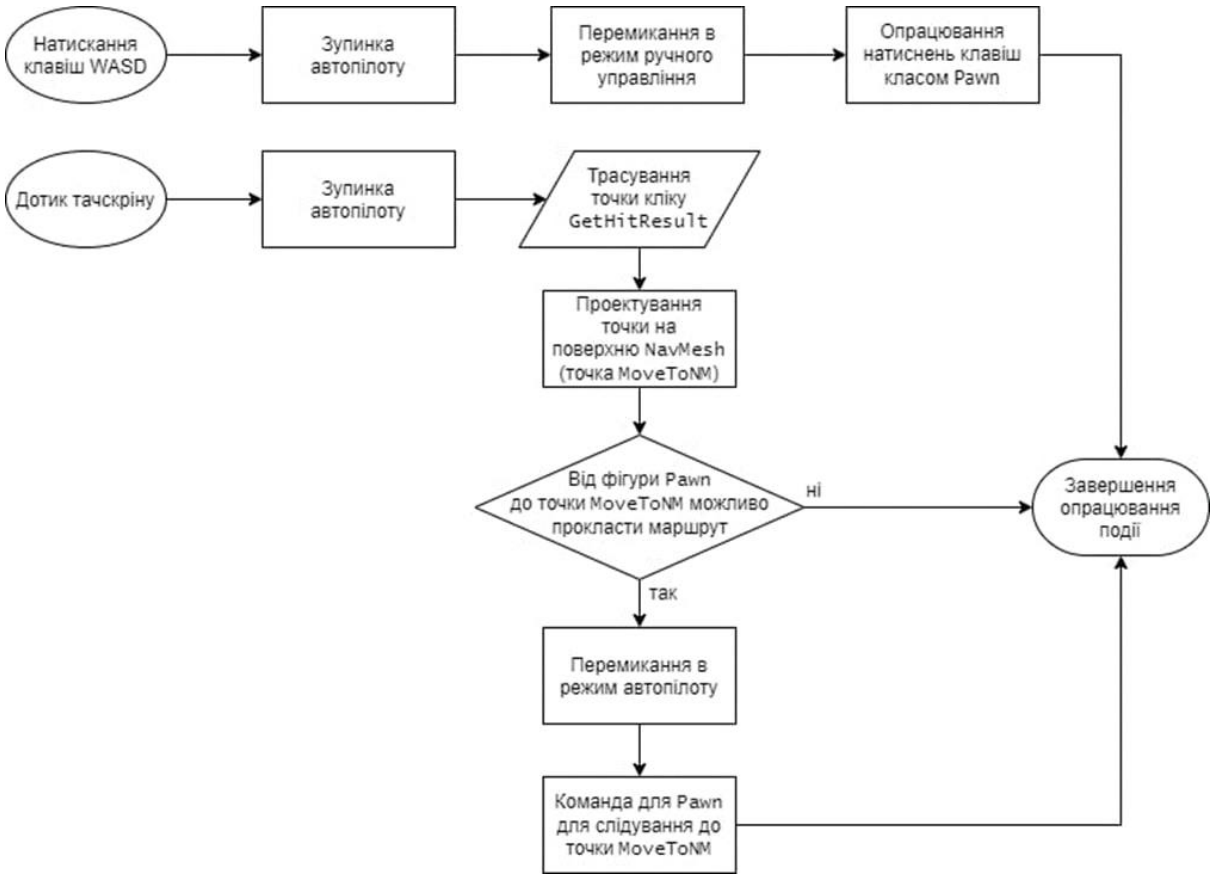

Рисунок 2.7 – Схема алгоритму мультимодального управління персонажем

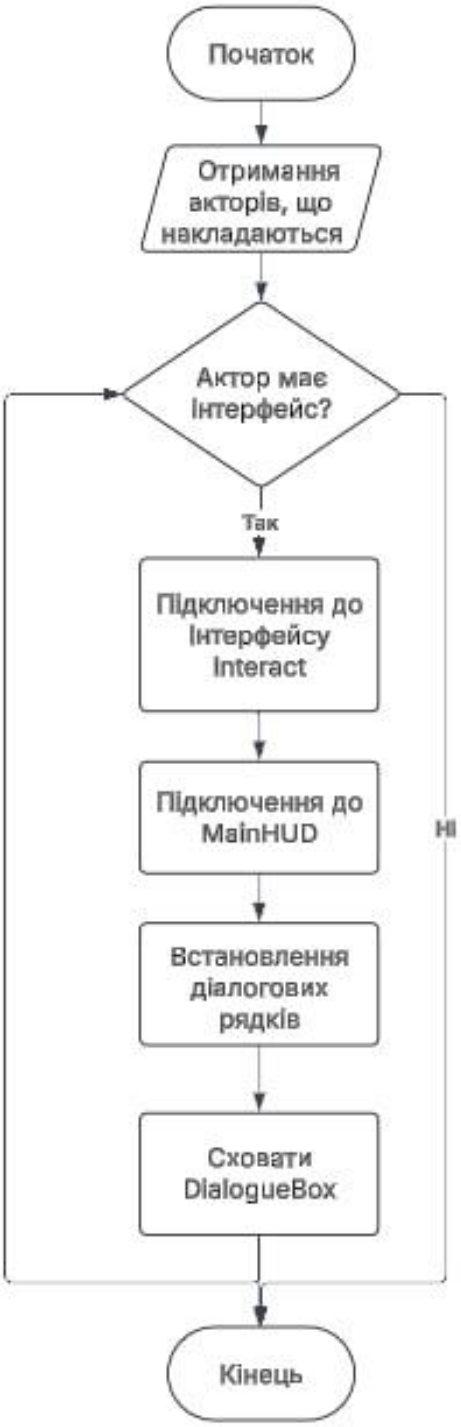

Рисунок 2.8 – Схема алгоритму взаємодії головного персонажа з іншими об’єктами

Алгоритм події взаємодії головного персонажа з іншими об’єктами (рис. 2.8) складається з таких кроків:

Крок 1. Отримання акторів що накладаються. На цьому етапі система ідентифікує та отримує список акторів які взаємодіють.

Крок 2. Чи актор має інтерфейс. Якщо так то наступний крок 3, якщо ні то алгоритм переходить безпосередньо до кроку «Кінець».

Крок 3. Підключення до інтерфейсу Interact. Якщо актор має інтерфейс то відбувається підключення функції взаємодії.

Крок 4. Підключення до MainHUD (основного екранного інтерфейсу).

Крок 5. Встановлення діалогових рядків. Встановлення тексту діалогу пов’язаного з поточною взаємодією.

Крок 6. Сховати діалогове вікно.

2.4 Висновок до розділу другого

У цьому розділі було розроблено структуру програмного модуля комп’ютерної гри Moonlight Song, що дозволяє налагодити та полегшити організовану розробку програмного забезпечення. Було визначено, що гра базуватиметься на «танковому» управлінні та використанні фіксованих камер.

Було створено UML-діаграми основних класів, що дозволяє краще структурувати архітектуру проєкту. Представлено загальну ієрархію класів з демонстрацією механізму спадкування і одночасно чітко виділено окремі класи, які не входять до цієї ієрархії. Описано ключові змінні та методи, які містяться у відповідних класах.

Також були створені схеми основних алгоритмів роботи програмного модуля комп’ютерної гри Moonlight Song. Вони допомогли детальніше спланувати та проаналізувати логіку роботи ігрового процесу. Ці алгоритмічні схеми значно спрощують подальший процес програмування, надаючи чітке уявлення про послідовність дій і взаємодію функцій між собою.

Здійснення подібних підготовчих кроків є надзвичайно важливим для ефективного створення комп’ютерної гри як складного програмного продукту.

Завчасне опрацювання архітектури, визначення логіки взаємодії між компонентами та моделювання ігрових процесів дає змогу сформувати чіткий і впорядкований план розробки. Це дозволяє уникнути численних труднощів на пізніших етапах розробки гри.

3 РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОЇ ГРИ MOONLIGHT SONG З ПОКРАЩЕНИМ СПОСОБОМ КЕРУВАННЯ У СТИЛІ КЛАСИЧНИХ ІГОР

3.1 Обґрунтування вибору ігрового рушія та мови програмування

Для реалізації гри Moonlight Song обрано ігровий рушій Unreal Engine 5. Оскільки гра розробляється у форматі 3D це цілком логічний вибір, адже Unreal Engine 5 пропонує широкі можливості для роботи з фізикою та освітленням.

Unreal Engine 5 – це високо функціональний ігровий рушій, який є безкоштовним та незважаючи на це пропонує велику кількість інструментів для розробки ігор, тому підходить як для великих студій, так і для незалежних розробників.

UE 5 надає можливість комбінувати традиційне програмування на C++, з системою візуального скриптінгу Blueprints. Blueprints у Unreal Engine – це повноцінна система побудови ігрової логіки на основі вузлового інтерфейсу в Unreal Editor. Подібно до багатьох сучасних мов сценаріїв, Blueprints підтримують об'єктно-орієнтований підхід, дозволяючи визначати класи та взаємодії між об'єктами гри. Завдяки своїй гнучкості та зрозумілості, ця система надає дизайнерам можливість використовувати широкий набір інструментів який зазвичай доступний розробникам з досвідом програмування. А також спеціальна розмітка Blueprints, доступна в C++ реалізації Unreal Engine, що дозволяє програмістам створювати базові системи, які можуть бути розширені дизайнерами. За допомогою цих візуальних сценаріїв можна розрбити повноцінну гру: від керування персонажем і обробки анімацій до складної логіки штучного інтелекту, взаємодії з оточенням та обробки подій.

Зазвичай під час розробки ігор на рушії Unreal Engine 5 застосовують комбінацію Blueprints та C++, хоча цілком можливо створити гру, використовуючи лише один із цих підходів. Як правило, C++ використовується

для розробки базових і системних компонентів, тоді як Blueprints слугує для опису логіки поведінки та взаємодії об'єктів.

До того ж, функціональні можливості Blueprints значно ширші, ніж просто розробка логіки для акторських класів – вони поширюються на великий перелік інших класів, серед яких ігрові режими (Game Modes), стани гри (Game States), а також персонажі (Characters) та пішаки (Pawns). Крім того, система Blueprints підтримує роботу з компонентами, бібліотеками функцій та макросами, що суттєво пришвидшує розробку та спрощує повторне використання напрацьованого коду. Тобто, Blueprints – є візуальною формою кодування C++, яка подається у зрозумілому та доступному вигляді. Водночас програмування на C++ має більш високий бар’єр для початку роботи, тому що потребує більш глибокого розуміння структур даних, синтаксису і основ об’єктно-орієнтованого програмування.

Попри свою складність, C++ має певні переваги:

- Легко створювати компоненти, які можна використовувати у різних проєктах, що скорочує час розробки;
- Надає можливість будувати складні та масштабовані системи з чіткою архітектурою;

Для розробки комп’ютерної гри Moonlight Song обрано ігровий рушій Unreal Engine 5 з поєднанням Blueprints та C++. Оскільки гра є 3D-проєктом, вибір UE 5 є обґрунтованим - рушій забезпечує широкий спектр інструментів для роботи з тривимірним середовищем.

3.2 Обґрунтування вибору програмного забезпечення для створення графічної частини

Задля створення 3D графіки, існує велика варіативність програмного забезпечення, але всі вони відрізняються своєю складністю та вартістю.

До прикладу, Autodesk Maya – є професійним інструментом для створення 3D-моделей, анімацій та візуальних ефектів, проте має складний інтерфейс та потребує платної підписки. 3ds Max також широко використовується у галузі, особливо для архітектурної візуалізації та геймдизайну, але є менш зручним для новачків. ZBrush – потужний інструмент для скульптингу, який дозволяє створювати високодеталізовані 3D-моделі. Проте він має доволі складний інтерфейс і є також платним, що може стати перешкодою для індивідуальних розробників або невеликих команд.

Однією з найкращих альтернатив є Blender – ще одне широко використовуване програмне забезпечення, яке добре підходить для роботи з Unreal Engine 5.

До переваг використання Blender можна віднести:

- Безкоштовність. Програмне забезпечення не вимагає фінансових витрат або придбання підписок;
- Сумісність з Unreal Engine 5. Програма має підтримку формату FBX, який є основним для імпорту 3D-моделей, анімацій та скелетів в Unreal Engine 5;
- Велика кількість навчальних матеріалів, які суттєво спрощують процес навчання;
- Широкий функціонал. Програма підтримує моделювання, текстурування, ріггінг, анімацію, скульптинг, рендеринг та композитинг;

Тому в даному випадку, серед всіх вище перерахованих програмних забезпечень було обрано Blender для розробки графічної частини у 3D грі.

Також, для полегшення процесу текстурування, у поєднанні з Blender було використано RizomUV – програмне забезпечення яке значно полегшує процес UV-розгортки (UV-unwrapping).

Серед найважливіших переваг цього інструменту варто виділити:

- Автоматизовані інструменти для розгортки, які значно економлять час у порівняні з ручною роботою в Blender;

- Контроль над швами та розміщенням UV-островів, що забезпечує кращу якість текстурування.
- Сумісність та підтримка форматів для роботи з Unreal Engine 5 та Blender;
- Інтуїтивний інтерфейс в якому легко освоїтись.

На рисунку 3.1 наведено приклад автоматизованої розгортки 3D-моделі за допомогою програмного забезпечення RizomUV.

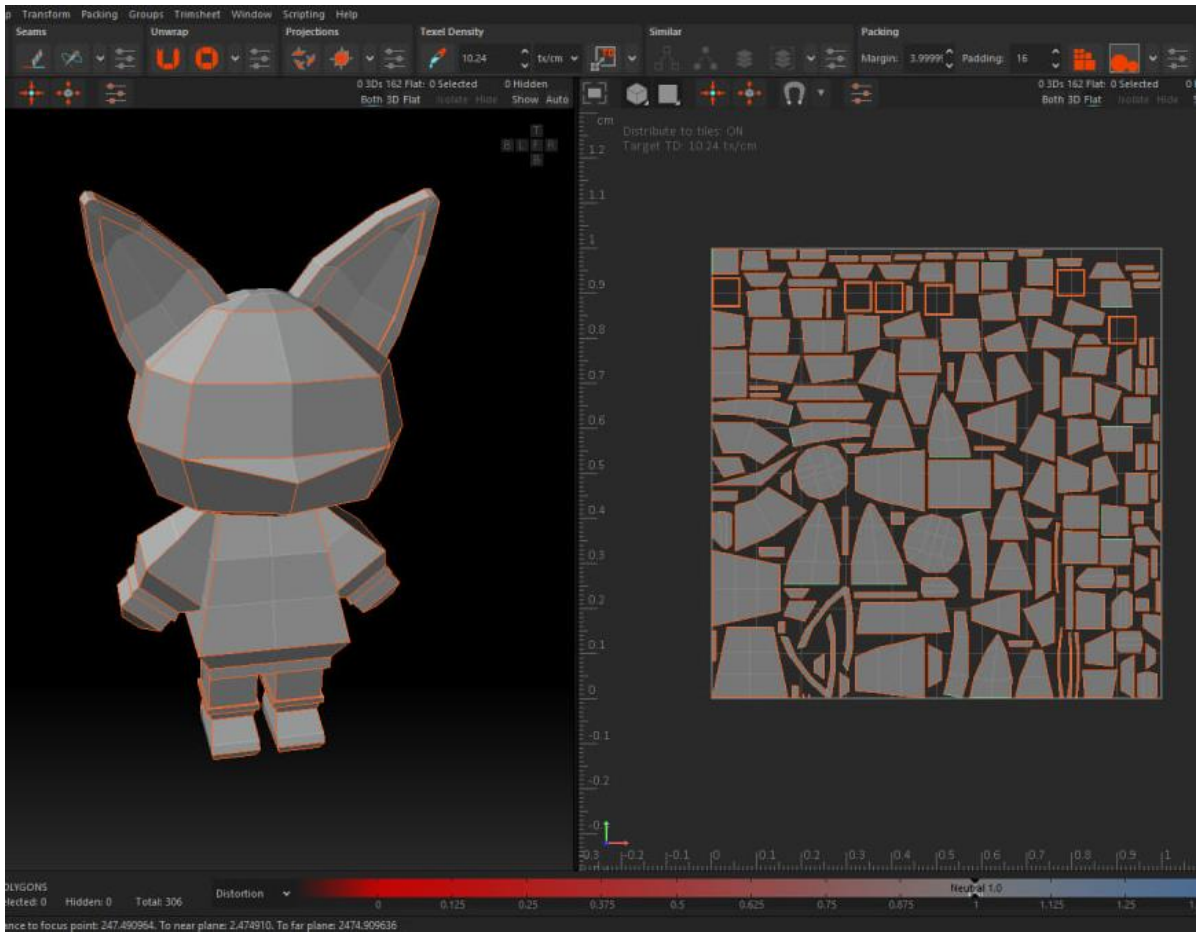

Рисунок 3.1 – Приклад автоматизованої розгортки 3D-моделі за допомогою програмного забезпечення RizomUV.

3.3 Підготування 3D-моделі до використання в ігровому рушії

Імпорт 3D-моделі, створеної в Blender, до ігрового рушія Unreal Engine 5 є одним із найбільш поширених, але водночас складних етапів у процесі розробки

тривимірних ігор. Незважаючи на те, що обидва інструменти підтримують формат FBX, який і використовується для обміну 3D-даними, існує ряд технічних нюансів, які можуть призвести до некоректного відображення моделі, її анімацій або скелетної структури.

Однією з перших проблем з якою можна часто зустрітись – є невідповідність масштабу моделі після імпорту в Unreal Engine 5.

Blender використовує метричну систему, де 1 одиниця = 1 метр, але за замовчуванням одиниця виміру в Blender відповідає одному сантиметру в Unreal Engine. Через це модель може виглядати надто великою або надто маленькою у сцені. Щоб уникнути цієї проблеми, перед експортом у Blender потрібно правильно виставити масштаб (наприклад, встановити Scale у вкладці Scene на 0.01). На рисунку 3.2 наведено неправильно виставлений масштаб моделі в Blender.

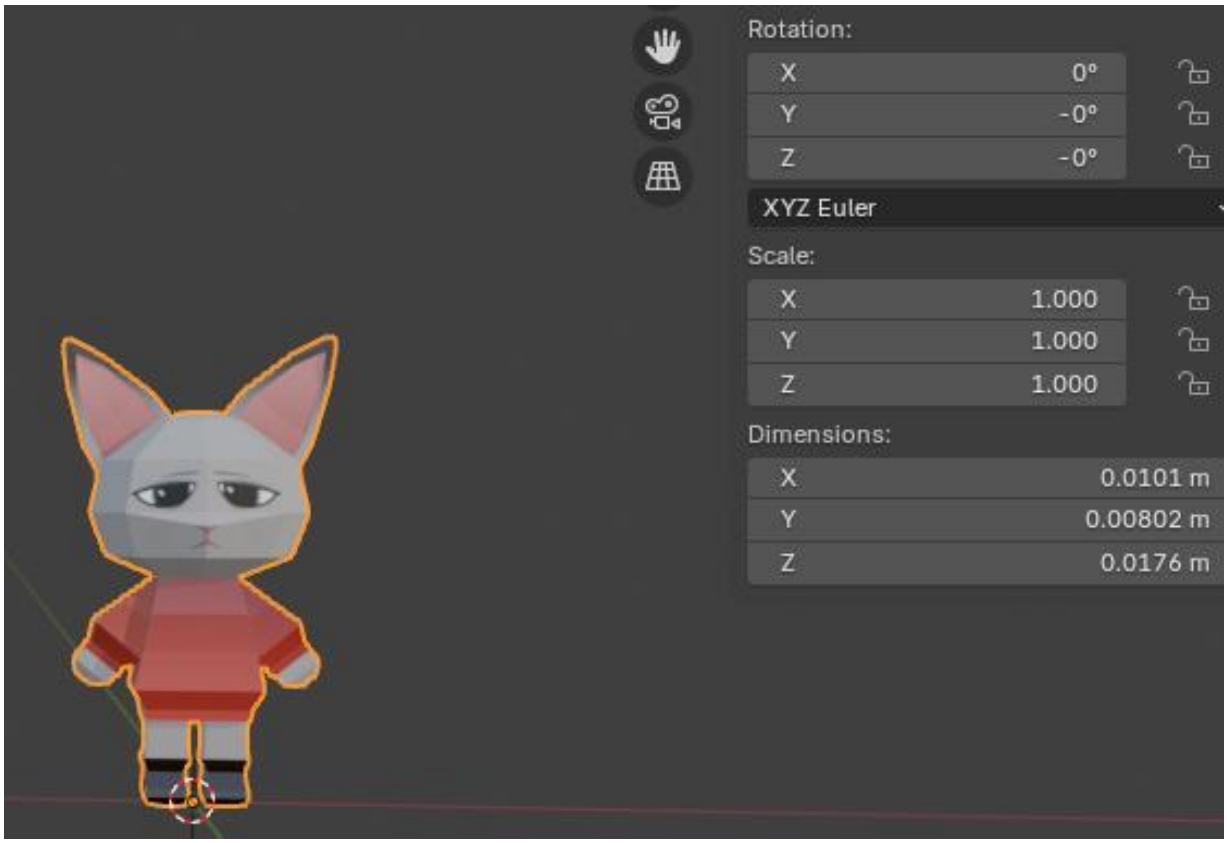

Рисунок 3.2 – Неправильно виставлений масштаб моделі в Blender

На рисунку 3.3 наведено правильно виставлений масштаб моделі в Blender.

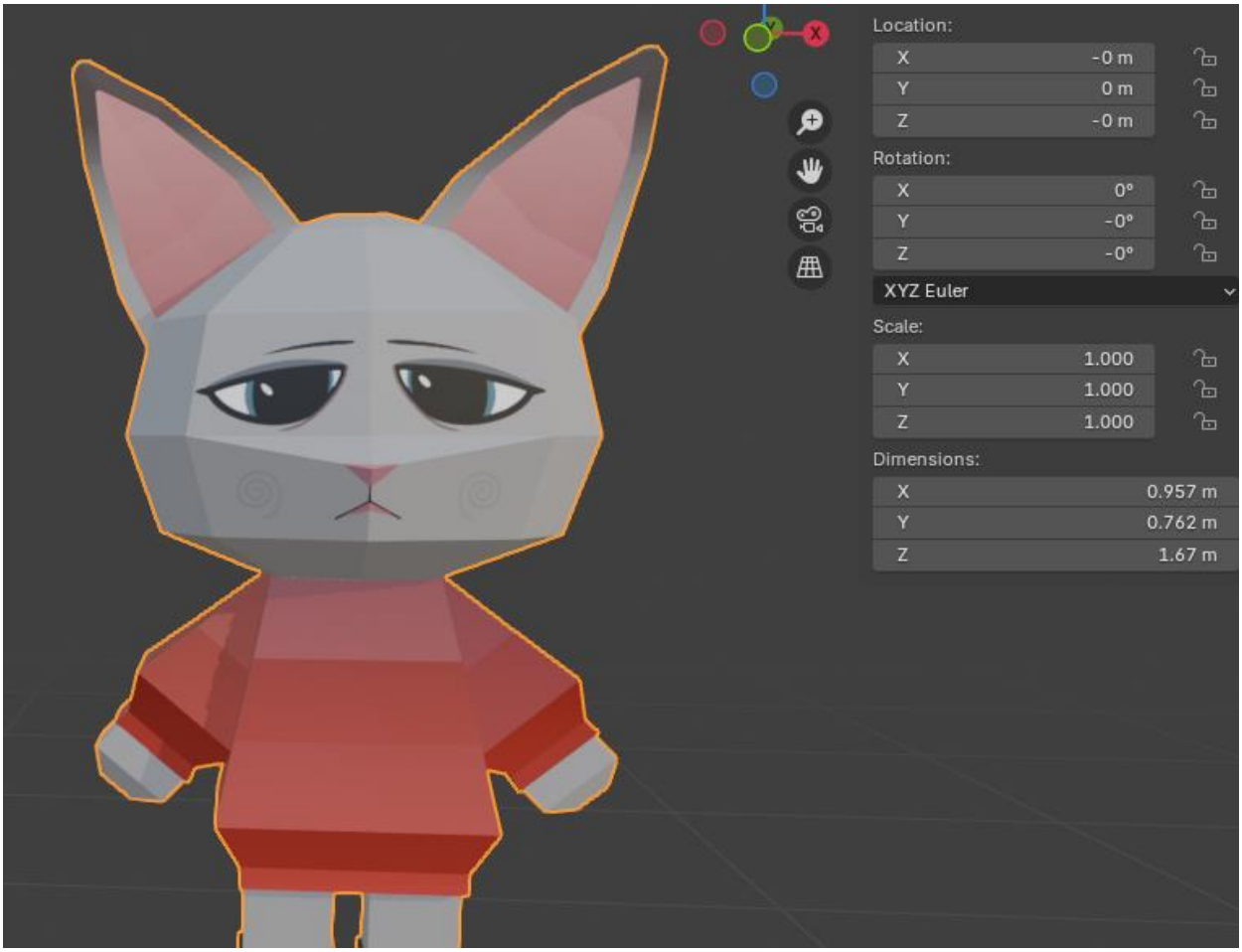

Рисунок 3.3 – Правильно виставлений масштаб моделі в Blender

Але й після цього нас може зустріти ряд проблем. Якщо модель створюється у Blender з унікальним скелетом, вона може не бути сумісною з уже наявними скелетними системами в Unreal Engine 5.

Якщо модель імпортується з наміром використовувати вже існуючий скелет в UE 5, важливо, щоб у Blender її Armature мала точно таку ж структуру кісток, як і оригінал.

У іншому випадку коли ми маємо модель з іншим скелетом і намагаємося замінити її скелет на вже наявний у проєкті, можуть виникнути значні складнощі. Після імпорту в Unreal Engine 5 рушій не завжди коректно розпізнає нову структуру, що може призвести до серйозних помилок в анімації: кістки можуть

рухатися неправильно, частини тіла будуть "рватися" або модель взагалі може рухатись в «замерзлому» вигляді через відсутність анімацій для нового скелету. На рисунку 3.4 наведено приклад базового скелету в Unreal Engine 5.

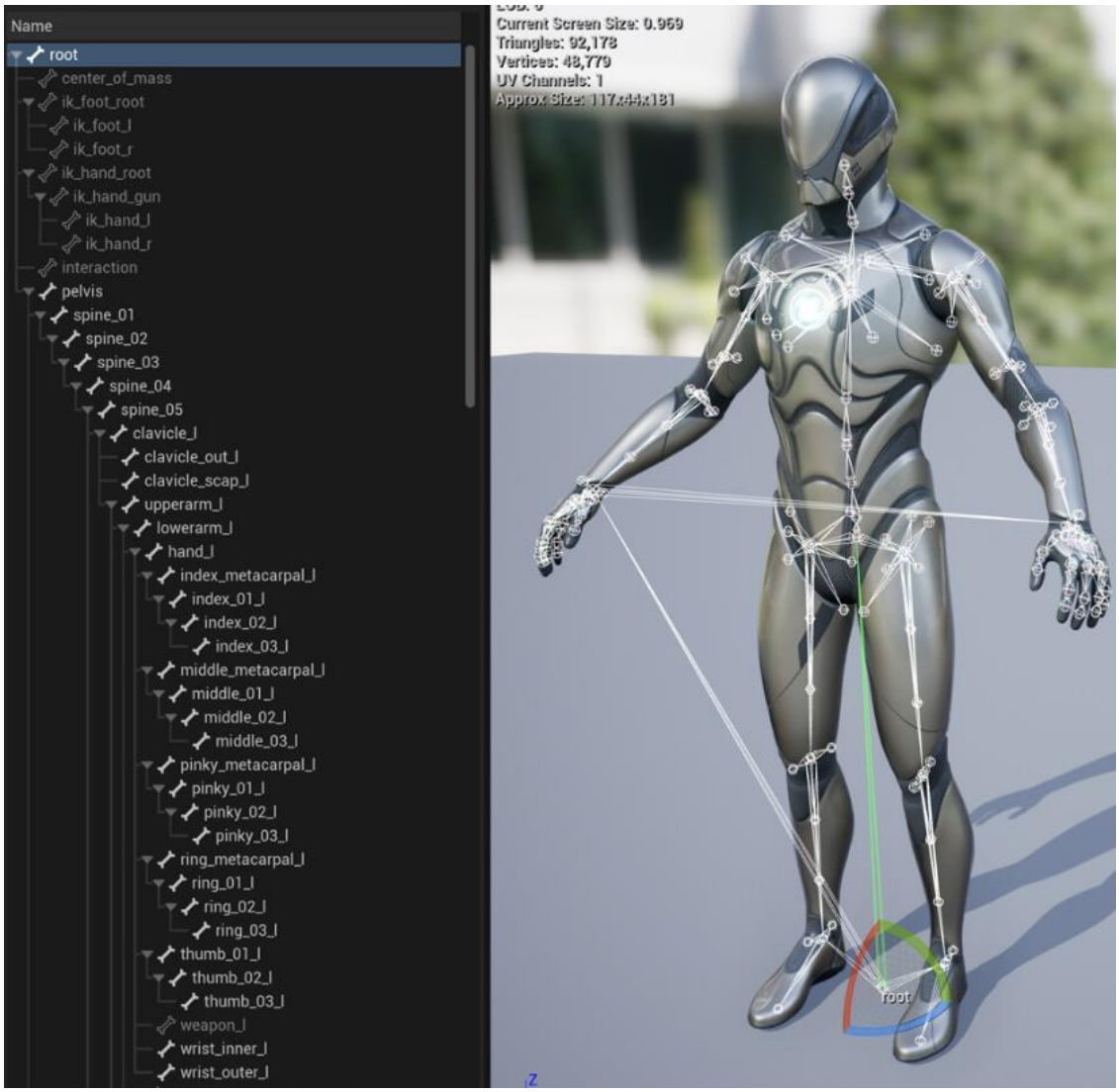

Рисунок 3.4 – Приклад базового скелету в Unreal Engine 5

Як наглядно видно, базовий скелет Unreal Engine 5 набагато складніший за своєю структурою – він містить велику кількість кісток, включаючи окремі сегменти для кожного пальця рук. Натомість власноруч розроблений скелет персонажа є набагато спрощеним, що дозволяє легко розробляти йому анімації, але не є настільки гнучким як структурованіший скелет [21].

На рисунку 3.5 наведено власноруч створений скелет для 3D-моделі.

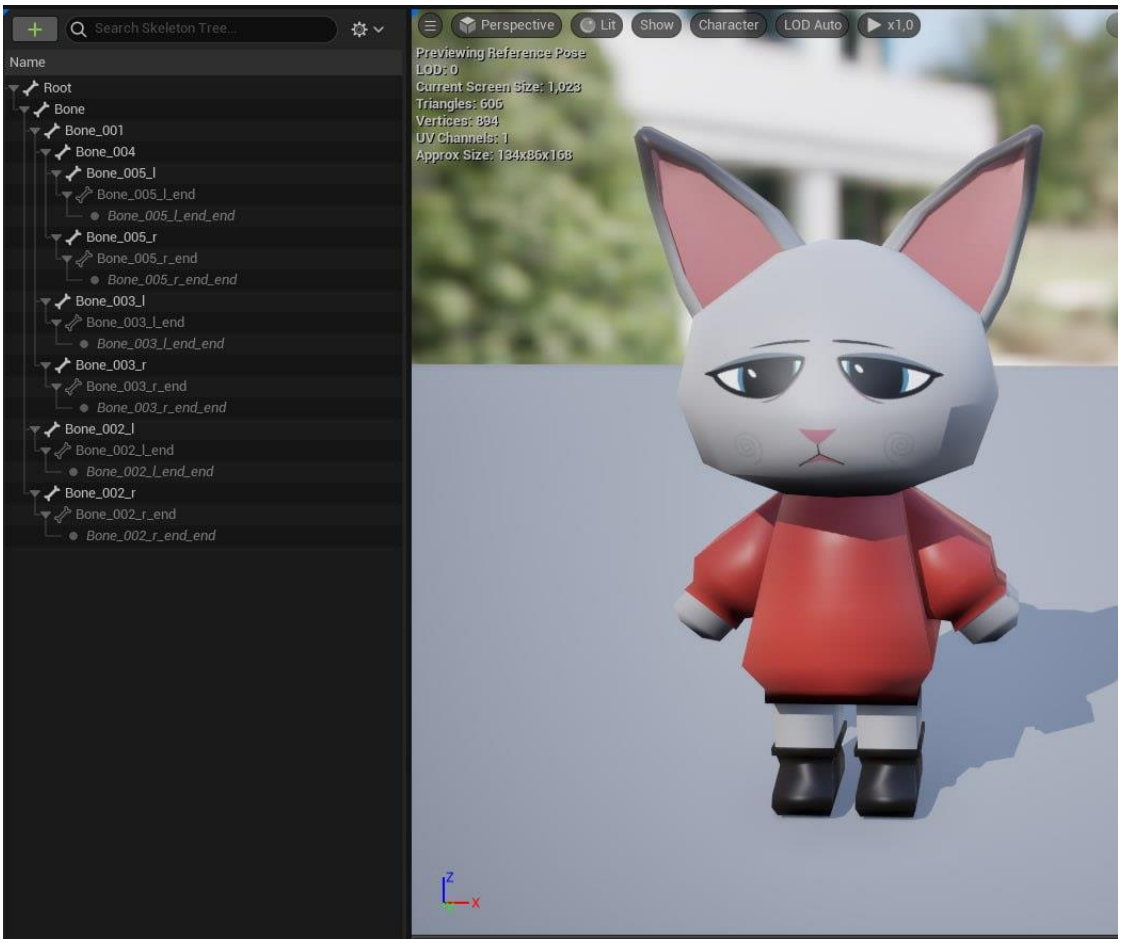

Рисунок 3.5 – Власноруч створений скелет для 3D-моделі

Для цього можна скористатися двома основними підходами. Перший — використання Blender, де за допомогою техніки keyframing (ключових кадрів) можна вручну створити необхідні анімації. У цьому процесі задаються основні пози, між якими Blender автоматично інтерполює рухи, формуючи плавну анімацію.

Другий підхід — використання вбудованих інструментів Unreal Engine 5, зокрема Sequencer або Control Rig. Sequencer надає можливість створювати анімаційні послідовності прямо в рушії, що зручно для синхронізації з іншими елементами сцени. Control Rig, зі свого боку, дозволяє створювати кастомні ріги та гнучко анімувати персонажа без потреби експорту в сторонні програми.

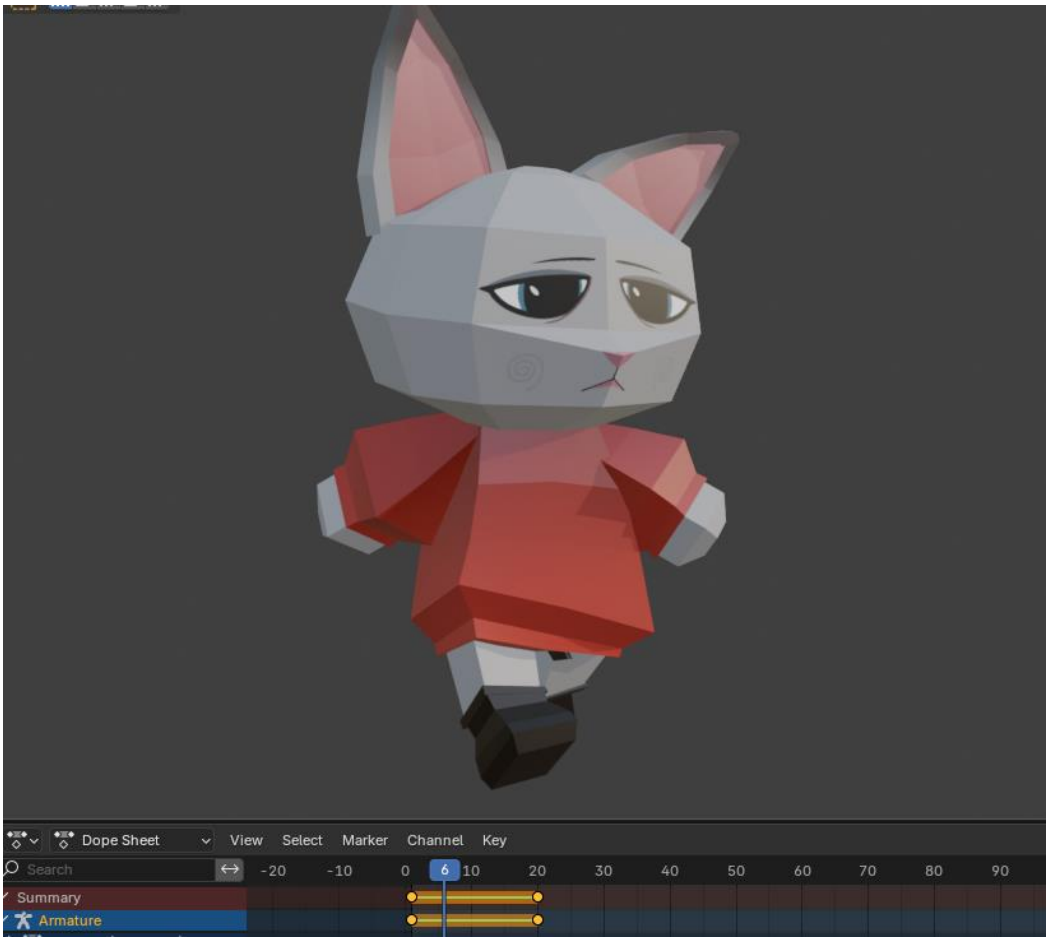

Рисунок 3.6 – Ключові кадри анімації ходьби персонажа в Blender

Після створення потрібних анімацій, наступним кроком є їх інтеграція до персонажу. Для цього використовується Animation Blueprint - це спеціалізовані Blueprint, які керують анімацією скелетного мешу під час симуляції або ігрового процесу. Графи редагуються у Animation Blueprint Editor, де можна поєднувати анімації, керувати кістками скелета або створювати логіку, яка визначатиме фінальну анімаційну позу для скелетного мешу на кожен кадр.

За допомогою цього, можна створити State Machine - модульна система, яка дозволяє задавати певні анімаційні стани та умови їхнього переходу. Вона переважно використовується для зв'язку анімацій із рухом персонажа, наприклад, коли він стоїть на місці (idle), йде, біжить або стрибає [22].

На рисунку 3.7 наведено State Machine у складі Animation Blueprint.

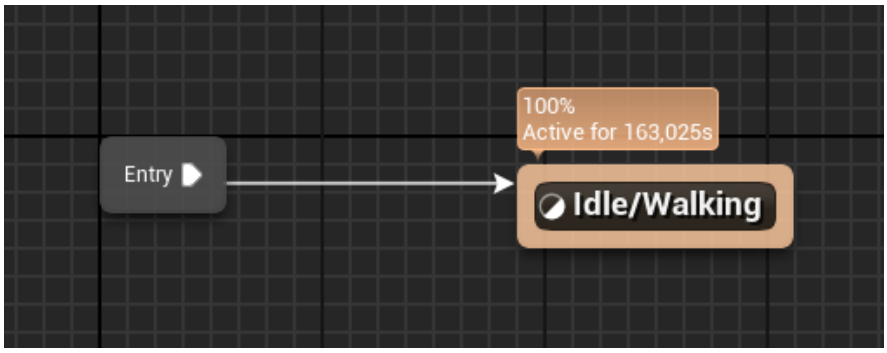

Рисунок 3.7 – State Machine у складі Animation Blueprint

За допомогою State Machine можна створити окремі стани, призначити для кожного з них конкретні анімації та налаштувати переходи між цими станами. Це значно спрощує процес побудови складних анімаційних логік, уникаючи перевантаження основного Anim Graph. На рисунку 3.8 наведено Anim Graph який керує відтворенням анімаційного переходу між станами.

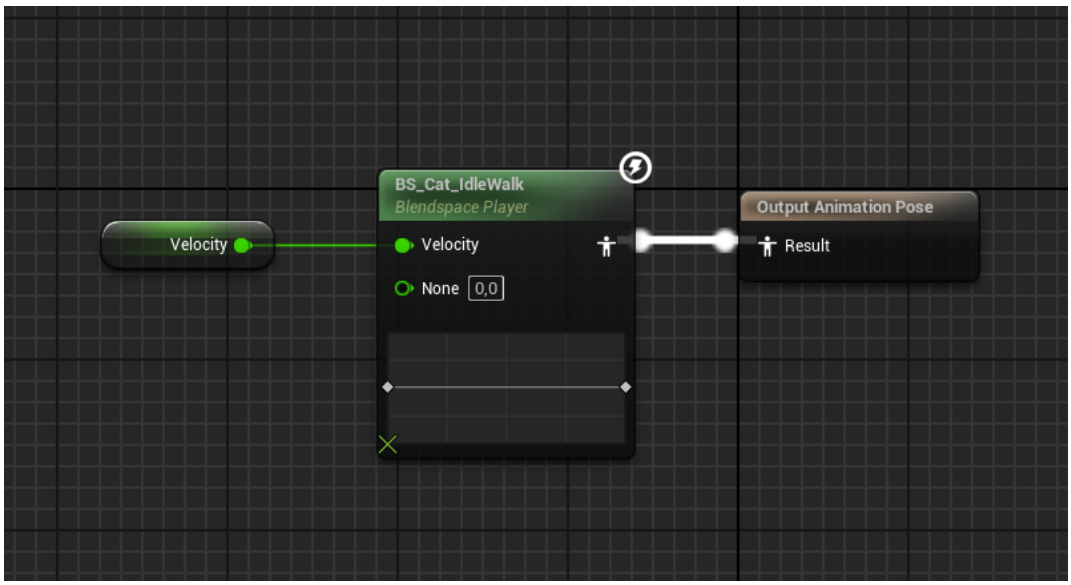

Рисунок 3.8 – Граф який керує відтворенням анімаційного переходу між станами.

Наступним кроком йде налаштування отримання швидкості персонажа, щоб керувати анімацією залежно від його руху. Спочатку ми отримуємо самого персонажа, до якого прив’язана анімація, і переконуємося, що це наш головний герой. Потім ми беремо його швидкість руху, рахуємо її довжину (наскільки

швидко він рухається), і зберігаємо це число у змінну Velocity. На рисунку 3.9 наведено налаштування Event Graph в Animation Blueprint.

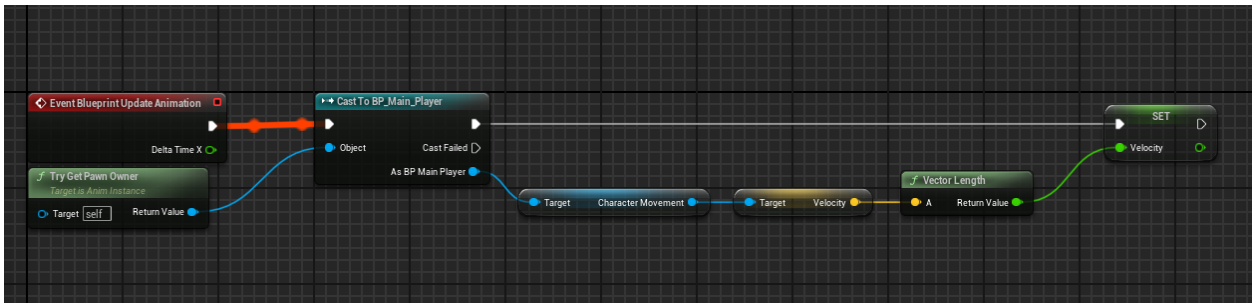

Рисунок 3.9 – Налаштування Event Graph в Animation Blueprint

У результаті правильної експортації та налаштування моделі в Unreal Engine 5 було успішно досягнуто її коректного відображення в ігровому середовищі.

3.4 Реалізація «танкового» управління та фіксованої камери

Фіксована камера у поєднанні з «танковим» управлінням – це класичний підхід до керування персонажем в проєктах кінця 90-х, який створював специфічну кінематографічну атмосферу.

Сутність цього підходу полягає в тому, що камера фіксується в заданій позиції на сцені (наприклад, кут кімнати), не слідкуючи динамічно за персонажем, як це робить сучасна третя особа камера. Зміна ракурсу відбувається лише під час переходу між зонами або тригерами, створюючи контрольовану композицію кадру.

«Танкове» управління полягає у тому, що персонаж рухається вперед або назад відповідно до свого поточного напрямку, а поворот здійснюється окремо – за допомогою клавіш вліво та вправо. Тобто, це означає, що персонаж буде рухатись вперед у напрямку в який він «дивиться», а не в тому напрямку який збігається з вектором камери.

Поєднання цих двох елементів дозволяє досягти певних дизайнерських цілей:

- Контроль над композицією – створення певних настроїв;
- Акцентування уваги гравця на певних деталях;
- Знизити навантаження на рендеринг, оскільки не потрібно обробляти все оточення;

Для створення фіксованих у грі Moonlight Song, було використано Blueprint-систему на основі тригерів. На рисунку 3.10 зображено налаштування логіки активації фіксованої камери.

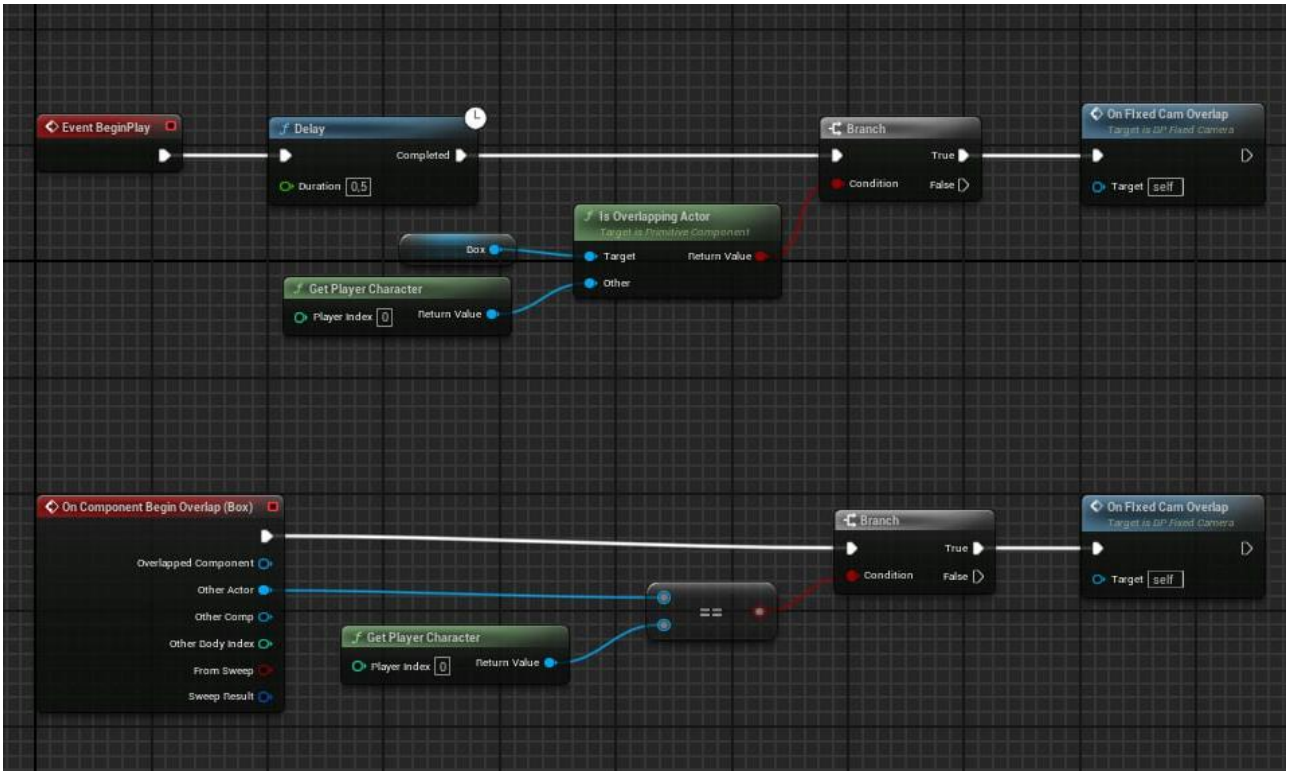

Рисунок 3.10 – Налаштування логіки активації фіксованої камери

Event BeginPlay – це подія, яка викликається автоматично один раз коли «актор» створюється або запускається на сцені. На верхньому фрагменті рисунку 3.10, зображено, що під час запуску гри відбувається затримка на 0.5 секунд, після чого відбувається перевірка чи персонаж гравця вже перебуває в зоні

камери. Якщо так – викликається подія On Fixed Cam Overlap, яка відповідає за активацію певної фіксованої камери [23].

На нижньому ж фрагменті обробляється динамічна подія On Component Begin Overlap (Box). Тобто коли гравець входить у зону за яку відповідає інша камера, то відбувається перевірка, чи об’єкт, що викликав подію є гравцем, і якщо це так то відбувається перехід до іншої камери яка відповідає за цю зону. Це забезпечує плавне перемикання між камерами.

На рисунку 3.11 наведено «зони» фіксованих камер.

Рисунок 3.11 – «Зони» фіксованих камер

Після налаштування логіки перемикання між фіксованими камерами, наступним кроком є налаштування системи керування. Для розробки системи керування за допомогою Enhanced Input, створюється Input Mapping Context в якому визначають усі необхідні дії управління персонажем. Кожен Context можна активувати або деактивувати динамічно в залежності від ситуації в грі — наприклад, у меню, в бойовому режимі або під час керування транспортом [24].

На рисунку 3.12 наведено ліву частину Blueprint-системи переміщення гравця.

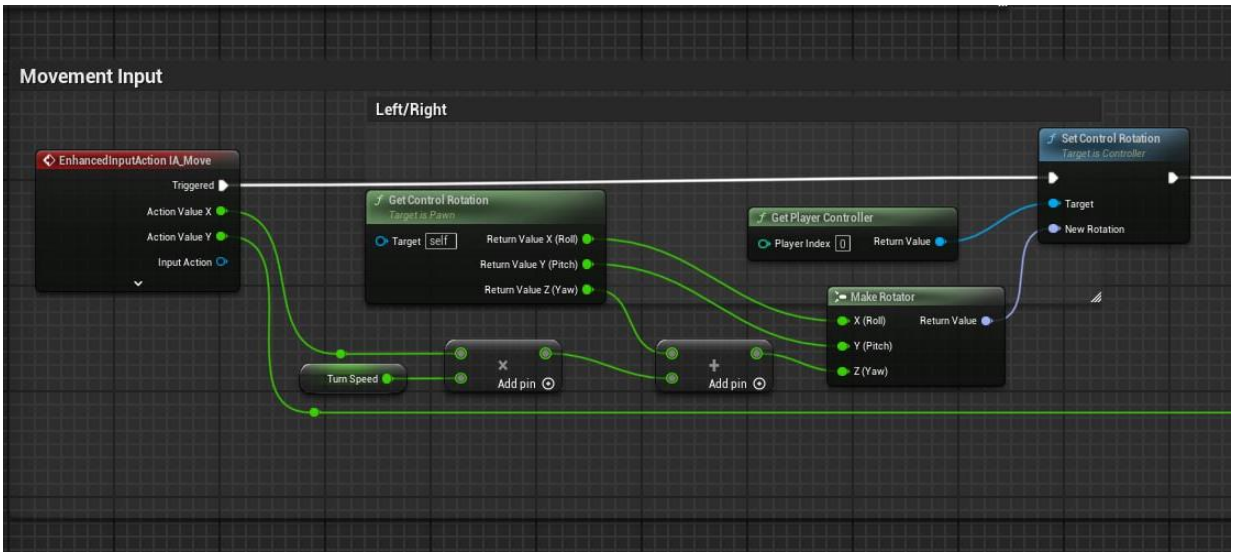

Рисунок 3.12 – Ліва частина Blueprint-системи переміщення гравця

Для налаштування управління вліво та вправо спочатку виконуються такі дії:

- Enhanced Input Action: IA_Move – використовується для читання двовимірного вектора. Значення X відповідає за рух вліво та вправо, Y за вперед та назад
- Get Control Rotation – отримується поточний оберт;
- Отриманий новий Rotator передається у Set Control Rotation, змінюючи кут огляду контролера, тобто змінюється напрямок, у який персонаж «дивиться» лицем.

Після цього йде налаштування управління вперед та назад, де знову отримується Get Control Rotation, і потім на основі куту повороту по вертикальній очі визначається вектор напряму вперед (Get Forward Vector). І Add Movement Input – додає рух у цьому напрямку.

На рисунку 3.13 наведено праву частину Blueprint-системи переміщення гравця.

Рисунок 3.13 – Права частина Blueprint-системи переміщення гравця

Також варто зазначити, що під час налаштування фіксованої камери, було використано Cinematic Camera (16:9 DSLR), яка має більший вибір налаштувань та допомагає досягти кінематографічного кадру, замість звичайної Camera (16:9 Film) яку пропонує Unreal Engine 5 на початку. На рисунку 3.14 наведено поточні налаштування камери [25].

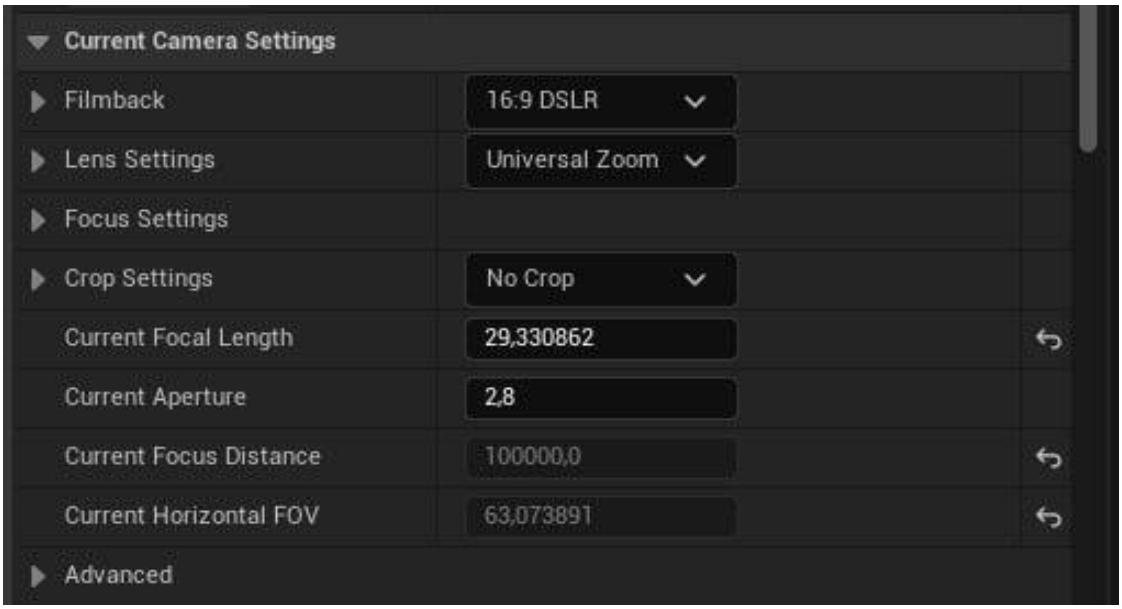

Рисунок 3.14 – Налаштування Cinematic Camera

І також було створено ефект Low Resolution для більш атмосферного вигляду, через Post Process Volume. На рисунку 3.15 зображено налаштування матеріалу Low Resolution.

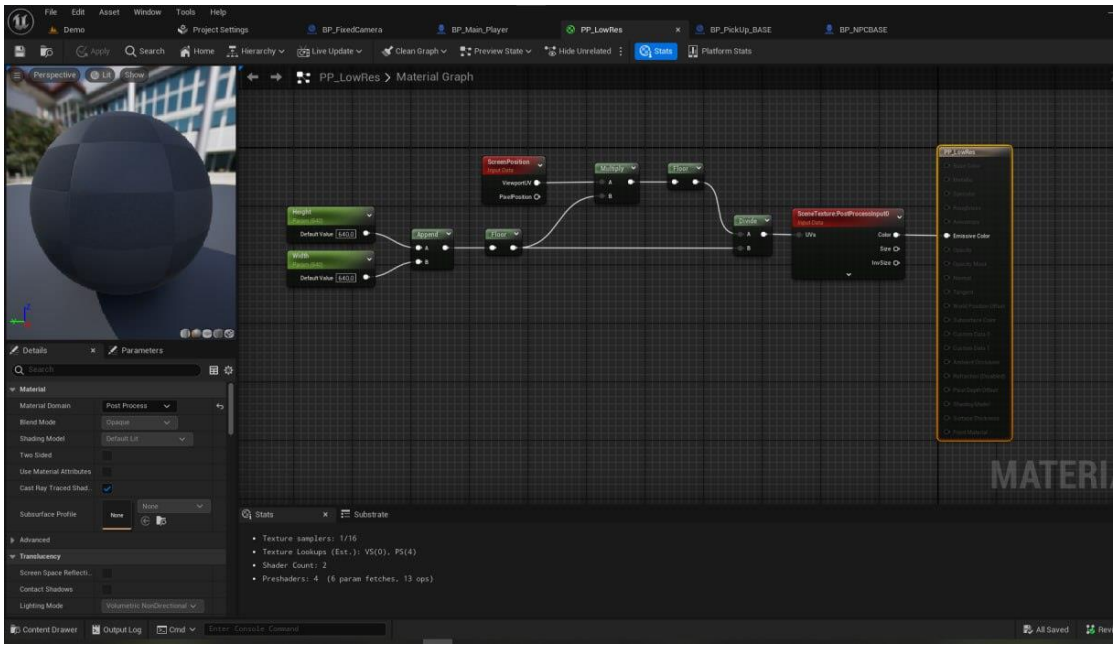

Рисунок 3.15 – Налаштування матеріалу Low Resolution

На рисунку 3.16 зображено вигляд ефекту у грі Moonlight Song.

Рисунок 3.16 – Вигляд ефекту у грі Moonlight Song

3.5 Реалізація мультимодального управління персонажем

Після реалізації «танкового» управління, також було реалізовано управління за допомогою автоматичного переміщення (автопілот), яке активувалось при дотику на екрані.

Це було реалізовано завдяки AIController – спеціальному контролеру штучного інтелекту в Unreal Engine 5, який дозволяє керувати персонажем через логіку автоматичного руху.

Основна логіка автопілоту була реалізована у класі AAutoMovePawn. Під час дотику до екрану викликається OnInputTouchBegin – який виявляє місце дотику. Якщо натискання виявлено (через GetHitResultUnderFinger), то визначаються координати на сцені, куди був здійснений дотик.

Далі ці координати проєктуються (ProjectPointToNavigation) на навігаційну сітку за допомогою навігаційної системи (UNavigationSystemV1). Якщо координати доступні для руху, то викликається ProcessAutoMove, який вже ініціює рух персонажа до обраної точки.

ProccesAutoMove – створює запит FAIMoveRequest, у якому вказується координата призначення – SetGoalLocation та AcceptanceRadius – радіус прийняття, після досягнення якого вважатиметься, що персонаж прибув у обране місце. Далі запит на пересування передається через MoveTo у AIController.

Також, для зручного та гнучкого керування, було реалізовано можливість переривати автопілот у будь-який момент руху за допомогою ручного керування. Будь-який рух вбік чи вперед або назад викликає метод (StopAutoPilot), який зупиняє поточне завдання AIController на переміщення і вимикає автопілот, надаючи користувачу управління в свої руки.

Поєднання цих двох способів керування персонажем – є досить зручним та потрібним сучасним іграм удосконаленням, адже дозволяє легко керувати персонажем як гравцям на мобільних пристроях, так і на PC або консолях.

3.6 Тестування роботи програми та аналіз результатів

Тестування гри є завершальним етапом розробки. У процесі створення було здійснено 50 тестових запусків для перевірки роботи впроваджених функцій та виявлення можливих проблем.

При запуску гри, нас одразу зустрічає головний герой який знаходиться серед лісової місцевості. Фіксована камера одразу фіксує персонажа всередині зони. На рисунку 3.17 зображено фрагмент початку гри.

Рисунок 3.17 – Фрагмент початку гри

Керування одразу переходить до гравця, і було спроектовано воно відбувається за принципом мультимодального управління, тобто може відбуватись за допомогою клавіш «WASD» або дотику тачскріну. У грі підтримується також керування контролером від серії Xbox на лівий стік.

Подорожуючи лісом, наш головний герой зустрічає привітна місцевість, де він може підібрати як гарбуз, так і ліхтар. Для взаємодії з предметом потрібно натиснути клавішу «E» на клавіатурі, або «A» на контролері. На рисунку 3.18 зображено фрагмент взаємодії з предметами та вид першої фіксованої камери.

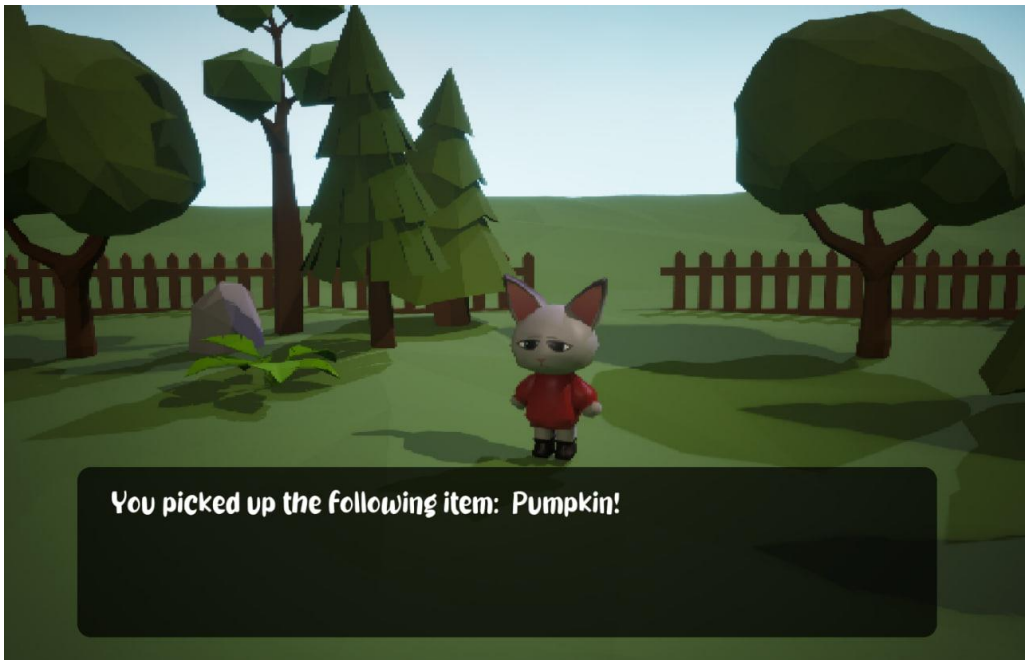

Рисунок 3.18 – Фрагмент взаємодії з предметом та вид першої фіксованої камери

При подальшому дослідженні лісу, головний герой знаходить котика, який при взаємодії з ним, веде себе досить підозріло. На рисунку 3.19 наведено фрагмент взаємодії з котом та вид другої фіксованої камери.

Рисунок 3.19 – Фрагмент взаємодії з котом та вид другої фіксованої камери

Також, по дорозі в глибини лісу, можна знайти галявину де відпочиває дівчинка і ще один котик. При взаємодії з дівчинкою, та натякне що вона достатньо голодна і їй хотілось би грибочка. На рисунку 3.20 наведено фрагмент прийняття квеста.

Рисунок 3.20 – Фрагмент прийняття квесту

Після цього, головний герой матиме ще можливість ще порозмовляти з іншими персонажами. До прикладу, котик запитає головного персонажа чи не бачив він поблизу його брата і поскаржиться нам на його відсутність. На рисунку 3.21 наведено фрагмент діалогу з котом

Рисунок 3.21 – Фрагмент діалогу з котом

Також, герой матиме можливість принести грибочок дівчинці, пошукавши його серед лісової місцевості. Його можна буде знайти поруч з дивним котом. На рисунку 3.22 зображено фрагмент процесу виконання квесту.

Рисунок 3.22 – Фрагмент процесу виконання квесту

Віднісши грибочок, дівчинка прийме його та віддячить головному герою за його допомогу. На рисунку 3.23 зображено фрагмент фінального завершення гри

Рисунок 3.23 – Фрагмент фінального завершення гри

Отже, розроблена гра Moonlight Song успішно пройшла тестування та відповідає всім заданим вимогам.

Під час розробки було виконано такі задачі:

- Налаштовано середовище розробки ігровий рушій Unreal Engine 5 для роботи з 3D графікою;
- Розроблено графічну частину гри у програмному забезпечені Blender та RizomUV.
- Підготовлено 3D-модель до використання в UE 5;
- Розроблено «танкове» управління та фіксовані камери;
- Здійснено тестування гри Moonlight Song.

Порівнюючи гру з іншими 3D інді-проєктами

Також було проведено порівняння розробленого програмного продукту з двома 3D-проєктами за ключовими аспектами. У таблиці 3.1 подано результати порівняння певних аспектів аналогів та розробленого програмного продукту.

Таблиця 3.1 – Результати порівняння певних аспектів аналогів та розробленого програмного продукту

	Розроблена гра	Crow Country	Fears to Fathom
Мультимодальне управління	+	+	–
Безкоштовна гра	+	–	+
Фіксовані камери	+	+	–
Підтримка контролера	+	+	–
Оптимізація для слабких ПК	+	–	–
Розроблено на Unreal Engine 5	+	–	+

Проаналізувавши ці ігри, їх можна описати так:

- Crow Country від розробника SFB Games – тривимірна гра з елементами survival horror, візуальний стиль якої виконаний у

малополігональному форматі та стилістиці ретро з ефектом низької роздільної здатності. Поєднує фіксовані камери з мультимодальним управлінням та має підтримку контролерів [26].

- Fears to Fathom від інді-розробника Rayll – тривимірна гра жанру жахів, створена у стилі «реальної історії» яку нам розповідає головний герой. Управління відбувається від першого лиця і поєднує пікселізовану графіку з ефектами, створюючи атмосферу напруженості [27].

Отже, протестувавши програмний продукт, та проаналізувавши його з аналогами, можна дійти висновку, що цілі які були поставлені на початку, успішно досягнуто.

3.7 Висновок до розділу третього

У даному розділі було обґрунтовано вибір ігрового рушія, а також проаналізовано вибір програмного забезпечення для створення графічної частини гри. Було оглянуто такі програми як Autodesk Maya, 3ds Max, ZBrush та Blender. Також були проаналізовані особливості процесу розробки.

У результаті проведеного аналізу для розробки було обрано ігровий рушій Unreal Engine 5 з поєднанням системи візуального програмування Blueprints та C++. Unreal Engine 5 є одним із найсучасніших і доступних рушіїв, що забезпечує широкі можливості для створення тривимірних ігор. До того ж, завдяки системі Blueprints, цей рушій дозволяє розробляти ігри навіть без глибоких знань програмування.

Для графічної частини обрано Blender у поєднанні з RizomUV, тому що це програмне забезпечення має нескладний інтерфейс, великий функціонал та зручність у використанні з UE 5.

Було підготовлено та налаштовано 3D-модель та її скелет до подальшого використання в ігровому рушії, замість базового скелету.

Також було розроблено систему «танкового» управління та фіксовані камери, за допомогою яких можна створити кінематографічну атмосферу. Додано ефект Low Resolution на камеру.

Було проведено тестування гри Moonlight Song. У результаті зроблено висновок, що всі основні механіки та функції було реалізовано. Розроблений програмний продукт має покращений функціонал порівняно з аналогами на 4 можливості, а саме: поєднання двох механік у грі, виконано у жанрі «затишна гра», відсутність багів, підтримка геймпаду.

ВИСНОВКИ

Усі поставлені задачі для реалізації комп’ютерної гри Moonlight Song були виконані у повному обсязі, а саме:

1. Проведено аналіз предметної області, існуючих ігор у жанрах 3D, ігор з різними видами управління, методів реалізації та обґрунтовано доцільність розробки;
2. Спроєктовано комп’ютерну гру Moonlight Song з покращеним способом керування в стилі класичних;
3. Реалізовано комп’ютерну гру Moonlight Song з покращеним способом керування в стилі класичних ігор за допомогою ігрового рушія Unreal Engine 5;
4. Виконано тестування програми та проаналізовано отримані результати;
5. Розроблено інструкцію користувача.

Для розробки використовувався ігровий рушій Unreal Engine 5 з використанням системи візуальних сценаріїв Blueprint та C++. Він має широкий функціонал для роботи з тривимірними проєктами.

Для графічної частини було використано безкоштовне програмне забезпечення Blender у поєднанні з RizomUV, який надає різноманітні інструменти для роботи з 3D-моделями перед їх експортацією в UE 5.

Розроблений продукт був успішно протестований і порівнянний з іграми аналогами. Робота виконана згідно вимог та методичних рекомендацій щодо підготовки бакалаврських кваліфікаційних робіт [28].

Мету роботи було досягнуто завдяки розробці гри Moonlight Song, в якій було покращено спосіб керування, який забезпечує можливість мультимодального управління рухом персонажа на різних платформах.

Отже, розроблена гра Moonlight Song з покращеним способом керування в стилі класичних ігор відповідає заданим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Поліщук А. І., Малініч І. П. Платформно-адаптивний метод керування ігровим персонажем у середовищі Unreal Engine 5. *LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації.* URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/24019 (дата звернення: 6.05.2025).

2. Xbox Adaptive Controller. URL: https://www.xbox.com/en-US/auth/msa?action=loggedIn&locale_hint=en-US (дата звернення: 6.05.2025)

3. How the Video Game Industry Is Changing. URL: https://www.investopedia.com/articles/investing/053115/how-video-game-industry-changing.asp (дата звернення 6.05.2025)

4. Baldur`s Gate 3. URL: https://baldursgate3.game (дата звернення: 7.05.2025).

5. Resident Evil. URL: https://www.britannica.com/topic/Resident-Evil (дата звернення: 7.05.2025).

6. GameFrame: Lessons on Fear Mechanics from the Video Game Series Silent Hill. URL: https://medium.com/@antonio.sadaric1/gameframe-lessons-on-fear-mechanics-from-the-video-game-series-silent-hill-29ca40104e99 (дата звернення: 7.05.2025).

7. Dead by Daylight. URL: https://deadbydaylight.com (дата звернення: 7.05.2025).

8. ANNO: Mutationem. URL: https://store.steampowered.com/app/1368030/ANNO_Mutationem (дата звернення: 7.05.2025)

9. 3D Modeling in Gaming Industry: Evolution and Impact. URL: https://www.nextechar.com/blog/3d-modeling-in-gaming-industry (дата звернення: 8.05.2025)

10. The Evolution of 3D Graphics in Video Games (The Hunt for Photorealism). URL: https://gameopedia.com/blogs/the-evolution-of-3d-graphics-in-video-games-(the-hunt-for-photorealism) (дата звернення: 8.05.2025).

11. Rental. URL: https://smarto-club.itch.io/rental (дата звернення: 8.05.2025).

12. Bendy and The Ink Machine. URL: https://www.joeydrewstudios.com/batim (дата звернення: 8.05.2025).

13. Designing Game Controls. URL: https://www.gamedeveloper.com/design/designing-game-controls (дата звернення: 9.05.2025)

14. Silent Hill 3. URL: https://silenthill.fandom.com/en/wiki/Silent_Hill_3 (дата звеpення: 9.05.2025)

15. Unity. URL: https://unity.com/products/unity-engine (дата звернення: 9.05.2025)

16. Unreal Engine. URL: https://www.unrealengine.com/en-US (дата звернення: 9.05.2025).

17. Blueprints Visual Scripting. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprints-visual-scripting-in-unreal-engine (дата звернення: 10.05.2025).

18. Blender. URL: https://www.blender.org (дата звернення: 11.05.2025).

19. RizomUV. URL: https://www.rizomuv.com (дата звернення: 11.05.2025).

20. Lost Records: Bloom & Rage URL: https://store.steampowered.com/app/1902960/Lost_Records_Bloom__Rage/ (дата звернення: 12.05.2025)

21. Skeletons. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/skeletons-in-unreal-engine (дата звернення: 14.05.2025)

22. State Machines. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/state-machines-in-unreal-engine (дата звернення: 14.05.2025).

23. Events. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/events-in-unreal-engine (дата звернення: 14.05.2025)

24. Enhanced Input. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/enhanced-input-in-unreal-engine (дата звернення: 15.05.2025)

25. Cine Camera Actor. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/cinematic-cameras-in-unreal-engine (дата звернення: 15.05.2025)

26. Crow Country. URL: https://store.steampowered.com/app/1996010/Crow_Country/ (дата звернення: 16.05.2025).

27. Fear to Fathom. URL: https://rayll.itch.io/fears-to-fathom (дата звернення: 16.05.2025).

28. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. Вінниця : ВНТУ, 2023. (PDF, 54 с.).

ДОДАТКИ

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Комп'ютерна гра Moonlight Song з покращеним способом керування в стилі класичних ігор

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,72 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Малініч І.П.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Поліщук А.І.
(підпис) (прізвище, ініціали)

ДОДАТОК Б

ІЛЮСТРАТИВНА ЧАСТИНА

«КОМП'ЮТЕРНА ГРА MOONLIGHT SONG З ПОКРАЩЕНИМ СПОСОБОМ КЕРУВАННЯ В СТИЛІ КЛАСИЧНИХ ІГОР»

Виконала: студентка 4 курсу, групи 4КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Поліщук А.І.
(прізвище та ініціали)

Керівник: асистент каф. КН
Малініч І. П.
(прізвище та ініціали)
«03» червня 2025 р.

Вінниця ВНТУ - 2025 рік

Рисунок Б.1 – Приклад використання малополігональної графіки в Resident Evil 1(1996)

Рисунок Б.2 – Приклад використання багатополігональної графіки в ремастері Resident Evil 1(2002)

Рисунок Б.3 – Приклад використання стилізованої 3D графіки у Rental

Рисунок Б.4 – Приклад слідкуючої камери з «танковим» управлінням у 3D-грі Silent Hill 3

Рисунок Б.5 – Приклад фіксованої камери з «танковим» управлінням у грі Resident Evil Code: Veronica

Рисунок Б.6 – Приклад з виглядом від першої особи у грі Cyberpunk 2077

Рисунок Б.7 – Приклад з виглядом від третьої особи у ремейку гри Resident Evil 4

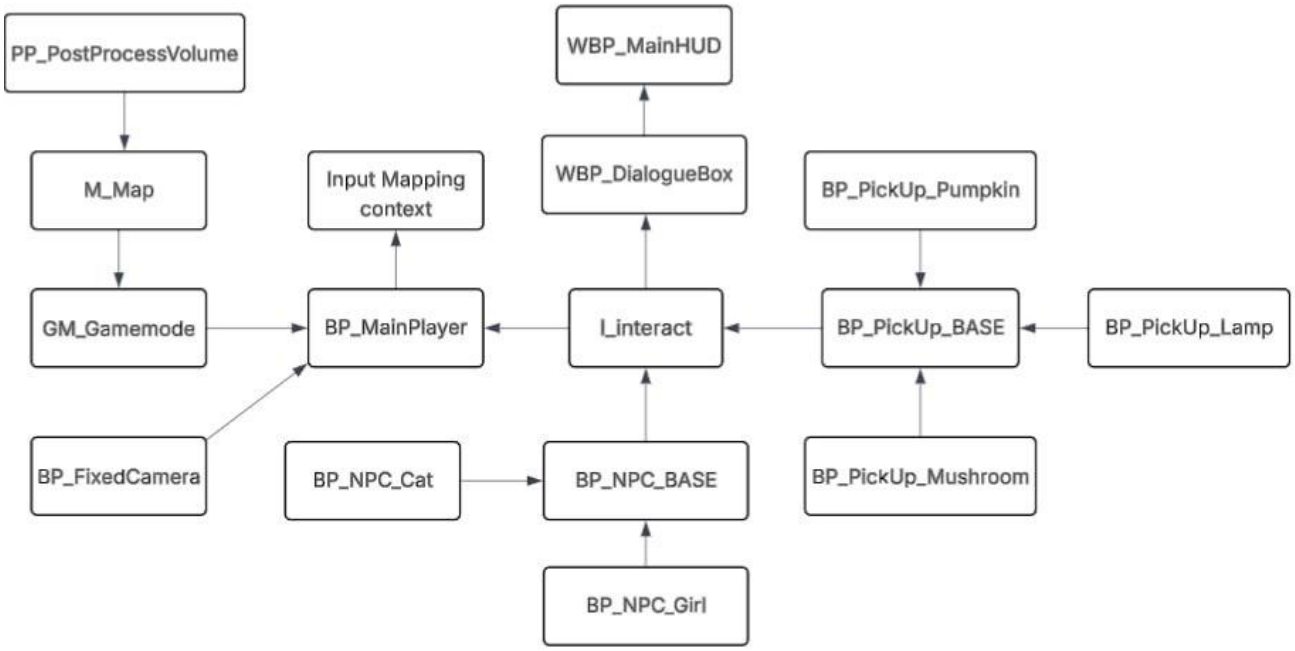

Рисунок Б.8 – Загальний план гри

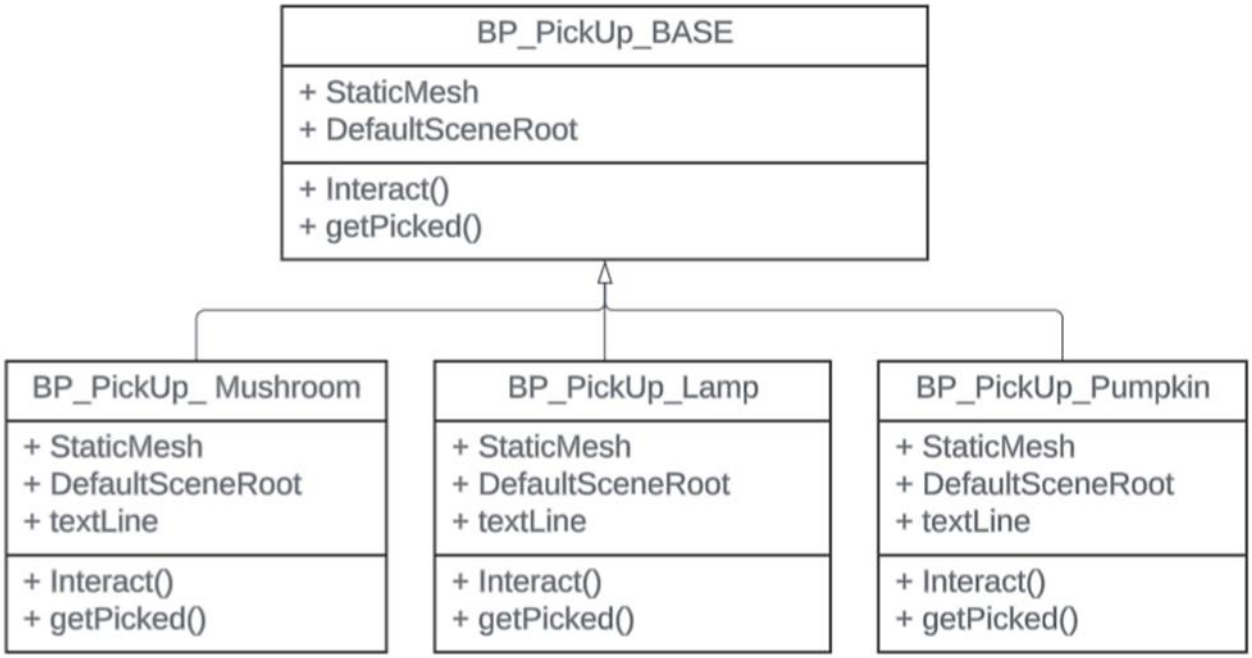

Рисунок Б.9 – UML-діаграма класів для предметів

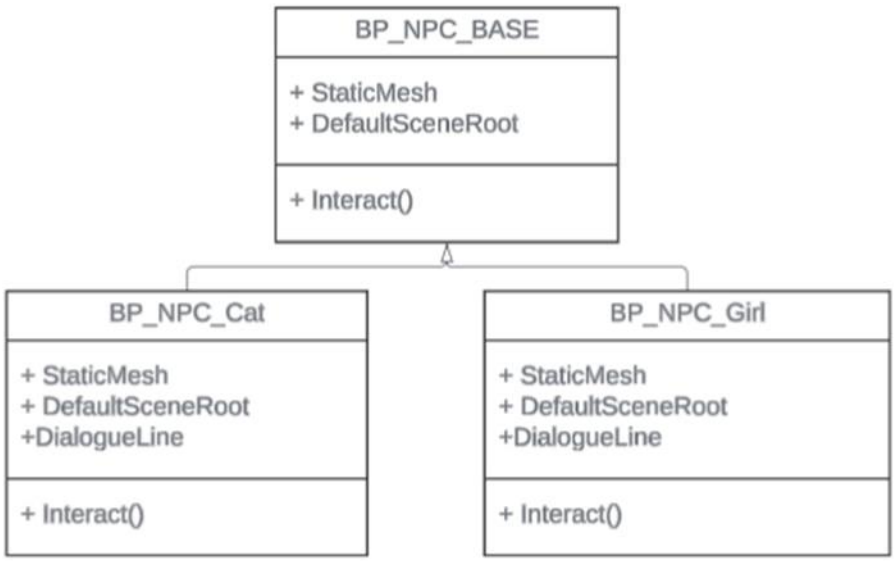

Рисунок Б.10 – UML-діаграма класів для NPC-персонажів

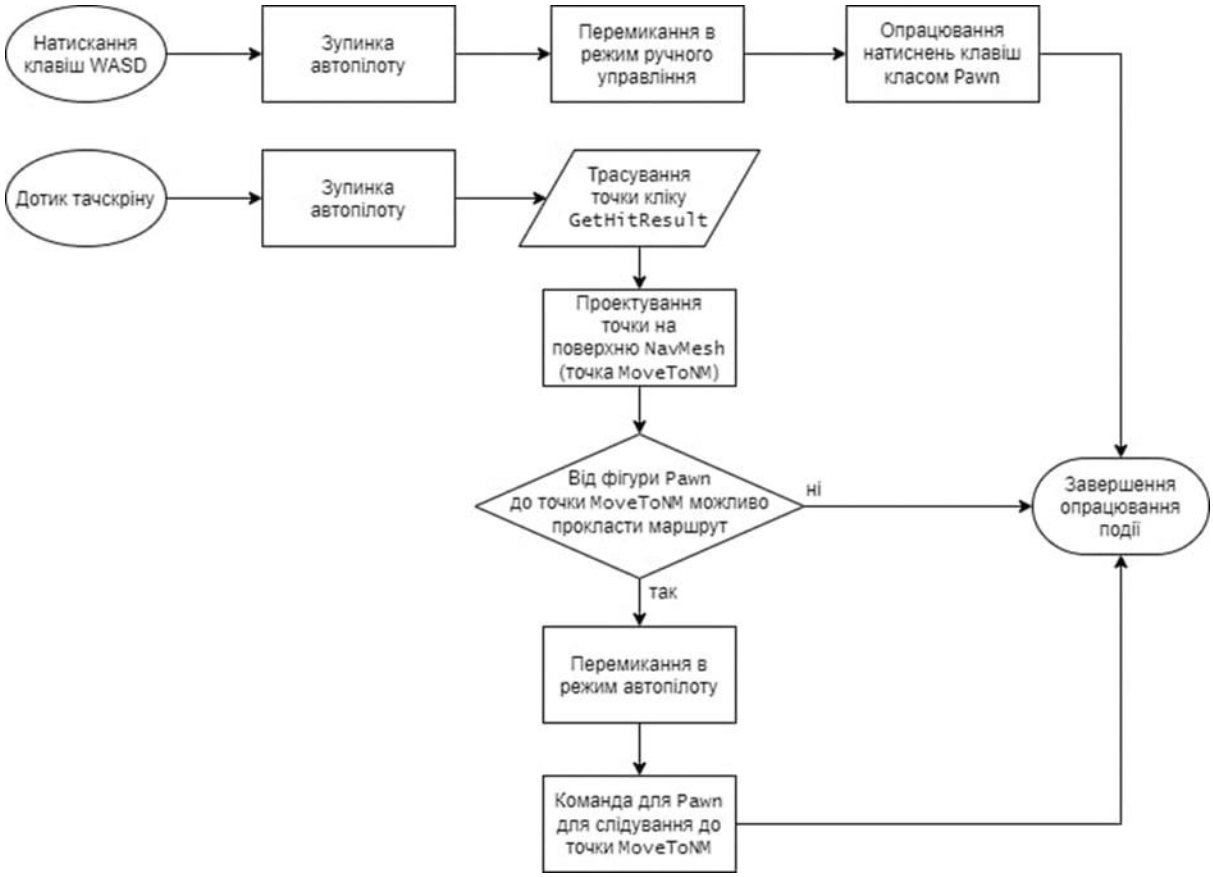

Рисунок Б.11 – Схема алгоритму мультимодального управління персонажем

Рисунок Б.12 – Схема опрацювання «танкового» управління

Рисунок Б.13 – «Зони» фіксованих камер

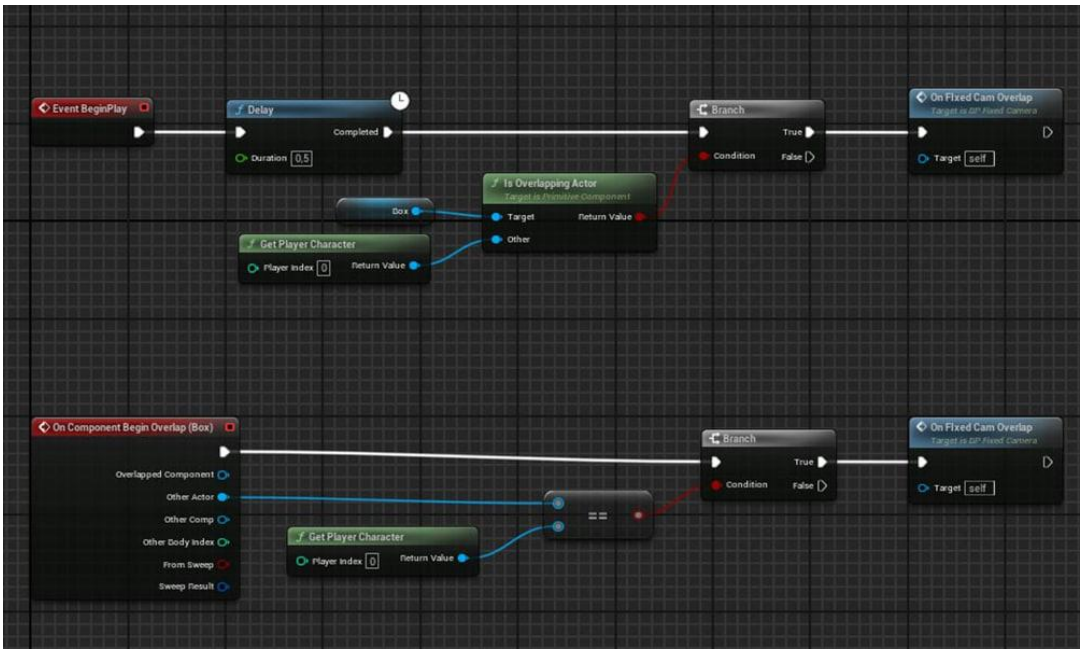

Рисунок Б.14 – Налаштування логіки активації фіксованої камери

Рисунок Б.15 – Фрагмент взаємодії з предметом та вид першої фіксованої камери

Рисунок Б.16 – Фрагмент взаємодії з котом та вид другої фіксованої камери

Рисунок Б.17 – Фрагмент прийняття квесту

ДОДАТОК В
ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
#include "AutoMovePawn.h"
#include "GameFramework/PlayerController.h"
#include "AIController.h"
#include "NavigationSystem.h"
#include "DrawDebugHelpers.h"

AAutoMovePawn::AAutoMovePawn()
{
   PrimaryActorTick.bCanEverTick = true;
   bIsAutopilotEnabled = false;
}

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

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

void AAutoMovePawn::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
   Super::SetupPlayerInputComponent(PlayerInputComponent);

   PlayerInputComponent->BindAxis("MoveForward", this, &AAutoMovePawn::MoveForward);
   PlayerInputComponent->BindAxis("MoveRight", this, &AAutoMovePawn::MoveRight);
   PlayerInputComponent->BindTouch(IE_Pressed, this, &AAutoMovePawn::OnInputTouchBegin);
}

void AAutoMovePawn::MoveForward(float Value)
{
   if (Value != 0.0f)
   {
      StopAutopilot();
      AddMovementInput(GetActorForwardVector(), Value);
```

```
    }
}

void AAutoMovePawn::MoveRight(float Value)
{
    if (Value != 0.0f)
    {
        StopAutopilot();
        AddMovementInput(GetActorRightVector(), Value);
    }
}

void AAutoMovePawn::StopAutopilot()
{
    if (bIsAutopilotEnabled)
    {
        AAIController* AIController = Cast<AAIController>(GetController());
        if (AIController)
        {
            AIController->StopMovement();
        }
        bIsAutopilotEnabled = false;
    }
}

void AAutoMovePawn::OnInputTouchBegin(ETouchIndex::Type FingerIndex, FVector Location)
{
    StopAutopilot();

    APlayerController* PC = Cast<APlayerController>(GetController());
    if (!PC) return;

    FHitResult Hit;
    if (PC->GetHitResultUnderFinger(FingerIndex, ECC_Visibility, true, Hit))
    {
        FVector HitLocation = Hit.Location;

        UNavigationSystemV1* NavSys = FNavigationSystem::GetCurrent<UNavigationSystemV1>(GetWorld());
        if (NavSys && NavSys->GetNavDataForProps(GetNavAgentPropertiesRef()))
        {
            FNavLocation ProjectedLocation;
```

```
            if (NavSys->ProjectPointToNavigation(HitLocation, ProjectedLocation))
            {
                bIsAutopilotEnabled = true;
                ProcessAutoMove(ProjectedLocation.Location);
            }
        }
    }
}

void AAutoMovePawn::ProcessAutoMove(const FVector& Destination)
{
    AAIController* AIController = Cast<AAIController>(GetController());
    if (AIController)
    {
        FAIMoveRequest MoveRequest;
        MoveRequest.SetGoalLocation(Destination);
        MoveRequest.SetAcceptanceRadius(AcceptanceRadius);

        FNavPathSharedPtr NavPath;
        if (AIController->MoveTo(MoveRequest, &NavPath) == EPathFollowingRequestResult::RequestSuccessful)
        {
            if (bDrawDebugPath && NavPath.IsValid())
            {
                for (int32 i = 0; i < NavPath->GetPathPoints().Num() - 1; ++i)
                {
                    DrawDebugLine(GetWorld(),
                        NavPath->GetPathPoints()[i].Location,
                        NavPath->GetPathPoints()[i + 1].Location,
                        FColor::Blue, false, 2.0f, 0, 3.0f);
                }
            }
        }
        else
        {
            bIsAutopilotEnabled = false;
        }
    }
}
```

ДОДАТОК Г
ІНСТРУКЦІЯ КОРИСТУВАЧА

Для запуску гри потрібно відкрити теку гри, та запустити файл DiplomaGame.exe. (рис. Г.1)

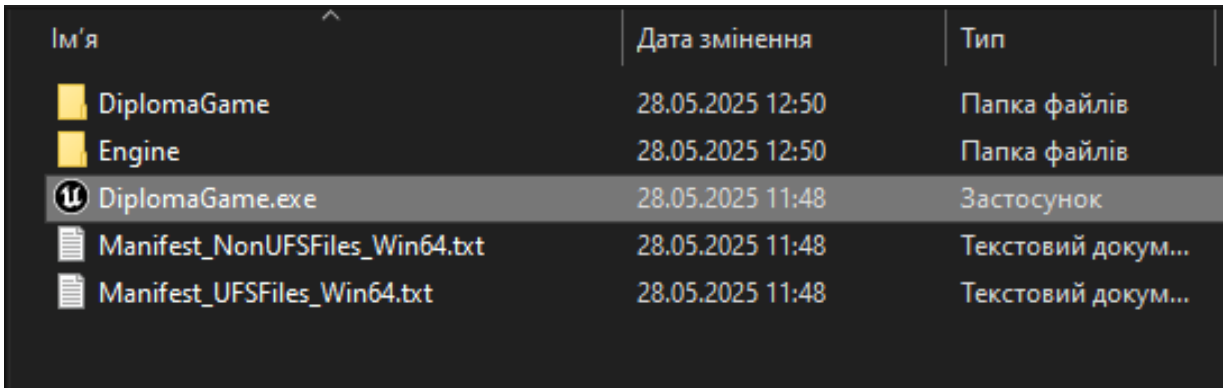

Рисунок Г.1 – Тека гри

Одразу після запуску, нас зустрічає гра. Взаємодія з предметами відбувається на клавішу E. Керування може відбуватись за допомогою джойстика, мишки або клавіатури. (рис. Г.2)

Рисунок Г.2 – Взаємодія з предметами