Dr. Babasaheb Ambedkar Technological University
Lonere, Dist. Raigad, Pin 402103, Maharashtra

A
PROJECT REPORT ON

"GAME DEVELOPMENT USING UNREAL ENGINE"

Under the Guidance of

Prof. A. G. Kadam

Submitted by

Ravi Ramshankar Singh (CS4122)
Ganesh Tatyarao Tagad (CS4141)
Vaibhav Dadasaheb Narode (CS4163)
Chaitanya Ranjeet Pawar (CS4164)

For the partial fulfillment for the award of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

CSMSS
CHH. SHAHU COLLEGE OF ENGINEERING
Chhatrapati Sambhajinagar – 431011
(2025-26)

CSMSS

Chh. Shahu College of Engineering,

Chhatrapati Sambhajinagar, Maharashtra - 431011

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Project report entitled

"Game Develpoment using Unreal Engine"

Submitted by

Ravi Ramshankar Singh (CS4122)
Ganesh Tatyarao Tagad (CS4141)
Vaibhav Dadasaheb Narode (CS4163)
Chaitanya Ranjeet Pawar (CS4164)

in partial fulfillment for award of the Degree **Bachelor of Technology Computer Science and Engineering** of **Dr. Babasaheb Ambedkar Technological University**, Lonere, Raigad, during academic year 2025-26 Part-I.

Prof. A. G. Kadam	**Dr. S. P. Abhang**	**Dr. G. B. Dongre**
Guide	**Head of the Department**	**Principal**

CSMSS

Chh. Shahu College of Engineering,

Chhatrapati Sambhajinagar, Maharashtra - 431011

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROJECT APPROVAL SHEET

Project entitled "**GAME DEVELPOMENT USING UNREAL ENGINE**" submitted by **Ravi Ramshankar Singh (CS4122), Ganesh Tatyarao Tagad (CS4141), Vaibhav Dadasaheb Naraode (CS4163), Chaitanya Ranjeet Pawar (CS4164)** is approved for partial fulfillment for the award of **Bachelor of Technology in Computer Science and Engineering** of **Dr. Babasaheb Ambedkar Technological University**, Lonere, Raigad (M.S.) during academic year 2025-26 Part-I.

Name and Sign	**Name and Sign**
Internal Examiners	**External Examiners**

Place: Chhatrapati Sambhajinagar

Date:

DECLARATION

We, the students enrolled in the seventh semester of the B.Tech program in **Computer Science and Engineering** at CSMSS Chh. Shahu College of Engineering, Chhatrapati Sambhajinagar, hereby assert that our project work titled " **GAME DEVLPOMENT USING UNREAL ENGINE** " submitted to Dr. Babasaheb Ambedkar Technological University, Lonere, Raigad, during the academic year 2025-26, represents original research conducted by us.

This project work is presented as a partial fulfillment of the requirements for the Bachelor of Technology degree in **Computer Science and Engineering**. The findings presented in this report have not been previously submitted to any other University or Institute for the purpose of obtaining any degree.

Roll No.	Name of the student	PRN No.	Signature
CS4122	Ravi Ramshankar Singh	2225331242022	
CS4141	Ganesh Tatyarao Tagad	2225331242044	
CS4163	Vaibhav Dadasaheb Narode	2225331242068	
CS4164	Chaitanya Ranjeet Pawar	2225331242069	

Place: Chhatrapati Sambhajinagar

Date

ACKNOWLEDGEMENT

We take immense pleasure in presenting this project report, as this page provides us with the opportunity to convey our heartfelt emotions and gratitude.

We extend our sincere thanks to our guide, **Prof. A. G. Kadam** whose guidance was invaluable at every step of this project. His motivation and confidence-boosting efforts were instrumental, and we acknowledge that this work would not have been possible without his support and encouragement.

Special appreciation is also extended to the project coordinator, **Dr. G. J. Sahani** for guiding and motivating us throughout the project. We would like to express our gratitude to the Head of the Department, **Dr. S. P. Abhang** and the respected Principal, **Dr. G. B. Dongre**, for providing valuable resources and dedicating their valuable time to review our report.

Finally, we express our thanks to all the staff members of the **Computer Science and Engineering Department** and our friends, without whom the completion of this project would not have been possible.

Ravi Ramshankar Singh (CS4122)
Ganesh Tatyarao Tagad (CS4141)
Vaibhav Dadasaheb Narode (CS4163)
Chaitanya Ranjeet Pawar (CS4164)
B. Tech. (Computer Science and Engineering)

CONTENTS

List of Figures

List of Tables

List of Abbreviations

SN	Symbol	Illustrations
1	UE	Unreal Engine
2	UE5	Unreal Engine 5
3	LOD	Level of Detail
4	AI	Artificial Intelligence
5	HUD	Heads-Up Display
6	UI	User Interface
7	BT	Behavior Tree
8	BP	Blueprint
9	FAB	Fully Automatic Bind
10	NavMesh	Navigation Mesh
11	FPS	Frames Per Second

ABSTRACT

This project focuses on the development of a 3D shooting game using Unreal Engine 5.7, utilizing its advanced tools such as Blueprint visual scripting, Nanite virtualized geometry, Lumen global illumination, and built-in AI systems. The primary aim of the project is to create a fully functional and visually enhanced game without relying on traditional programming, instead using Unreal Engine's graphical workflow to design gameplay mechanics, character controls, enemy behavior, and environment interactions.

The gameplay system is built around the player character, who can move, sprint, aim, jump, and shoot using the Enhanced Input System. All gameplay logic, including weapon firing, hit detection, animation transitions, and movement states, is implemented using Blueprints, offering a fast and flexible way to develop and test mechanics. Enemy bots are created using AI Controllers, Behavior Trees, NavMesh, and Perception Components, enabling them to patrol, detect the player, chase, and attack intelligently. A boss enemy with higher aggression, extended detection range, and unique attack patterns is added to make the gameplay more challenging and engaging.

The visual quality of the game relies on Unreal Engine 5.7's rendering technologies. Nanite enables the use of high-detail models while maintaining smooth performance, and Lumen provides realistic real-time lighting and reflections that react instantly to gameplay changes. The LOD system further ensures efficient performance by reducing asset complexity at larger distances. These features combine to deliver realistic visuals and stable performance during fast-paced gameplay.

The environment is designed using modular FAB assets, proper collision setup, and lighting adjustments to create a believable and interactive world. A minimal and clear HUD shows essential information such as player health, ammunition, and cross.

Chapter I
INTRODUCTION

1.1 Introduction:

Game development has evolved rapidly over the last several years due to advancements in real-time graphics, rendering technologies, game engines, and interactive design frameworks. Modern players expect high-quality visuals, smooth performance, realistic animations, and immersive experiences. To meet these expectations, game engines have become more powerful, optimized, and user-friendly. Among all the available engines, Unreal Engine, developed by Epic Games, has emerged as one of the most dominant and technically advanced platforms for building games across PC, console, mobile, and VR.

Unreal Engine provides a complete ecosystem for creating real-time 3D worlds, interactive simulations, cinematic experiences, and visually stunning environments. The engine is known for its cutting-edge rendering capabilities, advanced physics simulation, robust animation system, and strong support for both programmers and non-programmers. One of the most unique aspects of Unreal Engine is its Blueprint Visual Scripting System, which allows developers to build gameplay logic without writing traditional code. This makes it accessible for students, beginners, and artists while still being powerful enough for professional game studios.

In this project, we developed a game using Unreal Engine Blueprints, free assets from Unreal Marketplace, default Unreal Templates, and modern UE5 technologies such as Nanite, Lumen, and Level of Detail (LOD) systems. The use of Blueprints enabled us to create gameplay logic rapidly, prototype interactions, and implement character mechanics without manually coding in C++. Free assets helped in building environments, characters, props, weapons, sounds, and animations at a professional quality level while staying within academic constraints. Unreal Engine templates provided a strong base structure for movement, input mapping, camera control, and basic gameplay flow, which we extended based on project requirements.

1.1.1 Nanite and Its Impact:

One of the revolutionary features used in our project is Nanite, a virtualized micro-polygon geometry system introduced in Unreal Engine 5. Nanite allows the engine to render extremely high-detail models without major performance loss. This eliminated the need for manually reducing polygon counts or baking simplified meshes. With Nanite, we used highly detailed environment assets, rocks, buildings, and props while maintaining high frame rates. It greatly improved the visual fidelity of the game, especially during close-up views.

1.1.2 Lumen Global Illumination:

Another breakthrough feature used in the project is Lumen, Unreal Engine 5's fully dynamic global illumination and reflections system. Instead of manually baking lighting or using static lightmaps, Lumen provided real-time lighting updates based on scene changes, movement, and object interactions. This resulted in more realistic shadows, reflections, and light bounce effects. Lumen enhanced the atmosphere, depth, and mood of the game world, giving it a modern AAA-quality visual feel.

1.1.3 Level of Detail (LOD) Optimization:

Performance optimization is a crucial part of game development. Unreal Engine supports LOD (Level of Detail) to automatically switch between high-poly and low-poly versions of models based on camera distance. In our project, the use of LOD ensured that the game ran smoothly even when many assets were present in a level. LOD helped maintain FPS, reduce memory usage, and improve rendering efficiency without sacrificing visual quality.

1.2 Problem Statement:

In modern game development, creating a complete and functional game requires understanding multiple interconnected systems such as character control, animation, user interface, input handling, abilities, and overall gameplay logic. Beginners often face challenges in learning how these systems work together within a real-time game engine. While Unreal Engine provides powerful tools, building a full game architecture from scratch can be time-consuming and complex.

To simplify this process, the Lyra Starter Game Template offers a structured and production-ready example. However, understanding, customizing, and modifying such a large and modular framework presents its own difficulties. Students must learn how to navigate game systems, integrate new features, and adapt existing ones without breaking the overall structure.

The main problem addressed in this project is to study and analyze the Lyra Starter Template, understand its internal architecture, and customize its existing systems to create a functional and improved gameplay experience. This includes identifying how different modules interact, how abilities are implemented, where animations are controlled, how UI responds to gameplay, and how character systems are extended.

This project solves the problem of understanding real-world, scalable game design by providing hands-on experience with an industry-standard framework, enabling a clearer understanding of modern gameplay development practices

1.3 Objectives of the Project:

The objectives of this project are focused on developing a complete and immersive game experience using Unreal Engine 5. The aim is to combine modern rendering technologies, Blueprint-based gameplay logic, and high-quality assets to create a polished and interactive 3D game. These objectives guide the overall direction of the project by defining the key features we want to implement, such as smooth gameplay mechanics, intelligent enemy behavior, and visually detailed environments. Through these objectives, the project ensures a clear development strategy and helps in achieving a final game that is both engaging and technically strong.

1.3.1 High-Quality Game:

One objective of the project is to create a visually high-quality game that makes full use of Unreal Engine's advanced features such as Nanite for detailed models and Lumen for dynamic lighting. The aim is to design an environment that looks realistic, smooth, and modern while maintaining good performance. By using optimized assets and Level of

Detail (LOD), the project focuses on delivering a polished gaming experience with stable frame rates and professional visual effects.

1.3.2 3D Shooting Game:

Another objective is to develop a complete 3D shooting game where the player can move freely, aim accurately, and interact with the environment. The game includes shooting mechanics, weapon handling, character animations, and responsive controls built using Blueprint scripting. The goal is to create smooth gameplay where the player can explore the level, engage with enemies, and experience a fully functional third-person or first-person shooting system.

1.3.3 AI Bot Players:

A key objective of the project is to implement AI bot players that can navigate the level, detect the player, and respond intelligently. These AI bots should be able to patrol, chase, attack, take damage, and adapt to the player's actions. Using Unreal Engine's Behavior Trees and AI Controllers, the project aims to create opponents that make the game more challenging and realistic. This enhances gameplay depth and makes the experience more engaging for players.

1.3.4 Boss Mode:

The project also includes the creation of a Boss Mode, where the player faces a stronger and more advanced enemy. The boss will have higher health, unique abilities, stronger attacks, and more complex behavior compared to normal AI bots. This objective focuses on designing a special battle scenario that tests player skill and adds excitement to the game. The Boss Mode serves as a major milestone in gameplay progression and provides a memorable challenge at the end of the level

Chapter II

LITERATURE SURVEY

2.1 Literature Survey:

Game development has evolved rapidly over the last decade due to major advancements in real-time rendering, artificial intelligence, physics simulation, and content-creation workflows. Modern video games are no longer built from scratch; instead, they use powerful game engines that provide pre-built systems for graphics, animation, sound, input handling, and optimization. These engines allow developers to focus on gameplay design and creativity rather than low-level technical implementation. As a result, game engines have become one of the most important tools in both the gaming industry and academic research.

In recent years, 3D action and shooting games have gained enormous popularity because of their immersive environments, realistic movements, and interactive combat systems. To create such experiences, developers require advanced rendering technologies, efficient AI behavior models, and optimized asset pipelines. Studies highlight that the combination of physics, animation blending, real-time lighting, and AI decision-making forms the foundation of any successful 3D shooting game. With the availability of high-performance GPUs and real-time rendering engines, developers can now achieve movie-level visual quality inside a playable game environment.

Another important development in modern game creation is the rise of visual scripting systems. Traditional game programming often required strong coding knowledge, but with tools like Unreal Engine’s Blueprint system, developers can design complex gameplay logic using node-based visual scripts. Research shows that this approach significantly reduces development time and is especially beneficial for student projects, rapid prototyping, and beginners entering the gaming field. Blueprint visual scripting has been adopted widely in academic game development projects due to its flexibility, readability, and ease of debugging.

Artificial Intelligence is also a major area of focus in current game development studies. Researchers have explored techniques such as Behavior Trees, Finite State Machines, and Utility AI to create intelligent enemy characters in FPS games. AI must be able to detect players, navigate the environment, decide when to attack, and respond to combat situations realistically. Academic sources consistently highlight that strong AI design improves both the challenge level and the overall player experience. These findings are highly relevant to 3D shooting games like the one developed in this project.

Real-time rendering technologies have also advanced significantly. Recent innovations such as virtualized geometry, dynamic global illumination, real-time reflections, and physically based lighting have transformed how games display environments. Unreal Engine's Nanite technology allows extremely high-detail models to render smoothly, while Lumen provides global illumination and real-time reflection updates without pre-baking. This combination delivers highly realistic scenes suitable for fast-paced shooting games. Studies confirm that such rendering systems help maintain stable performance while improving visual quality.

In summary, the literature shows that modern game development relies heavily on three major components: powerful game engines, efficient AI systems, and advanced real-time rendering technologies. Unreal Engine 5 brings these components together through its Blueprint workflow, Behavior Tree system, Nanite geometry pipeline, Lumen lighting model, and optimized performance tools. These technologies form the foundational background for the 3D shooting game developed in this project.

2.1.1 Game Engines and Modern Development Trends:

Game engines play an essential role in modern game development by providing ready-made tools for graphics, animation, physics, sound, AI, and optimization. Instead of writing all systems from scratch, developers can build games faster and more efficiently using these engines. Popular engines like Unreal Engine, Unity, CryEngine, and Godot support real-time rendering, multi-platform deployment, and flexible workflows suitable for both beginners and professionals.

Recent trends in game development show a strong shift toward high-quality graphics, realistic lighting, and large open environments. Modern engines now include advanced features such as real-time global illumination, virtualized geometry, dynamic weather, improved physics, and visual scripting. These trends allow developers to create more immersive and technically advanced games while reducing development time.

2.1.2 Evolution of Unreal Engine:

Unreal Engine has evolved consistently over the years, starting from a basic 3D engine in the late 1990s to one of the most powerful real-time engines available today. Early versions focused mainly on rendering and level design, while later versions introduced advanced materials, skeletal animation, physics engines, and improved visual quality. Unreal Engine 4 marked a major shift by introducing Blueprint visual scripting, physically based rendering, improved animations, and better performance tools.

Unreal Engine 5 further transformed game development with new technologies like Nanite virtualized geometry, Lumen dynamic lighting, MetaSounds, World Partition, and an updated editor. These improvements make it easier to create realistic worlds with high detail while maintaining stable performance. Unreal Engine’s evolution reflects its strong position as an industry-leading engine for high-quality games, simulations, virtual production, and academic projects.

2.1.3 Blueprint Visual Scripting:

Blueprint visual scripting is a node-based system in Unreal Engine that allows developers to build gameplay logic without writing code. It provides a clear and visual way to connect functions, events, and variables, making it easy to create mechanics such as player movement, shooting, AI behavior, UI interactions, and environmental actions. Blueprint reduces the learning curve for beginners and helps teams work faster during prototyping and testing. It is widely used in academic projects because it is powerful, easy to understand, and extremely efficient for rapid development.

2.1.4 Rendering Technologies: Nanite, Lumen, and LOD:

Modern game rendering focuses on achieving realistic visuals while maintaining high performance. Nanite virtualized geometry allows artists to use film-quality 3D models directly in the game without manually creating optimized low-poly versions. This results in detailed environments and smooth surfaces even at close range. Lumen provides dynamic global illumination, meaning light, shadows, and reflections update instantly based on scene changes, giving a realistic and natural look without any pre-baking.

The Level of Detail (LOD) system automatically adjusts the complexity of objects based on their distance from the camera. This ensures that performance remains stable even in large or detailed environments. Together, Nanite, Lumen, and LOD significantly improve visual quality and optimization in Unreal Engine, making them ideal for fast-paced and visually rich shooting games.

2.1.5 AI Techniques Used in Shooting Games:

Artificial Intelligence is a core component of shooting games because it controls how enemies behave, react, and interact with the player. Common techniques include Behavior Trees, which organize decision-making, and Finite State Machines, which represent states such as patrol, chase, attack, or search. Enemy AI also uses perception systems to detect the player based on sight, sound, and distance. Navigation Meshes help AI characters move smoothly through the environment while avoiding obstacles.

Research shows that well-designed AI makes gameplay more engaging and challenging by creating dynamic and unpredictable enemy behavior. Advanced AI can choose strategies, use cover, follow paths, and adjust difficulty based on the player's actions. These techniques are widely used in FPS games and form the foundation for enemy bots, boss characters, and multiplayer AI opponents in modern game development.

2.2 Related Work:

Research in the field of game development has grown significantly with the increasing use of advanced game engines such as Unreal Engine. Many academic studies and projects have explored how real-time rendering, AI-based enemy behavior, and interactive gameplay mechanics can be implemented efficiently using modern tools. Several researchers have focused on creating first-person and third-person shooter games to analyze player movement, aiming accuracy, combat response, and environment interaction. These studies demonstrate that shooting games require strong coordination between input systems, animation logic, AI decision-making, and optimized rendering pipelines.

Previous works also highlight the importance of visual scripting tools like Unreal Engine's Blueprint system, which allow rapid development of prototype gameplay features without deep coding knowledge. Many university-level projects have successfully used Blueprints to design player movement, shooting mechanics, weapon systems, enemy reactions, and UI behavior. Research further shows that the combination of Blueprints and Behavior Trees provides an efficient workflow for implementing AI logic in shooting games, enabling enemies to patrol, detect players, chase, and attack based on dynamic conditions.

In addition to gameplay and AI, several studies have examined the impact of advanced rendering technologies on game performance. Recent research emphasizes the use of real-time global illumination, mesh streaming, and Level of Detail systems to improve both visual fidelity and frame stability. Techniques like virtualized geometry, dynamic lighting, and optimized materials have been widely explored to enhance the visual quality of shooting environments while maintaining smooth and responsive gameplay. These findings support the use of Nanite and Lumen in modern Unreal Engine projects, especially for high-quality 3D action games.

2.2.1 Work on Unreal Engine Game Development:

Research on Unreal Engine development highlights its strong capabilities for creating high-quality 3D games with realistic environments and smooth gameplay. Many academic and professional projects use Unreal Engine because it provides powerful real-time rendering, a flexible visual scripting system, and a large library of ready-made assets. Studies show that Blueprint scripting reduces development complexity and makes rapid prototyping easier for students and beginners. Research also emphasizes the impact of advanced UE5 features such as Nanite, Lumen, and optimized LOD systems in achieving high visual fidelity while maintaining stable performance. These findings directly align with the tools and features used in this project, making Unreal Engine an ideal choice for developing a 3D shooting game.

2.2.2 Studies on Blueprint Visual Scripting:

Blueprint visual scripting has been widely adopted in academic and professional game development because it allows developers to implement gameplay mechanics without writing complex code. Research shows that Blueprint helps students and beginners build interactive systems quickly by using node-based functions for inputs, character behaviors, and event handling. Many studies highlight that Blueprint offers a clear visual representation of gameplay logic, making it easier to design and debug features such as movement, shooting, interaction, and animation transitions. This visual workflow speeds up experimentation and reduces development time, which is why it is frequently used in prototypes and academic projects.

Existing work also demonstrates that Blueprint is powerful enough to build complete gameplay systems in 3D action and shooting games. Researchers often use it to develop combat mechanics, AI communication events, HUD interactions, and weapon systems. Due to its flexibility and tight integration with Unreal Engine's visual tools, Blueprint-based development has become a dependable method for implementing core gameplay features in student projects. These studies support the use of Blueprints in this project, where player movement, shooting logic, AI triggers, and environment interactions were all created using visual scripting.

2.2.3 Work on Real-Time Rendering and Lighting;

Recent research on real-time rendering highlights the major advancements introduced in modern game engines, especially with the launch of Unreal Engine 5. Studies often focus on how technologies like Nanite virtualized geometry allow developers to use ultra-high detail models without manually creating low-poly versions. This significantly reduces asset creation time while maintaining strong performance. Research also discusses the improvements in dynamic lighting and reflections, demonstrating how real-time global illumination creates more natural and visually appealing game environments. These advancements make modern engines capable of producing film-like visuals while still supporting interactive gameplay.

Further studies show that lighting accuracy plays a key role in creating immersive 3D shooting environments. Unreal Engine's Lumen system has been widely analyzed for its ability to generate realistic light bounce, soft shadows, and dynamic reflections without pre-baking. Researchers also highlight the importance of Level of Detail (LOD) systems for maintaining stable frame rates in large game worlds. LOD ensures that distant objects are simplified automatically, reducing GPU load without affecting perceived quality. Together, these technologies create a balance between high visual fidelity and performance, supporting the development of smooth and responsive gameplay such as the shooting mechanics used in this project.

2.2.4 Work on Player Controls and Gameplay Mechanics:

Research on player control systems emphasizes the importance of responsiveness, accuracy, and smooth input handling, especially in fast-paced shooting games. Studies show that modern engines like Unreal Engine provide frameworks that help developers implement reliable movement, aiming, weapon handling, and interaction mechanics. Researchers highlight that player responsiveness directly affects gameplay quality, immersion, and difficulty balancing. Various publications discuss the importance of input mapping, animation blending, and character state management for ensuring that movement and actions feel natural and predictable to the player.

Further studies explore how gameplay mechanics influence overall game experience, including shooting accuracy, recoil patterns, enemy reaction timing, and combat flow. Research demonstrates that systems such as Enhanced Input, animation graphs, and physics-based interactions significantly enhance the realism of player actions. Many academic projects use Unreal Engine’s Blueprint system to prototype these mechanics quickly and test combat scenarios efficiently. These findings support the choices made in this project, where player movement, shooting behavior, and interaction systems are all designed using Blueprint logic and Unreal Engine’s built-in tools to create a smooth and engaging gameplay experience.

Chapter III
SYSTEM MODELLING

3.1 Modelling Diagrams:

This section presents the different modelling diagrams used to represent the structure, behaviour, and functional flow of the 3D shooting game developed in Unreal Engine 5.7. These diagrams help visualize how the system operates, how different components interact, and how the player and AI communicate with the environment. Each model provides a specific viewpoint of the system—ranging from high-level functional requirements to internal architecture and detailed behavioural flows. The following diagrams collectively offer a clear and organized understanding of the game's overall design and working process.

3.1.1 Use-Case Diagram:

The Use-Case Diagram provides a high-level functional overview of the 3D shooting game developed in Unreal Engine. It represents how different actors interact with the game system and what core actions the system allows. The primary actor in this project is the Player, who performs gameplay actions such as moving, aiming, shooting, reloading weapons, interacting with objects, and monitoring the HUD for essential game information. Additionally, the AI System acts as another actor responsible for performing enemy actions such as patrolling, detecting the player, chasing, and attacking during combat.

This diagram helps in understanding the overall functionality before going into the internal architecture of the game. It shows how both the Player and the AI communicate with the game environment through different use-cases. By presenting the system at this functional level, the use-case model ensures clear identification of all major gameplay requirements and provides a foundation for the detailed system architecture and flow diagrams that follow.

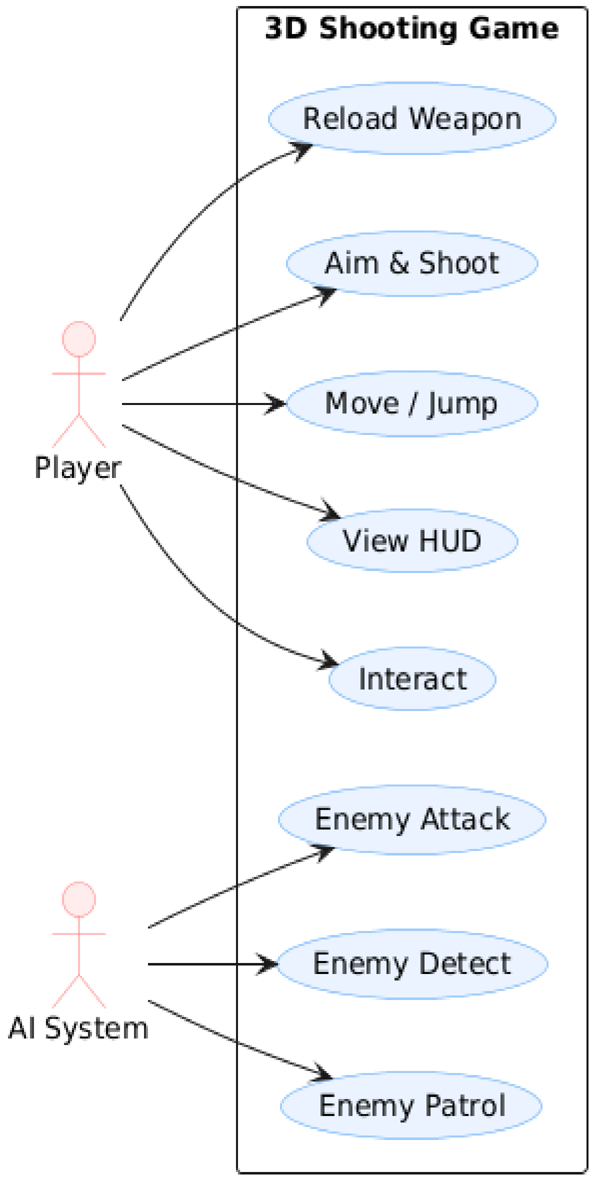


Figure 3.1.1: Use-Case Diagram

Figure 3.1.1 shows the Use-Case Diagram of the 3D shooting game, outlining how the Player interacts with core actions like movement, shooting, and reloading, while the AI System handles enemy behaviors such as patrol, detection, and attack. It provides a quick overview of the main gameplay functions.

3.1.2 System Architecture Diagram:

The System Architecture of the 3D shooting game is designed using a modular structure to ensure smooth interaction between all major gameplay systems. Each module is separated according to its functionality but remains fully connected through Unreal Engine's Blueprint communication. This modular structure helps in managing gameplay logic, rendering processes, environment data, and AI behavior while maintaining clarity and performance throughout the project.

The Player Character System forms the core interactive layer of the architecture. It handles essential player mechanics such as movement, aiming, shooting, jumping, and camera control. The Enhanced Input System converts user inputs into actions, while the Blueprint logic processes these actions and triggers the appropriate responses. The Animation Blueprint ensures every movement and action transitions smoothly, providing realistic and responsive gameplay.

The AI System manages all enemy behavior using multiple components such as the AI Controller, Behavior Tree, Perception System, and Navigation Mesh. These elements work together to allow enemies to patrol, detect the player, follow navigation paths, chase targets, and perform attack sequences. By separating AI logic from player logic, the system ensures that enemy behavior remains stable, organized, and independent from direct gameplay input.

The architecture also includes major subsystems such as the Rendering System, Environment System, and UI/HUD System. Nanite, Lumen, and LOD are responsible for visual quality and performance. Level design, assets, lighting, and collision setup are controlled by the Environment System. The UI/HUD System displays dynamic information such as health, ammo, and enemy indicators. All these modules communicate seamlessly to produce a smooth, optimized, and visually rich gaming experience.

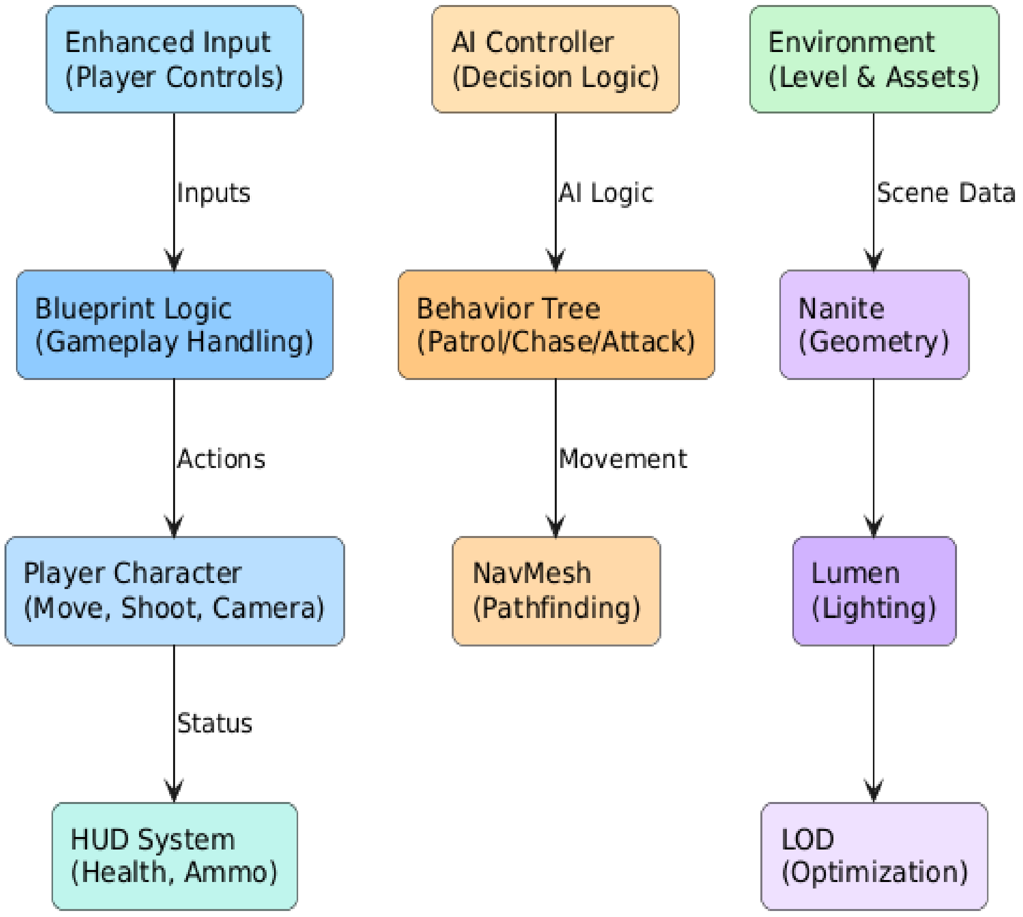

Enhanced Input (Player Controls)
Inputs
Blueprint Logic (Gameplay Handling)
Actions
Player Character (Move, Shoot, Camera)
Status
HUD System (Health, Ammo)
AI Controller (Decision Logic)
AI Logic
Behavior Tree (Patrol/Chase/Attack)
Movement
NavMesh (Pathfinding)
Environment (Level & Assets)
Scene Data
Nanite (Geometry)
Lumen (Lighting)
LOD (Optimization)

Figure 3.1.2: System Architecture Diagram

Figure 3.1.1 shows the System Architecture of the 3D shooting game, illustrating how the Player System, AI System, Rendering System, Environment, and HUD components interact with each other to manage gameplay, visuals, and game logic.enables AI to patrol, detect the player, chase, and attack intelligently. The diagram below shows the basic flow of the AI components.

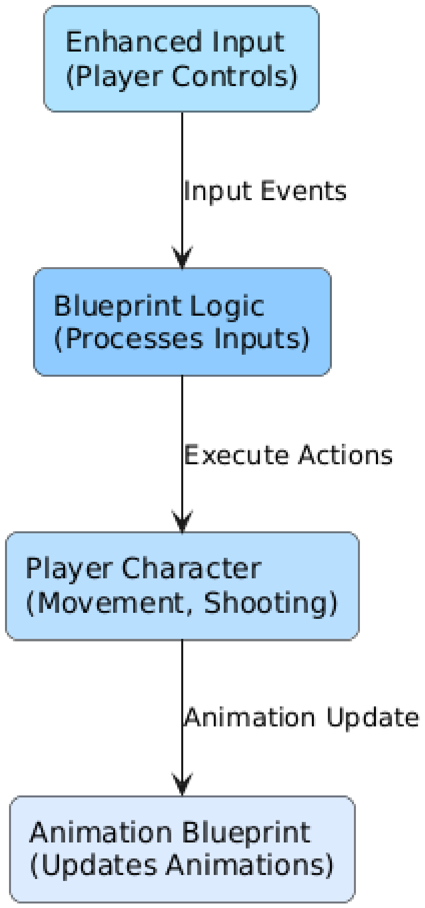
3.1.3 Player Input Flow Diagram:

The Player Input Flow Diagram explains how user actions such as movement, aiming, shooting, and jumping are captured and processed inside the game. When the player interacts using the keyboard or mouse, the Enhanced Input System of Unreal Engine converts these raw inputs into structured gameplay events. These events are then interpreted by Blueprint logic, where different gameplay actions like firing a weapon, switching animations, moving the character, or adjusting the camera are executed. This ensures that all user controls are responsive and accurately mapped to real-time gameplay behavior.

After the inputs are processed, the Player Character system updates animations, movement states, and gameplay responses. The Animation Blueprint synchronizes character actions such as running, crouching, firing, and reloading to ensure smooth visual transitions. This flow creates a seamless connection between the player’s controls and the in-game actions, resulting in a fluid and immersive gameplay experience. The diagram visually represents this entire chain—from input detection to character response—making it easier to understand how player interaction drives the core gameplay.

Key Components of the Player Input System:

- **Enhanced Input Mapping:** Converts keyboard/mouse actions into game events.
- **Input Actions:** Defines specific actions like jump, shoot, sprint, or reload.
- **Blueprint Event Graph:** Processes input events and triggers gameplay logic.
- **Animation Blueprint:** Updates movement and shooting animations based on input.
- **Camera Controller:** Adjusts the player’s viewpoint according to mouse movement.



3.1.3 Player Input Flow Diagram

Figure 3.1.3 illustrates how player actions are captured and processed inside the game. The Enhanced Input System receives keyboard and mouse inputs and sends them to the Blueprint logic, which interprets the commands and applies the corresponding gameplay actions. The Player Character then executes movement or shooting responses, while the Animation Blueprint updates the character's visual states to ensure smooth and realistic transitions.

3.1.4 AI Behavior Tree Diagram:

The AI Behavior Tree Diagram explains how enemy bots in the 3D shooting game make decisions during gameplay. The Behavior Tree controls enemy actions such as patrolling, detecting the player, chasing, attacking, and resetting their state. Each task in the tree is processed step-by-step to ensure that AI responds intelligently and consistently based on player actions and environmental conditions.

The Behavior Tree works together with the AI Controller and Perception System to create dynamic enemy behavior. The Perception System detects the player's presence, the AI Controller decides the appropriate response, and the Behavior Tree executes the logic flow. This structure ensures that enemy bots behave realistically, transition smoothly between states, and provide challenging gameplay interactions. The diagram shows the complete decision-making flow used by the AI system.

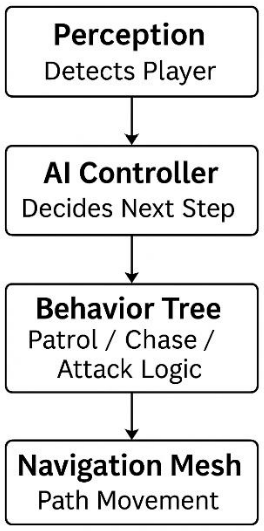


Figure 3.1.4(a): AI Behavior Mini Flow Diagram

This figure shows the simplified decision flow of the enemy AI, including how the system detects the player, selects an action through the AI Controller, executes logic using the Behavior Tree, and navigates the level using the Navigation Mesh.

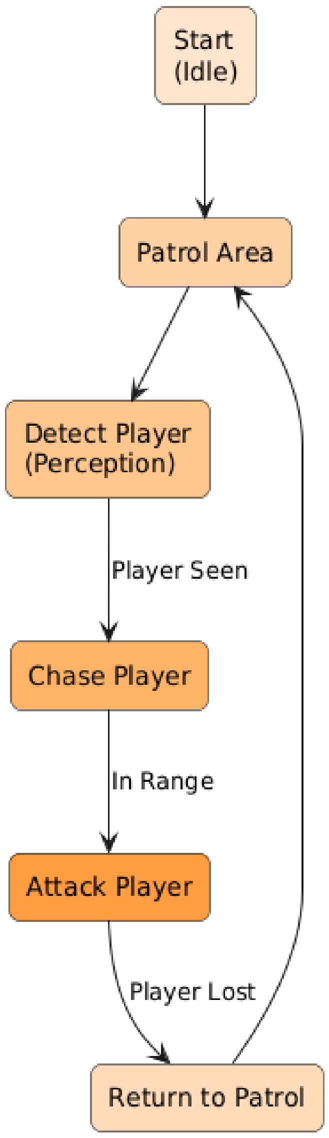


Figure 3.1.4 AI Behavior Tree Diagram

Figure 3.1.4 shows the AI Behavior Tree used to control enemy decision-making, illustrating how enemies patrol, detect the player, chase, and attack based on in-game conditions.

3.1.5 Rendering Pipeline Diagram:

The Rendering Pipeline Diagram illustrates the complete visual processing workflow of the 3D shooting game developed in Unreal Engine 5.7. This pipeline represents how raw level assets, high-detail geometry, lighting information, and optimization systems work together to generate the final on-screen visuals. The process begins with the engine receiving detailed meshes and environmental data from the scene, after which the rendering system prepares these assets for further processing. Unreal Engine’s modern rendering technologies ensure that each stage of the pipeline contributes to realistic visuals, smooth gameplay, and efficient performance even in large and complex levels.

Nanite, Unreal Engine’s virtualized geometry technology, handles high-poly meshes with exceptional efficiency. It eliminates the need for manual optimization by automatically streaming only the required levels of geometric detail. This allows the game to include highly detailed environments, realistic props, and dense structures without causing frame rate drops. Once Nanite processes the geometry, the data is passed to Lumen, the real-time global illumination and lighting system. Lumen calculates accurate light bounces, reflections, shadows, and ambient lighting, responding instantly to every change in the scene—whether it is player movement, enemy activity, dynamic lights, or environmental interaction. These lighting adjustments significantly enhance visual realism and immersion.

After the lighting stage is complete, the Level of Detail (LOD) system performs additional optimization to maintain smooth performance. LOD automatically reduces the polygon count of distant objects, ensuring that only nearby elements are rendered with full detail. This technique prevents unnecessary GPU load while preserving visual quality for the player. The optimized geometry, lighting data, and rendering passes are then combined into the final frame output, which is delivered to the screen. This complete rendering pipeline ensures that the game maintains high graphical fidelity, stable performance, and visually impressive scenes throughout all gameplay moments.

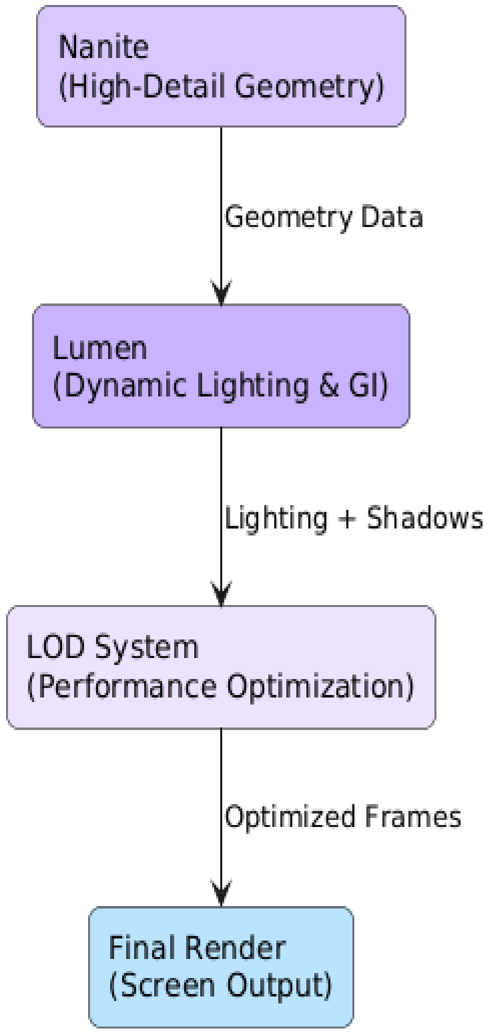


Figure 3.1.5 Rendering Pipeline Diagram

Figure 3.1.5 shows the Rendering Pipeline, highlighting how Nanite, Lumen, and LOD work together to produce high-quality and optimized visuals in the game. These systems collectively ensure that the environment, lighting, and performance remain stable even during fast-paced gameplay.

3.1.6 Level / Environment Flow Diagram:

The Level / Environment Flow Diagram explains how Unreal Engine 5.7 loads and manages the game world during gameplay. When the level starts, the engine initializes terrain, buildings, props, FAB assets, collisions, and lighting needed for the scene. As the player moves, Unreal's level streaming system loads new areas and unloads unused parts to maintain smooth performance without visible loading screens. This ensures the game world feels continuous and responsive.

Once the environment is loaded, it interacts with several gameplay systems. Collision components define how the player and AI interact with walls and objects, while the Navigation Mesh guides AI movement across walkable areas. Dynamic lighting from Lumen updates instantly based on environmental changes, and Nanite provides high-detail environmental meshes. Together, these elements create an immersive and optimized game world where visual quality and performance stay balanced.

Key Components of the Environment System:

- Level Streaming: Loads only necessary portions of the map to reduce memory usage.
- FAB Assets: High-quality prebuilt assets used for designing buildings, props, and environmental structures.
- Collision Detection: Ensures realistic interaction between players, enemies, and the environment.
- Navigation Mesh (NavMesh): Guides AI movement across the level.
- Dynamic Lighting: Updates based on player and enemy movement using Lumen.
- Nanite Geometry: Provides detailed environmental models with no performance loss.
- LOD Optimization: Reduces detail of distant objects to maintain smooth performance.

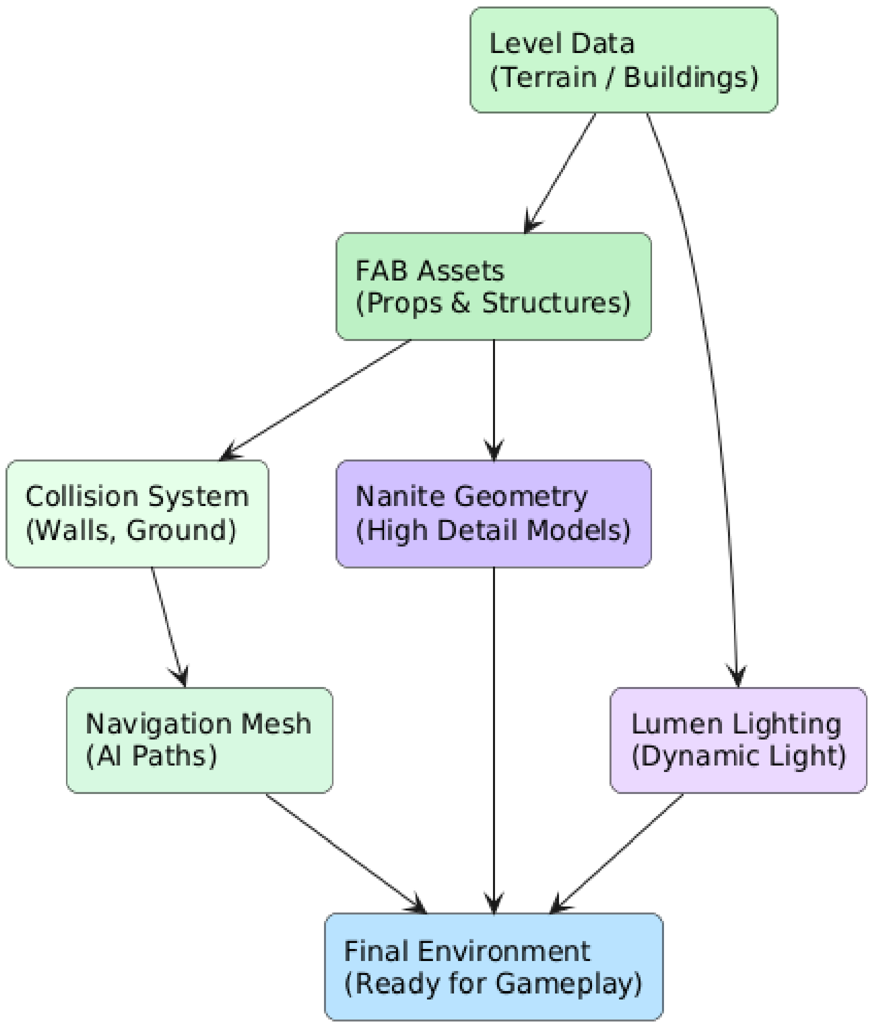


Figure 3.1.6 Level / Environment Flow Diagram

Figure 3.1.6 shows the Level / Environment Flow, explaining how terrain, buildings, FAB assets, collisions, lighting, navigation, and geometry data are processed together to generate the final playable environment in the game. The flow illustrates how each environmental component contributes to world-building and ensures smooth interaction during gameplay.

3.2 Model Development:

The Model Development phase involves transforming the conceptual design of the game into a functional and interactive system using the tools and technologies provided by Unreal Engine 5.7. In this phase, all the core gameplay elements—including the player character, input handling, environment layout, enemy AI, visual rendering, and UI—are implemented step by step. Unreal Engine's Blueprint system was used extensively to build the behaviors and mechanics of the game without writing complex C++ code. This made the development process more visual, modular, and beginner-friendly while still supporting advanced functionality needed for a 3D shooting game.

The development process began with designing the Player Character Model, where movement, camera handling, shooting mechanics, and jumping actions were created. Enhanced Input Mapping was configured to link keyboard and mouse inputs with gameplay actions. Blueprint Event Graphs were then used to script the shooting logic, projectile spawning, recoil, weapon animations, and hit detection. The Animation Blueprint was responsible for blending idle, walk, run, jump, and shooting animations to create smooth transitions that match the player's actions.

Next, the AI Model was developed using Unreal Engine's AI Controller, Behavior Tree, Perception System, and Navigation Mesh. The Behavior Tree defined enemy behavior such as patrol, chase, search, and attack. The Perception System enabled enemies to detect the player based on sight and distance. Navigation Mesh allowed enemies to move intelligently within the map, selecting paths automatically while avoiding obstacles. Together, these systems created smart and responsive enemy bots that react dynamically to the player's actions and game environment.

The Environment Model was created using FAB assets, high-detail Nanite meshes, and prebuilt level structures. The world layout included buildings, obstacles, cover points, floors, and pathways that support both player movement and AI navigation. Collision components were added to every object to ensure smooth interaction with the character and

weapons. Lumen was used to enhance the environmental lighting, producing realistic shadows, reflections, and ambient color changes as the player moves through the level.
The Rendering Model forms another essential part of the development. Nanite was used for detailed geometry, allowing the use of complex assets without causing performance drops. Lumen was configured to generate real-time global illumination and reflections. Level of Detail (LOD) settings were applied to optimize performance by reducing mesh complexity when objects are far away. These rendering techniques ensured that the game maintained high visual quality while running efficiently.

Finally, the UI Model was created using Widget Blueprints. The HUD includes elements such as the player's health bar, ammo count, crosshair, and enemy indicators. The UI updates dynamically based on gameplay actions—for example, ammo reduces when the gun is fired, and health decreases when the player takes damage. This makes the game more interactive and informative for the player.

Main Components Developed:

- Player Model (movement, shooting, camera, animations)
- AI Model (patrol, chase, attack behavior)
- Environment Model (level layout, collisions, FAB assets)
- Rendering Model (Nanite, Lumen, LOD setup)
- UI Model (health, ammo, crosshair, enemy indicators)

3.2.1 Player Model Development:

The Player Model Development focuses on creating all the core mechanics that allow the player to move, shoot, jump, aim, and interact with the game world. This part of development is crucial because the player character acts as the primary control element of the entire gameplay experience. Using Unreal Engine 5.7, the player model was created by combining Enhanced Input, Blueprint logic, Animation Blueprint, collision components, and camera controls. This modular setup ensures that the player reacts smoothly to input and performs all actions without lag or inconsistent behavior.

The first phase involved implementing the Enhanced Input System, which maps keyboard and mouse actions to specific gameplay functions. Movement keys (W, A, S, D), mouse input for camera rotation, left-click for shooting, and space bar for jumping were connected through Input Actions. These inputs are processed inside Blueprint Event Graphs, where each event triggers the appropriate gameplay response. Shooting logic was developed using projectile blueprints and line traces for hit detection, ensuring bullets register accurately on enemies and environmental objects.

The next step was constructing the Player Character Blueprint. This blueprint handles all essential player mechanics such as movement speed, sprinting, jumping height, crouching, recoil system, camera positioning, and weapon firing. Logic for animations, weapon firing intervals, projectile spawning, and camera shake was implemented here to maintain smooth transitions and natural feedback. The camera component was positioned and adjusted to give the player a clear third-person or first-person perspective based on the chosen template.

To ensure smooth visual transitions, an Animation Blueprint was integrated with the player model. This system blends animations such as idle, walk, run, jump, aim, recoil, and shooting based on player input. Each movement state triggers a different animation, and Unreal's State Machine ensures that transitions between these states occur fluidly. This creates a polished gameplay experience where every input feels responsive and realistic.

Finally, the player model was connected to the environment through collision capsules and physics settings. This ensures the player interacts correctly with floors, walls, obstacles, and enemy objects. The final implementation created a fully functional, responsive, and visually polished player character capable of performing all core actions required in a 3D shooting game.

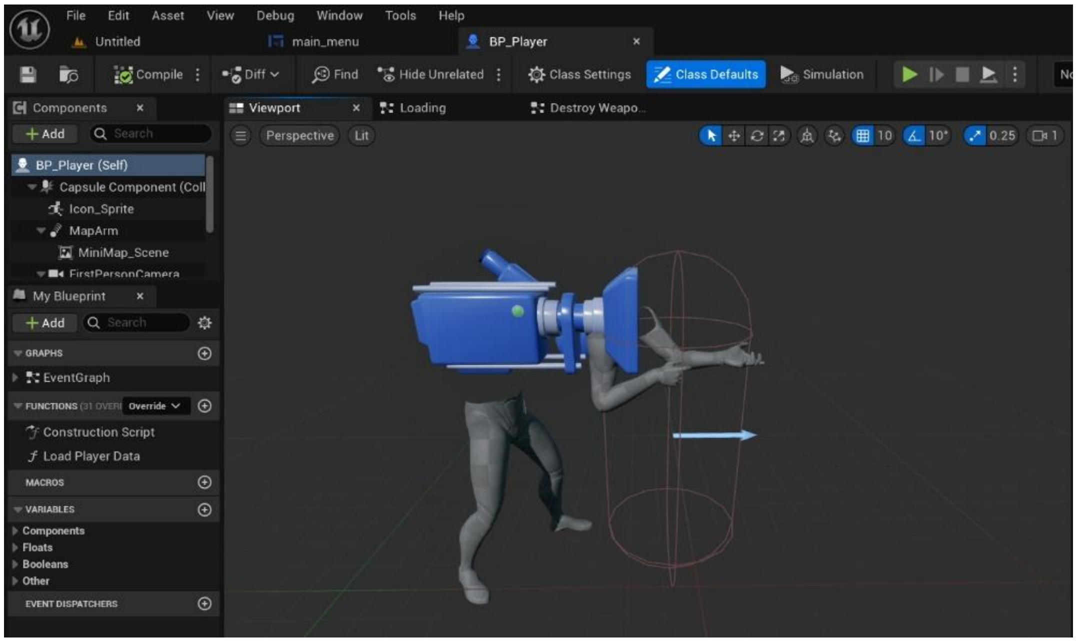


Figure 3.2.1 Character Model

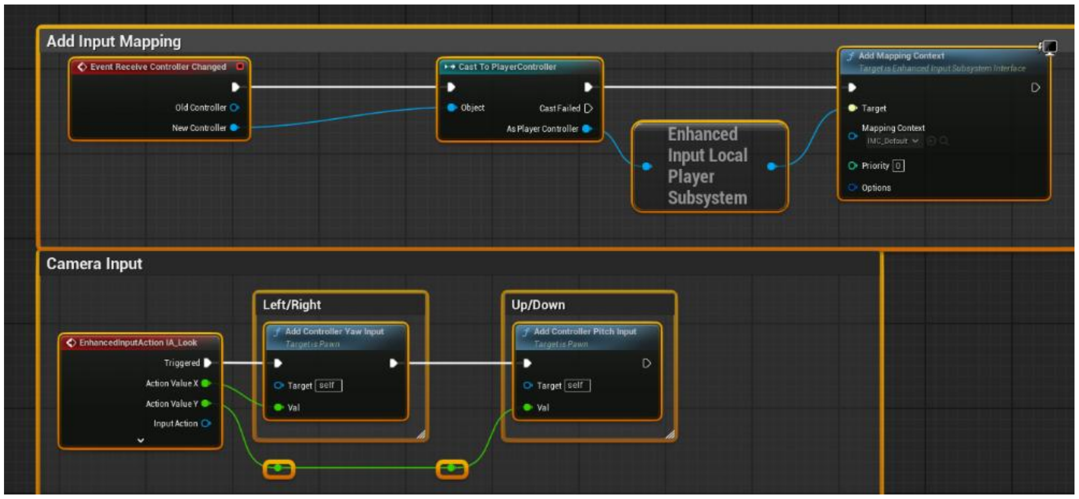


Figure 3.2.2 (a) Player Movement Blueprint

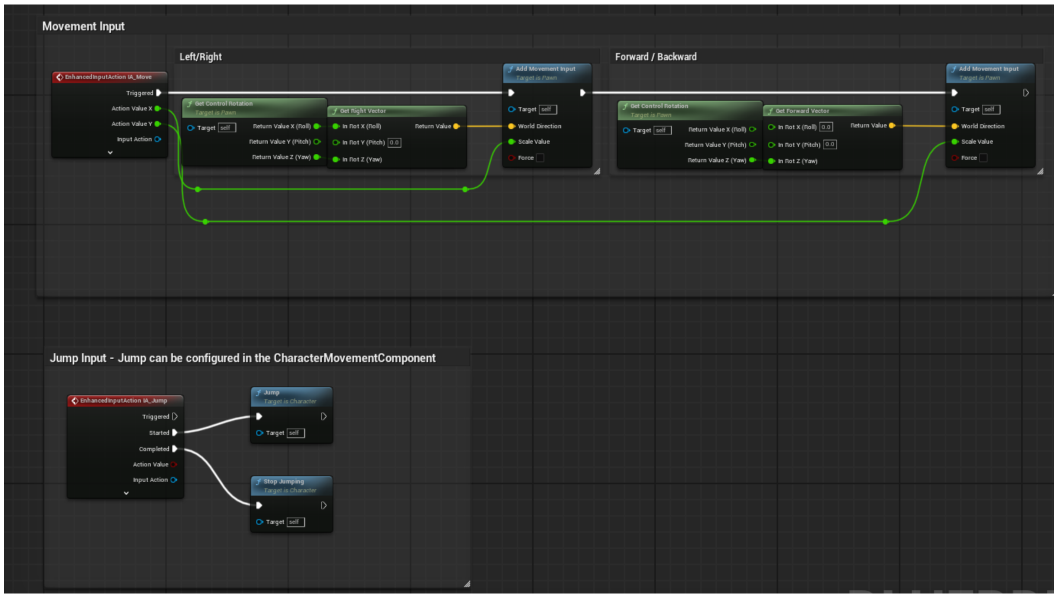


Figure 3.2.2 (b) Player Movement Blueprint

3.2.2 AI Model Development:

The AI Model Development focuses on creating intelligent and dynamic enemy behavior that reacts to the player's actions in real time. In a 3D shooting game, enemy AI is one of the most important elements because it directly affects the difficulty, excitement, and replay value of the gameplay. Unreal Engine 5.7 provides a powerful AI framework that includes the AI Controller, Behavior Tree, Perception System, and Navigation Mesh, which together form the backbone of enemy logic. These components were used to develop enemies that can patrol, detect the player, chase targets, attack intelligently, and return to their original state when the player is lost from view.

The development process began with the AI Controller, which acts as the brain of the enemy. It handles decisions such as when the enemy should patrol, when it should follow the player, and when to engage in combat. This controller is linked with the Behavior Tree, which organizes the logic into a structured flow. The Behavior Tree was designed with different tasks and states, including idle, patrol, detect, chase, attack, investigate, and reset. These tasks execute step-by-step based on game conditions, making enemy behavior smooth and natural during gameplay.

To enable enemies to sense the player, Unreal Engine’s AI Perception System was used. The sight sense was configured so that enemies could detect the player within a specific radius and field of view. When the player enters this detection range, the perception system sends an alert to the AI Controller, triggering a transition from patrol state to chase or attack mode. This makes enemy reactions immediate and context-aware, improving gameplay intensity.

For realistic movement, the game environment was processed using a Navigation Mesh (NavMesh). This system automatically calculates walkable surfaces in the level and allows AI characters to find the best path to reach the player. Thanks to NavMesh, enemies can navigate around obstacles, move through complex structures, and reposition themselves during combat. Additional movement logic, such as speed variation, stopping distance, and attacking range, was fine-tuned to make enemy behavior more believable.

The combat behavior was developed using Blueprint logic integrated with the Behavior Tree. Enemies can fire at the player, deal damage, follow attack cooldowns, and switch between aggressive and defensive states depending on distance and visibility. When the player hides or escapes detection, the AI searches nearby areas and eventually returns to patrol mode. This complete implementation results in enemies that behave intelligently and significantly enhance the overall gameplay experience.

Core Components of the AI Model:

- AI Controller (decision making)
- Behavior Tree (logic structure)
- Perception System (player detection)
- Navigation Mesh (pathfinding)
- Combat Blueprint Logic
- Patrol and Chase Behavior

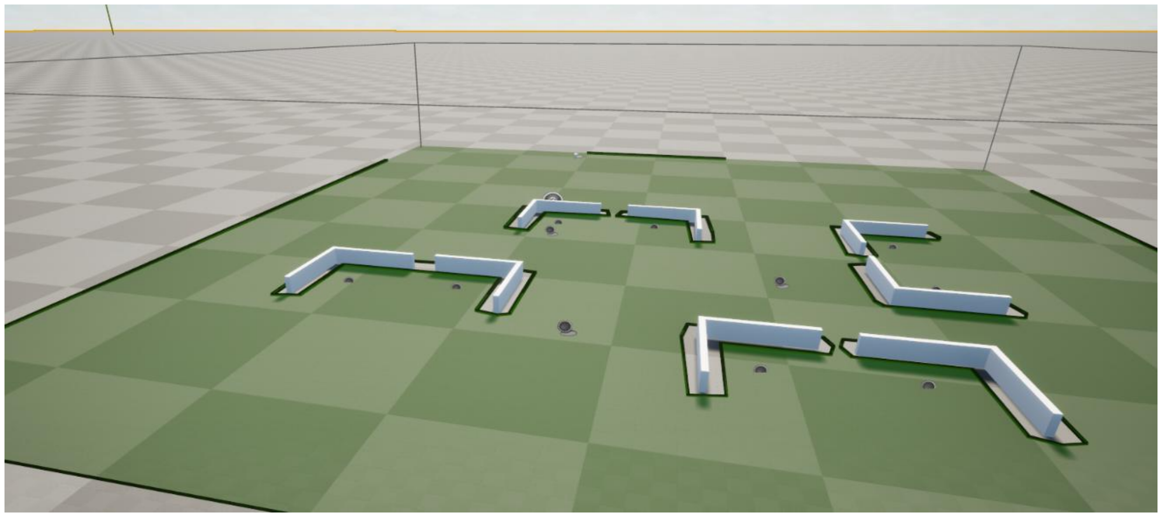


Figure 3.2.3 AI Spawn Area

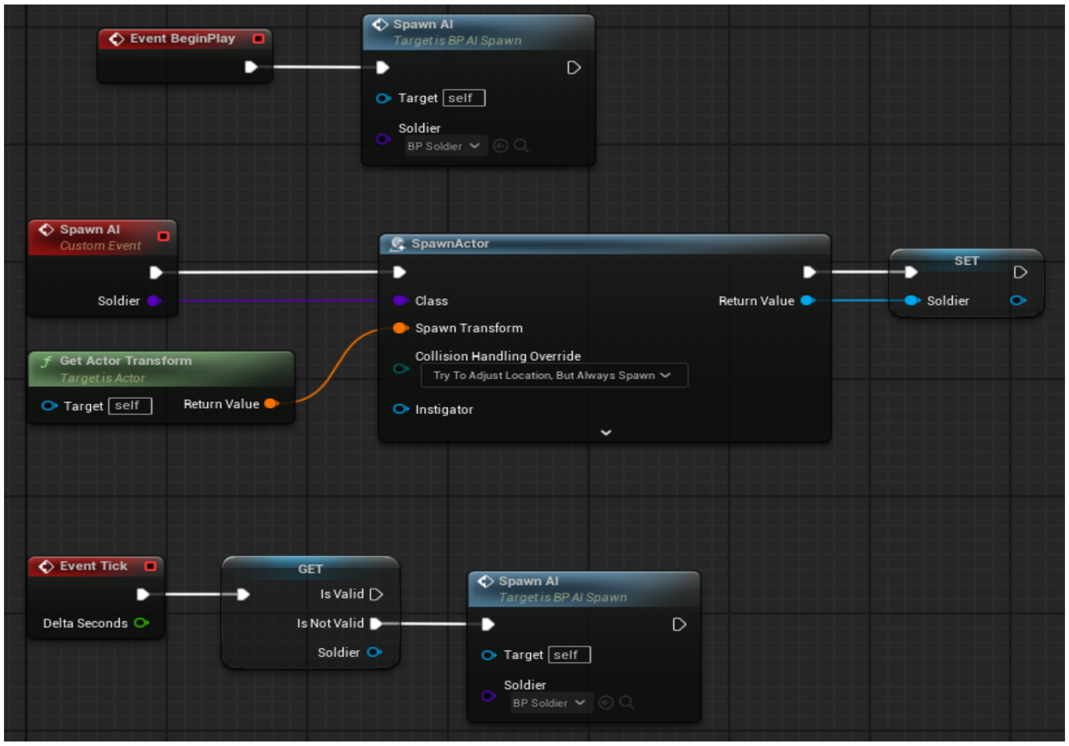


Figure 3.2.4 AI Spawn Blueprint

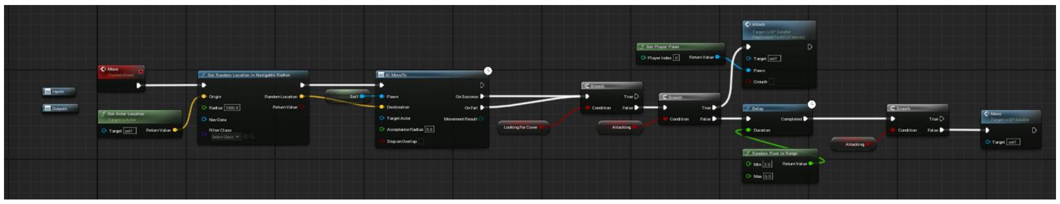


Figure 3.2.5 AI Patrol System

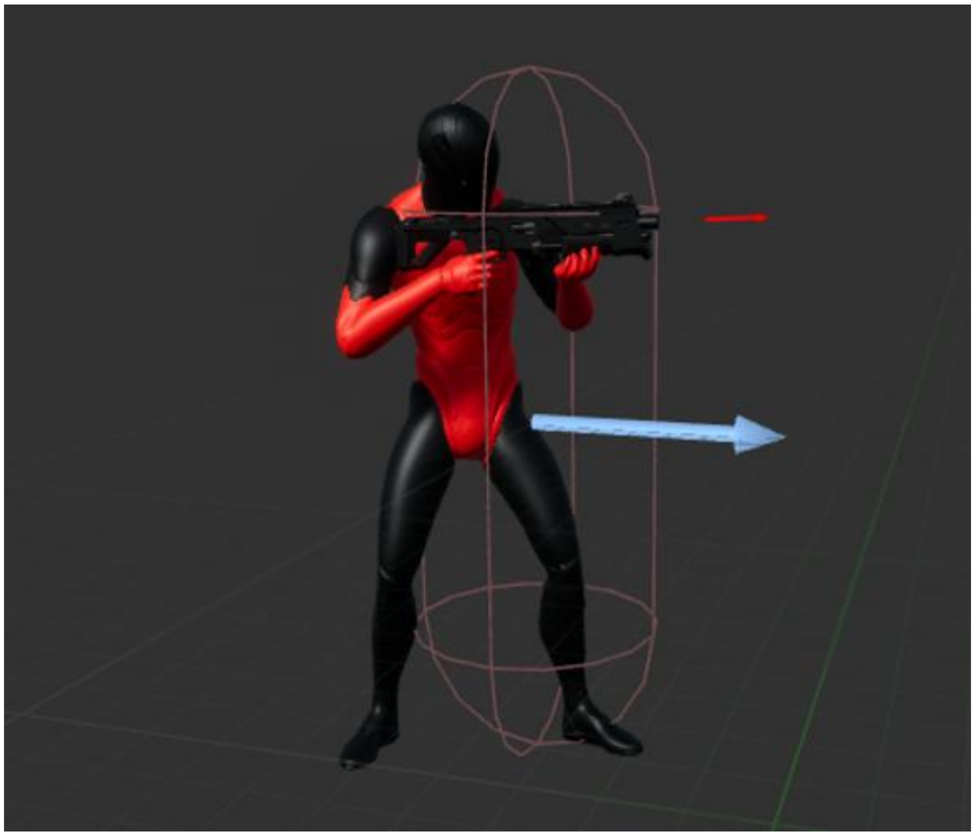


Figure 3.2.6 AI Character Model

3.2.3 Environment & Level Model Development:

The Environment & Level Model Development focuses on designing the complete game world in which the player and enemies interact. In Unreal Engine 5.7, the environment plays a critical role because it not only creates the visual atmosphere but also determines how players and AI navigate, take cover, and engage in combat. The development process began by creating the base terrain, adding modular building structures, placing FAB assets, and designing pathways that support smooth movement throughout the map. The level layout was planned in a way that provides clear combat zones, movement routes, cover positions, spawn points, and exploration areas for the player.

To build a detailed and realistic world, FAB assets were used extensively. These high-quality ready-made assets helped in constructing buildings, props, walls, barriers, crates, lighting posts, and various environmental elements that make the game world visually rich. Nanite was enabled for these assets to ensure smooth performance even when using high-detail models. The environment was arranged in layers such as ground meshes, structural meshes, vegetation, props, and interactive elements so that each component could be edited independently during development.

Collision components were added to every major structure to ensure proper interaction between the player, AI enemies, and projectiles. Walls were assigned blocking collisions to stop bullets and movement, floors were given walking surfaces, and objects like crates and obstacles were designed to act as cover during combat. Proper collision setup improves gameplay flow and prevents the player from getting stuck or passing through objects unintentionally. This step ensures the environment behaves as expected during fast-paced movement and shooting.

A crucial part of environmental development was generating the Navigation Mesh (NavMesh). This mesh defines all walkable surfaces and helps AI enemies move intelligently throughout the map. When the NavMesh was built, enemies were able to patrol paths, chase the player, reposition during fights, and avoid obstacles. Adjustments were made to ensure there were no dead zones, blocked paths, or unreachable areas. The NavMesh greatly enhances enemy behavior and makes the level feel more dynamic.

Lighting and atmosphere were enhanced using Lumen, which provides dynamic global illumination, soft shadows, and realistic reflections. As the player moves around the level, Lumen updates the lighting instantly, creating a more immersive and visually appealing environment. This allows both indoor and outdoor areas to maintain consistent and natural lighting without manually baking shadows.

Together, these components create a detailed, optimized, and interactive game world where the player can navigate freely and engage in realistic combat scenarios. The final environment supports smooth performance, high visual fidelity, and stable AI navigation, making it an essential foundation for the entire gameplay experience.

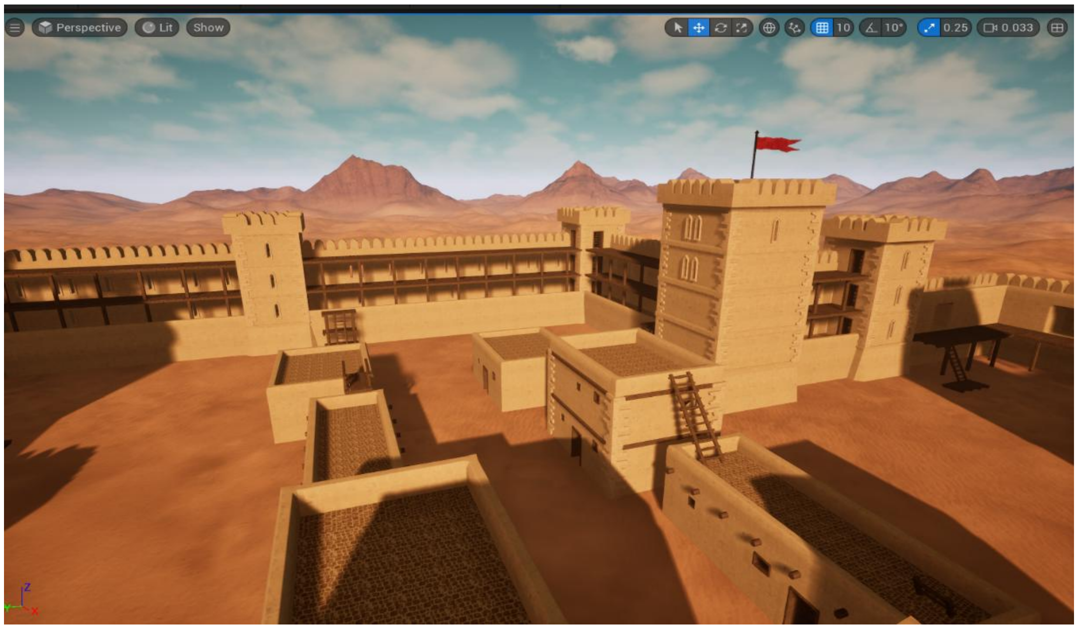


Figure 3.2.7 Desert Map

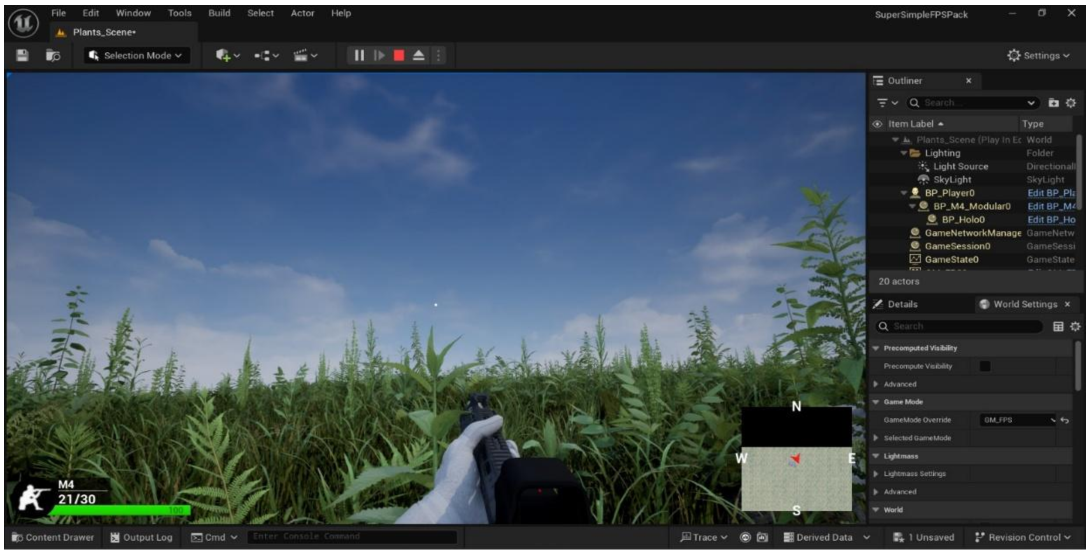


Figure 3.2.8 Flora Map

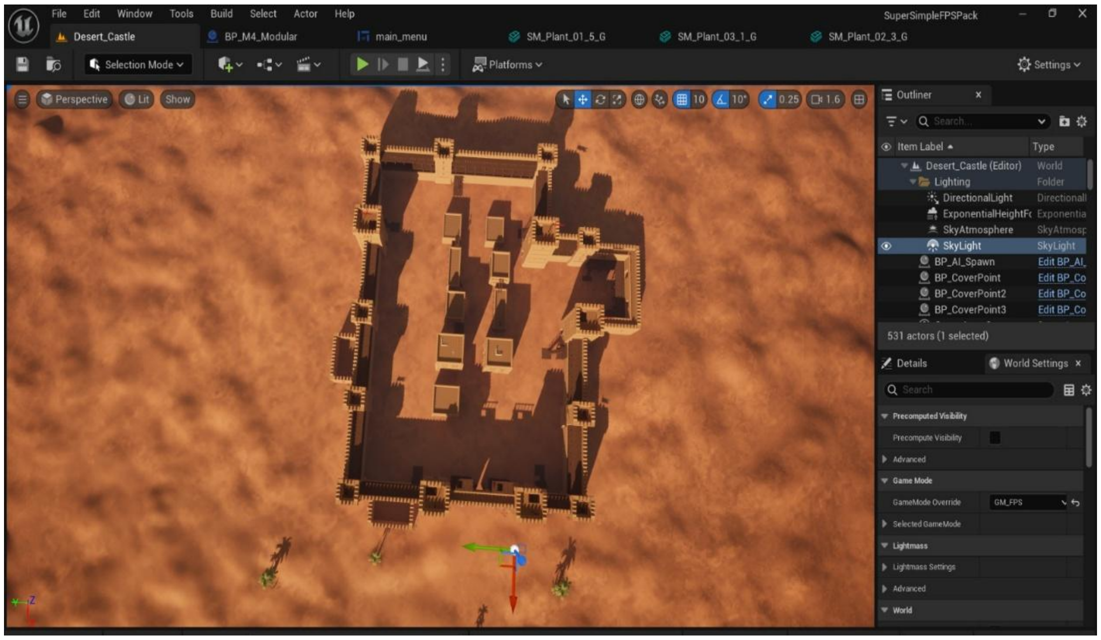


Figure 3.2.9 Top View of Desert Map

3.2.4 Rendering & UI Model Development:

The Rendering & UI Model Development focuses on creating the visual quality, lighting, optimization, and user interface systems that make the game visually appealing and easy to interact with. Unreal Engine 5.7 provides advanced graphical technologies such as Nanite, Lumen, and LOD systems, which were used to build a high-quality and smooth gaming experience. These components ensure that the environment, characters, weapons, and objects are rendered with realistic lighting and geometry while maintaining stable performance throughout gameplay. The rendering model is essential because it directly affects how the player experiences the game world.

Nanite was used as the primary geometry system for rendering detailed environment assets. It allows extremely high-poly meshes to be used without reducing frame rate, making the buildings, props, and terrain look sharp and realistic. After Nanite processes the models, Lumen generates dynamic lighting, reflections, and global illumination based on the player's movement and environmental changes. This real-time lighting system removes the need for baking lightmaps and ensures that both indoor and outdoor areas look natural, with soft shadows and accurate reflections. The LOD (Level of Detail) system further optimizes the performance by automatically reducing the complexity of distant objects, ensuring the game runs smoothly even in heavy combat scenes.

The UI (User Interface) Model was developed using Unreal Engine's Widget Blueprint system. The HUD (Heads-Up Display) includes several elements that help the player track important game information. These include the health bar, ammo counter, crosshair, enemy indicators, and boss health bar for special encounters. Each of these elements updates in real time based on gameplay actions. For example, ammo decreases when the weapon fires, health is reduced when the player takes damage, and enemy indicators appear when enemies are nearby. The crosshair helps in aiming accurately, while the boss health bar appears only during critical fights to improve player awareness.

Blueprint logic was connected to the UI system to ensure that gameplay events are reflected correctly on the screen. When the player reloads the weapon, the ammo value resets. When the player enters a danger zone, alert indicators appear. Damage feedback is shown through red screen flashes or directional indicators. These dynamic UI features improve the player's experience and make the game more engaging and intuitive. The rendering and UI components together ensure that the game not only functions correctly but also looks polished and feels satisfying to interact with.

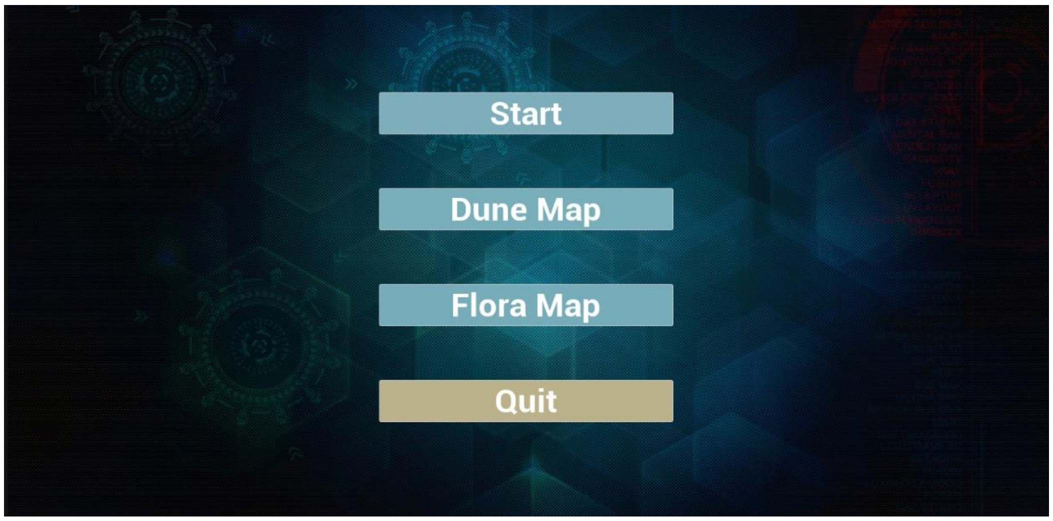


Figure 3.2.10 UI/HUD

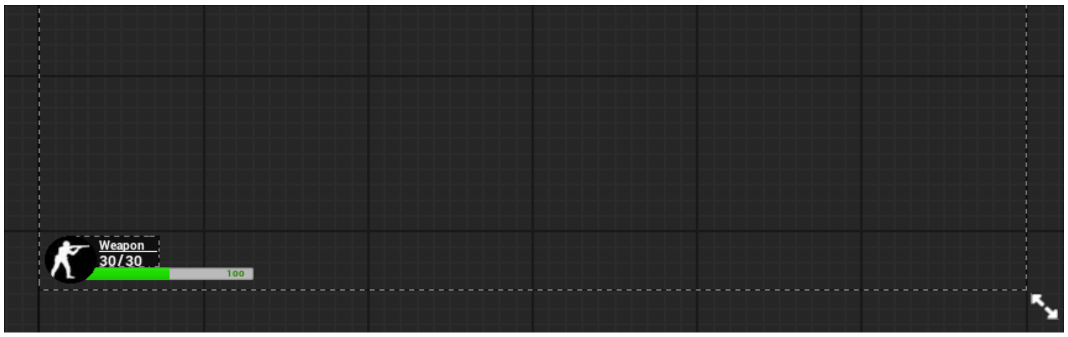


Figure 3.2.11 Health UI Widget

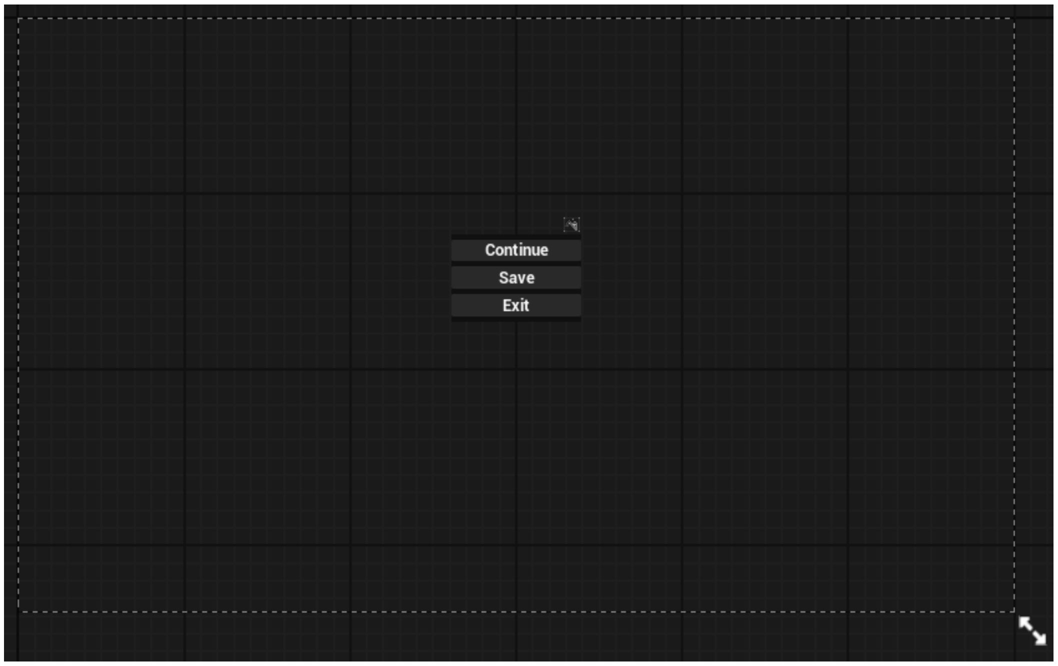


Figure 3.2.12 Pause Menu

Chapter IV
RESULT AND DISCUSSION

4.1 Gameplay Results and Observations:

The gameplay results highlight that the complete 3D shooting game functions reliably with smooth mechanics, realistic AI behavior, and consistent visual performance inside Unreal Engine 5.7. During testing, the player character reacted accurately to keyboard and mouse inputs through the Enhanced Input System. Movement transitions such as walking, running, crouching, aiming, and jumping occurred without delay, and the camera smoothly followed the player's direction. Shooting mechanics, including recoil, bullet travel, muzzle flashes, and hit detection, responded instantly, creating an engaging and responsive combat experience. The animations synced perfectly with player actions, ensuring that movements felt natural and visually polished.

Enemy AI performance was evaluated in multiple environments, and the bots behaved intelligently based on the Behavior Tree logic. Enemies patrolled their assigned paths, scanned the area using the Perception System, and reacted quickly when the player entered their detection range. Once the player was spotted, enemies transitioned into chase or attack modes depending on distance, obstacles, and line of sight. NavMesh-based pathfinding allowed AI characters to move around corners, climb slopes, and reposition during combat without taking incorrect paths. These behaviors created intense and realistic gameplay encounters, making the game more challenging and dynamic.

The environment and level structure were also tested thoroughly. All FAB assets, buildings, walls, and props behaved correctly during collisions. The player and enemy characters did not clip through objects or get stuck in any part of the environment. Nanite-enabled geometry delivered detailed textures and models without affecting performance, even when large scenes were loaded. Lumen lighting updated in real time, producing soft shadows, reflections, and global illumination that reacted to player actions and environmental elements.

The entire level streamed smoothly, ensuring there were no sudden loading delays or frame drops.

User Interface elements were also verified under different gameplay conditions. The health bar decreased correctly when the player received damage, the ammo counter updated instantly after each shot or reload, and enemy indicators appeared only when opponents were nearby. The crosshair remained stable and accurate during movement, sprinting, and firing. The boss health bar appeared properly during special encounters and disappeared after the fight, confirming its correct functionality. These UI elements contributed to a clear, interactive, and player-friendly experience.

Key Gameplay Observations

- Player movement, aiming, and shooting were smooth and responsive.
- Enemy AI detected the player accurately and behaved intelligently.
- NavMesh allowed enemies to navigate the map without errors.
- Collisions were stable for both environment and characters.
- Nanite and Lumen delivered high-quality visuals with reliable performance.
- HUD elements such as health, ammo, and crosshair updated instantly.

4.1.1 Gameplay Testing Results:

Gameplay testing was conducted to evaluate how well the core systems of the game performed during actual play sessions. The player character responded smoothly to all movement inputs including walking, sprinting, jumping, and aiming. Shooting mechanics worked accurately, with projectiles hitting targets as expected and recoil effects displaying correctly. Animation transitions such as idle-to-run, run-to-aim, and aim-to-shoot were consistent and did not show any visible delays or glitches. Camera rotation and player control remained stable even during fast movement and combat situations, ensuring a responsive gameplay experience.

Enemy behavior was also tested across multiple scenarios. AI bots were able to patrol predefined paths, detect the player within their vision range, and transition into chase or attack modes reliably. The Navigation Mesh allowed enemies to move naturally around obstacles and approach the player without getting stuck. Combat interactions between the player and AI opponents happened smoothly, with damage values, hit detection, and attack animations working correctly. Overall, gameplay testing confirmed that the movement, combat mechanics, AI behavior, environment interactions, and HUD updates all functioned exactly as intended.

Table 4.1: Gameplay Performance Results Table:

Sr	Test Performed	Result / Observation
1	Player Movement Test	Movement smooth and responsive
2	Shooting Mechanics Test	Accurate hit detection, no delay
3	AI Patrol Behavior Test	AI patrolled correctly on NavMesh
4	AI Chase & Attack Test	AI responded to player presence
5	Environment Collision Test	No clipping or stuck issues
6	UI/HUD Functionality Test	Health, ammo & crosshair updated

4.1.2 AI Performance Results:

AI performance testing was carried out to evaluate how effectively the enemy bots behaved in different gameplay scenarios. During testing, the AI responded accurately to the player's presence through the Perception System. Enemies were able to identify the player when entering their vision range and reacted immediately by switching from patrol mode to chase or attack mode. Detection timing, distance sensitivity, and angle-based visibility all worked consistently throughout the tests.

AI enemies also demonstrated intelligent navigation using the Navigation Mesh. They followed patrol routes correctly and moved around obstacles without getting stuck or taking incorrect paths. When engaging the player, bots repositioned themselves, maintained attack distance, and attempted to track the player's movement. If the player hid or moved out of sight, the AI briefly searched nearby areas before returning to their patrol state, confirming that the Behavior Tree logic was functioning correctly.

Combat interactions between the AI bots and player were smooth and predictable. The AI fired projectiles at the correct intervals, dealt damage accurately, and followed attack cooldown rules. Attack frequency and movement patterns remained stable even when multiple enemies were active simultaneously. Overall, the AI system behaved reliably and delivered consistent results, contributing to challenging and dynamic gameplay.

Key AI Behavior Observations:

➢ **Patrol Movement:**

AI bots followed predefined patrol paths smoothly.
No stuck movement or random direction change observed.

➢ **Player Detection:**

AI successfully detected the player based on distance and line-of-sight.
Bots reacted immediately once the player entered their vision cone.

➢ **Chase Behavior:**

Bots chased the player using accurate NavMesh pathfinding.
AI smoothly avoided obstacles and maintained proper chase speed.

➢ **Attack Execution:**

Bots attacked the player at the correct range.
Firing logic triggered with accurate timing and consistent cooldowns.

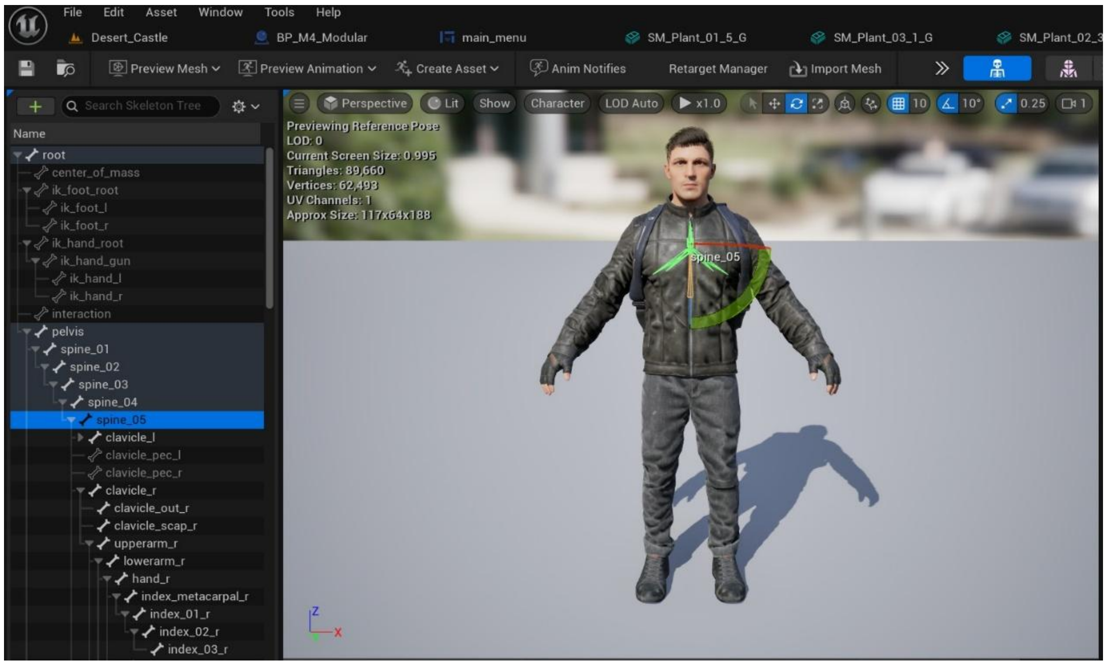
➢ **State Transitions:**

AI switched cleanly between patrol → chase → attack → search → return states.

No looping, freezing, or unexpected behavior was encountered.

Boss AI Behavior Highlights:

- **Extended Detection Range:** Boss detected the player earlier than normal enemies.
- **Special Attack Logic:** Performed unique high-damage attacks at set intervals.
- **Smooth Animation Transition:** Boss attacks and movement blended naturally.
- **Higher Aggression:** Boss maintained constant pressure during combat.
- **Advanced Tracking:** Reacted quickly to player movement, cover, and dodging.



Previewing Reference Pose
LOD: 0
Current Screen Size: 0.995
Triangles: 89,660
Vertices: 62,493
UV Channels: 1
Approx Size: 117x64x188

Figure 4.1.1 Skeleton

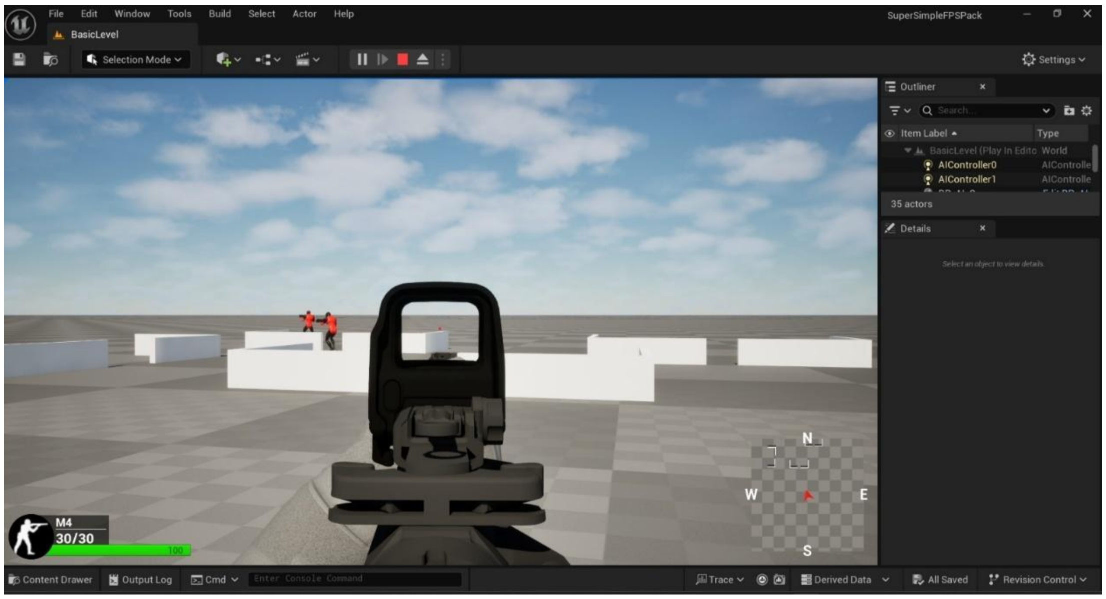


Figure 4.1.2 Shooting System

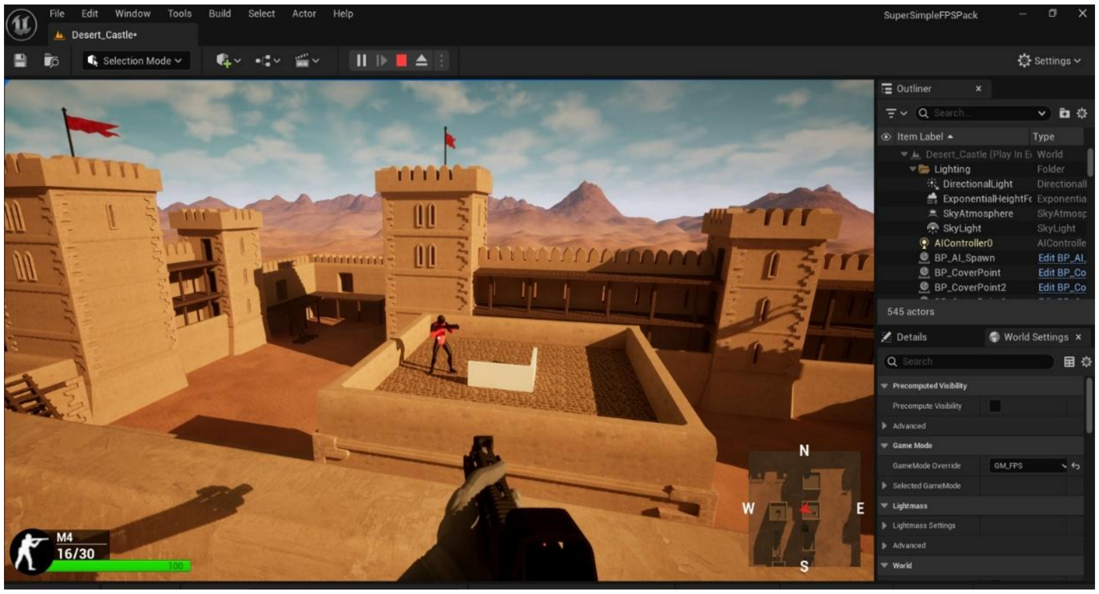


Figure 4.1.3 Player view

4.1.3 Visual and Rendering Performance Results:

The visual and rendering performance of the developed 3D shooting game was tested under different gameplay scenarios, including indoor areas, outdoor environments, enemy combat zones, and boss fights. Unreal Engine 5.7's advanced rendering systems — Nanite, Lumen, and Level of Detail (LOD) — worked smoothly throughout the testing process. Nanite allowed high-poly meshes and detailed FAB assets to render without visible frame drops, making the environment appear sharp, realistic, and stable even during fast camera rotation and intense gameplay.

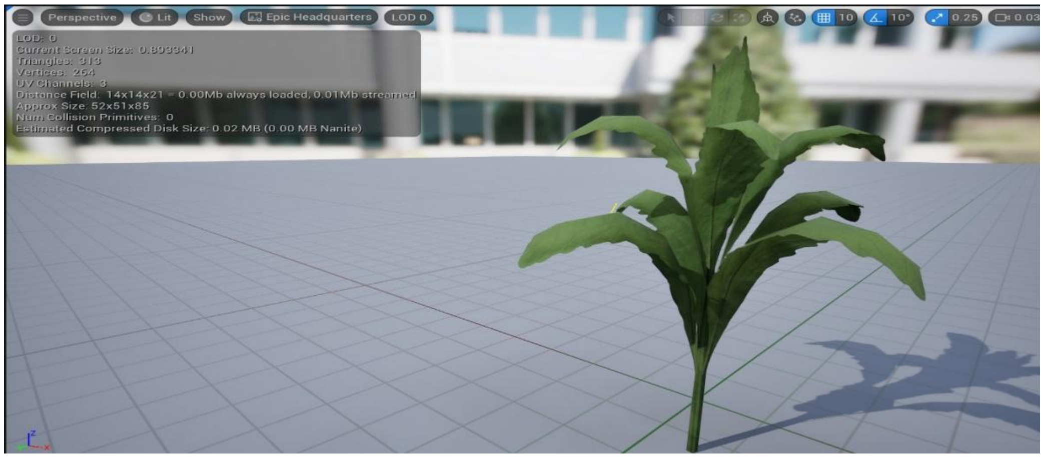
Lumen lighting provided dynamic global illumination, soft shadows, and real-time reflections that adapted instantly to player movement and environmental changes. Light sources reacted naturally when the player moved through dark corridors, open areas, and rooms with multiple objects. Shadow quality remained consistent and stable, and no lighting artifacts or flickering issues were observed. The overall illumination improved depth, realism, and visual clarity throughout the level.

The Level of Detail (LOD) system performed effectively by reducing polygon density for distant objects, improving overall frame stability. LOD transitions occurred smoothly without popping or visual glitches. While testing heavy scenes with multiple enemies and high-detail structures, the game maintained stable visual output and consistent frame performance. Special effects such as muzzle flashes, bullet impacts, explosions, and boss attack effects also rendered correctly and matched the fast-paced gameplay.

- **Nanite Rendering:** High-detail meshes rendered smoothly with no visible artifacts.
- **Lumen Lighting:** Real-time shadows, reflections, and illumination responded instantly.
- **LOD Optimization:** Distance-based detail reduction happened smoothly without popping.

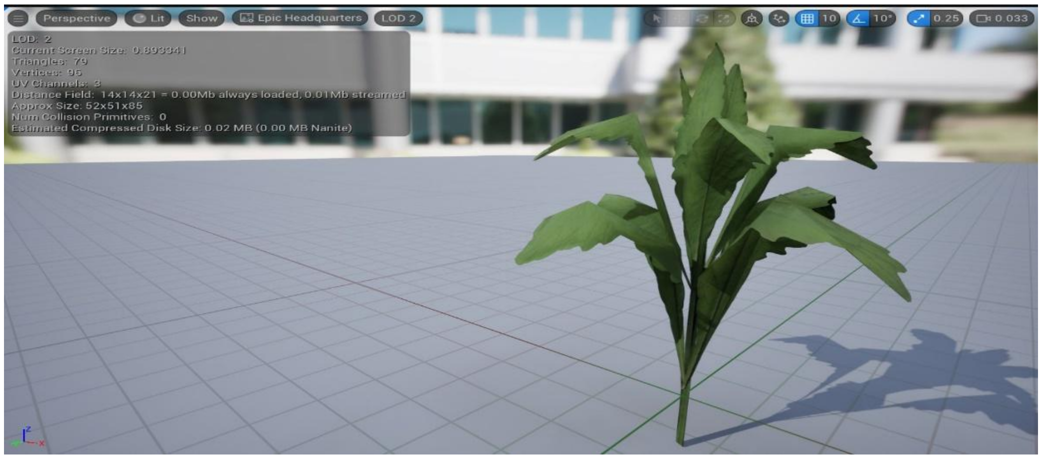
- **Combat Visuals:** Muzzle flashes, bullet impacts, and effects appeared sharp and stable.

- **Overall Frame Stability:** Gameplay maintained consistent visual quality even during heavy action.



```
LOD: 0
Current Screen Size: 0.893341
Triangles: 313
Vertices: 264
UV Channels: 3
Distance Field: 14x14x21 = 0.00Mb always loaded, 0.01Mb streamed
Approx Size: 52x51x85
Num Collision Primitives: 0
Estimated Compressed Disk Size: 0.02 MB (0.00 MB Nanite)
```

Figure 4.1.4 (a) LOD 0



```
LOD: 2
Current Screen Size: 0.893341
Triangles: 79
Vertices: 96
UV Channels: 3
Distance Field: 14x14x21 = 0.00Mb always loaded, 0.01Mb streamed
Approx Size: 52x51x85
Num Collision Primitives: 0
Estimated Compressed Disk Size: 0.02 MB (0.00 MB Nanite)
```

Figure 4.1.4 (b) LOD 2

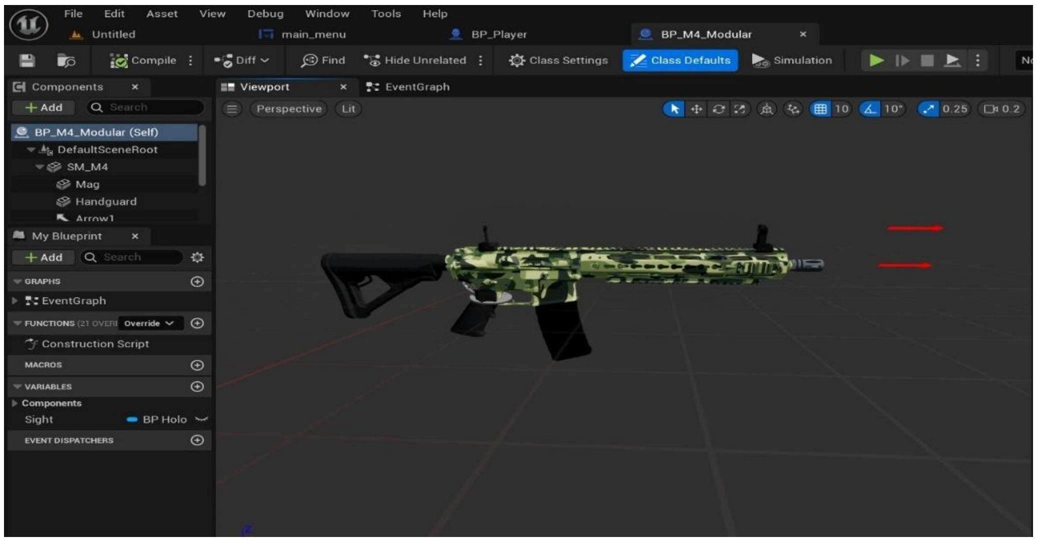


Figure 4.1.5 Shooting AR Gun

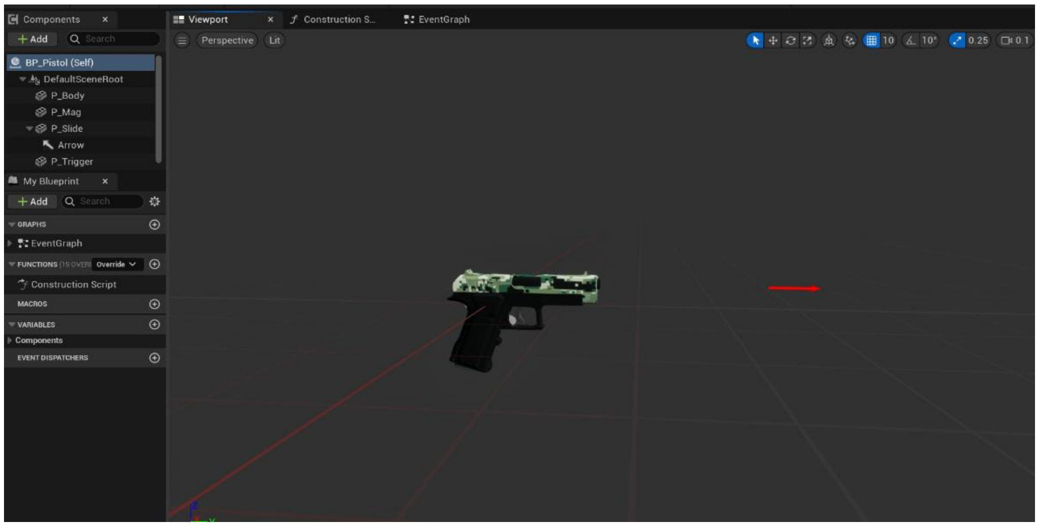


Figure 4.1.6 Shooting Pistol Gun

4.2 Discussion:

The overall results show that all major systems of the 3D shooting game worked together smoothly during testing. Player controls such as movement, aiming, shooting, and animation transitions responded accurately and maintained stable performance. The AI system reacted correctly to the player's presence, shifted between patrol, chase, and attack states, and navigated the level effectively using the Navigation Mesh. These behaviors indicate that the game logic, movement flow, and combat interactions were designed and implemented reliably.

Rendering and environment performance were also consistent across different areas of the map. Nanite delivered high-detail visuals, Lumen provided dynamic lighting and shadow updates, and the LOD system handled distance-based optimization without visual issues. The HUD elements such as health, ammo, crosshair, and boss indicators updated correctly, helping the player stay aware during gameplay. Overall, all systems worked cohesively and supported a smooth and engaging gameplay experience.

- Player controls and combat mechanics remained stable during all tests.
- AI behavior transitions worked without errors or stuck states.
- Navigation Mesh ensured smooth enemy movement across the map.
- Lumen and Nanite maintained strong visual quality without frame drops.
- HUD elements responded instantly to gameplay events.

Chapter V
CONCLUSION AND FUTURE SCOPE

5.1 Conclusion:

The development of the 3D shooting game using Unreal Engine 5.7 successfully met all the intended objectives of the project. By integrating Blueprint scripting, Enhanced Input, AI behavior systems, Nanite rendering, and Lumen lighting, the game provides smooth control, intelligent enemy responses, and visually realistic environments. The project demonstrates how modern game engine features can be combined to create an engaging and stable gameplay experience. From movement and shooting mechanics to boss battles and visual effects, all systems worked cohesively and delivered reliable performance during testing.

This project also provided valuable learning outcomes in the areas of real-time rendering, AI design, level development, and system optimization. The use of Nanite and Lumen helped understand advanced rendering workflows, while Blueprint scripting simplified rapid prototyping of gameplay mechanics. The overall execution of the project shows that Unreal Engine is a powerful and scalable tool for academic as well as professional game development. The final output successfully reflects the objectives defined at the beginning and establishes a strong foundation for future enhancements or expansion of the game.

5.2 Future Scope:

The developed 3D shooting game has a strong foundation and can be expanded significantly with future improvements. Unreal Engine 5.7 provides scalable tools that allow the game to grow in terms of gameplay, performance, visuals, and user experience. The following future scopes can enhance the overall game quality and make it suitable for advanced academic or commercial development.

- **Game Development Industry:**

This project can be extended into professional PC, console, or mobile games. The skills gained in Unreal Engine, Blueprints, AI systems, and rendering technologies can be applied directly to real-world game studios and indie game development teams.

- **Simulation & Training Systems:**

The core mechanics of the game can be adapted for military shooting simulators, police training modules, security-force training, and safety or emergency-response simulation systems.

- **Virtual Reality (VR) Applications:**

With Unreal Engine's VR support, the game can be converted into a VR shooting experience. This can be used for immersive training, VR entertainment centers, and interactive shooting simulations.

- **Education & Research:**

The project can serve as a reference for teaching topics like game design, AI behavior trees, real-time rendering, Lumen lighting, Nanite optimization, and Blueprint scripting. It can be used in colleges, labs, workshops, and research studies.

- **Entertainment & E-Sports:**

By adding competitive modes, leaderboards, and multiplayer support, the game can evolve into a small e-sports title or a competitive shooting challenge, attracting players and communities.

- **Entertainment & E-Sports:**

By adding competitive modes, leaderboards, and multiplayer support, the game can evolve into a small e-sports title or a competitive shooting challenge, attracting players and communities.

References:

[1] Anderson, Erik, and John McCaffrey. “Using Unreal Engine 4 to Teach Game Design and Development.” *Proceedings of the 2018 ACM Conference on Innovation and Technology in Computer Science Education*, pp. 264–269, 2018.

[2] Kojic, Dusko, and Milos Prica. “Development of AI Behavior in Modern Game Engines Using Behavior Trees.” *IEEE 9th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1–4, 2020.

[3] Karvelis, Petros, and Nikolaos Papagiannis. “Real-Time Rendering Improvements with Nanite and Lumen in Unreal Engine 5.” *Journal of Computer Graphics and Applications*, vol. 12, no. 4, pp. 22–30, 2022.

[4] Gomes, Rui, and José Pereira. “AI Navigation and Pathfinding Techniques for 3D Game Environments.” *Procedia Computer Science*, vol. 179, pp. 728–737, 2020.

[5] Nakamura, Hideki, and Taichi Hayashi. “Dynamic Lighting and Global Illumination Techniques for Real-Time Games.” *Computer Animation and Virtual Worlds*, vol. 31, no. 3–4, e1959, 2020.

[6] Pan, Zhiqiang, and Rui Chen. “Performance Optimization Techniques for Real-Time 3D Games Using Level of Detail (LOD) Systems.” *Entertainment Computing*, vol. 35, pp. 100–110, 2020.

[7] Shaw, Timothy. “User Interface (UI) and HUD Design Principles in Modern FPS Games.” *International Journal of Human–Computer Interaction*, vol. 36, no. 10, pp. 923–933, 2020.