ARCHITECTURAL FRAMEWORK

FOR ROLE-PLAYING GAMES

PROJECT ID: 2658

BY

LUQMAN HAKIM BIN BASIR

1221303485

PROJECT REPORT SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF COMPUTER SCIENCE (HONOURS)

(SOFTWARE ENGINEERING)

in the

Faculty of Computing and Informatics

MULTIMEDIA UNIVERSITY

MALAYSIA

MAY 2024

DECLARATION

I hereby declare that the work has been done by myself and no portion of the work contained in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institution of learning.

Name of candidate: LUQMAN HAKIM BIN BASIR

Faculty of Computing & Informatics

Multimedia University

Date: 27: 05: 2024

ACKNOWLEDGMENTS

I would like to express my gratitude to all those who have contributed to the successful completion of my Final Year Project, "Architectural Framework for Role-Playing Games."

I extend my heartfelt thanks to my project supervisor, Dr Wong Ya Ping, for his invaluable guidance, and support throughout the entire research and development process. Their insights and constructive feedback have been instrumental in shaping the direction of this project. Thus, Warm thanks to the Multimedia University for providing the necessary resources and a conducive environment for the development of this project.

Special appreciation goes to the RPG enthusiasts and communities that have been a wellspring of inspiration. Their passion for storytelling and game design has been a driving force, and I am honoured to contribute to the shared world of role-playing games.

Furthermore, to my family and friends, thank you for your unwavering support and understanding as I delved into the intricacies of creating an architectural framework for RPGs. Your encouragement has been instrumental in this journey. Finally, I would like to thank God Almighty for providing me guidance to success.

Thank you.

LUQMAN HAKIM BIN BASIR

MULTIMEDIA UNIVERSITY

31/1/2024

ABSTRACT

This project aims to develop a custom architectural framework for Role-Playing Games (RPGs) using Unreal Engine 5, emphasizing scalability, maintainability, and performance. The framework addresses the significant challenge faced by game developers of constructing complex RPG systems from scratch. Many developers face challenges in implementing efficient and maintainable systems that ensure optimal performance. The proposed solution alleviates these challenges by providing pre-built, highly customizable systems that streamline the development process. Leveraging Unreal Engine 5's Blueprint visual scripting, the framework makes game development accessible to developers with varying levels of coding expertise.

The primary problem addressed in this project is the significant time and resource investment required by game developers to create RPG systems from the ground up in Unreal Engine 5. The developed framework integrates key RPG components: Character System, Combat System, Quest System, and Item System. These components are modular and can be customized to fit various RPG styles, providing a robust and versatile foundation for game development.

The project's objectives included conducting a detailed analysis of challenges related to scalability, maintainability, and performance in RPGs; developing a custom-made architectural framework prioritizing features such as Character Designer, Quest Generator, Item Creator, and Combat Creator; and creating a playable RPG prototype to showcase the framework's functionality and user experience. Comprehensive research was conducted to identify best practices and design patterns, which were then applied to develop a robust architectural framework. The successful implementation and integration of all four systems using Unreal Engine 5 Blueprints ensured ease of use and high customization.

The testing phase involved functional, integration, and usability testing. The framework was evaluated for character stat calculations, leveling mechanics, user interface accuracy, enemy AI behavior, quest functionalities, and item equipping mechanics. Integration tests ensured seamless interaction between systems, while usability tests validated the framework's intuitiveness from a developer's perspective. The results indicated high functionality with minor bugs that were systematically addressed. The framework exhibited excellent performance and scalability, though some areas such as attack tracing and combo handling require further refinement.

This project provided valuable insights into game development using Unreal Engine 5, highlighting the importance of time management, logical problem-solving, and continuous learning. The framework demonstrated significant potential for improving RPG development efficiency. Future work will focus on refining combat mechanics, adding more character and item variations, implementing advanced animations, and enhancing overall user experience. The framework's commercialization on the Unreal Engine Marketplace is planned, aiming to provide game developers with a powerful tool to expedite their RPG projects.

In summary, this project successfully developed a comprehensive architectural framework for RPGs, addressing critical challenges and offering a valuable resource for game developers. The framework's modular design and ease of use promise significant contributions to the field of RPG development, paving the way for future enhancements and broader adoption.

TABLE OF CONTENTS

DECLARATION iii

ACKNOWLEDGMENTS iv

ABSTRACT v

TABLE OF CONTENTS vii

LIST OF TABLES x

LIST OF FIGURES xi

LIST OF ABBREAVIATION xii

CHAPTER 1 INTRODUCTION 1

1.1 Problem Statement 1

1.1.1 Problem Identification 1

1.1.2 Personal Interest 2

1.2 Project Objectives: 2

1.3 Deliverables 3

1.3.1 Custom Architectural Framework for RPGs with Unreal Engine 3

1.3.2 Playable RPG Prototype 4

1.3.3 Evaluation Report 4

1.4 Scope Definition 5

1.4.1 System Boundaries 5

1.4.2 Functionality Limitations 5

1.5 Chapter Organization: 6

CHAPTER 2 BACKGROUND STUDY 9

2.1 Scalability in RPGs 10

2.1.1 Definition and Importance 10

2.1.2 Existing Solutions and Approaches 11

2.1.3 Case Study on Scalability 13

2.2 Maintainability in RPGs 14

2.2.1 Definition and Importance: 14

2.2.2 Best Practices and Design Patterns 16

2.2.3 Case Study on Maintainability 17
2.3 Performance in RPGs 19
2.3.1 Definition and Importance: 19
2.3.2 Optimization Techniques and Strategies: 22
2.3.3 Case Studies on Performance Optimization 24
2.4 Overview of Architecture Styles 26
2.4.1 Entity-Component-System (ECS) 26
2.4.2 Model-View-Controller (MVC) 27
2.4.3 Component-Based Architecture 27
2.4.4 Hierarchical Level Streaming 28
2.4.5 Advantages and Disadvantages 28
2.4.6 Impact on Scalability, Maintainability, and Performance: 30
CHAPTER 3 REQUIREMENTS ANALYSIS 31
3.1 User Requirements: 32
3.1.1 User Personas 33
3.1.2 Use Case Scenarios 34
3.1.3 Use Case Diagram 36
3.2 Functional Requirements: 38
3.2.1 Character System Requirements: 38
3.2.2 Quest System Requirements: 39
3.2.3 Item & Weapon System Requirements: 39
3.2.4 Combat System Requirements: 40
3.3 Non-Functional Requirements 40
3.4 Technical Requirements 41
CHAPTER 4 DESIGN 44
4.1 Features of the RPG Architectural Framework 45
4.1.1 Features of the Character Designer 45
4.1.2 Features of the Quest Creator 47
4.1.3 Features of the Item Creator 48
4.1.4 Features of the Combat Creator 49
4.2 Technical Representations 50
4.2.1 Entity Relationships Diagram 50
4.2.2 Flow Chart for Character Designer 52
CHAPTER 5 IMPLEMENTATION 54

5.1 Technical Implementation 54
5.2 Development Process 55
5.3 System Implementation Details 56
5.3.1 Character System Framework 56
5.3.2 Combat System Framework 63
5.3.3 Quest System Framework 68
5.3.4 Item System Framework 73
CHAPTER 6 TESTING 75
6.1 Testing Plan and Execution 75
6.1.1 Functional Testing 75
6.1.2 Integration Testing 77
6.1.3 Usability Testing 78
6.2 Results Analysis and Discussion 78
6.2.1 Test Results 79
6.2.2 Encountered Issues and Bug Fixes 80
6.2.3 Overall Analysis 81
CHAPTER 7 CONCLUSION 82
7.1 Evaluation of Project Objectives 83
7.2 Major Learnings from the Project 85
7.3 Remaining Work, Potential Improvements and Future Plans 85
REFFERENCES 88
APPENDIX 91
Appendix A: Turnitin Report 91

LIST OF TABLES

Table 2.1 Advantages and Disadvantages of Architecture Styles 29

LIST OF FIGURES

Figure 3.1 Use Case Diagram of RPG Architectural Framework 36
Figure 4.1 Entity Relationships Diagram of the RPG Architectural Framework 50
Figure 4.2 Flow Chart of Character Creation with Character Designer 52
Figure 5.1 Increase Max Health Function 56
Figure 5.2 Increase EXP Function 57
Figure 5.3 Increase Level Function 57
Figure 5.4 EventGraph of BPC_CharacterStat 58
Figure 5.5 BPC_ThirdPersonHero Viewport 59
Figure 5.6 Main_HUD Designer Mode 59
Figure 5.7 Setting UI Event 60
Figure 5.8 Enemy AI Radius HitBox 61
Figure 5.9 Enemy AI Reference Flow 62
Figure 5.10 Attack 1 Animation Montage 63
Figure 5.11 Attack 1 Animation Implementation 64
Figure 5.12 Sphere Tracing to Detect Enemies 64
Figure 5.13 Event AnyDamage for BP_Dummy Blueprint 65
Figure 5.14 Continuation of Event AnyDamage for BP_Dummy Blueprint 65
Figure 5.15 BP_Projectile_Fireball Viewport 66
Figure 5.16 BP_AOE_FireTornado 67
Figure 5.17 Quest System Related Files 68
Figure 5.18 WB_Quest List UI Designer 69
Figure 5.19 Blueprint Graph of WB_QuestList 69
Figure 5.20 Quest View UI 70
Figure 5.21 Quest NPC 71
Figure 5.22 BPC_Equipping Reference Viewer 73

LIST OF ABBREAVIATION

Abbreviation	Definition
RPG	Role Playing Games
FYP	Final Year Project
ECS	Entity-Component-System
MVC	Model-View-Controller
SE	Software Engineering
FPS	First Person Shooter
NPC	Non-Playable Character
GPU	Graphic Processing Unit
AI	Artificial Intelligence
HP	Health Points
EXP	Experience Points
UI	User Interface
BP	Blueprint
WB	Widget Blueprint
AOE	Area of Effect
VFX	Visual Effects

CHAPTER 1
INTRODUCTION

1.1 Problem Statement

Role Playing Games (RPG) is a genre within the industry of video games where it consists of the intricate worlds and deep gameplay from combat to forging items or weapons. RPGs have captivated players for decades, yet developers often face architectural issues that limit their creations' potential. Current frameworks struggle with inflating storage demands, balancing act inaccuracies, and performance bottlenecks, hindering creativity and scalability. These issues are logically derived from the field of Software Engineering (SE). As a passionate gamer and aspiring developer, I faced multiple problems when I was trying to enjoy playing existing RPGs to the point where I wanted to solve those game developers' problems myself.

1.1.1 Problem Identification

The problems that the game developers are encountering would be classified as these challenges:

- Scaling: Current frameworks often buckle under the weight of expanding content, burdening players with excessive storage requirements.
- Maintainability: Games are evolving with updates and player feedback. Yet, modifying core components within current frameworks can feel like navigating a maze, delaying long-term and short-term maintainability and agility.
- Performance: Immersing players in the real world demands efficient resource utilization because not everyone can afford good computers. However, many existing architectures out there fall short, resulting in performance inconsistencies that disrupt the flow of gameplay.

1.1.2 Personal Interest

This project is more than just academic for me; it's about revolutionizing RPG game development. I'm passionate about creating games that go beyond imagination and set new industry standards. With every line of code, I aim to unveil a game that resonates with players worldwide and makes a lasting impact on the gaming world. My experience with RPGs has given me a deep love for their storytelling and strategy. Seeing the constraints of current frameworks has inspired me to innovate. Maybe one day my upcoming development of the RPG video game would attract the whole world’s attention.

1.2 Project Objectives:

This project aims to address the limitations of existing RPG development frameworks by creating a novel framework that enhances scalability, maintainability, and performance. To achieve this goal, I have established the following objectives:

Objective 1: Develop a Character Designer

Create a robust character design system that allows for customizable character stats, leveling, and UI. This system will focus on scalability and maintainability, ensuring that it can handle a variety of character types and abilities.

Objective 2: Develop a Quest Generator

Build a flexible quest generation system that enables the creation of complex quest lines with NPC interactions and objective tracking. This system will be designed to maintain high performance even as the number of quests and interactions increases.

Objective 3: Develop an Item Creator

Implement an item creation system that supports diverse item types, equipping mechanics, and inventory management. The system will be built with scalability in mind to accommodate a large number of items and ensure ease of maintenance.

Objective 4: Develop a Combat Creator

Establish a modular combat system with customizable attack and defense logic. This system will be optimized for performance and designed to integrate seamlessly with the character and item systems.

Objective 5: Implement a Playable RPG Prototype

Utilize the custom-made framework to develop a playable RPG prototype. This prototype will demonstrate the functionality and user experience of the framework. Comprehensive playtesting will be conducted to evaluate the framework's scalability, maintainability, and performance compared to a baseline prototype built with a traditional framework.

1.3 Deliverables

This project will produce several key deliverables demonstrating the research, design, implementation, and evaluation of the custom architectural framework for RPG development within Unreal Engine.

1.3.1 Custom Architectural Framework for RPGs with Unreal Engine

This architectural framework will be the main product of FYP Phase 2 concerning Objective 2. The framework will be tailored to fit the creation of video

games related to the genre of RPG by using its templates. The main features of the framework would be:

1. Character System
2. Quest System
3. Items & Weapons System
4. Combat System

The user document will also be made to ensure transparency and reproducibility, making it easy for others to understand the design choices and potentially contribute into my work.

1.3.2 Playable RPG Prototype

This functional prototype will demonstrate the practical implementation of my custom architectural framework in a basic RPG setting:

- Core gameplay mechanics (e.g., character movement, inventory management, simple combat) showcasing the framework's capabilities.
- User-friendly interface allowing interaction with the game world and experimentation with the framework's functionalities.

The prototype will serve as a tangible proof-of-concept and provide valuable feedback for further development.

1.3.3 Evaluation Report

This report will assess the effectiveness of the prototype and framework based on the objectives outlined in Section 1.2:

- Playtesting results and user feedback, gathered through structured testing sessions or questionnaires.
- Evaluation of the framework's success in achieving scalability, maintainability, and performance goals through comparisons with benchmarks.

- Discussion of the prototype's performance, limitations, and areas for improvement.

1.4 Scope Definition

This project aims to develop a custom architectural framework within Unreal Engine 5 specifically designed to facilitate the creation of open-world RPGs with a focus on scalability, dynamic content, and diverse entity interactions. While drawing inspiration from existing frameworks and approaches, this project will not attempt to replicate their full development of the game but rather focus on core mechanics and key features tailored to the required specific design choices.

1.4.1 System Boundaries

The framework will encompass core functionalities like entity management, data-driven design, world streaming, and basic gameplay. User interface elements and complex narrative systems will be included but limited in the upcoming development phase, potentially being addressed in future iterations. Integration with external asset packs or third-party Artificial Intelligence (AI) solutions might be considered based on compatibility and project needs.

1.4.2 Functionality Limitations

Due to project scope and time constraints, the framework iteration might not support features like multiplayer functionality, persistent online worlds, or highly complex character customization systems. These features are not cored to demonstrating the framework's potential and can be explored in future development stages.

Technology Stack:

- Programming Language: Visual Scripting Blueprint
- Game Engine: Unreal Engine 5.1

- Libraries/Frameworks: Potential integration of open-source libraries for specific functionalities (e.g., data parsing, networking) will be evaluated during development.

In FYP Phase 2, these are the expected outcomes if the development is completed within the scope of the project:

1. A functional prototype showcasing core gameplay mechanics and framework capabilities within a basic open-world environment.
2. Detailed technical documentation outlining the framework's design, implementation, and usage.
3. An analysis report evaluating the chosen architectural approaches and the impact on the framework's performance and scalability.

1.5 Chapter Organization:

This report delves into the development of a custom architectural framework for open-world RPGs within Unreal Engine. Here's a short overview of the chapters and their key topics within this FYP 2 Final Report:

Chapter 1: Introduction

This introductory chapter serves as a gateway to the project. It provides a concise overview of open-world RPGs, shedding light on the inherent challenges encountered during their development. The chapter discusses the motivation, defines the project's scope, objectives, and outlines the anticipated deliverables.

Chapter 2: Background Study

Exploring deep into existing architectural approaches for open-world RPGs, this chapter accurately examines their strengths and limitations. It intricately dissects and defines pressing challenges such as scalability, maintainability, and performance, offering clear examples from well-known RPG titles to illustrate these challenges.

Furthermore, it studies the efforts of other developers in addressing similar issues, providing valuable insights into the landscape of RPG game development.

Chapter 3: Requirements Analysis

With a focus on clarity and precision, this chapter articulates the functional and non-functional requirements essential for the architectural framework's success. It categorizes requirements based on priority and impact, emphasizing crucial features vital for addressing scalability, performance, and maintainability concerns. Additionally, the chapter outlines evaluation criteria and metrics for measuring the framework's efficacy, while also highlighting any unique project demands or considerations.

Chapter 4: Design

Detailing the chosen architectural approach, this chapter unveils the framework's core components and functionalities. It translates the system requirements into concrete technical representations, employing diagrams and flowcharts to elucidate the framework's internal structure and interactions. Design decisions regarding technology, data structures, and communication protocols are meticulously explained, alongside potential challenges and mitigation strategies.

Chapter 5: Implementation

This chapter provides a comprehensive account of the technical implementation of the framework. It details the development process, including the tools and methodologies used, and gives an in-depth description of the system's implementation. Each of the core systems; Character System, Combat System, Quest System, and Item System is discussed in detail, highlighting the technical challenges faced and the solutions implemented.

Chapter 6: Testing

This chapter outlines the testing plan and execution, encompassing functional, integration, and usability testing. It presents the test results, analyses, and discusses encountered issues and their resolutions. The overall analysis of the testing phase provides insights into the framework's reliability and performance.

Chapter 7: Conclusion

The final chapter offers a reflective summary of key achievements and contributions made throughout the project's trajectory. It discusses the evaluation of project objectives, major learnings from the project, and the remaining work, potential improvements, and plans.

CHAPTER 2
BACKGROUND STUDY

Embarking on any ambitious development journey necessitates a thorough understanding of the scenery upon which it unfolds. This chapter investigates into the landscape of open-world RPG development, specifically illuminating the intricate challenges and solutions surrounding scalability, maintainability, and performance. By wisely examining existing approaches, analysing ongoing research, and exploring industry trends, this study aims to navigate these interconnected issues and lay the foundation for informed decision-making within our own project.

Our approach in defining the challenges within this chapter explains in three stages. First, the method of defining would unpack each challenge individually, defining them broadly in SE, then within the specific context of video games and especially open-world RPGs. These three stages would explain the specifics of the four challenges

Next, we'll shift our focus to examine the interwoven fabric of these challenges. By delving into their correlations and interdependencies, it aims to reveal how they often influence each other within the complex tapestry of development. Understanding these connections is crucial for making informed decisions that holistically address various needs, rather than tackling them in isolation.

Throughout this chapter, the background study serves as a crucial basis, propelling us forward towards the design and implementation of a framework that effectively addresses these obstacles, paving the way for a successful project.

2.1 Scalability in RPGs

2.1.1 Definition and Importance

Definition of Scalability in SE:

According to CyberLink ASP (2021), software scalability refers to its ability to adapt to varying demands and workloads. This includes handling increasing user numbers, data volumes, and complexity while maintaining performance and cost-effectiveness.

CyberLink ASP (2021) further emphasizes the importance of software scalability, highlighting its impact beyond seasonal businesses. It allows for cost-effective adaptation to various demands and complexities, ensuring system integrity and user experience even during fluctuations.

Scalability in the Context of Video Game:

According to a definition found on dictionary.com, scalability refers to how effectively a solution performs as the size of the problem grows. This concept is particularly relevant for game developers, especially in the context of multiplayer networked games where the number of simultaneous users serves as an indicator of the problem's scale. However, within the game developers' community, Dean Macri (2004) explains that scalability commonly denotes the difficulty of ensuring that a game operates satisfactorily across diverse system configurations, which may differ in terms of features, performance, or both.

Scalability in the context of video games encompasses multiple facets, including **storage scalability, content scalability, player count scalability,** and **world scalability**. A scalable video game architecture ensures smooth and consistent performance across different hardware setups, supports varying player counts and play styles, and enables seamless integration of additional content and features without compromising the overall gaming experience.

Scalability in RPG Genre:

Scalability in RPGs refers to the capability of the game's architecture to accommodate the evolving needs and expectations of players as they progress through the game world. This includes the ability to seamlessly integrate new characters, quests, items, and storylines without overwhelming the game engine or causing performance issues. Importantly, scalability in RPGs extends to supporting dynamic player interactions, expansive open worlds, and robust multiplayer experiences while maintaining stability and immersion. A scalable RPG framework empowers developers to create rich, expansive game worlds that grow and evolve alongside the player community, ensuring enduring enjoyment and engagement.

Moreover, an article written by Craig Stern (2017) in Sinister Design blog, he emphasized regarding the definition of Content Scalability and how it affects RPGs. Based on the article, content scalability refers to the *"design that permits the usefulness of (or challenge posed by) existing game content to scale so that the content remains relevant throughout the whole game."*

2.1.2 Existing Solutions and Approaches

Content Scalability Solutions:

In Craig Stern's (2017) article "Designing RPG Mechanics for Scalability," published on Sinister Design's website, prominent indie developer Craig Stern, known for the turn-based tactics game Telepath Tactics, explores into the crucial issue of balancing content creation with maintaining engaging gameplay throughout an RPG's progression. He proposes three key strategies to achieve "content scalability," ensuring players encounter fresh challenges and meaningful choices without overwhelming developers with content creation burdens.

1. **Dynamic Enemy Scaling:** This technique involves spawning enemies with variable power levels based on the player's progress. For example, a basic

goblin encounter in early levels could dynamically scale to be a more formidable foe if the player encounters it later in the game. This ensures individual encounters remain relevant and challenging throughout the experience, mitigating the need for excessive enemy types while promoting a sense of growth and adaptation.

2. **Linear Stat Progression:** Stern suggests employing linear instead of exponential growth for item and equipment stats. This approach avoids the "equipment treadmill" syndrome, where players constantly replace old items with newer, more powerful ones as they progress. Linear progression encourages players to strategically utilize and personalize their gear throughout the game, making each equipment decision meaningful and extending the lifespan of individual items.

3. **Percentage-Based Effects:** Implementing percentage-based modifiers for various in-game effects and character defences is another key strategy. This ensures mechanics remain balanced and engaging as the player progresses. For instance, instead of granting flat attack bonuses, applying a percentage increase to damage output adapts to the player's overall stats, maintaining meaningful challenges and strategic depth throughout the gameplay experience.

Stern's article offers valuable insights and practical methods for addressing content scalability challenges in RPG development. These strategies could be particularly relevant to the project in FYP Phase 2. If these strategies are wisely applied to my development, my architectural framework would provide good content scalability to the users. Further exploration and analysis of their applicability and potential adaptations within the framework will be important for developing a robust and engaging scalability solution.

2.1.3 Case Study on Scalability

Genshin Impact's Scalability Issues:

Genshin Impact, a popular open-world RPG developed by miHoYo, presents a compelling case study on scalability challenges within game development. Initially released with a modest installation size of 30GB for Version 1.0 in September 2020, the game boasted 22 playable characters, 87 weapons, and two explorable nations (Game8, 2023). However, as the game evolved, subsequent updates saw a significant expansion in content, resulting in Version 4.4 released in January 2024 with a staggering size of 150.0 GB. After this update, the game has a total up of 79 playable characters, 179 weapons, and five explorable nations (Sharma, 2024).

Lessons Learned:

The lessons learned from Genshin Impact's scalability challenges are highly relevant. By proactively addressing scalability considerations in the design and implementation of my framework, I can ensure that it remains resilient to content expansion and capable of accommodating the evolving needs of developers. This entails adopting modular design principles, implementing optimization strategies, and soliciting feedback from the developer community to iteratively refine and enhance the framework's scalability capabilities. By learning from real-world case studies such as Genshin Impact, I can effectively navigate scalability challenges and deliver a robust and future-proof architectural solution for RPG development.

2.2 Maintainability in RPGs

2.2.1 Definition and Importance:

Definition of Maintainability in SE:

Based on the IEEE Standard Glossary of Software Engineering Terminology, the formal definition of maintainability is *"the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment."* (Institute of Electrical and Electronics Engineers, 1990)

The importance of applying maintainability in SE is to make the development of an application easy to understand by developers while making it easy for them to update or repair the application. A good maintainability of a software application would lead to cost reduction of the real- world SE projects. Chen, Alfayez, Srisopha, Boehm, Shi (2017) stated that highly maintainable systems would lead to decreasing 75% life cycle's cost.

Maintainability in the Context of Video Games:

In general, maintainability in game development is all about keeping the video game alive and thriving. It refers to the ease with which a game can be modified, updated, and improved over its lifetime. The importance of having good maintainability in a video game would lead to smooth implementation of maintenance.

In a thesis written by Jarman (2010), he mentioned the three types of maintenance in the video game industry. The three types are corrective maintenance, perfective maintenance, and adaptive maintenance. Corrective maintenance involves addressing bugs and errors directly after release. Perfective maintenance focuses on improving and adding new features to a game over its lifecycle. Adaptive maintenance involves updating a game to accommodate changes in the environment

where it operates, such as updating for new service packs or operating system changes.

While corrective maintenance is mainly achieved through patches or updates, perfective maintenance often involves downloadable content, which can be released to users via network connections. Adaptive maintenance is less common in the gaming industry but may involve updating firmware or operating systems on gaming consoles. The text emphasizes the importance of networks in enabling maintenance activities, such as patching bugs and releasing downloadable content, which contribute to user satisfaction and prolonged profitability of video games.

Maintainability in RPG Genre:

Maintainability in the context of open-world RPG refers to the ease with which the game's architecture or design can be updated, modified, and extended over time while ensuring its stability, performance, and coherence. It encompasses various aspects, including code maintainability, content management, and overall game design flexibility.

When it comes to minor content expansion, continuously adding new quests, characters, locations, items, and other content to keep the game fresh and engaging for players. This may involve designing and implementing new game mechanics, narratives, and assets while maintaining consistency with the existing game world and lore.

Furthermore, performing balancing and fine-tuning towards game mechanics, combat systems, character progression, and economy would be required to ensure a satisfying and balanced gameplay experience. This involves analysing player feedback, monitoring gameplay metrics, and adjusting parameters such as difficulty levels, resource availability, and enemy AI behaviour. Thus, structuring an upright architecture would lead to smooth maintainability.

2.2.2 Best Practices and Design Patterns

General Principles:

A fundamental approach to creating maintainable RPG architectures is to adhere to general principles that promote modularization, clarity, and flexibility.

Modularization involves breaking down the game into smaller, independent modules responsible for specific functionalities, such as combat, inventory, and quests. This promotes isolation of concerns and facilitates easier modification. Furthermore, minimizing dependencies between modules through loose coupling allows for modules to evolve independently, achieved by using interfaces and events for communication (Nystrom, 2014). For example, when a player interacts with an NPC to initiate a quest, the quest system module can subscribe to events triggered by the NPC interaction, allowing for decoupled communication between the two systems.

Following the Single Responsibility Principle ensures that each class or component has a single, well-defined responsibility, reducing complexity and enhancing code maintainability (Gamma, Helm, Johnson, Vlissides, 1994). The Single Responsibility Principle would be applied to each module or component within the RPG game. For example, a combat system module should focus solely on managing combat mechanics, while an inventory system module should handle inventory-related functionality exclusively.

Implementing a data-driven design approach involves storing game data, such as items, characters, and quests, in external files or data tables, allowing for easy modification without recompiling code. In addition, maintaining clear naming conventions for variables, functions, and classes enhances code readability and

understanding, contributing to overall code maintainability (Unreal Engine Documentation, n.d.).

Design Patterns:

Incorporating design patterns into the architecture of RPG games can significantly contribute to their maintainability and extensibility. The Observer Pattern facilitates notifying multiple objects about changes in another object, beneficial for quests, events, and UI updates (Unreal Engine Documentation, n.d.). Additionally, employing the Component Pattern enables the assembly of complex entities from smaller, reusable components, promoting flexibility and customization.

Utilizing the State Pattern allows for managing an object's behaviour based on its internal state, applicable for character states such as idle, running, and attacking. Additionally, the Repository Pattern provides a unified access point to data, simplifying data access and management within the game (Nystrom, 2014).

2.2.3 Case Study on Maintainability

Maintainability Issues in Over the Edge:

In the RPG game industry, one game known to have maintainability issues is ‘Over the Edge’. This surreal role-playing game, created by Jonathan Tweet and Robin Laws and published by Atlas Games, is based on the mysterious Island of Al Amarja (Wikipedia, 2017). The game's maintainability issues stem from its departure from the conventional character representation system. Instead of using attributes and skills, characters are quantified with "traits", which are created and defined by the player. Each character has one primary trait, two secondary traits, and one flaw, and actions are based on these traits 3.

The maintainability issues arise due to the unconventional way characters are represented and the complexity introduced by the traits system. This departure from traditional RPG character representation could potentially lead to maintainability challenges in terms of game balance, rule complexity, and long-term support 3.

Lesson Learned:

Over the Edge innovated by using a trait-based character representation system, but this innovation led to maintainability issues. When developing game architecture, it's important to balance innovation with maintainability. While innovation is crucial for creating unique gameplay experiences, it's equally important to consider the long-term maintainability of the game. When introducing new systems or mechanics, assess their potential impact on maintainability and plan for their long-term support.

The complexity introduced by the trait-based system in Over the Edge contributed to maintainability issues. Application: In game architecture development, strive for simplicity and consistency. Complex systems can introduce maintainability challenges, so it's essential to keep the architecture as simple and consistent as possible. This includes using standardized coding practices, clear documentation, and modular design to facilitate easier maintenance and updates.

The unconventional character representation system in Over the Edge may have hindered long-term support and updates. When developing game architecture, consider future-proofing the design. Anticipate potential updates, expansions, and modifications, and build the architecture to accommodate these changes. This includes creating flexible and scalable systems that can adapt to future gameplay mechanics, content additions, and technological advancements.

2.3 Performance in RPGs

2.3.1 Definition and Importance:

Definition of Performance in SE:

Based on the IEEE Standard Glossary of Software Engineering Terminology, the formal definition of performance is *"The degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage."* (Institute of Electrical and Electronics Engineers, 1990).

This includes speed, responsiveness, resource utilization, scalability, and availability. Speed pertains to the software's responsiveness to user input and task completion time. Responsiveness reflects the software's ability to display updates promptly and react swiftly to user interaction. Resource utilization gauges the efficiency with which the software employs system resources such as memory, CPU, and network bandwidth. Scalability assesses the software's capacity to accommodate increased workloads or user requests without performance deterioration, while availability underscores the software's consistency and reliability in accessibility and operation.

The importance of performance in software engineering is manifold. It directly influences user experience, operational costs, system reliability, scalability, and competitive advantage. Poor performance can lead to user dissatisfaction, increased expenses, system instability, scalability limitations, and reduced competitiveness in the market (Bejawada, 2019).

Performance in the Context of Video Games:

In the context of video games, performance extends beyond technical metrics to encompass aspects crucial for player engagement and enjoyment. It encompasses frame rate, input latency, visual fidelity, stability, and loading times. Frame rate

denotes the number of frames rendered per second, impacting visual smoothness and responsiveness. Input latency measures the delay between player input and in-game response, critical for action-based gameplay. Visual fidelity encompasses the graphical quality and complexity of the game world, influencing immersion and aesthetics. Stability refers to the absence of crashes, bugs, or glitches that disrupt gameplay, while loading times affect immersion and flow.

The significance of performance in video games lies in its direct impact on gameplay experience, immersion, accessibility, and competitive advantage. High performance enhances gameplay fluidity, player enjoyment, accessibility, and skill expression, while technical limitations necessitate optimization for diverse hardware configurations.

Performance in RPG Genre:

In the field of game genres, different categories prioritize performance aspects according to their gameplay requirements. Fast-paced genres such as Action First-Person Shooter, and rhythm games prioritize high frame rates per second (FPS) to ensure smooth reaction-based gameplay. Even minor frame rate dips can significantly impact performance in these genres.

Conversely, story-driven genres like RPGs and adventure games place more emphasis on stability and visual fidelity to create immersion and maintain atmosphere, where frame rate may be of lesser importance. Simulation and strategy games prioritize responsiveness and input latency to facilitate accurate decision-making and control. For mobile games, careful optimization is necessary due to hardware limitations, focusing on battery life and efficient resource usage.

For instance, a competitive First-Person Shooter game on PC may target ultra-high frame rates may be around 144+ FPS to ensure the smoothest possible

gameplay, necessitating powerful hardware and meticulous optimization. In contrast, a story-driven RPG on console might prioritize a stable 30 FPS with high-quality textures and environments to enhance immersion, valuing visual fidelity over raw speed. Similarly, a mobile RPG may leverage stylized graphics and efficient rendering techniques to deliver smooth gameplay even on low-end devices.

Performance assumes added importance in RPG games due to their expansive worlds, intricate systems, and narrative emphasis. It encompasses world streaming, NPC interaction, combat fluidity, inventory management, and stability. World streaming ensures seamless loading of game elements, NPC interaction facilitates smooth conversations and gameplay, while combat fluidity and inventory management ensure engaging experiences. Stability maintains consistent performance, crucial for immersion, combat satisfaction, character engagement, large-scale battles, complex environments, and emotional connection to the narrative.

The importance of performance in RPG games lies in its pivotal role in sustaining immersion, enhancing gameplay experiences, facilitating character engagement, managing complex encounters, and fostering emotional connection to the narrative. To do so, building a good architecture that would complement the necessary game design would lead to a higher quality of game performance.

2.3.2 Optimization Techniques and Strategies:

3D Game Development Optimization:

Yasmin Curren (2021), an associate technical designer known for the game "Perfection," which faced performance issues such as lag and low frame rates impacting player immersion, shares her top 10 tips for optimizing game performance.

Curren emphasizes the importance of using a profiler to assess performance and identify areas needing improvement, including script code, assets, and resources. She advises against excessive rendering with multiple cameras and suggests using the dot product method to optimize rendering. Curren recommends utilizing layers to specify what assets a camera can render, optimizing models by reducing poly count and removing unnecessary objects.

She advocates for draw call batching, occlusion culling, and light baking techniques to improve rendering efficiency. Additionally, Curren advises optimizing code by reducing update function complexity and organizing logic into separate functions. These strategies aim to elevate game performance from under 20 frames per second to at least 60 frames or more.

RPG-Specific Strategies with Unreal Engine:

Based on the Unreal Engine Documentation (n.d.), Level Streaming would manage large open worlds by loading and unloading sections dynamically based on player proximity. Implementation of efficient streaming strategies in Unreal Engine can be done with Level Streaming Volumes. Furthermore, when it comes to the optimization of non-player characters (NPC), NPC Culling can be done by reducing the number of active NPCs in the scene based on distance or relevance to the player. Within Unreal Engine, developers shall utilize "Hidden GameObjects" functionality and visibility checks to perform NPC Culling.

In the perspective of graphics optimization regarding the details of characters, props, and foliage, developers can use Levels of Detail (LOD) to adjust complexity of models based on the distance from the camera. LOD optimization would enhance game's performance by reducing workload on the graphics processing unit (GPU). In Unreal Engine, LOD settings can be configured for static meshes through LOD Group settings in the Details panel. For better graphics in RPG, this LOD for models near to the cameras would be maxed out while the furthest models would be minimised but not much.

In Unreal Engine, leveraging GPU instancing for repeating assets can significantly enhance rendering efficiency by minimizing draw calls and reducing the GPU workload. This optimization technique is particularly beneficial when dealing with numerous identical objects in a scene. Developers can implement GPU instancing using Unreal Engine's "InstancedStaticMeshComponent" class, which allows multiple instances of a static mesh to be rendered with a single draw call. By utilizing this class, developers can efficiently render large numbers of identical objects while maintaining optimal performance. This approach is particularly useful for environments containing repetitive elements such as foliage, rocks, or props. By incorporating GPU instancing into Unreal Engine projects, developers can achieve smoother performance and enhance the visual fidelity of their games or applications.

In Unreal Engine, optimizing AI pathfinding involves implementing efficient algorithms and strategies to enhance the performance of artificial intelligence navigation systems. Utilizing Unreal Engine's built-in behaviour tree system, developers can design complex AI behaviours while ensuring streamlined pathfinding. By carefully designing behaviour trees, developers can minimize unnecessary calculations and improve the efficiency of AI decision-making processes. Additionally, spatial partitioning techniques, such as quad trees or navigation meshes, can be employed to optimize pathfinding computations by reducing the search space and eliminating redundant calculations. These techniques partition the game world into smaller, manageable regions, allowing AI agents to

navigate more efficiently. By incorporating these methods into Unreal Engine projects, developers can achieve smoother and more responsive AI behaviour, enhancing the overall gameplay experience for players.

2.3.3 Case Studies on Performance Optimization

Cyberpunk 2077 serves as a notable example of poor performance optimization in open-world RPGs, highlighting the repercussions of inadequate technical preparation and rushed development timelines. Upon its highly anticipated release, the game faced substantial criticism for its subpar performance, particularly on last-generation consoles such as PlayStation 4 and Xbox One.

Cyberpunk 2077's performance issues encompassed frequent crashes, low frame rates, and texture pop-in, significantly detracting from the overall gaming experience. The game's optimization shortcomings were attributed to a combination of factors, including rushed development schedules, ambitious scope, and technical challenges in adapting the game for diverse hardware configurations. CD Projekt Red's failure to prioritize optimization for older consoles led to severe backlash from players and critics alike, tarnishing the game's reputation and resulting in substantial financial losses for the company.

Lessons Learned:

The case of Cyberpunk 2077 underscores the critical importance of performance optimization in game development, particularly for large-scale open-world RPGs. Prioritizing thorough testing, iteration, and optimization throughout the development process is essential to identify and address performance bottlenecks effectively.

Adopting modular and scalable architectural designs can facilitate smoother optimization efforts, allowing for targeted improvements to specific components without compromising overall system integrity.

Implementing efficient resource management techniques, such as level-of-detail scaling and texture streaming, can help mitigate performance issues and ensure a more seamless gameplay experience for players.

By learning from the mistakes of Cyberpunk 2077, our architectural framework for RPGs will emphasize meticulous optimization efforts from the outset, aiming to deliver optimal performance across various hardware configurations while maintaining the immersive and expansive nature of open-world gameplay.

2.4 Overview of Architecture Styles

Based on the IEEE Standard Glossary of Software Engineering Terminology, the formal definition of architecture is *"The organizational structure of a system or component. See also: component; module; subprogram; routine."* (Institute of Electrical and Electronics Engineers, 1990)

Software architectural styles refer to the fundamental structural choices and design patterns that guide the organization and interaction of system components within a software application. These styles provide a framework for addressing key architectural concerns such as scalability, maintainability, and performance. These architectural styles that will be analysed in this subchapter would be ECS, MVC, Component-Based and Hierarchical Level Streaming

Each architectural style offers distinct advantages and disadvantages. The styles are required to be integrated to shape the overall structure and behaviour of software systems. A large and complex system usually implements more than one architectural style, this would be done by aligning each of the styles into the specific requirements and objectives of the software.

2.4.1 Entity-Component-System (ECS)

In 1996, Martin explained that ECS is an architectural pattern used in game development where entities are composed of independent components that define their behaviour and attributes. The system separates data from behaviour, allowing for flexible composition and reusability of components across different entities.

ECS is well-suited for RPG game development because it allows for flexible composition of game entities such as characters, items, and environments. In RPGs, where characters and objects often have diverse behaviours and attributes, ECS enables developers to easily create and modify entities by combining reusable

components. For example, a character in an RPG may consist of components for movement, health, inventory, and AI behaviour. (Unity Technologies, 2023)

2.4.2 Model-View-Controller (MVC)

MVC is a software architectural pattern commonly used in web and application development. It separates an application into three interconnected components: the Model (data and logic), the View (user interface), and the Controller (handles user input). MVC promotes modularity and separation of concerns, making it easier to maintain and extend applications. (Reenskaug, 2003)

While MVC is more commonly associated with web and application development, its principles can still be applied to game development, especially in the context of user interfaces and game logic separation (Hod, 2014). In an RPG, the Model may represent game data such as player stats and world state, the View may handle rendering and display of the game world, and the Controller may manage player input and interaction.

2.4.3 Component-Based Architecture

Component-based architecture is a design approach where complex systems are built from reusable, self-contained modules, or components. Each component encapsulates a specific piece of functionality and can be easily added, removed, or modified without affecting other parts of the system just like components of Lego Block explained by Gillin (2024). This promotes flexibility, modularity, and code reusability.

RPGs often feature complex systems with various gameplay mechanics, character classes, and abilities. Component-based architecture allows developers to design and manage these systems more efficiently by breaking them down into modular components (Nystrom, 2014). For instance, in an RPG, abilities and spells

could be implemented as reusable components that can be attached to different characters or items, enhancing flexibility and code reuse.

2.4.4 Hierarchical Level Streaming

Hierarchical level streaming is a technique commonly used in game development to manage large, open-world environments efficiently. It involves dividing the game world into hierarchical regions or levels and loading/unloading them dynamically based on the player's position and movement. This allows games to maintain a seamless and immersive experience while conserving memory and processing resources. (Unreal Engine Documentation, n.d.)

In RPGs with expansive worlds and seamless exploration, hierarchical level streaming can optimize resource usage and improve performance. By dynamically loading and unloading game regions based on the player's location, developers can create large and detailed environments without sacrificing memory or processing resources. This is particularly beneficial for RPGs with open-world exploration and non-linear gameplay, enhancing immersion and player freedom.

2.4.5 Advantages and Disadvantages

Every architectural style cannot fit in every scenario in system development due to having different sets of advantages and disadvantages. Even in the realm of game development, various architectural styles offer distinct advantages and challenges.

Table 2.1 Advantages and Disadvantages of Architecture Styles

Architectural Styles	Advantages	Disadvantages
Entity-Component-System	• Flexibility • Code reusability	• Complexity • Potential performance overhead
Model-View-Controller	• Modularity • Code organization • Maintainability	• Complexity • Potential tight coupling
Component-Based Architecture	• Code reusability • Flexibility • Maintainability • Scalability • Code reusability	• Complexity • Potential overhead
Hierarchical Level Streaming	• Performance • Player immersion	• Potential delays

Based on the Table 2.1 above, The Entity-Component-System (ECS) architecture provides flexibility in designing entities and encourages code reusability and separation of concerns. However, managing entity-component relationships can become complex, and large-scale systems may experience performance overhead. Conversely, the Model-View-Controller (MVC) pattern, although more commonly associated with web and application development, promotes modularity, and facilitates maintenance and testing by separating concerns.

Yet, it may introduce complexity in smaller applications and demands careful design to prevent tight coupling between components. Component-Based Architecture further enhances code reusability and system flexibility, simplifying maintenance and scalability. Nevertheless, managing component dependencies can be challenging, potentially leading to communication overhead. Hierarchical Level Streaming optimizes resource usage and enhances performance by dynamically loading game regions, fostering seamless exploration and immersion.

However, it necessitates meticulous level design and implementation, with the risk of loading delays or artifacts. Each architectural style presents a unique set of pros and cons, affecting aspects such as scalability, maintainability, and performance in RPG game development.

2.4.6 Impact on Scalability, Maintainability, and Performance:

ECS can enhance scalability by allowing for flexible entity composition and parallel processing of components. However, improper design or management of entity-component relationships can impact maintainability and introduce performance bottlenecks.

MVC promotes modularity and separation of concerns, improving maintainability and scalability. However, inefficient communication between model, view, and controller components can impact performance, especially in complex applications.

Component-based architecture enhances scalability and maintainability by promoting code reusability and modularity. However, excessive component dependencies or inefficient communication can affect performance, particularly in large-scale systems.

Hierarchical level streaming optimizes resource usage and enhances performance by dynamically loading and unloading game regions. It also improves scalability by allowing for the creation of expansive game worlds. However, improper implementation or level design can impact performance and player experience.

CHAPTER 3
REQUIREMENTS ANALYSIS

Chapter 3 delves deeply into the foundational aspects of our RPG architectural framework within Unreal Engine 5 by meticulously analysing user requirements. Through an intricate exploration, this chapter aims to gain a comprehensive understanding of the diverse needs and expectations of game developers who are potential users of the framework.

It begins by examining various user personas, shedding light on the unique profiles and demands of developers across different spectrums. Transitioning from personas to practical scenarios, the chapter vividly illustrates real-world situations, offering invaluable insights into how the framework might be employed. Moving beyond theoretical constructs, the exploration delves into the realm of functional requirements, dissecting critical elements like the character system, quest system, item and weapon system, and combat system.

Additionally, non-functional dimensions such as usability, reliability, and performance are scrutinized, aiming to ensure that the framework not only meets functional demands but also excels in user experience and technical robustness. Ultimately, the chapter concludes with a thorough examination of the technical requirements indispensable for the successful implementation and utilization of the framework. Through this meticulous analysis, Chapter 3 endeavours to provide an exhaustive understanding of the intricate needs and aspirations that shape the development of our RPG framework.

3.1 User Requirements:

When it comes to the users of my product, it would not be the gamers playing the RPG video game itself, instead it would be other game developers using my framework as a template to create their own RPG. Practically, the platform where my architectural framework can be downloaded or purchased would be Unreal Engine Marketplace.

As a solo developer or indie studio seeking to create an RPG, the user aims to develop the game without the extensive time and effort required. The framework provides a comprehensive set of tools and templates focusing on key RPG elements, including character, quest, item & weapons, and combat systems. This framework streamlines the development process by offering modular blueprint components that can be easily integrated into existing projects in Unreal Engine.

Actor:

Solo developer or developer of an indie studio

Preconditions:

- The developer possesses basic knowledge of game development principles and programming.
- The development environment, such as a game engine like Unreal Engine or Unity, is set up and ready for use.

Postconditions:

- The RPG framework components are successfully integrated into the game project.
- The game project benefits from enhanced gameplay mechanics and streamlined development process.

3.1.1 User Personas

Solo Developer:

- Persona: Alex, an aspiring game developer juggling a full-time job and passion project of developing RPGs.
- Challenges: Limited time, budget constraints, and novice-level technical skills hinder Alex's ability to create complex RPGs efficiently.
- Needs: Alex requires a user-friendly framework that offers ease of learning, usage, and customization, enabling him to develop captivating RPGs within his constrained resources.

By using this upcoming architectural framework that I will produce, Alex is able to develop his own RPG game efficiently. Since he lacks manpower, he will save time and effort by using the template of creating and adjusting characters, quests, items, weapons, and combat systems within my framework and implementing them into his RPG.

Indie Studio Developer:

- Persona: Sarah, the lead of a small indie game development team comprising 4 members embarking on their first commercial RPG venture.
- Challenges: Sarah and her team face challenges in balancing costs, managing workflows effectively, and ensuring scalability as their team expands.
- Needs: Sarah seeks a robust framework that facilitates seamless collaboration, promotes workflow consistency, and supports future scalability to accommodate the studio's growth and ambitions.

For a team of indie developers however, a smooth and flexible workflow would be implemented within the game development cycle by using my framework. For instance, if Sarah is the developer that handles the characters' game logic and her teammate Alisya handles the logic of items and weapons, Sarah can be use the character system template and Alisya can use the item & weapon system template in my framework. This will improve the game development's maintainability.

3.1.2 Use Case Scenarios

Setting Up the Unreal Engine Environment:

- Scenario: The user initiates the Unreal Engine environment and acquires the RPG framework package.
- Action: Following the provided setup guidelines, the user configures the environment to ensure compatibility with Unreal Engine, downloading and integrating the framework from Unreal Engine Marketplace.
- Outcome: The Unreal Engine environment is primed, with the RPG framework seamlessly integrated, enabling the user to commence RPG development within the familiar Unreal Engine ecosystem.

Implementing Character System within Unreal Engine:

- Scenario: The user aims to introduce diverse characters with unique attributes and abilities into their RPG project.
- Action: Accessing the character system module within the framework tailored for Unreal Engine, the user utilizes pre-designed character classes and customizable attributes, by using Unreal Engine's Blueprint system for visual scripting if needed that my framework provides.
- Outcome: The user successfully integrates the character system into their Unreal Engine project, empowering them to create and manage dynamic in-game personas efficiently.

Designing Quest System in Unreal Engine:

- Scenario: The user seeks to construct engaging quests and narrative arcs for player progression within the Unreal Engine environment.
- Action: Leveraging the quest system module customized for Unreal Engine, the user selects from a variety of quest templates and defines objectives,

rewards, and branching storylines using Unreal Engine's Blueprint visual scripting capabilities that can be found in my framework.

- Outcome: The user implements a diverse range of quests within their Unreal Engine RPG project, enhancing player engagement and providing meaningful gameplay experiences seamlessly within the engine.

Integrating Item and Weapon System in Unreal Engine:

- Scenario: The user aims to incorporate a wide array of items and weapons, each with distinct attributes and effects, utilizing Unreal Engine's robust capabilities.
- Action: Utilizing the item and weapon system templates customized for Unreal Engine, the user adds various equipment types and assigns corresponding properties using Unreal Engine's Blueprint system.
- Outcome: The user enhances gameplay depth by introducing a comprehensive inventory system within their Unreal Engine RPG project, leveraging the engine's power to create immersive and customizable equipment mechanics.

Implementing Combat System within Unreal Engine:

- Scenario: The user attempts to design dynamic combat encounters leveraging Unreal Engine's capabilities to create immersive gameplay experiences.
- Action: Employing the combat system framework customized for Unreal Engine, the user defines combat mechanics, including action selection, turn order, and skill execution, utilizing Blueprint scripting provided in my framework.
- Outcome: The user seamlessly integrates the combat system into their Unreal Engine RPG project, harnessing the engine's advanced features to deliver immersive and tactical gameplay experiences for players.

3.1.3 Use Case Diagram

The utilization of the architectural framework within the Unreal Engine ecosystem presents a promising avenue for solo developers and indie studio teams alike. In this context, the actor, representing either a solo developer or a member of an indie studio team, interacts with the framework to harness its capabilities in RPG game development. Whether working independently or as part of a small team, developers benefit from the framework's comprehensive suite of features, tailored specifically for RPG creation within Unreal Engine.

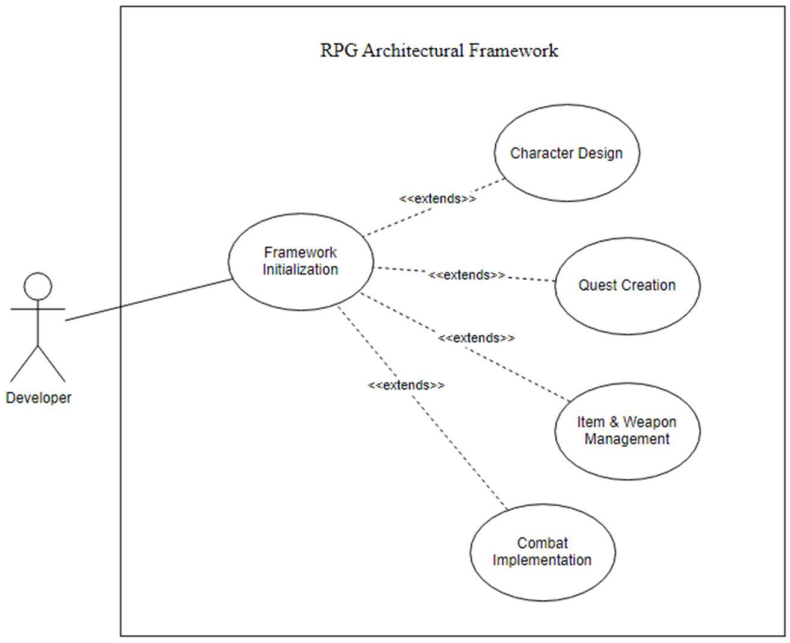

Figure 3.1 Use Case Diagram of RPG Architectural Framework

Initializing the Framework:

As developers embark on their RPG game development journey within the Unreal Engine environment, the first step involves initializing the architectural framework by downloading it from the Unreal Engine Marketplace. This action is undertaken by the developer, who accesses the framework's initialization function or module, configures settings, and integrates it into the project.

Designing Characters:

With the framework in place, developers get into the process of character design for their RPG game. Empowered by the framework's character system, developers can craft diverse characters ranging from playable heroes to non-playable entities and formidable enemies. Leveraging the comprehensive character creation tools provided by the framework, developers define attributes, skills, appearances, and behaviours, infusing life into their virtual roles.

Creating Quests:

Quest design unfolds as developers harness the capabilities of the framework's quest system. Through intuitive quest creation tools, developers outline quest objectives, triggers, conditions, rewards, and dialogue, meticulously sculpting immersive quests that captivate players' imaginations and drive gameplay progression.

Managing Items and Weapons:

The framework's item and weapon system equip developers with the tools necessary to curate a vast array of items and weapons. Developers can manage item types, properties, effects, equipment customization, crafting systems, and inventory management, ensuring a unified integration of loot, gear, and equipment within their RPG.

Implementing Combat Encounters:

Developers would compose thrilling battles and strategic skirmishes, shaping flow of combat within their RPG realms. With access to a comprehensive suite of combat mechanics, enemy behaviours, attack types, and AI logic, developers craft exciting combat experiences that resonate with players and elevate immersion.

3.2 Functional Requirements:

This framework aims to provide game developers with a robust and flexible toolkit for creating immersive role-playing experiences. At its core, the framework focuses on four key components: the Character System, Quest System, Item & Weapon System, and Combat System. With this framework, developers will have the tools they need to bring their RPG visions to life efficiently and effectively within the Unreal Engine ecosystem.

3.2.1 Character System Requirements:

The Character System requirements outlines the core functionalities essential for character creation and management within the architectural framework. These requirements are tailored to cater to the diverse needs of developers in the RPG Genre. The character system template in my framework will be able to:

1. Create diverse playable character logic with customizable attributes, skills, and stats, allowing users to define unique traits for each character.
2. Create assorted enemy character logic with customizable attributes, skills, and stats, allowing users to define traits for each enemy.
3. Provide tools to customize AI behaviours for NPCs with varying roles and personalities, enabling developers to create dynamic and lifelike non-player characters.
4. Manage character progression, including experience points (EXP) and character levels, allowing for customizable levelling systems and character growth mechanics.
5. Handle equipment management through inventories, enabling characters to acquire, equip, and manage various items and weapons as they progress through the game.

In short, the character system template in my framework will offer pre-built logic for playable heores, NPCs (for quest interactions or city decoration), and enemies, providing flexibility for diverse character types.

3.2.2 Quest System Requirements:

In this section on Quest System Requirements, the essential functionalities necessary for constructing captivating quests within the architectural framework is detailed. Aligned with user needs, these requirements encompass a comprehensive range of capabilities:

1. Crafting branching quests featuring diverse objectives and decision points to immerse players in dynamic narratives.
2. Defining quest triggers, conditions, and corresponding rewards to incentivize player progression and engagement.
3. Seamlessly integrating dialogue and cutscenes to enhance storytelling and player immersion throughout the questing experience.
4. Implementing robust quest tracking mechanisms and progress indicators to facilitate player navigation and provide clear guidance on quest objectives and completion status.

3.2.3 Item & Weapon System Requirements:

This section delineates the essential functionalities pertaining to item and equipment management within the architectural framework, tailored to meet user requirements. The item and weapon system will encompass a range of capabilities, including:

1. Providing diverse item types with distinct properties and effects, such as quest objects, weapons, armoury, and accessories, catering to the varied needs of RPG scenarios.
2. Creating different types of weapons such as swords, bows, daggers, spears, magic books and so on.
3. Enabling comprehensive equipment customization and modification options for characters, allowing for personalized gameplay experiences.

4. Implementing crafting and upgrading for inventory systems to facilitate the seamless integration and management of items within the game environment.

3.2.4 Combat System Requirements:

This section delineates the essential functionalities required for implementing dynamic and engaging combat encounters within the architectural framework, tailored to meet user needs. The combat system will encompass a range of capabilities, including:

1. Supporting real-time combat mechanics suitable for open-world RPG environments, ensuring fluid and immersive gameplay experiences.
2. Providing comprehensive logic for handling various combat actions, including normal attacks, skills, and damage calculations, fostering strategic depth and player choice.
3. Allowing for extensive customization of enemy behaviours, attack types (such as slow or fast), and AI patterns, enabling developers to craft diverse and challenging combat encounters.
4. Facilitating seamless integration of visual effects and sound design elements to enhance the overall impact and immersion of combat experiences, enriching the player's engagement and enjoyment.

These functionalities aim to equip developers with robust tools for creating dynamic and compelling combat systems within their RPG projects, enhancing the overall gameplay experience and player satisfaction.

3.3 Non-Functional Requirements

Optimizing Performance:

To ensure optimal performance, the architectural framework will prioritize efficiency using streamlined data structures and optimized algorithms tailored to each feature. Leveraging Unreal Engine's resource management tools, the framework will

meticulously manage memory allocation and deallocation to minimize overhead and maximize runtime performance. Additionally, a component-based design approach will facilitate modularization, enabling lightweight components for seamless processing and rendering within the game environment.

Improving Maintainability:

For maintainability, the framework will adopt a modular design strategy, breaking down each feature into distinct, well-defined components with clear interfaces. Extensive documentation, including comprehensive API references and tutorials, will empower developers to understand and maintain the framework with ease. Robust error handling mechanisms and logging functionalities will enhance debugging capabilities, ensuring swift resolution of issues, and minimizing downtime.

Refining Scalability:

In terms of scalability, the framework will offer a flexible architecture that accommodates the evolving needs of developers as their games grow in complexity and scope. Dynamic asset loading capabilities will enable games to scale seamlessly to larger worlds by intelligently managing asset loading and unloading based on player proximity and other criteria. Rigorous performance testing throughout the development process will identify and address potential bottlenecks early on, ensuring that the framework can scale effectively under diverse conditions.

3.4 Technical Requirements

Hardware Specifications:

The framework is optimized to run on hardware configurations that adhere to or surpass the recommended specifications for operating Unreal Engine 5. This includes a quad-core Intel or AMD processor clocked at 2.5 GHz or higher, a minimum of 8 GB RAM, and a DirectX 11 or 12 compatible graphics card. Adequate

storage space is also required to accommodate project files and assets. Additionally, compatibility with Vulkan API is supported for select AMD and NVIDIA graphics cards.

Software Dependencies:

Seamless integration and functionality within Unreal Engine 5 are contingent upon compatibility with specific software dependencies and versions. The framework requires a Windows 10 64-bit operating system, version 1909 or higher, or Windows 11. DirectX End-User Runtimes (June 2010) are essential for DirectX runtime support. These software dependencies ensure optimal performance and functionality when utilizing the architectural framework for RPG development within Unreal Engine 5.

Version Control Systems:

To maintain code integrity and facilitate collaborative development efforts, a version control system, such as Git, will be implemented. This allows for centralized management of project assets, seamless branching and merging of code changes, and comprehensive tracking of project history, ensuring project stability and enabling effective team collaboration.

Testing Methodologies:

Rigorous testing methodologies will be employed throughout the development lifecycle to validate the functionality, performance, and reliability of the framework. This includes comprehensive unit testing to verify individual components, integration testing to assess system-wide interactions, and user acceptance testing to validate end-user functionality. Automated testing frameworks and continuous integration practices will be utilized to streamline the testing process and identify potential issues early, ensuring the delivery of a robust and high-quality architectural framework for RPG development within Unreal Engine 5.

CHAPTER 4
DESIGN

Chapter 4 serves as the cornerstone of the project, exploring deep into the details of designing the RPG Architectural Framework. At its core, this chapter clarifies the framework's fundamental features, crafted to meet the diverse needs of game developers.

Beyond a mere conceptualization, this section delves into the technical underpinnings of the framework, offering detailed representations and illustrations to elucidate its inner workings. Through a systematic exploration of the framework's key components, developers gain invaluable insights into its structure, functionality, and implementation distinctions.

By transcending abstract theories and delving into practical design considerations, this chapter provides a robust roadmap for translating conceptual ideas into tangible solutions. Through a rigorous design approach, the architectural framework is composed to empower developers with the tools they need to navigate the complexities of RPG development seamlessly.

With a focus on user-centric design principles and technical feasibility, Chapter 4 epitomizes the project's commitment to delivering a cutting-edge solution that revolutionizes the landscape of RPG game development.

4.1 Features of the RPG Architectural Framework

In this section, the architectural framework's pivotal features are outlined, offering developers a comprehensive overview of its capabilities. Each component of the RPG Architectural Framework is dissected, emphasizing its unique functionalities and contributions to the game development process by using Unreal Engine. From the Character Designer, empowering developers to craft diverse and customizable characters, to the Quest Creator, facilitating the creation of dynamic and engaging quests, every feature is tailored to streamline development workflows and enhance gameplay experiences. The Item Creator provides developers with a versatile toolset for managing items and equipment within the game, while the Combat Creator enables the implementation of dynamic and immersive combat encounters.

4.1.1 Features of the Character Designer

Within the RPG Architectural Framework, the Character Designer offers developers a versatile toolkit for crafting a diverse array of characters, including playable heroes, enemies, and NPCs. Through streamlined Blueprint classes in Unreal Engine 5, developers can access the following functionalities:

1. **Customizable Attributes:** Developers have the flexibility to define key attributes such as Health Points (HP), Mana, and Damage Stats for each character, influenced by factors like strength, agility, and intelligence. This customization allows for the creation of unique and balanced heroes tailored to various playstyles.

2. **Enemy Configuration:** In addition to playable heroes, developers can create formidable enemies with adjustable HP and Damage Stats. Combat styles can be assigned to enemies, dictating their AI behaviour and tactics during engagements, ensuring dynamic and challenging encounters for players.

3. **NPC Customization:** The Character Designer enables the creation of NPCs personalized for diverse roles within the game world. Serving as integral quest givers, NPCs can be customized to fulfil specific narrative and gameplay requirements, enhancing immersion and player engagement.

4. **Progression Systems:** Each character within the framework is equipped with default level and EXP systems, allowing for seamless progression as players traverse the game world. Developers can also fine-tune character progression mechanics, including levelling up and acquiring new abilities, to create rewarding gameplay experiences.

5. **Weapon Assignment:** Developers can assign weapon types to heroes or enemies, defining their combat capabilities and playstyle. This feature enables precise customization, ensuring that each character is equipped with suitable weapons that complement their design and role within the game.

6. **Combat Management:** The Character Designer includes combat management tools, facilitating the implementation of engaging combat mechanics. From basic attacks to complex skill interactions, developers can design intricate combat systems that offer depth and strategic depth to gameplay.

7. **Character Models Presets:** To improve development, the framework provides a collection of character model presets, offering a diverse selection of visual assets for developers to choose from. These presets serve as a foundation for character creation, accelerating the design process while maintaining visual coherence and quality.

4.1.2 Features of the Quest Creator

The Quest Creator feature of the architectural framework facilitates the integration of quest systems into RPGs by providing essential game logic and functionality. This includes:

1. **Location Tracking:** The framework enables developers to define specific locations that players must visit to progress through quests. This functionality streamlines quest development by allowing developers to set precise waypoints and objectives within the game world.

2. **Enemy Assignment:** With the Quest Creator, developers can easily designate which enemies' players must defeat to complete quests. This feature includes the ability to specify enemy types, spawn locations, and encounter conditions, ensuring dynamic and engaging combat encounters throughout the game.

3. **NPC Interaction Templates:** The framework includes pre-designed templates for NPC interactions, simplifying the implementation of dialogue and conversation systems within quests. Developers can utilize these templates to create branching dialogue trees, quest-giver NPCs, and interactive story elements, enhancing the narrative depth and immersion of the game.

By incorporating these features into the Quest Creator, the architectural framework allows developers to integrate immersive and engaging quest systems into their RPGs. This approach reduces development time and complexity while providing the tools necessary to create compelling and memorable gameplay experiences for players.

4.1.3 Features of the Item Creator

The Item Creator feature within the architectural framework offers robust functionality for defining and managing various in-game items, enhancing the depth and complexity of RPG gameplay. Key features of the Item Creator include:

1. **Weapon and Armour Logic:** Developers can define game logic for RPG weapons and armour. This includes attributes like damage, durability, and special effects, offering flexibility for in-game equipment.

2. **Custom Weapon Types:** The Item Creator allows for creating new weapon types, expanding the character system's arsenal. From exotic weapons to futuristic gadgets, developers can tailor the gameplay experience.

3. **Accessories Creation:** Developers can design accessories like amulets and rings. These enhance player stats or provide benefits during gameplay, aiding in overcoming challenges or gaining advantages.

4. **Quest Item Generation:** With the Item Creator, developers can generate quest items. These items range from key artifacts to mundane objects, adding depth to quest objectives and storylines.

5. **Inventory Integration:** The Item Creator seamlessly integrates with inventory systems. Players can manage their collected items, including sorting, stacking, and equipping, improving the overall player experience.

By incorporating these comprehensive features into the Item Creator, the architectural framework provides developers with the tools and flexibility to design intricate item systems that enrich the RPG gameplay experience.

4.1.4 Features of the Combat Creator

There would be multiple features of the Combat Creator to assist developers to create astonishing attacks and abilities that can be assigned in Character Designer. The features include:

1. **Combat Style Presets:** Developers have access to a variety of combat style presets, each with unique animations and mechanics. These presets include different attack sequences, abilities, and ultimate moves tailored for diverse gameplay experiences. These combats can be assigned for playable heroes or enemies in Character Designer

2. **Custom Animation Support:** The Combat Creator allows developers to incorporate custom animations into their combat sequences. Whether it's a signature move for a character or a unique attack animation, developers can bring their creative vision to life.

3. **Weapon-specific Attacks:** Combat presets are aligned with weapon types, ensuring that each character's combat style is cohesive and thematically appropriate. Normal attacks, special abilities, and ultimate moves are designed to complement the chosen weapon type, enhancing immersion and gameplay depth.

4. **Damage and Mana Systems:** Developers can utilize the Combat Creator to fine-tune the game's damage and mana systems. This includes adjusting damage scaling, hit detection mechanics, and mana cost for abilities, offering precise control over combat balance and dynamics.

4.2 Technical Representations

4.2.1 Entity Relationships Diagram

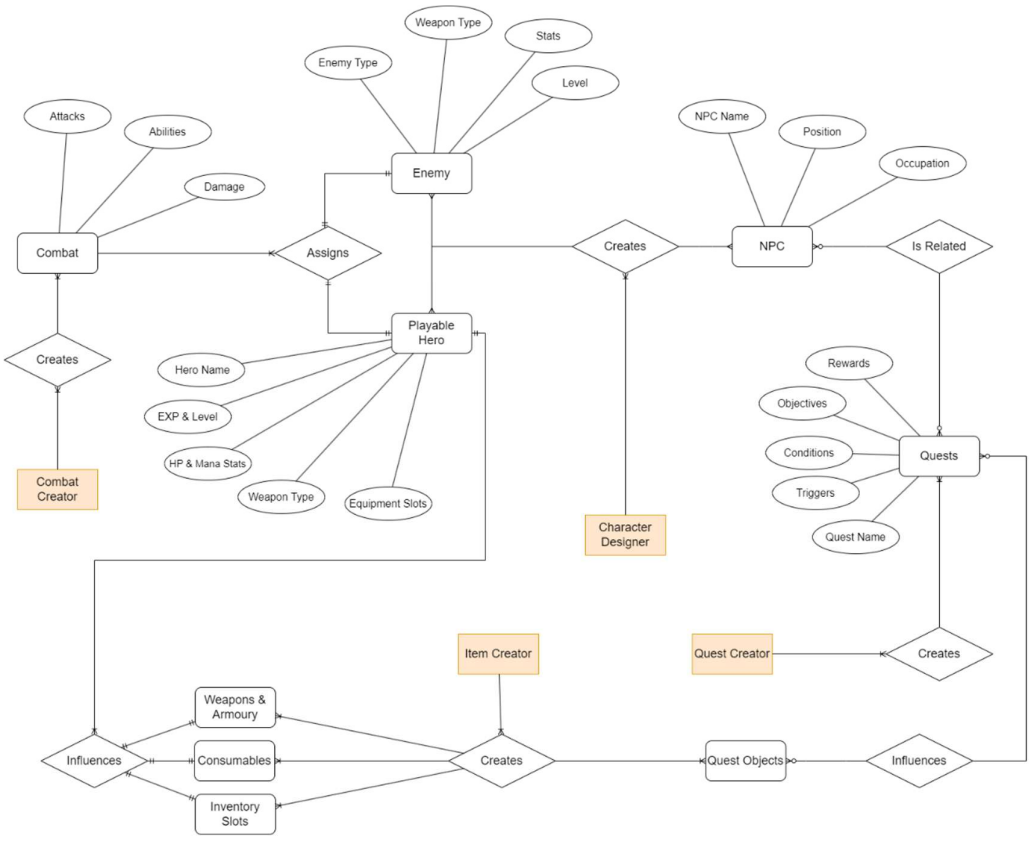

Figure 4.1 Entity Relationships Diagram of the RPG Architectural Framework

Based on the Figure 4.1 Entity Relationships Diagram of the RPG Architectural Framework offers a visual representation of the RPG Architectural Framework, illustrating the intricate relationships between its components. Designed to empower game developers utilizing Unreal Engine in crafting immersive RPG experiences, this framework encompasses several key features:

Character Designer: This component enables developers to create a diverse range of characters, including playable heroes, enemies, and NPCs, each with customizable attributes such as name, EXP & Level, HP, Mana, and Damage Stats. Through intuitive tools provided by the Character Designer, developers can define

the characteristics and behaviours of their characters, enriching the gameplay experience.

Quest Creator: The Quest Creator empowers developers to design engaging quests within their RPG games. From defining quest objectives and triggers to scripting dialogue and cutscenes, this feature facilitates the creation of dynamic and compelling narrative experiences. Quests created using this tool can drive the progression of the game and immerse players in its richly crafted world.

Combat Creator: With the Combat Creator, developers can design and implement thrilling combat encounters tailored to their game's design and setting. This feature allows for the creation of diverse combat styles, abilities, and enemy behaviours, enabling developers to craft challenging and strategic gameplay experiences. By integrating the Combat Creator into their projects, developers can elevate the intensity and excitement of combat within their RPG games.

Item Creator: The Item Creator component provides developers with the tools to populate their RPG worlds with a variety of items and equipment. From weapons and armour to accessories and quest objects, developers can define the properties and functionalities of each item, enriching the gameplay with strategic choices and rewards. The Item Creator also includes presets for inventory management, allowing developers to seamlessly integrate items into their game's economy and progression systems.

From character customization to quest design, combat mechanics, and item management, this framework provides a comprehensive suite of tools and features to support developers in realizing their creative visions and captivating players with unforgettable adventures.

4.2.2 Flow Chart for Character Designer

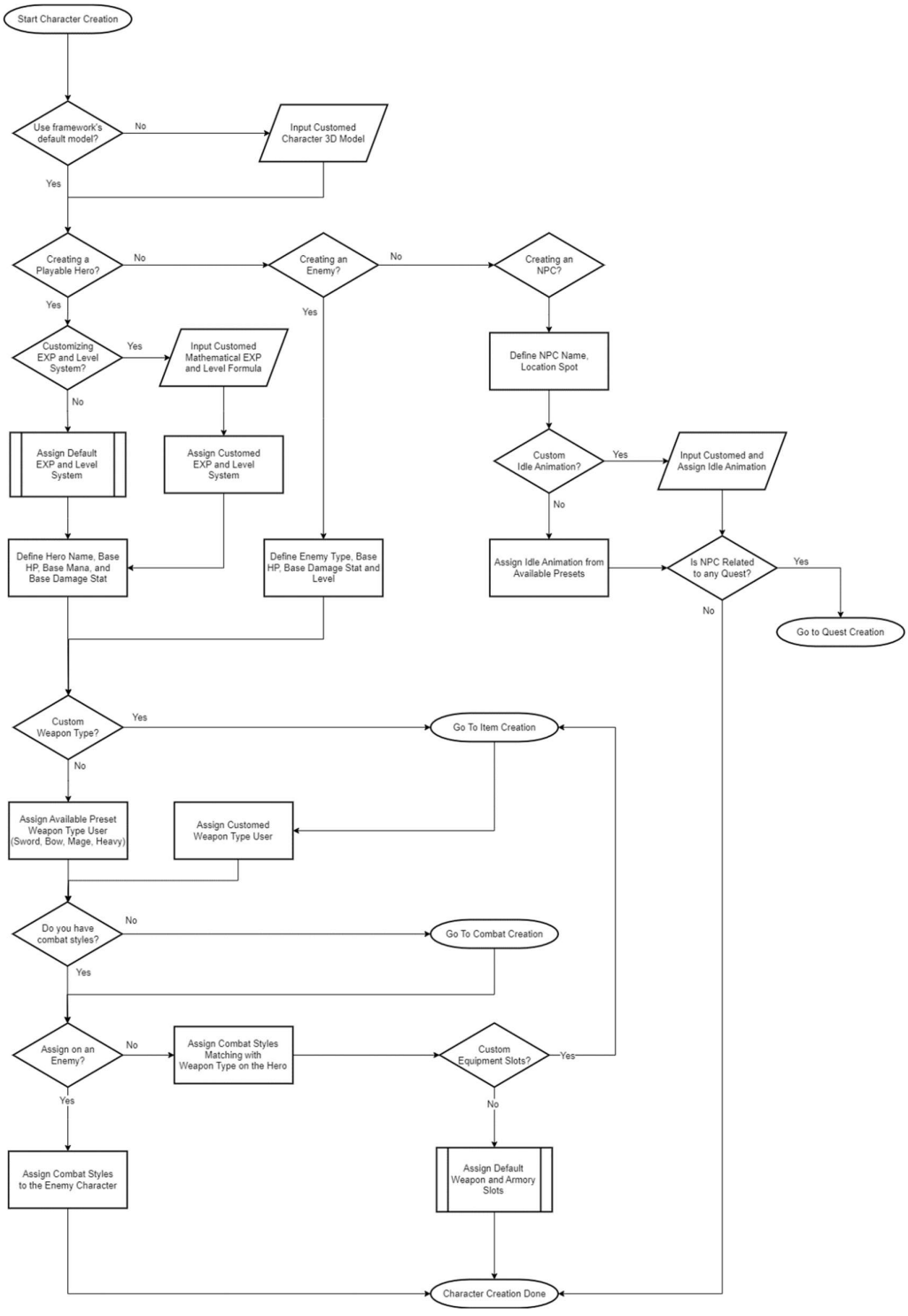

Figure 4.2 Flow Chart of Character Creation with Character Designer

Based on Figure 4.2 Flow Chart of Character Creation with Character Designer illustrates the step-by-step process for developers to create characters within the RPG Architectural Framework:

- **Input Custom 3D Character Model:** Developers have the option to input their own custom 3D character models into the framework, providing flexibility in character design and visual representation.
- **Character Type Selection:** The flowchart depicts the selection process for creating various character types, including heroes, enemies, or NPCs. Developers can choose the appropriate character type based on their intended role in the game.
- **Level System Specification:** Developers have the flexibility to utilize either the default level system provided by the framework or implement a custom level system tailored to their specific game mechanics and progression design.
- **Attribute Definition:** Developers define the attributes of the character, such as HP, Mana, and Damage Stats, which contribute to their overall capabilities and performance within the game world.
- **Weapon Type Assignment:** Developers assign weapon types to the character, determining the weapons they can wield and the associated combat styles available to them.
- **Combat Style Assignment:** Developers select from a range of available combat styles and assign them to the character, influencing their fighting techniques and strategies during combat encounters.

Weapon and Armory Slot Assignment: Finally, developers allocate default weapon and armory slots to the character, allowing them to equip various weapons and armor to enhance their combat effectiveness and customization options.

CHAPTER 5
IMPLEMENTATION

5.1 Technical Implementation

The cornerstone of the RPG framework's development lies in Unreal Engine 5's Blueprint system. Unlike traditional coding, Blueprints offer a visual scripting approach, where developers construct game mechanics and systems by connecting nodes and functions within a user-friendly interface. This intuitive environment empowers even those with limited coding experience to create complex functionalities.

Blueprints boast a vast library of pre-built nodes, each serving a specific purpose. This extensive toolkit allows the framework to handle diverse RPG elements through visual scripting. Developers can utilize Blueprints to craft intricate character creation systems, where players can customize their avatars. Similarly, quest generation and management become streamlined processes with Blueprints, allowing for the creation of engaging narrative experiences within the game. Furthermore, item management and inventory functionalities become readily achievable through Blueprint scripting. The framework even leverages Blueprints for crafting combat mechanics, enabling developers to design dynamic attack sequences and responsive battle systems.

The decision to prioritize Blueprints brings several advantages to the development process. First and foremost, the visual nature of Blueprints simplifies game development. Without the need for extensive coding knowledge, developers can quickly grasp the functionality of each node and construct complex systems with relative ease. This emphasis on user-friendliness opens the framework to a wider range of developers, fostering creativity and innovation. Additionally, Blueprints excel in rapid prototyping. By visually connecting nodes, developers can test new ideas and mechanics swiftly, allowing for faster iteration and refinement during development. The extensive library of pre-built nodes further contributes to

development efficiency. By leveraging these pre-built functionalities, developers can avoid reinventing the wheel, saving valuable time and resources. Ultimately, by prioritizing Blueprints, the framework achieves a streamlined development process, leading to faster creation cycles and RPG experience.

5.2 Development Process

The conceptual design phase involved making critical design choices for the character system, combat system, quest system, and item system. Diagrams and flowcharts in chapter 4 were created to visualize the structure and flow of these components, ensuring a cohesive framework. Detailed design documents are available in the appendix for reference.

My journey with Blueprints involved referring to Unreal Engine 5 documentation, engaging with community forums, and watching numerous YouTube tutorials. These resources provided comprehensive guidance on using Blueprints effectively. Utilizing Blueprints significantly reduced development time compared to hardcoding in C++. The visual nature of Blueprints simplifies the creation and management of game logic, making it an ideal choice for this project.

The integration process involved combining different components of the framework seamlessly. Manual testing was conducted extensively to ensure each system functioned correctly and interacted as expected. Potential automated testing methods were considered for future implementation to streamline the testing process further.

5.3 System Implementation Details

5.3.1 Character System Framework

The character system framework is a crucial part of any RPG, as it manages the player's attributes and progression. In this project, the BPC_CharacterStat Blueprint Class was designed to handle various character attributes, including experience points (EXP), level, health points (HP), and mana. This class contains several functions to modify these attributes based on in-game events, ensuring a dynamic and engaging gameplay experience.

BPC_CharacterStat Blueprint Class

- Increase Max Health

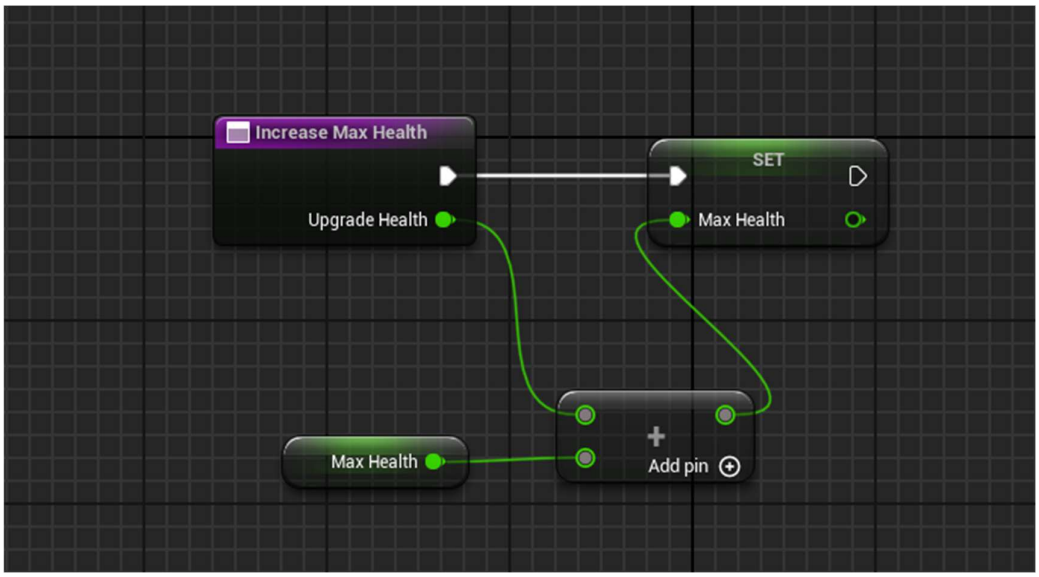

Figure 5.1 Increase Max Health Function

The function to increase max health is visually represented using a node setup in the Blueprint system. This setup allows the game to increase the character's maximum health based on a specified input value, providing a clear and efficient way to manage health upgrades within the game. Figure 5.1 Increase Max Health Function demonstrates the specific nodes and connections used to implement the increase in max health.

- Increase EXP

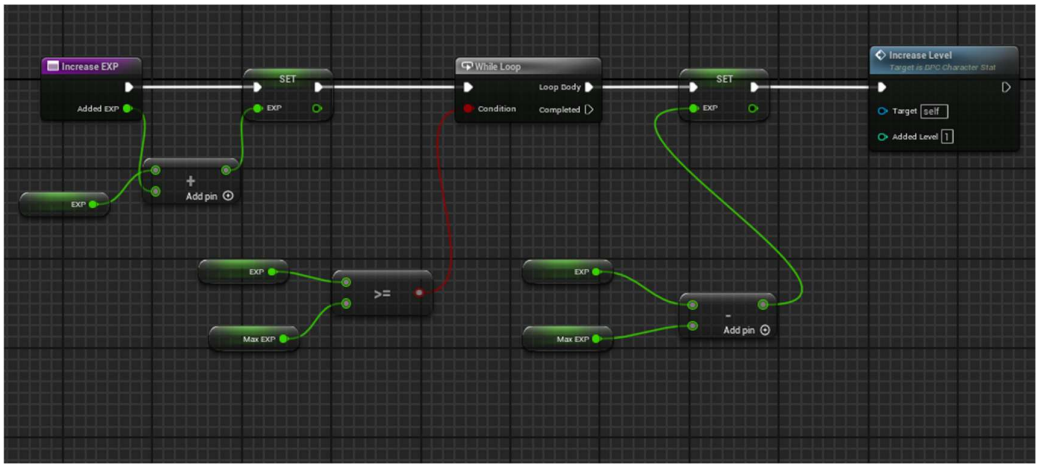

Figure 5.2 Increase EXP Function

The process of adding EXP to a character is illustrated through a Blueprint diagram. When a character gains experience, this function updates the current EXP and checks if the character has enough EXP to level up. If so, a while loop is used to handle large EXP gains, ensuring the character's level is accurately updated. Figure 5.2 Increase EXP Function shows the node setup for adding and managing EXP, including the logic for handling level ups.

- Increase Level:

Figure 5.3 Increase Level Function

This function details how the character's level is updated and how this affects other stats such as max EXP, max health, max mana, attack stat, and defense stat. By incrementing the level, the character's overall power and capabilities are enhanced.

Figure 5.3 Increase Level Function provides a visual representation of the level-up process, highlighting the interconnectedness of different character attributes.

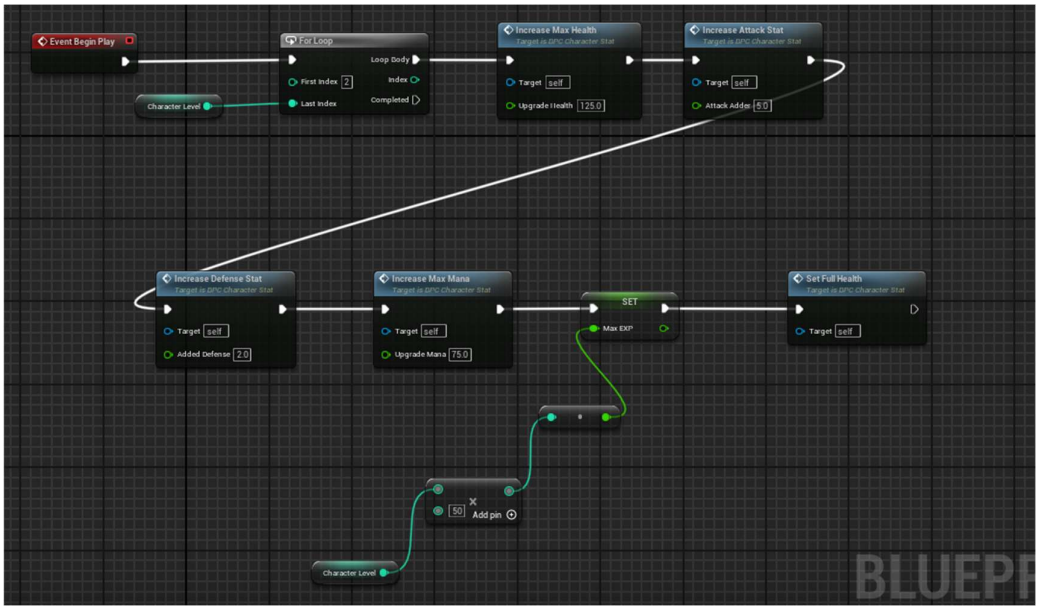

Figure 5.4 EventGraph of BPC_CharacterStat

The EventGraph within the BPC_CharacterStat class initializes the character's stats based on the current level when the game begins. This initialization ensures that all character attributes are set correctly at the start of gameplay. Figure 5.4 EventGraph of BPC_CharacterStat demonstrates the initialization logic within the EventGraph, showing how various nodes work together to set up the character's initial stats.

BP_ThirdPersonHero Blueprint Class

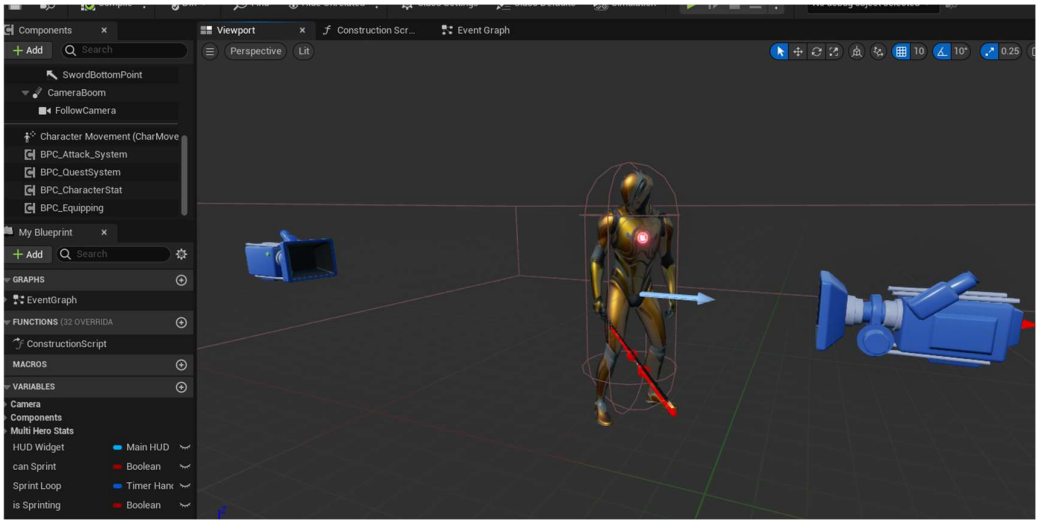

Figure 5.5 BPC_ThirdPersonHero Viewport

The BP_ThirdPersonHero class represents the player-controlled character. It includes essential functionalities such as death and respawn logic, ensuring that the player has a seamless experience. This class serves as the primary blueprint for player interactions and character management within the game.

User Interface (UI) for Character System:

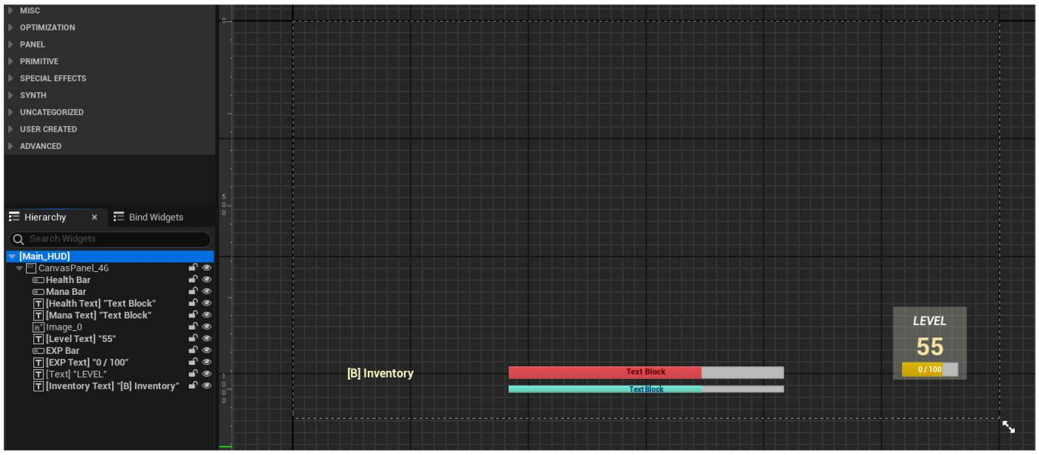

Figure 5.6 Main_HUD Designer Mode

The Main_HUD widget blueprint is designed to display essential character stats, including HP, mana, level, and EXP. This UI provides players with real-time

information about their character's status, enhancing the overall gameplay experience. Figure 5.6 Main_HUD Designer Modeshows the UI elements within the Main_HUD widget, illustrating how character stats are presented to the player.

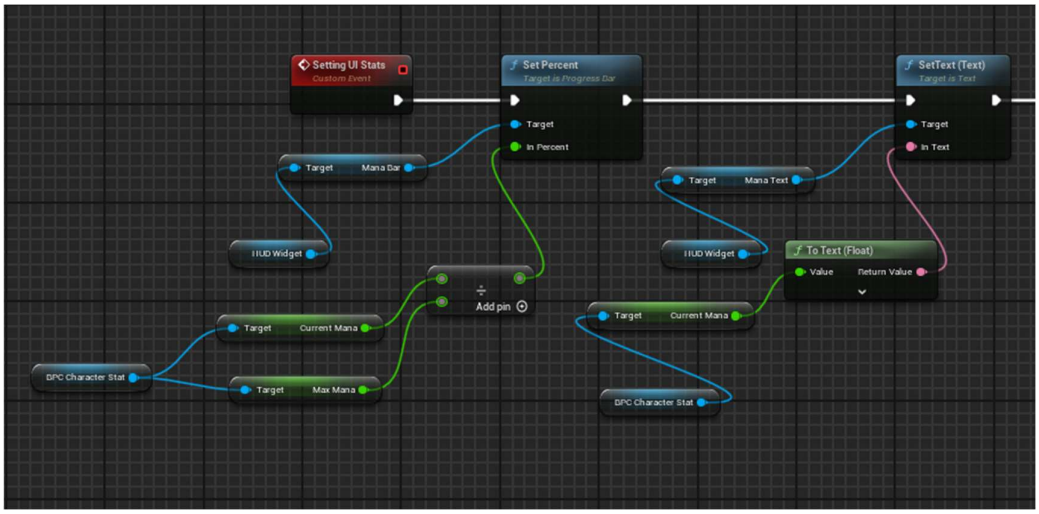

Figure 5.7 Setting UI Event

This function connects character stats to UI text elements, ensuring that the displayed information is accurate and up-to-date. Figure 5.7 Setting UI EventThis image demonstrates how the current mana value is connected to the corresponding text element in the UI. This function links character stats to UI percentage bars, such as health and mana bars, providing a visual representation of the character's status. Figure 5.7 Setting UI Event shows the logic for updating the mana bar percentage, highlighting the connection between character stats and their visual representation in the UI.

Enemy AI

Figure 5.8 Enemy AI Radius HitBox

One of the foundational elements of AI behaviour in Unreal Engine 5 is the use of detection hitboxes, which are visual representations of the AI's perception ranges. These hitboxes define the zones within which the AI can detect various stimuli such as sight and sound. As depicted in the accompanying figure, three distinct colors represent different types of perception ranges: green, pink, and yellow.

The green radius represents the AI's sight range, known as AI Sight Config. This is a visual trigger that allows the AI to perceive objects or characters within its line of sight. When an enemy AI detects the player or any other object within this range, it can trigger a variety of responses, such as engaging in combat, issuing an alert, or changing its patrol route. The setup of AI Sight Config involves specifying parameters such as sight radius, peripheral vision angle, and the detection logic.

The pink radius indicates the lose sight radius. If the player exits the green sight zone and enters the pink zone, the AI will eventually lose sight of the player. This triggers a different set of behaviours, typically causing the AI to abandon its pursuit and return to its default state, such as patrolling or guarding a specific area. This mechanic is essential for creating realistic and fair AI behaviour, ensuring that players have opportunities to evade detection and plan strategic manoeuvres.

The yellow radius is associated with AI Sense Hearing, which enables the AI to detect sounds generated by the player or other sources. Actions such as running, firing a weapon, or interacting with objects can produce sounds that alert the AI to the player's presence. Upon detecting a sound within this radius, the AI will move towards the source of the noise to investigate. If the player manages to escape without being seen, the AI will eventually forget about the incident and resume its original behaviour after a short duration.

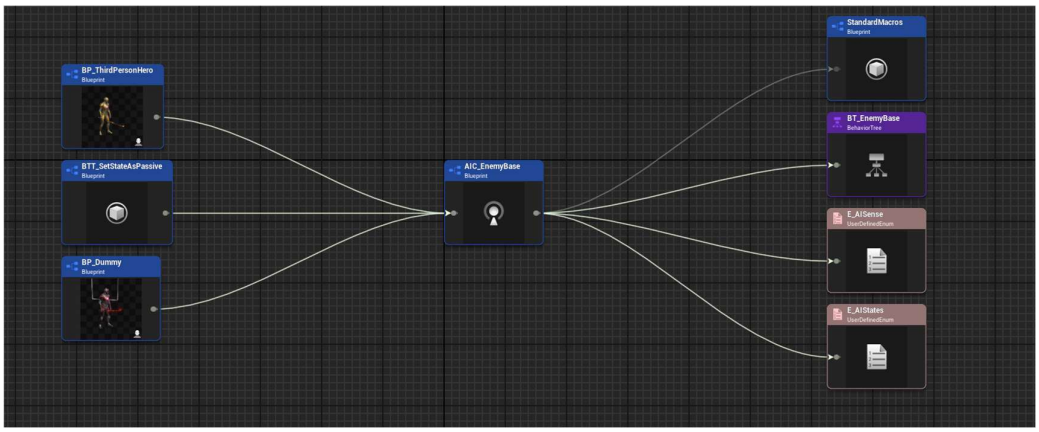

Figure 5.9 Enemy AI Reference Flow

While the perception systems define how the AI detects provocations, the actual logic dictating the AI's responses to these stimuli is implemented through Blueprints. In my project, most of this logic is encapsulated within a custom AI controller class named AIC_EnemyBase. This class contains the rules and procedures that govern how the AI reacts to different types of detections.

5.3.2 Combat System Framework

The combat system is a fundamental aspect of RPG gameplay, providing players with engaging and dynamic interactions. The combat system in this project includes various attack combos, projectile abilities, and area-of-effect (AOE) abilities, each implemented using Unreal Engine 5's Blueprint system.

Attack Combos

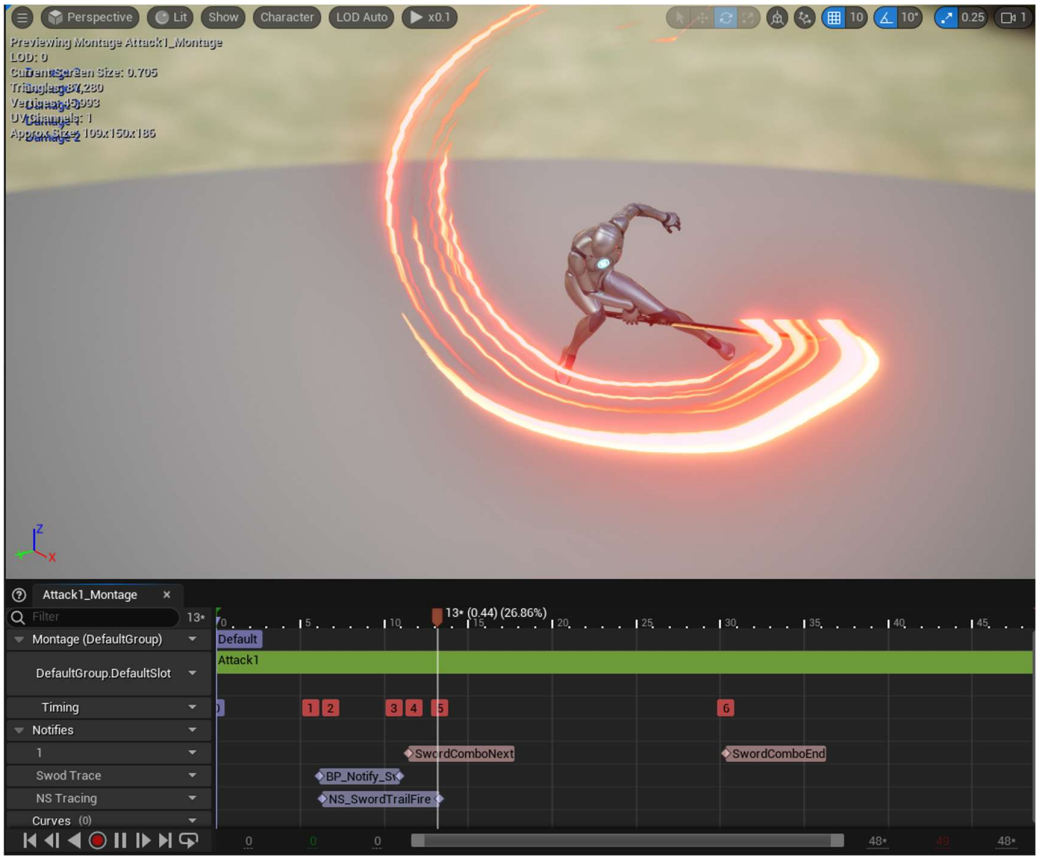

Figure 5.10 Attack 1 Animation Montage

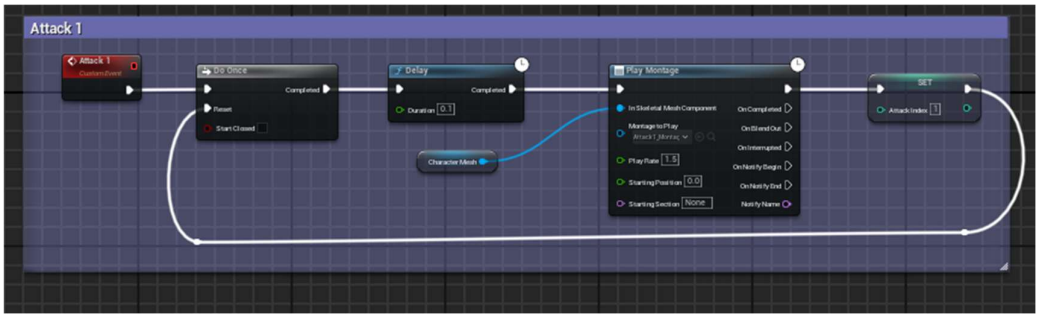

Figure 5.11 Attack 1 Animation Implementation

The combat system features four primary attack combos: a slash, a stab, a kick, and a slam. These actions are named as Attack 1, Attack 2, Attack 3 and Attack 4 respectively. These combos provide a range of offensive options for the player, enhancing the combat experience. Figure 5.10 Attack 1 Animation Montage displays the first attack animation, complete with visual effects (VFX), demonstrating the dynamic nature of combat animations. Figure 5.9 Attack 1 Animation Implementation shows how the animation montage of Attack 1 is applied. Other attacks are applied in the same way.

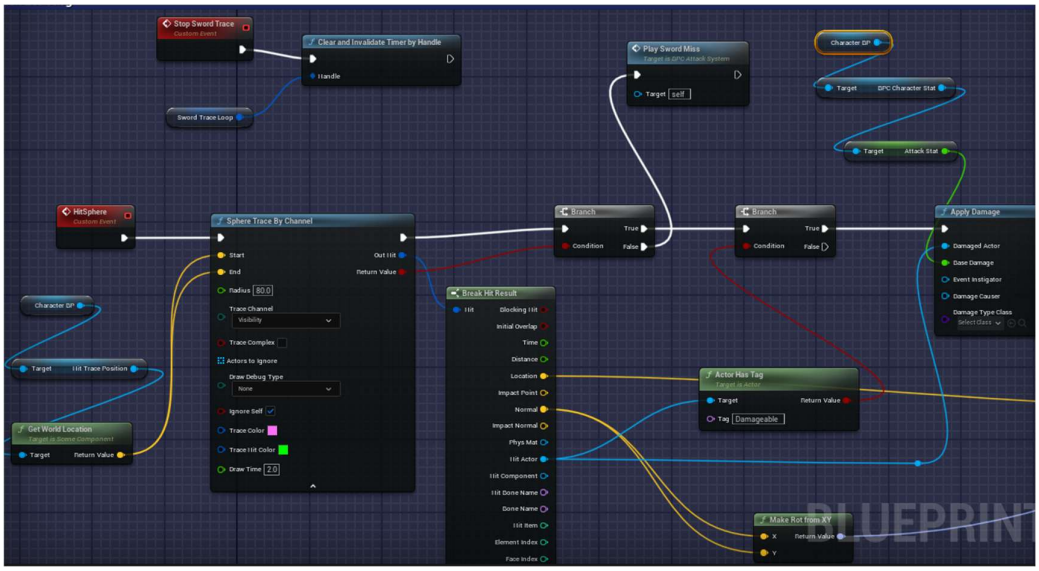

Figure 5.12 Sphere Tracing to Detect Enemies

Each attack is implemented sphere tracing to detect enemies. Sphere tracing allows for precise hit detection, ensuring that attacks accurately register hits on

enemies. Figure 5.10 Sphere Tracing to Detect Enemies shows the logic for detecting hits and applying damage to actors tagged as "Damageable." This setup ensures that only valid targets are affected by the player's attacks.

BP_Dummy Blueprint

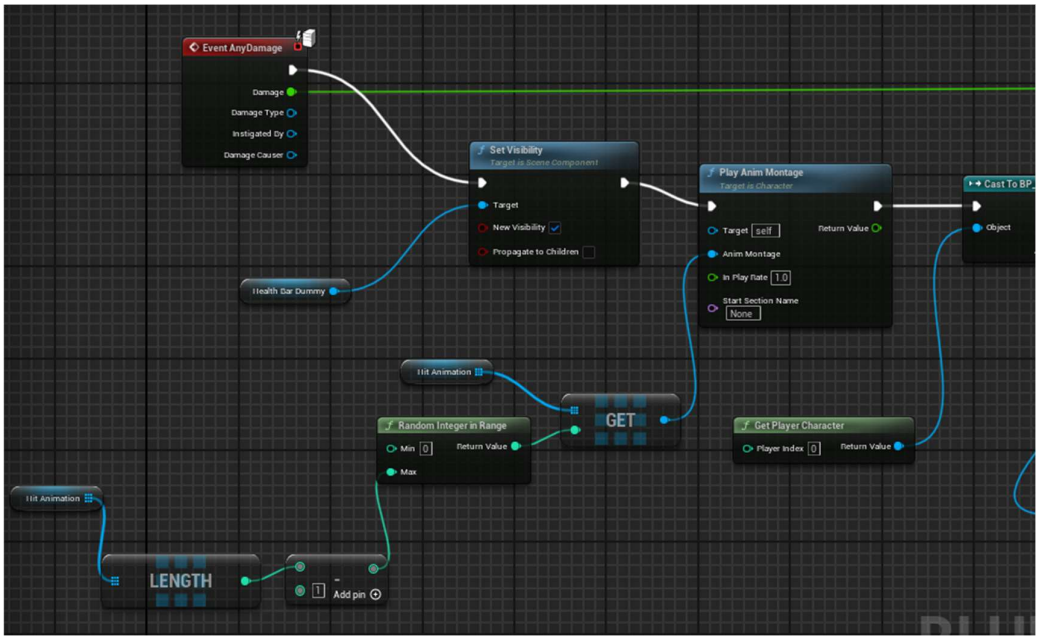

Figure 5.13 Event AnyDamage for BP_Dummy Blueprint

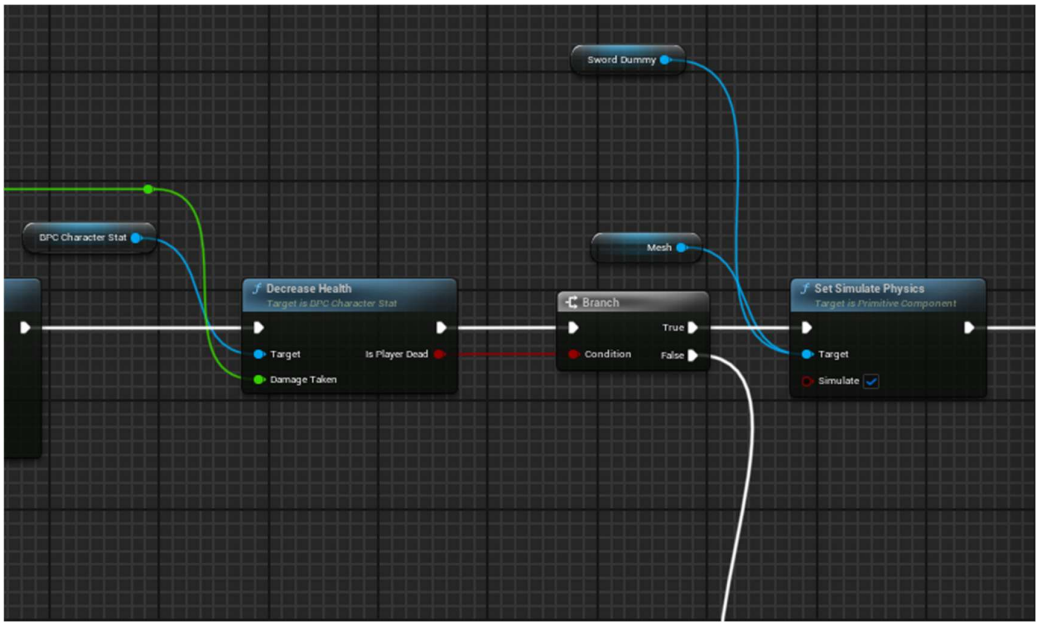

Figure 5.14 Continuation of Event AnyDamage for BP_Dummy Blueprint

The BP_Dummy class serves as a target enemy for testing combat mechanics. It includes logic for damage detection and health bar visibility, providing a straightforward way to test and refine the combat system. Figure 5.13 Event AnyDamage for BP_Dummy Blueprint and Figure 5.14 Continuation of Event AnyDamage for BP_Dummy Blueprint demonstrates the damage detection and health bar visibility logic within the BP_Dummy class.

Projectile and AOE Abilities

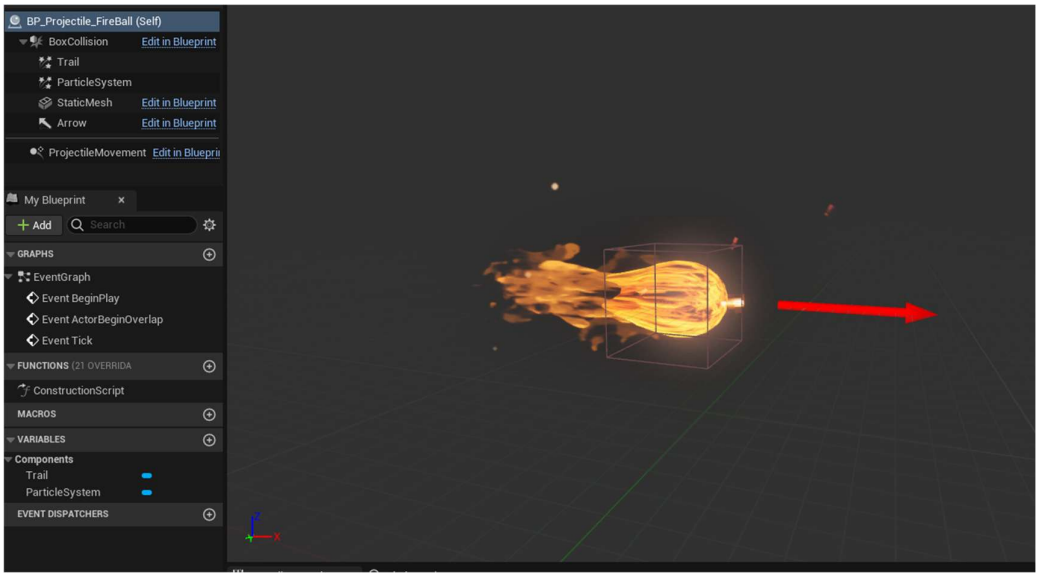

Figure 5.15 BP_Projectile_Fireball Viewport

This class is a child of the BP_ProjectilesBase class, specifically designed for projectile abilities. This is practiced so that any projectile abilities that would be created in the future can be created as the child class of the base class. The BP_Projectile_Fireball is an ability that can be performed as the Hero Character by pressing Q on the keyboard. The logic is by calling on hit, and if the Hit Actor has the Damageable tag, the actor will receive damage and reduce its health, just like the attack swings. Figure 5.15 BP_Projectile_Fireball Viewport illustrates the setup for the fireball projectile, highlighting its unique properties and behaviours.

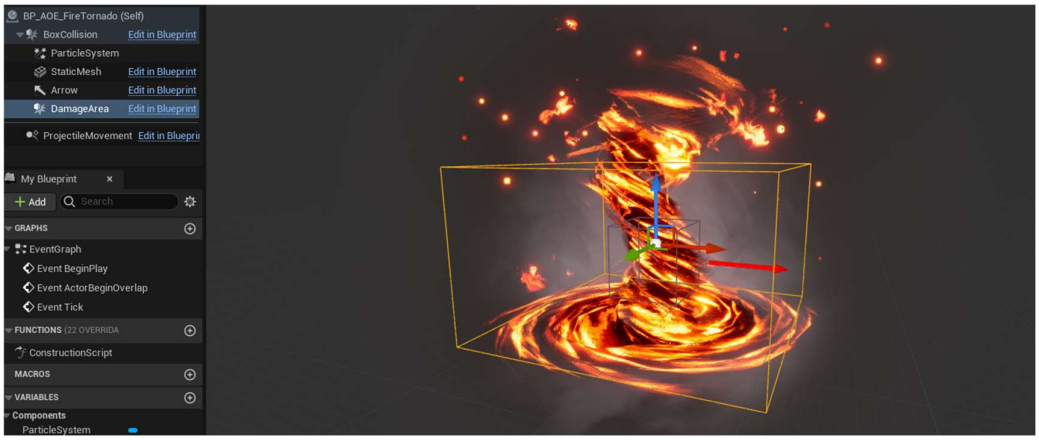

Figure 5.16 BP_AOE_FireTornado

This class is a child of the BP_AOE_Base_DamageOverTime class, designed for AOE abilities that cause damage over time. The base class has the logic of applying damage over time by using Set Timer by Event default unreal function and connecting it to an event that applies damage on an actor with the tag "Damageable". The Timer is set to 0.5 seconds, that means the damage would be applied every 0.5 seconds thus giving the Damage Over Time effect. The fire tornado ability provides a powerful and visually impressive AOE attack, this can be done by pressing the keyboard button E. Figure 5.16 BP_AOE_FireTornado shows the setup for the fire tornado ability, emphasizing its damage-over-time properties and VFX.

5.3.3 Quest System Framework

The quest system is a core component of RPGs, providing players with objectives and goals to achieve. This project implements a solid quest system using numerous blueprint classes and structures.

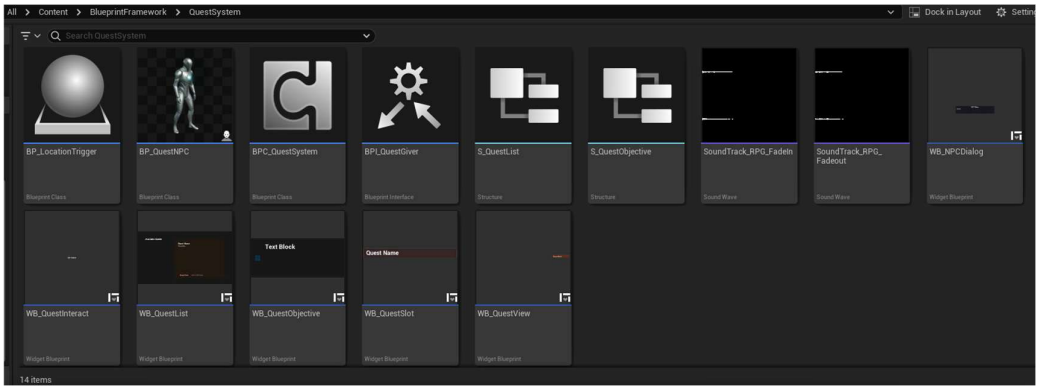

Figure 5.17 Quest System Related Files

The quest system is organized using several blueprint classes, including Widget Blueprints for the UI, Actor Blueprints for location triggers, and structure files for quest data. This organization ensures a clear and manageable framework for quest implementation. Figure 5.17 Quest System Related Files shows the organization of quest-related files, highlighting the various blueprint classes and structures used in the quest system.

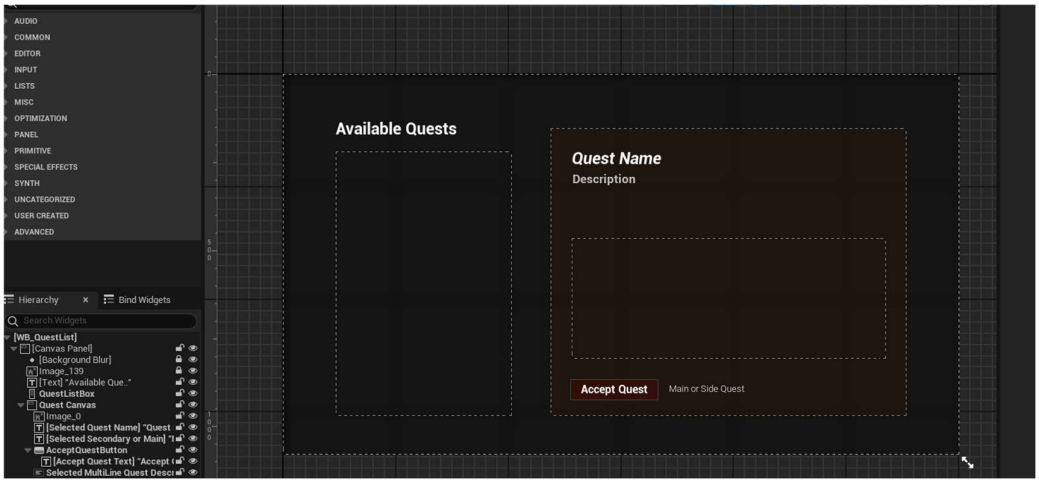

Figure 5.18 WB_Quest List UI Designer

The WB_QuestList widget blueprint manages the quest list interface, allowing players to view and accept quests. Figure 5.18 WB_Quest List UI Designer displays the quest list UI, providing a visual representation of the available quests. The UI of the quest list that would appear once the player interacts with the quest NPC by pressing the button F. The quest selected would be accepted when the Accept Quest button is clicked.

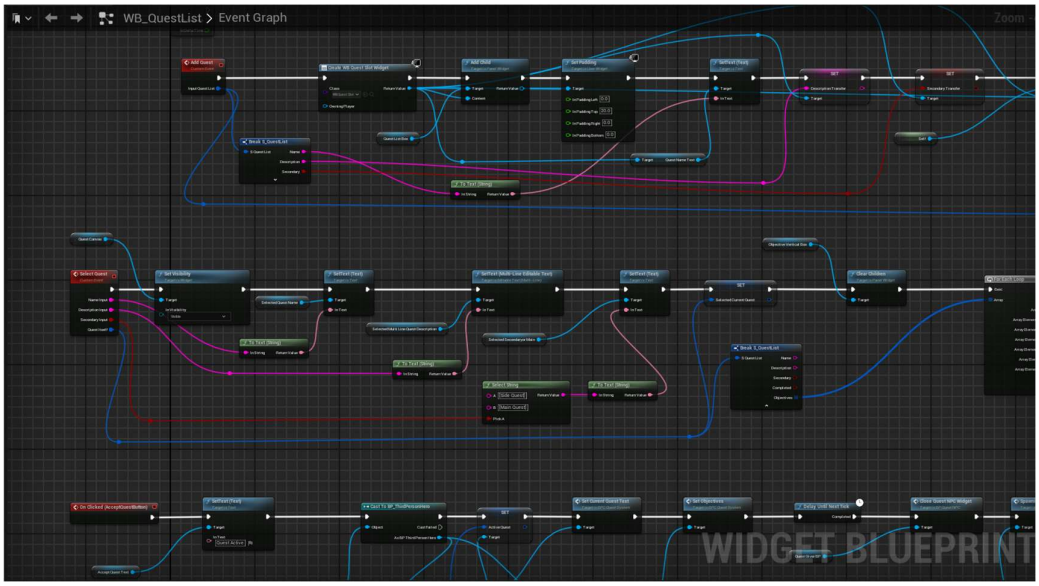

Figure 5.19 Blueprint Graph of WB_QuestList

The blueprint graph of WB_QuestList demonstrates the logic for selecting and accepting quests, ensuring that the quest system functions smoothly and intuitively for the player. Figure 5.19 Blueprint Graph of WB_QuestList shows the logic for selecting and accepting quests within the WB_QuestList blueprint, highlighting the connections and nodes involved in quest management. Quest item itself is stored in a structure Blueprint class. Structure Blueprint allows developers to store data in their own desired customized variables. Quest objectives are stored in arrays and updated based on player actions, ensuring that the quest system accurately tracks progress and completion.

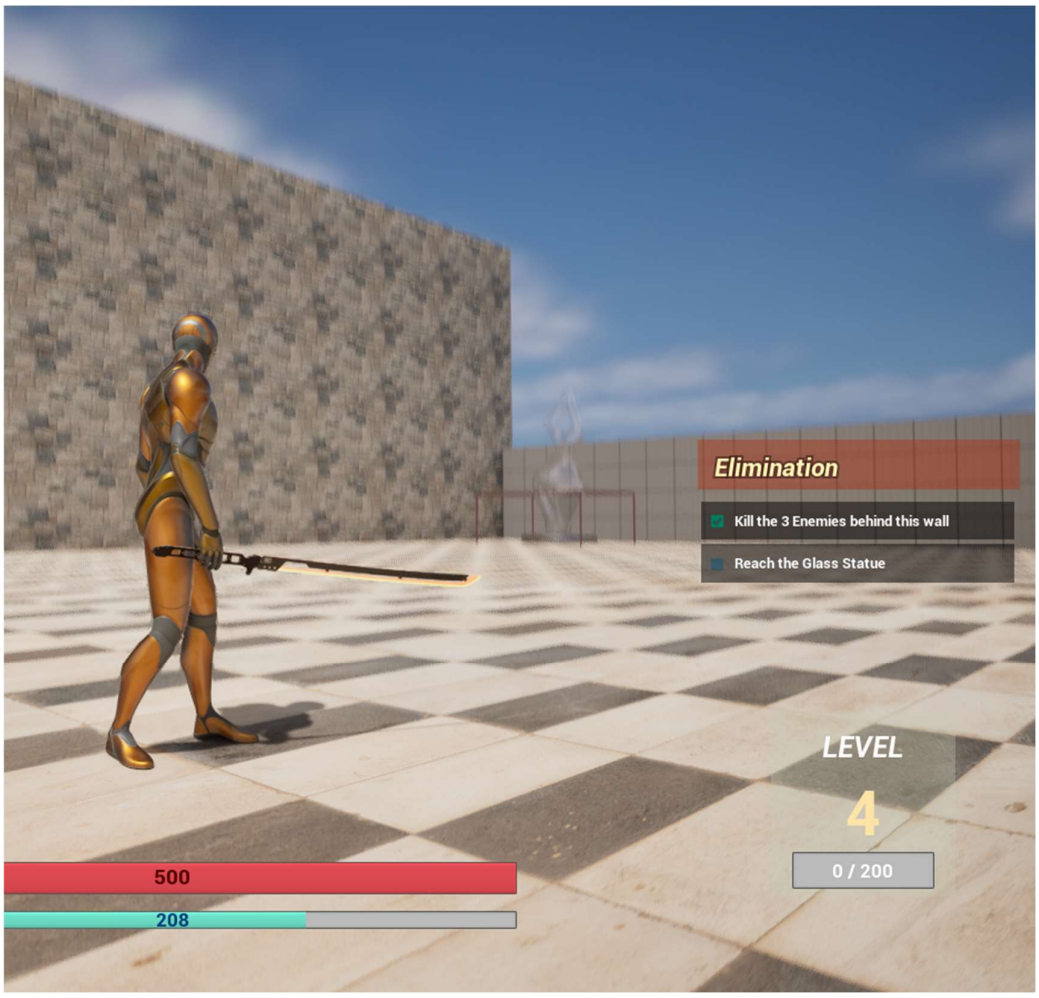

Figure 5.20 Quest View UI

Figure 5.20 Quest View UI illustrates how quest objectives are tracked and marked as completed, providing a clear and efficient way to manage quest progress. Once a quest is accepted, there would be the quest name and objectives that would appear on the right side of the screen. The quest objectives are stored in the form of array as mentioned earlier. When it comes to location related objective, the location trigger box would be the actor to sense any character that enters the box and tick the objective itself and marked as completed. When it comes to eliminating enemies in an objective, every time an enemy is killed, the game will check whether the enemy itself is related to a quest or not. If it is, then the enemy actor that is stored in an array of actors within the objective will be reduced by 1. Once it's reduced to 0, the objective that is related to killing enemies will be ticked and mark as completed.

Quest NPC

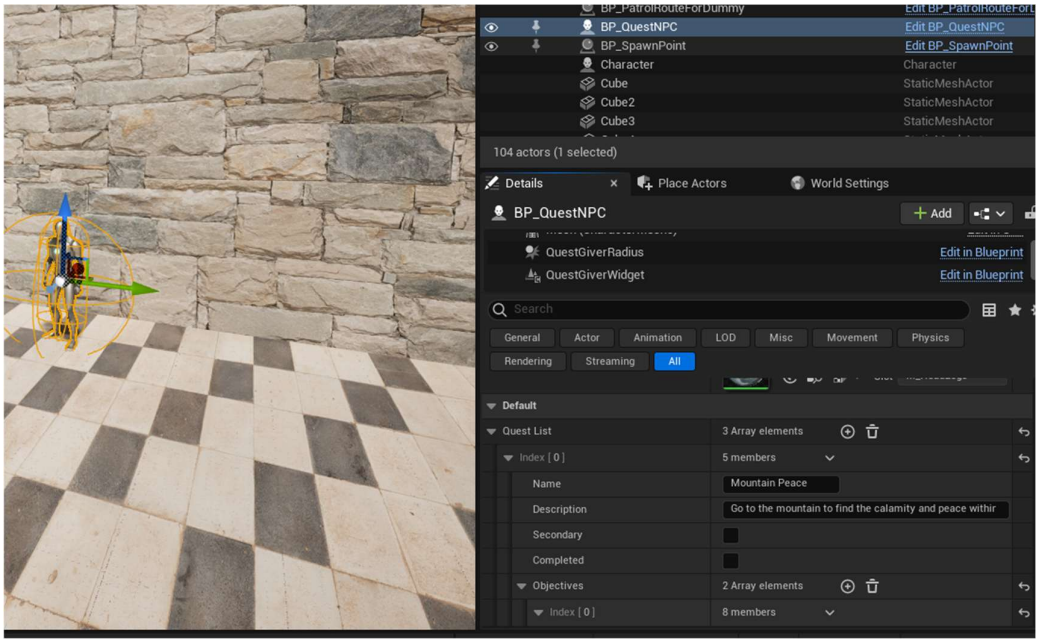

Figure 5.21 Quest NPC

The Quest NPC serves as a pivotal component in creating an immersive and interactive RPG experience. This NPC acts as a conduit for players to receive, track, and complete quests, thereby driving the narrative and providing a structured path for gameplay.

When a Quest NPC is placed into the game world, developers have access to a streamlined process for configuring quests through the details panel on the right side of the Unreal Engine editor. This panel offers a user-friendly interface where developers can populate various attributes related to quests, ensuring that the NPC can effectively interact with players and provide them with tasks to complete.

The Quest List is an integral part of the Quest NPC's setup. Within the details panel, developers can add multiple quests to this list. Each quest entry can be configured with the following attributes:

- Name: The name of the quest, which is displayed to the player when they interact with the Quest NPC.
- Description: A detailed description of the quest, providing players with context and narrative background.
- Secondary: Additional information or secondary objectives that might be part of the quest.
- Completed: A flag indicating whether the quest has been completed, used to track progress and trigger subsequent events or rewards.
- Objectives: A comprehensive list of tasks that players must complete to finish the quest. These tasks can involve eliminating specific enemies or reaching certain locations.

The Objectives index is where developers can specify the exact requirements for completing a quest. Developers can list the enemies that need to be defeated. Each enemy type or specific enemy can be added to the objectives index, ensuring that the quest tracking system can accurately monitor the player's progress. For quests that require players to visit certain locations, developers can use the BP_LocationTrigger blueprint. This blueprint acts as a trigger point that players must reach to complete the location-based objective. Developers can place BP_LocationTrigger actors within the game world and then link these triggers to the relevant objectives in the Quest NPC's details panel.

5.3.4 Item System Framework

The item system is an essential part of RPGs, allowing players to collect, equip, and use various items to enhance their characters. This project implements a comprehensive item system using several blueprint classes.

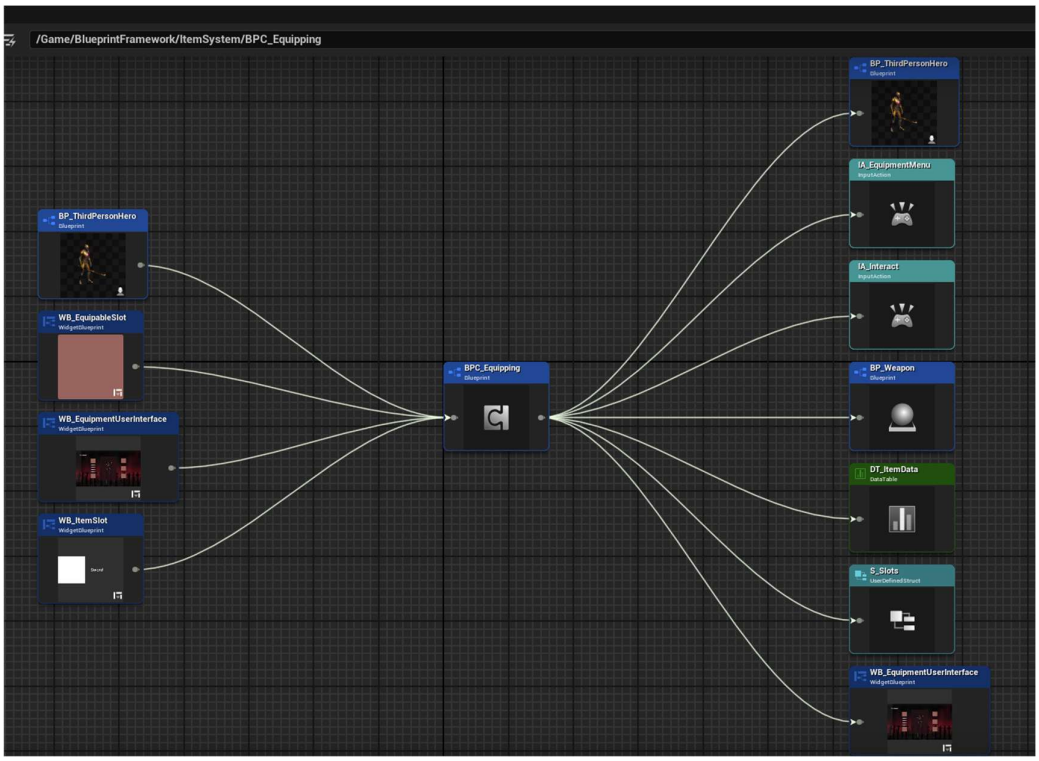

Figure 5.22 BPC_Equipping Reference Viewer

The item system includes various blueprint classes, such as BP_Weapon for weapons and DT_ItemData for item data. These classes are organized to ensure efficient management and implementation of items. Figure 5.22 BPC_Equipping Reference Viewer shows the reference flow of item-related files, highlighting the various blueprint classes used in the item system.

Items are categorized using enumerators, with current categories including Sword, Spear, Mage, Archer, Bracelets, Armours, Amulets, and Rings. This

categorization allows for easy management and access to different types of items within the game. The BPC_Equipping blueprint class handles equipping items and accessing the equipment UI. This class ensures that players can easily equip and manage their items, enhancing the overall gameplay experience.

One of the most compelling reasons for prioritizing development with Blueprints in Unreal Engine 5 is the simplicity and efficiency they offer. Blueprints allow developers to create intricate gameplay mechanics using a visual interface where logic is represented with nodes and lines of connection. This visual approach makes it easier to understand and manage complex systems, reducing the likelihood of errors and increasing productivity. For instance, in my project, the high amount of logic needed for character attributes, combat mechanics, quest systems, and item management is efficiently handled with a relatively small number of nodes and connections. This streamlined approach contrasts sharply with traditional coding, where such complexity would require extensive lines of code and intricate debugging.

The visual nature of Blueprints also enhances accessibility, enabling individuals without extensive programming knowledge to participate in game development. This democratization of game development means that artists, designers, and other team members can contribute more directly to the creation of game mechanics and features. Blueprints facilitate collaboration by providing a common, intuitive platform that all team members can understand and manipulate. In the context of my FYP, this accessibility allowed for a more collaborative approach to developing the various RPG systems, ensuring that each aspect of the game could be fine-tuned and perfected by experts in different fields.

CHAPTER 6
TESTING

This chapter is about the testing methodologies and results of the RPG framework developed in Unreal Engine 5 using Blueprints. The purpose of this RPG framework is to provide a comprehensive architectural structure for various RPG fundamentals, including character, combat, quest, and item systems, thus enabling the creation of engaging and interactive RPG experiences. By using a systematic testing approach, we aim to ensure that the framework functions as intended and delivers a vigorous and user-friendly development environment.

6.1 Testing Plan and Execution

The testing process for the RPG framework is divided into several phases: functional testing, integration testing, and usability testing. Each phase targets different aspects of the framework to verify its functionality, interaction, and user-friendliness.

6.1.1 Functional Testing

Functional testing focuses on verifying that each individual feature and functionality within the framework operates as intended. The following specific functionalities were meticulously tested:

Character Stat Calculations

The system was tested to ensure that leveling up appropriately increases various stats, including maximum HP, maximum mana, experience points, attack, and defense. Each of these increases was verified to be accurate based on the mathematical functions integrated into the system.

Additionally, the accuracy of character stats was confirmed to be precise, reflecting the correct calculations within the function scripts. The player UI was scrutinized to ensure it accurately displayed the level, experience points, mana, and HP, thereby providing the player with reliable and up-to-date information about their character.

Enemy AI

Rigorous tests were conducted to verify that the enemy AI detects, chases, and interacts with the player correctly. This involved ensuring that the AI behaves as expected within the defined sight and hearing radii, reacting to the player's presence and actions appropriately. The correct functioning of enemy death triggers was confirmed, ensuring that the system responds accurately when an enemy is defeated.

Player Mechanics

Player death and respawn functions were thoroughly tested to confirm that they work as intended. This included verifying that the player character respawns correctly after death, with a brief delay to enhance gameplay realism. The item equipping system was tested to ensure that equipping and changing sword models function correctly, reflecting the appropriate visual and statistical changes.

Quest System

Quest acceptance, NPC interactions, and quest UI functionality were examined to confirm that the quest system operates smoothly. This involved verifying that quests could be accepted, tracked, and completed accurately. Quest location triggers and enemy elimination objectives were tested to ensure they function as expected, providing the player with clear and achievable objectives. Completing all quest objectives was verified to lead to successful quest completion, thereby rewarding the player appropriately.

Item Equipping

The item equipping system was scrutinized to ensure that equipping different items accurately updates the player's stats. This involved verifying that the appropriate stat changes were reflected in the player's attributes. The equipment UI was tested to ensure it displayed accurate stats and item details, providing the player with a clear and informative interface.

6.1.2 Integration Testing

Integration testing focuses on ensuring that different components within the framework interact seamlessly and exchange data correctly. The following interactions were tested to confirm their functionality:

Character and Combat System

Tests were conducted to verify that character attack stats accurately affect ability damage and sword attack damage. This involved ensuring that the damage calculations reflect the correct attack stats of the character. Enemy damage reception was inspected to confirm that enemies take accurate damage based on the player's attack stats. This ensured that combat interactions were realistic and balanced. The system was tested to ensure that equipping different swords updates the attack stats correctly, providing the player with the intended combat advantages.

Quest and Combat System

The interaction between the quest and combat systems was tested to verify that completing quests grants experience points accurately. This involved ensuring that the quest completion logic was correctly integrated with the experience point system. The system was examined to confirm that eliminating enemies provides experience points as intended, rewarding the player for their combat efforts. Mana consumption during ability usage was tested to ensure that abilities consume the correct amount of mana. Additionally, scenarios where the player has insufficient mana were tested to confirm that the system handles these situations properly.

6.1.3 Usability Testing

Usability testing evaluates the framework's user-friendliness and intuitiveness from a developer’s perspective. The following steps were taken to ensure the framework's usability.

A fresh Unreal Engine project was created, and the architectural framework was integrated. This process involved applying the character system and placing the hero in the blank project. Enemy AI, including the NAV mesh, was added to the project, and character stats were tested to confirm their accuracy and functionality. The combat system was integrated and tested to verify damage calculations and attack functionality. This involved ensuring that the combat mechanics were realistic and responsive.

Quest NPCs were added to the blank project, and objectives were set up. This included configuring quest elimination and location triggers to ensure they functioned correctly. The quest system was tested to verify that it accurately tracked and completed objectives, providing the player with clear and achievable goals. The item functionality was integrated and tested to ensure that equipping items updated the player’s stats accurately. This involved verifying that the appropriate stat changes were reflected in the player’s attributes.

6.2 Results Analysis and Discussion

The testing process involved playing the game and systematically testing each functionality. For complex logic, such as quest objective completion, debug prints were used to verify the execution of scripts. For instance, quest location triggers were configured to print "user has entered," and enemy kills were tracked by printing the length of the array, ensuring that the system accurately monitored objective completion.

6.2.1 Test Results

The results of the functional, integration, and usability tests are presented below in a comprehensive manner:

Table 6.1 Test Cases Results

Test Case	Status	Notes
Character stat calculations	Pass	Stats updated accurately with leveling up.
Player UI	Pass	Displayed correct level, experience points, mana, and HP.
Enemy AI functionality	Pass	Detected and interacted with the player correctly.
Player death and respawn	Pass	Respawned correctly after death with a delay.
Combat Attacks	Pass	Sword Swings are applied correctly
Attack Tracing	Pass	Attack Trace works but needs improvement
Damage System	Pass	Enemy can take damage correctly according to the Attack Stats of player
Abilities	Pass	Projectiles and AOE skills works accordingly with the damage
Visual effects for abilities	Pass	Visual effects triggered correctly for different abilities, enhancing gameplay immersion.
Quest acceptance and NPC interactions	Pass	Quests accepted and tracked correctly.
Quest UI functionality	Pass	Displayed quest details accurately.
Quest location triggers	Pass	Triggered correctly upon player entry.
Quest enemy elimination	Pass	Tracked enemy kills accurately and completed quests upon fulfilling objectives.
Item equipping	Pass	Updated player stats and displayed correct sword models.
Equipment UI functionality	Pass	Displayed accurate stats and item details.
Inventory management	Pass	Inventory UI and functionality worked correctly, allowing item pickup and use
Integration of character and combat system	Pass	Attack stats affected damage calculations correctly.
Quest and combat system interaction	Pass	Quests granted experience points upon completion.

Ability mana consumption	Pass	Mana was consumed correctly, and insufficient mana scenarios were handled properly.
Sound triggers for abilities and actions	Pass	Sound effects played correctly based on player actions and abilities.
UI responsiveness and updates	Pass	UI elements updated in real-time, reflecting changes in character stats, quest progress, and inventory accordingly

6.2.2 Encountered Issues and Bug Fixes

During the testing process, several issues were encountered, each requiring specific solutions.

Enemy AI:

- Initial implementation caused enemies to chase the player indefinitely, failing to return to their default state.
- Solution: Adjusted the AI behavior tree and logic to include a lose sight radius, allowing enemies to return to their default state when the player is out of range.

Player Respawn:

- Respawn logic had multiple issues, particularly with resetting stats and position.
- Solution: Instead of destroying the actor, the approach was modified to disable input and enable physics, allowing the character to fall off the map. Following a delay, the character respawns at a designated spawn point with input re-enabled.

Quest Objective Tracking:

- Arrays did not reduce properly for enemy elimination objectives, causing incorrect tracking of enemy kills.
- Solution: Refined the array reduction logic to track kills accurately, ensuring quest completion upon fulfilling objectives.

Item Stat Updates:

- Issues with item stat updates not reversing correctly when switching items, leading to incorrect stat calculations.
- Solution: Implemented a reverse function to handle stat updates when changing items and adjusted the leveling up logic to add stats instead of setting them, ensuring accurate stat tracking.

Attack Tracing:

- Attack tracing detected only one enemy even if multiple enemies were within the hitbox.
- Status: Issue remains unresolved, requiring further investigation and refinement.

6.2.3 Overall Analysis

The testing process was largely successful, with most functionalities working as intended. The primary issues that remain unresolved pertain to attack tracing and combo handling. Despite these challenges, the framework provides a solid foundation for RPG development, with a user-friendly and flexible architecture.

There are few unresolved issues, such as attack tracing detects only one enemy at a time, even when multiple enemies are within the hitbox. Rapid clicking for attacks skips combo sequences, requiring a specific click pace.

From a developer's perspective, the framework is functional but not as smooth as anticipated. While it effectively facilitates RPG creation, there are areas where the user experience could be improved. On a scale of 1 to 10, with 10 being the smoothest experience akin to popular RPGs like Genshin Impact or Wuthering Waves, the final product of this RPG Architectural Framework is rated at 7.

CHAPTER 7 CONCLUSION

The development of the RPG framework through Unreal Engine 5 has been a comprehensive and insightful journey, culminating in the successful production of four key systems: Character System, Combat System, Quest System, and Item System. Each of these systems has been meticulously crafted to enhance the functionality and user experience of RPG development.

1. Character System

The Character System was designed to handle all character-related functionalities, including stat calculations, levelling up, health, mana, and experience management. The system allows for dynamic updates to character stats based on player actions and item equipping, ensuring a responsive gameplay experience.

2. Combat System

The Combat System encompasses all combat-related mechanics, from basic attacks to special abilities. This system integrates seamlessly with the Character System, ensuring that attack stats and abilities accurately reflect the character's current state. The system also includes enemy AI logic, allowing for challenging and engaging combat scenarios.

3. Quest System

The Quest System enables the creation and management of various quests within the game. This system supports multiple quest types, including enemy elimination and location-based objectives. It also integrates with the Quest NPC and location triggers, providing a cohesive questing experience that tracks and updates quest progress in real-time.

4. Item System

The Item System manages all aspects of item creation, equipping, and stat modification. This system ensures that items correctly affect character stats and are visually represented in the game world. It also includes functionality for updating the Equipment UI, allowing players to easily manage their inventory and equipped items.

7.1 Evaluation of Project Objectives

Conclusion of Objective 1: Develop a Character Designer

The first objective was to create a robust character design system that allows for customizable character stats, levelling, and UI. This system aimed to ensure scalability and maintainability, capable of handling a variety of character types and abilities. This objective was successfully achieved. The character design system developed is highly flexible, allowing for detailed customization and efficient management of character stats and progression. The integration of the system with Unreal Engine's Blueprint system ensured ease of use and adaptability. However, there are still areas for improvement, such as enhancing the user interface and adding more complex character abilities and traits.

Conclusion of Objective 2: Develop a Quest Generator

The second objective focused on building a flexible quest generation system that enables the creation of complex quest lines with NPC interactions and objective tracking. This system was designed to maintain high performance even as the number of quests and interactions increases. This objective was also successfully achieved. The quest generator developed can create dialogues with NPC interactions and objective tracking. Performance optimization was a key consideration, and the system handled an increasing number of quests efficiently. Future work could include refining the complexity of quests and improving the user interface for better quest management.

Conclusion of Objective 3: Develop an Item Creator

The third objective was to implement an item creation system that supports diverse item types, equipping mechanics, and inventory management. This system needed to be scalable to accommodate a large number of items and ensure ease of maintenance. This objective was achieved, resulting in a comprehensive item creator that supports a wide variety of item types and seamless inventory management. The system's scalability was ensured through efficient data management practices. Future enhancements could involve adding more item types, improving the equipping mechanics, and refining the inventory UI for better user experience.

Conclusion of Objective 4: Develop a Combat Creator

The fourth objective was to establish a modular combat system with customizable attack and defence logic. This system was optimized for performance and designed to integrate seamlessly with the character and item systems. This objective was successfully met, with the development of a versatile combat system that allows for detailed customization of combat mechanics. The integration with the character and item systems was smooth, and performance remained high even during intensive combat scenarios. However, further work is needed to refine combat animations, enhance enemy AI, and implement more sophisticated combat strategies.

Conclusion of Objective 5: Implement a Playable RPG Prototype

The final objective was to implement a playable RPG prototype using the custom-made framework, showcasing the framework's functionality and user experience. This prototype was also intended to evaluate scalability, maintainability, and performance. This objective was achieved to a considerable extent. A functional RPG prototype was created, demonstrating the core capabilities of the developed framework. The prototype provided a good user experience but was relatively small in scale. Future work involves expanding the game's content, improving the AI behavior, and addressing minor bugs that were identified during testing.

7.2 Major Learnings from the Project

Throughout the project, several valuable lessons and skills were acquired:

- Game Development with Unreal Engine: Gained in-depth knowledge of Unreal Engine 5, including its Blueprint system and various development tools.
- Time Management: Learned the importance of effective time management to meet project milestones and deadlines.
- Logical and Problem-Solving Skills: Enhanced logical thinking and problem-solving abilities through the development and debugging of complex game mechanics.
- Productivity: Improved productivity by optimizing workflows and utilizing Unreal Engine's features efficiently.
- User Requirements Solution: Developed a keen understanding of user requirements and how to translate them into functional game features.
- Staying Updated with Technology: Recognized the importance of staying up-to-date with the latest technological advancements, particularly in game development.

7.3 Remaining Work, Potential Improvements and Future Plans

While significant progress has been made, there are still areas that require further development and enhancement:

- Implement logic to allow attack sequences to follow spam clicks smoothly.
- Add more character weapon styles to increase variety.
- Integrate new animation retargeting features in Unreal Engine 5.4 for abilities.
- Fix issues with enemy AI attacking each other and forgetting perception triggers.
- Expand the item system to include more items and display item names above boxes.

Potential Enhancements

- Introduce more abilities and skills with cooldown reduction features.
- Add a wider range of attack and skill animations.
- Implement dialogue animations for a more immersive experience.
- Increase the variety of NPCs for decoration and interaction.
- Expand the stats system to include more complex attributes like crit rate, crit damage, movement speed, and attack speed.
- Develop a more visually appealing and user-friendly interface.
- Add more item categories and introduce crafting or blacksmithing systems.
- Enhance character progression with additional upgrade options.
- Enhance the character design system with more complex abilities
- Refining the quest generator to handle more intricate quest scenarios and improve the user interface.
- Improving the combat system with better animations, advanced AI, and more combat strategies.

Future Plans for the Software Package

Looking ahead, there are plans to continue developing and refining this RPG framework. The goal is to elevate it to a professional, commercial-level product that can be sold on the Unreal Engine Marketplace. This will involve addressing the remaining issues, adding new features, and ensuring the framework meets high standards of quality and usability. As an indie developer, I plan to work on this project in my free time, with the aim of creating a comprehensive and polished RPG development tool.

Application of Program-Specific Skills and Knowledge

The successful completion of this project was supported by a combination of skills and knowledge gained from various sources:

- Official Documentation: Utilized Unreal Engine's official documentation to understand and implement key features.
- Community Forums: Engaged with community forums and online discussions to solve problems and gather insights.
- YouTube Tutorials: Followed numerous YouTube tutorials to learn best practices and advanced techniques in Unreal Engine.

In summary, this project has been a significant learning experience, providing valuable insights into game development and the use of Unreal Engine 5. The accomplishments, coupled with the identified areas for improvement, set a solid foundation for future development and professional growth in the field of game development.