

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN



NGUYỄN VĨNH HƯNG - 52200097
KHUU TRÙNG DƯƠNG - 52200154
NGUYỄN HÒA AN - 52200182

TÌM HIỂU VÀ DEMO VỀ UNREAL ENGINE

BÁO CÁO GIỮA KỲ PHÁT TRIỂN TRÒ CHƠI

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2025

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN



NGUYỄN VĨNH HƯNG - 52200097
KUU TRÙNG DƯƠNG - 52200154
NGUYỄN HÒA AN - 52200182

TÌM HIỂU VÀ DEMO VỀ UNREAL ENGINE

BÁO CÁO GIỮA KỲ PHÁT TRIỂN TRÒ CHƠI

Người hướng dẫn
ThS. Vũ Đình Hồng

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2025

LỜI CẢM ƠN

Chúng em xin chân thành cảm ơn ThS. Vũ Đình Hồng đã tận tình hướng dẫn, hỗ trợ và cung cấp những kiến thức quý báu trong suốt quá trình thực hiện đề tài. Chúng em cũng xin cảm ơn các thầy cô trong Khoa Công Nghệ Thông Tin, Trường Đại học Tôn Đức Thắng đã tạo điều kiện để nhóm có môi trường học tập và thực hành tốt nhất.

TP. Hồ Chí Minh, ngày 25 tháng 11 năm 2025

Tác giả

(Ký tên và ghi rõ họ tên)

Hung

Nguyễn Vĩnh Hưng

Duong

Khưu Trùng Dương

An

Nguyễn Hòa An

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH

TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của ThS. Vũ Đình Hồng. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Dự án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Dự án của mình. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 25 tháng 11 năm 2025

Tác giả

(Ký tên và ghi rõ họ tên)

Hung

Nguyễn Vĩnh Hưng

Duong

Khưu Trùng Dương

An

Nguyễn Hòa An

TÌM HIỂU VÀ DEMO VỀ UNREAL ENGINE

TÓM TẮT

Đề tài “Tìm hiểu và Demo về Unreal Engine” tập trung nghiên cứu tổng quan về Unreal Engine 5 – một trong những game engine mạnh mẽ nhất hiện nay trong lĩnh vực phát triển trò chơi, phim ảnh và mô phỏng. Báo cáo trình bày kiến trúc lõi của Unreal Engine, quy trình xây dựng một dự án game, các công cụ quan trọng trong Unreal Editor, gameplay framework, hệ thống nhân vật, AI, UI và quy trình triển khai bản demo.

Nhóm đã tiến hành xây dựng một demo gameplay hoàn chỉnh ở góc nhìn TPS (Third-Person Shooter), bao gồm hệ thống di chuyển, chiến đấu, AI địch, inventory, UI và điều khiển phương tiện. Thông qua đó, nhóm thể hiện khả năng áp dụng thực tế những kiến thức đã tìm hiểu, đồng thời đánh giá tiềm năng của Unreal Engine trong phát triển game hiện đại. Kết quả đạt được cho thấy Unreal Engine là nền tảng mạnh mẽ, hỗ trợ tốt cho cả đội ngũ developer và artist, đặc biệt trong các dự án 3D chất lượng cao.

MỤC LỤC

DANH MỤC HÌNH VẼ	x
DANH MỤC BẢNG BIỂU	xiii
DANH MỤC CÁC CHỮ VIẾT TẮT.....	xiv
CHƯƠNG 1. MỞ ĐẦU VÀ TỔNG QUAN ĐỀ TÀI.....	1
1.1 Giới thiệu về Unreal Engine.....	1
1.2 Lịch sử ra đời và phát triển	1
1.3 So sánh với Unity	3
1.3.1 <i>Đồ họa và render (chất lượng hình ảnh)</i>	5
1.3.2 <i>Hiệu năng và tối ưu (scale, open-world, mobile)</i>	6
1.3.3 <i>Công cụ phát triển và workflow (scripting, visual tools, art pipeline)</i>	6
1.3.4 <i>Đa nền tảng và mục tiêu thiết bị</i>	7
1.3.5 <i>Hệ sinh thái, marketplace và cộng đồng</i>	7
1.3.6 <i>Chi phí và licensing</i>	7
1.3.7 <i>Rủi ro và hạn chế thực tế</i>	7
1.4 Ứng dụng của Unreal Engine	8
1.5 Ưu – nhược điểm thực tế khi sử dụng Unreal Engine	8
1.5.1 <i>Ưu điểm</i>	8
1.5.2 <i>Nhược điểm</i>	9
1.5.3 <i>Thách thức</i>	9
1.6 Mục tiêu thực hiện đề tài.....	9
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT	11
2.1 Kiến trúc nền tảng của Unreal Engine 5 (Core Architecture).....	11

2.1.1 <i>UObject system và Reflection</i>	11
2.1.2 <i>Actor – Component Architecture</i>	13
2.1.3 <i>Pawn – Character – Controller</i>	14
2.1.4 <i>World, Level, World Partition</i>	16
2.1.5 <i>Subsystem Architecture</i>	17
2.1.6 <i>Module System</i>	19
2.1.7 <i>Unreal Build Tool (UBT) và Build Modules</i>	21
2.1.8 <i>Garbage Collection và Memory Model</i>	23
2.2 Unreal Editor 5 – Giao diện và Công cụ chính	25
2.2.1 <i>Level Editor</i>	25
2.2.2 <i>World Outliner</i>	27
2.2.3 <i>Details Panel</i>	30
2.2.4 <i>Content Browser</i>	32
2.2.5 <i>Material Editor</i>	34
2.2.6 <i>Animation Editor</i>	36
2.2.7 <i>Niagara Editor</i>	38
2.2.8 <i>Sequencer</i>	39
2.2.9 <i>MetaHuman tools</i>	41
2.3 Hệ thống Gameplay Framework	43
2.3.1 <i>GameMode</i>	43
2.3.2 <i>GameState và PlayerState</i>	45
2.3.3 <i>Actor Lifecycle</i>	47
2.3.4 <i>Input System (Enhanced Input)</i>	49

2.3.5 <i>UI Framework</i>	51
2.3.6 <i>Gameplay Ability System (GAS)</i>	54
2.3.7 <i>Networking và Replication</i>	56
2.4 Blueprint Scripting và C++ Programming	57
2.4.1 <i>Blueprint Visual Scripting</i>	57
2.4.2 <i>C++ Programming</i>	59
2.4.3 <i>Khi nào dùng Blueprint – khi nào dùng C++</i>	59
2.5 Asset và Content System.....	60
2.5.1 <i>Asset Types (Mesh, Material, Texture, Animation, Audio ...)</i>	60
2.5.2 <i>Level</i>	61
2.5.3 <i>Blueprint Class và Blueprint Asset</i>	61
2.5.4 <i>Asset Metadata</i>	61
2.5.5 <i>Soft/Hard References</i>	61
2.5.6 <i>Asset Cooking Pipeline</i>	62
2.5.7 <i>Asset Streaming</i>	62
2.6 Animation System	62
2.6.1 <i>Skeleton</i>	62
2.6.2 <i>Animation Blueprint</i>	63
2.6.3 <i>Animation State Machine</i>	63
2.6.4 <i>Blend Space</i>	63
2.6.5 <i>Control Rig</i>	63
2.6.6 <i>IK System</i>	63
2.6.7 <i>Retargeting</i>	64

2.7 Hệ thống đồ họa và VFX trong UE5.....	64
2.7.1 Công nghệ đồ họa UE5.....	64
2.7.2 Niagara VFX System.....	68
2.7.3 Chaos Physics	69
2.8 Audio System	72
CHƯƠNG 3. QUY TRÌNH XÂY DỰNG, LẬP TRÌNH VÀ TRIỂN KHAI DEMO 77	
3.1 Quy trình xây dựng hệ thống	77
3.1.1 Môi trường phát triển	77
3.1.2 Cấu trúc project	78
3.1.3 Các asset và tài nguyên được sử dụng.....	79
CHƯƠNG 4. HỆ THỐNG NHÂN VẬT NGƯỜI CHƠI.....83	
4.1 Player.....	83
4.1.1 Cấu trúc tổng thể của Player.....	83
4.1.2 Góc nhìn và điều khiển di chuyển	84
4.1.3 Hệ thống chiến đấu (Combat System).....	85
4.1.4 Hệ thống trang bị (Equipment System).....	88
4.1.5 Vũ khí – Drop Equipment	90
4.1.6 Hệ thống tương tác (Interaction System).....	90
4.1.7 Hệ thống sức khỏe – Health System.....	91
4.1.8 Hệ thống lái xe (Vehicle Control System).....	92
4.1.9 Animation Blueprint.....	93
4.1.10 Giao diện người chơi (Player UI).....	94

<i>4.1.11 Collision & tương tác gameplay.....</i>	96
4.2 Enermy AI	96
<i>4.2.1 Hệ thống cảm biến – AI Perception (Sight).....</i>	97
<i>4.2.2 Behavior Tree (BT_Guard).....</i>	97
<i>4.2.3 Cơ chế tấn công – Line Trace Shooting.....</i>	102
<i>4.2.4 Cơ chế di chuyển & chết.....</i>	103
4.3 UI.....	103
<i>4.3.1 Kiến trúc tổ chức UI.....</i>	103
<i>4.3.2 UI điều khiển (Control Guide).....</i>	104
<i>4.3.3 UI hiển thị vũ khí & đạn (Weapon HUD).....</i>	105
<i>4.3.4 Hệ thống Inventory</i>	106
<i>4.3.5 UI Pause Menu (ESC).....</i>	107
<i>4.3.6 HP Bar & Health UI.....</i>	108
<i>4.3.7 Pickup Prompt UI (tự động)</i>	109
CHƯƠNG 5. DEMO GAMEPLAY.....	110
5.1 Mục tiêu của bài demo	110
5.2 Bối cảnh bản đồ (Map Context)	110
5.3 Luồng trải nghiệm gameplay tổng quát	111
5.4 Trình diễn từng hệ thống Gameplay	112
<i>5.4.1 Di chuyển nhân vật & Góc nhìn</i>	112
<i>5.4.2 Chiến đấu: Bắn – Nhảm – Nạp đạn.....</i>	114
<i>5.4.3 Hệ thống Inventory</i>	116
<i>5.4.4 Enemy AI Combat</i>	117

5.4.5 Hệ thống điều khiển xe (Vehicle Driving)	119
5.4.6 Hệ thống UI.....	119
CHƯƠNG 6. KẾT LUẬN.....	122
6.1 Kết luận	122
6.2 Hướng phát triển	122
TÀI LIỆU THAM KHẢO	123

DANH MỤC HÌNH VẼ

Hình 3.1 Cấu trúc tổ chức project	79
Hình 3.2 Thư mục asset	79
Hình 4.1 Cấu trúc tổng thể của Player	83
Hình 4.2 Góc nhìn TPS của nhân vật trong môi trường demo	85
Hình 4.3 Blueprint xử lý bắn súng (Start Shooting)	86
Hình 4.4 Blueprint xử lý nhắm bắn (Start Aiming)	87
Hình 4.5 Blueprint nạp đạn và animation (Reloading)	88
Hình 4.6 Blueprint chuyển đổi vũ khí (Next/Previous Weapon).....	89
Hình 4.7 Blueprint cất vũ khí (Holster Weapon)	89
Hình 4.8 Blueprint vứt vũ khí (Drop Weapon)	90
Hình 4.9 Blueprint tương tác (Enter Vehicle / Pickup Items).....	91
Hình 4.10 Blueprint xử lý xe cán người chơi (Instant Kill).....	92
Hình 4.11 Blueprint hệ thống xe (Accelerate, Steering, Get in/Exit Vehicle).....	93
Hình 4.12 Animation State Machine của Player	94
Hình 4.13 UI hiển thị bảng hướng dẫn điều khiển.....	95
Hình 4.14 UI hiển thị HP Bar.....	95
Hình 4.15 UI hiển thị inventory	96
Hình 4.16 Components của BP_Guard_Custom.....	97
Hình 4.17 Behavior Tree tổng quan (BT_Guard)	98
Hình 4.18 Nhánh Taking Damage	99
Hình 4.19 Nhánh Is Armed / Is Unarmed	100
Hình 4.20 Nhánh Engage Enemy (AI bắn Player).....	101

Hình 4.21 Nhánh Reload Weapon	101
Hình 4.22 Kiểm tra tiếng động (Noise Investigation).....	102
Hình 4.23 Nhánh Overwatch (AI quan sát môi trường)	102
Hình 4.24 Giao diện Control Guide	105
Hình 4.25 UI fullscreen của giao diện Control Guide	105
Hình 4.26 UI fullscreen khi trang bị Rocket Launcher	106
Hình 4.27 Giao diện Inventory (Crafting + Inventory + Quick Slots)	107
Hình 4.28 Giao diện Pause Menu	108
Hình 4.29 UI hiển thị máu và tương tác khi bị bắn.....	109
Hình 4.30 UI hiển thị vật phẩm.....	109
Hình 5.1 Toàn cảnh bản đồ và nhân vật khi bắt đầu demo	111
Hình 5.2 Toàn cảnh bản đồ và nhân vật khi bắt đầu demo	111
Hình 5.3 Gameplay nhân vật idle ở góc nhìn TPS.....	113
Hình 5.4 Gameplay nhân vật walk ở góc nhìn TPS	113
Hình 5.5 Gameplay nhân vật jump ở góc nhìn TPS	114
Hình 5.6 Nhân vật chết.....	114
Hình 5.7 UI vũ khí khi Player aim	115
Hình 5.8 UI vũ khí khi Player bắn	116
Hình 5.9 UI vũ khí khi Player nạp đạn	116
Hình 5.10 Giao diện Inventory với Crafting – Inventory – Quick Slots	117
Hình 5.11 Enemy AI trong lúc giao tranh với Player đang trong Car	118
Hình 5.12 Enemy AI chết.....	118
Hình 5.13 Người chơi điều khiển xe trong demo	119

Hình 5.14 Control Guide hiển thị trong gameplay.....	120
Hình 5.15 HUD vũ khí	120
Hình 5.16 Inventory	121
Hình 5.17 Menu Pause	121

DANH MỤC BẢNG BIỂU

Bảng 1.1 Bảng so sánh các phiên bản Unreal và Unity	2
Bảng 1.2 Tổng hợp so sánh Unreal Engine với Unity	4

DANH MỤC CÁC CHỮ VIẾT TẮT

TPS	Third-Person Shooter
UE	Unreal Engine
VR	Virtual Reality

CHƯƠNG 1. MỞ ĐẦU VÀ TỔNG QUAN ĐỀ TÀI

1.1 Giới thiệu về Unreal Engine

Unreal Engine (UE) là một hệ thống game engine đa mục đích do Epic Games phát triển, được ứng dụng rộng rãi trong các lĩnh vực như phát triển trò chơi điện tử, sản xuất phim ảnh, mô phỏng kỹ thuật, kiến trúc và các ứng dụng thực tế ảo (VR/AR). Với khả năng xử lý đồ họa 3D chất lượng cao cùng bộ công cụ tích hợp toàn diện, Unreal Engine đã trở thành một nền tảng quan trọng trong ngành công nghiệp nội dung số [1].

Engine này được khởi phát từ giữa thập niên 1990 bởi Tim Sweeney – nhà sáng lập Epic Games – nhằm giải quyết các hạn chế của công nghệ đồ họa thời kỳ đó, bao gồm khả năng hiển thị còn thô sơ, thiếu công cụ trực quan và khó mở rộng đối với các nhà phát triển bên thứ ba. Unreal Engine ban đầu được xây dựng để phục vụ cho tựa game *Unreal* (1998), đồng thời hướng đến mục tiêu trở thành một nền tảng mở và linh hoạt cho các nhà phát triển khác [2].

Qua nhiều thế hệ phát triển, Unreal Engine đã khẳng định vị thế nhờ khả năng dựng hình tiên tiến, tính linh hoạt trong lập trình và hệ thống Blueprint trực quan. Bên cạnh đó, engine cung cấp môi trường làm việc theo thời gian thực, cho phép các nhóm nghệ sĩ, kỹ sư và nhà thiết kế phối hợp hiệu quả trong toàn bộ quy trình sản xuất, góp phần đưa Unreal Engine trở thành nền tảng công nghệ cốt lõi trong nhiều lĩnh vực sáng tạo hiện đại.

1.2 Lịch sử ra đời và phát triển

Qua các phiên bản, Unreal Engine liên tục được cải tiến để đáp ứng nhu cầu ngày càng cao của ngành công nghiệp giải trí. UE1 (1998) tập trung vào dựng thế giới 3D cơ bản, ánh sáng động và hệ thống scripting riêng (UnrealScript), hỗ trợ modding, giúp các nhà phát triển tạo ra các bản mở rộng dễ dàng [2]. UE2 (2002) cải thiện vật lý bằng Karma Physics, nâng cao animation và đồ họa, đồng thời hỗ trợ thêm console và PC [2]. UE3 (2006) là bước đột phá với shader model, Lightmass, công cụ Kismet, và khả năng chạy trên consoles như PS3 và Xbox360, trở thành

engine phổ biến cho các game AAA như Gears of War, Mass Effect, Batman Arkham Series [2] [3].

Unreal Engine 4 (2014) đánh dấu bước ngoặt lớn: Epic Games mở mã nguồn, cung cấp engine miễn phí, tích hợp hệ thống lập trình trực quan Blueprint và renderer PBR, giúp các nhà phát triển dễ dàng tạo game, VR, phim và ứng dụng kiến trúc [2] [3]. Gần đây, Unreal Engine 5 (2022) giới thiệu các công nghệ đột phá như Nanite (render hàng tỉ polygon real-time), Lumen (global illumination real-time), World Partition và MetaHuman, cho phép dựng các thế giới 3D chất lượng phim điện ảnh và open-world quy mô lớn. UE5 được sử dụng trong các game AAA và dự án cinematic như Fortnite, Hellblade II, Black Myth: Wukong [4].

Mỗi đời Unreal Engine ra đời đều nhằm giải quyết các hạn chế đời trước và thích ứng với phần cứng mới. Ví dụ, UE3 tối ưu cho consoles thế hệ PS3/Xbox360, UE4 mở rộng cho lập trình trực quan và đa nền tảng, UE5 tập trung vào real-time photorealism và thế giới mở khổng lồ.

Bảng 1.1 Bảng so sánh các phiên bản Unreal và Unity

Engine	Năm ra mắt	Điểm nổi bật	Công cụ lập trình	Ứng dụng chính	Ưu điểm	Nhược điểm
UE1	1998	Engine 3D đầu tiên, ánh sáng động, modding	UnrealScript	Game PC 3D	Đồ họa 3D tốt, dễ mod	Hạn chế về vật lý, animation
UE2	2002	Physics Karma, animation nâng cao	UnrealScript	Game PC và console	Cải thiện đồ họa, physics	Hơi cứng nhắc, khó học

UE3	2006	Shader model, Lightmass, Kismet	UnrealScript, C++	Game AAA, console	Đồ họa đẹp, hỗ trợ console	Yêu cầu phần cứng mạnh
UE4	2014	Mã nguồn mở, Blueprint, PBR, Sequencer	C++, Blueprint	Game AAA, VR, phim, kiến trúc	Dễ học với Blueprint, đa nền tảng	Kích thước engine lớn, cần GPU mạnh
UE5	2022	Nanite, Lumen, World Partition, MetaHuman	C++, Blueprint	Game AAA next-gen, open-world, cinematic	Đồ họa gần phim, real-time, mở thẻ giới lớn	Yêu cầu PC/console mạnh, còn mới, học phí cao cho team
Unity	2005	Đa nền tảng, lightweight, Asset Store lớn	C#	Mobile, indie, VR/AR, game 2D/3D	Dễ học, nhanh triển khai, nhẹ	Đồ họa kém hơn UE, render AAA phức tạp hơn

1.3 So sánh với Unity

So với Unity, Unreal Engine nổi bật về đồ họa cao cấp, hiệu ứng ánh sáng và công cụ cinematic, thường được các studio AAA và dự án VR/phim sử dụng. Unity mạnh về linh hoạt, nhẹ, dễ học, thích hợp cho mobile, indie game hoặc các dự án nhanh và đa nền tảng. Lựa chọn engine phụ thuộc vào mục tiêu dự án, chất lượng đồ họa mong muốn và kinh phí [5] [6].

Nhờ sự phát triển liên tục, cộng đồng rộng lớn và hệ sinh thái phong phú, Unreal Engine hiện là một trong những công cụ mạnh mẽ nhất trong ngành công nghiệp giải trí, được ứng dụng từ các game AAA như Fortnite, Gears of War, Hellblade, đến các dự án phim, kiến trúc và mô phỏng kỹ thuật.

Dưới đây là bảng tổng hợp so sánh Unreal Engine với Unity qua một số tiêu chí:

Bảng 1.2 Tổng hợp so sánh Unreal Engine với Unity

Tiêu chí	Unreal Engine 5 (UE5)	Unity
Mục tiêu thiết kế	Tối ưu cho photorealism, game AAA, cinematic, film, kiến trúc 3D.	Tối ưu cho đa nền tảng, đặc biệt mobile, 2D, game indie, AR/VR.
Chất lượng đồ họa	Mạnh nhất hiện nay trong game engine thương mại. Hỗ trợ Nanite (virtualized geometry) và Lumen (real-time GI). Chỉ cần ít tối ưu LOD thủ công.	HDRP cho đồ họa cao cấp nhưng không đạt “out-of-the-box” như UE5. Cần tự setup LOD, ánh sáng, shader nhiều hơn. URP tối ưu cho mobile.
Hiệu năng	Khả năng xử lý cảnh lớn rất mạnh nhưng dễ tụt FPS nếu không tối ưu. Yêu cầu PC mạnh.	Nhỏ hơn, phù hợp mobile và game quy mô vừa/nhỏ. DOTS giúp tăng hiệu năng nhưng khó dùng.
Scripting	C++ + Blueprint Visual Scripting mạnh mẽ; designer có thể tạo gameplay mà không cần code.	C# dễ học, dễ prototyping. Không có visual scripting mặc định mạnh bằng Blueprint.
Công cụ cinematic	Sequencer, Control Rig, MetaHuman, Virtual Production → mạnh nhất hiện nay.	Có Timeline + Cinemachine, nhưng không ngang UE5 cho phim/cinematic lớn.

Artist pipeline	Drop-in asset đa giác cực lớn nhờ Nanite; phù hợp team làm cinematic/AAA.	Cần tối ưu asset nhiều hơn, pipeline thiên về mobile/indie. Asset Store rất phong phú.
Đa nền tảng	PC, console, mobile, AR/VR. Nhưng mobile phải tối ưu rất mạnh.	Rất mạnh: PC, mobile, console, WebGL, AR/VR. Standalone nhỏ gọn, build nhanh.
Cộng đồng – Hệ sinh thái	Marketplace chất lượng cao, nhiều asset AAA, Quixel Megascans miễn phí.	Cộng đồng lớn nhất trong mảng indie/mobile; Asset Store cực kỳ đa dạng.
Chi phí	Miễn phí, doanh thu vượt ngưỡng sẽ phải chia doanh thu theo quy định của Epic.	Các gói Personal/Pro/Enterprise. Một số gói thu phí, tùy mức doanh thu.
Độ khó học	Khó hơn cho người mới (C++), nhưng Blueprint giúp học nhanh.	Dễ hơn cho beginner nhờ C# và tài liệu phong phú.
Loại dự án phù hợp	Game AAA, open-world, cinematic, kiến trúc, phim ảnh, ứng dụng 3D chất lượng cao.	Game mobile, 2D, VR/AR, indie, app đa nền tảng, game casual.

1.3.1 Đồ họa và render (chất lượng hình ảnh)

Unreal (UE5): tập trung vào photorealism real-time với hai công nghệ cốt lõi: Nanite (virtualized geometry — render lượng đa giác cực lớn mà không cần LOD thủ công) và Lumen (global illumination real-time). Kết quả: dễ đạt chất lượng gần phim cho cảnh có nhiều chi tiết và ánh sáng động.

Unity: hiện có hai đường ray/đồ họa chính — HDRP (High Definition Render Pipeline) cho fidelity cao trên PC/console và URP (Universal Render Pipeline) cho hiệu năng trên mobile/khung nhẹ. HDRP đã tiến bộ mạnh nhưng về tính “out-of-the-box” để đạt photorealism ở quy mô UE5 thì Unity thường cần nhiều cấu hình, shader và pipeline tùy chỉnh hơn.

Từ đây, có thể thấy UE5 có lợi thế kỹ thuật lớn; Unity có thể đạt gần mức đó nhưng thường tồn công cấu hình hơn.

1.3.2 Hiệu năng và tối ưu (scale, open-world, mobile)

UE5 mạnh cho các dự án AAA, open-world lớn nhưng yêu cầu phần cứng mạnh; nếu không tối ưu sớm, có nguy cơ gặp vấn đề hiệu năng (vấn đề thực tế nhiều studio gặp phải khi build hình ảnh cao cấp). Epic đang liên tục cải tiến công cụ tối ưu nhưng developer discipline vẫn quan trọng.

Unity có lợi thế nhẹ hơn và nhiều tính năng tối ưu cho mobile/2D; URP giúp tối ưu cross-platform. Unity cũng phát triển hướng Data-Oriented Tech Stack (DOTS) để giải quyết bottleneck CPU/GPU cho project lớn, nhưng DOTS/Hybrid còn trong giai đoạn chuyển tiếp và cần kiến thức chuyên sâu.

Có thể nói Unity ưu thế cho mobile, 2D, dự án indie và những nơi cần chạy trên cấu hình thấp; UE5 phù hợp cho PC/console AAA và cinematic, nhưng cần lập kế hoạch tối ưu hóa ngay từ đầu.

1.3.3 Công cụ phát triển và workflow (scripting, visual tools, art pipeline)

UE có bộ công cụ “tập trung cinematic” mạnh hơn sẵn sàng; Unity cung cấp workflow linh hoạt, thân thiện dev bằng C# và hệ thống asset/extension lớn.

1.3.3.1 Scripting

- UE: C++ (hiệu năng cao) + Blueprints (visual scripting mạnh, dùng phổ biến cho gameplay/thiết kế nhanh).
- Unity: C# là ngôn ngữ chính, dễ học với nhiều tutorial; scripting C# rất nhanh để prototype.

1.3.3.2 Visual Tools / Cinematics

- UE có Sequencer, Control Rig, MetaHuman — mạnh cho cinematic, animation và character pipeline.
- Unity có Timeline, Cinemachine, Visual Effect Graph; khá tốt nhưng nhiều studio thấy UE tiện cho cinematic phức tạp hơn.

1.3.3.3 Asset / Art pipeline

- UE hỗ trợ trực tiếp các asset có độ phân giải rất cao nhờ Nanite; pipeline nghệ sĩ dễ dàng drop-in mô hình nhiều đa giác.
- Unity cần quản lý LOD và tối ưu hơn (HDRP đã cải thiện), nhưng Unity Asset Store + tooling cho mobile/2D rất phong phú.

1.3.4 Đa nền tảng và mục tiêu thiết bị

Unity: truyền thống mạnh về đa nền tảng (mobile, WebGL, console, AR/VR) và thường là lựa chọn hàng đầu cho phát triển nhanh đa nền tảng.

Unreal: cũng đa nền tảng (PC/console/mobile/console/AR/VR) nhưng thường được chọn cho PC/console/console-nextgen và các ứng dụng yêu cầu đồ họa cao.

Vậy nên nếu target mobile/web nhỏ gọn thì nên sử dụng Unity; nếu target PC/console AAA hoặc film/archviz thì sử dụng Unreal.

1.3.5 Hệ sinh thái, marketplace và cộng đồng

Unity Asset Store rất lớn, nhiều plugin, template cho mobile/2D/AR/VR; cộng đồng dev indie rất đông.

Unreal Marketplace có asset chất lượng cao, nhiều content AAA-oriented. UE cũng có kho sample cinematic, MetaHumans, megascans (Quixel).

Cả hai có cộng đồng lớn, nhưng Unity có lợi thế trong mảng indie/mobile, UE có cộng đồng mạnh mẽ ở AAA/film.

1.3.6 Chi phí và licensing

Unreal: chính sách thương mại hóa (Epic lấy % doanh thu theo điều kiện — trước đây 5% trên một số ngưỡng; cần kiểm tra điều khoản hiện hành cho chính xác trước khi ký hợp đồng).

Unity: có nhiều tier license (Personal/Plus/Pro/Enterprise) và đã có nhiều thay đổi về chính sách giá trong các năm gần đây; tùy nhu cầu doanh nghiệp mà phải trả license.

1.3.7 Rủi ro và hạn chế thực tế

UE5: đem lại fidelity lớn nhưng dễ dẫn tới vấn đề performance nếu không tối ưu sóm; nhiều studio đã chia sẻ khó khăn khi cố gắng port/delivery cho phần cứng yêu.

Unity: khá “đa dụng” nhưng để đạt chuẩn visual AAA hoặc xử lý open-world không lồ có thể cần đầu tư lớn vào hệ thống render và tối ưu riêng (HDRP + custom shaders + DOTS).

1.4 Ứng dụng của Unreal Engine

- Phim ảo (Virtual Production) & kỹ xảo điện ảnh: UE hỗ trợ dựng bối cảnh ảo real-time trên màn hình LED, giúp quay phim linh hoạt, giảm chi phí set thật và hậu kỳ.
- VR/AR: Cung cấp khả năng render real-time chất lượng cao, phù hợp cho ứng dụng mô phỏng, đào tạo, giải trí, triển lãm ảo.
- Visualization công nghiệp: Dùng để mô phỏng sản phẩm, quy trình kỹ thuật, thử nghiệm thiết kế trong môi trường 3D tương tác.
- Kiến trúc – bất động sản (Arch-viz): UE hỗ trợ tạo walkthrough, phối cảnh nội thất – ngoại thất, xem dự án trước khi xây dựng với chất lượng hình ảnh gần như thật.
- Mô phỏng – đào tạo quân sự / an toàn / y tế: Tạo môi trường ảo an toàn để huấn luyện, diễn tập hoặc kiểm thử các tình huống nguy hiểm mà không cần mô phỏng vật lý thật.

1.5 Ưu – nhược điểm thực tế khi sử dụng Unreal Engine

1.5.1 Ưu điểm

- Đồ họa mạnh, chất lượng photorealistic nhờ Lumen, Nanite và hệ thống vật liệu tiên tiến.
- Render real-time, chỉnh sửa nhanh, phù hợp cho phim ảnh, VR, trình diễn.

- Đa ngành, sử dụng được cho game, phim, kiến trúc, công nghiệp và mô phỏng.
- Tiết kiệm chi phí sản xuất so với dựng cảnh thật hoặc render offline.

1.5.2 Nhược điểm

- Yêu cầu phần cứng cao, đặc biệt khi dùng mesh/texture nặng.
- Khó học với người mới vì có nhiều hệ thống phức tạp.
- Dự án lớn dễ trở nên nặng nề, nếu không quản lý asset tốt thì load lâu, dễ crash.
- Tối ưu hóa tốn thời gian, cần kỹ năng về đồ họa, asset, ánh sáng và pipeline

1.5.3 Thách thức

- Dễ gặp tình trạng lag, drop FPS, crash editor nếu không tối ưu
- Cần quản lý asset khoa học: LOD, streaming, chia level, tránh trùng lặp.
- Build / cook dự án có thể rất lâu nếu dung lượng lớn.
- Yêu cầu nhóm phát triển có kỹ năng đa dạng (3D, lighting, technical art, dev).

1.6 Mục tiêu thực hiện đề tài

Đề tài được thực hiện với các mục tiêu sau:

1. Tìm hiểu tổng quan Unreal Engine 5

Bao gồm kiến trúc lõi, hệ thống UObject, Actor–Component, Gameplay Framework, Animation, Graphics, Blueprint và C++.

2. Nắm vững cách sử dụng Unreal Editor

Hiểu và thao tác các công cụ như Level Editor, Content Browser, Material Editor, Animation Editor, Niagara, Sequencer...

3. Xây dựng một demo gameplay hoàn chỉnh

- Điều khiển nhân vật TPS

- Hệ thống chiến đấu (bắn – nhắm – nạp đạn)
- AI địch sử dụng Behavior Tree
- Hệ thống trang bị và inventory
- UI (HUD, Pause Menu, Pickup Prompt...)
- Điều khiển xe (Vehicle System)

4. Ứng dụng kiến thức để xây dựng logic gameplay thực tế

Sử dụng Blueprint và C++ nhằm hiểu cách UE5 xử lý logic nội bộ.

5. Đánh giá khả năng của Unreal Engine trong phát triển game

Phân tích ưu – nhược điểm qua quá trình làm demo.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1 Kiến trúc nền tảng của Unreal Engine 5 (Core Architecture)

2.1.1 *UObject system và Reflection*

UObject là lớp cơ sở quan trọng nhất trong Unreal Engine, đóng vai trò như một **đơn vị quản lý vòng đời (lifecycle unit)** cho mọi tài nguyên bậc cao trong engine, bao gồm Actor, Component, Material, Blueprint, Animation Asset, v.v. Khác với các lớp C++ thông thường, UObject được liên kết trực tiếp với **UObject Ecosystem**—một cơ chế nội bộ đảm bảo quản lý bộ nhớ, nhận diện đối tượng, theo dõi tham chiếu, và hỗ trợ kiểm tra thời gian chạy. Mọi lớp kế thừa từ UObject đều được đặt dưới sự quản lý của **Object Cluster** và **Global Object Array**, cho phép Unreal Engine có thể thực hiện các thao tác như tuần tự hóa (serialization), Garbage Collection và Reflection một cách nhất quán trên toàn bộ hệ thống.

Không như C++ truyền thống vốn dựa vào manual memory management hoặc RAII, hệ thống UObject sử dụng **Garbage Collector theo kiểu Mark-and-Sweep**. GC này không vận hành độc lập, mà dựa vào **Reference Token Stream**—một cấu trúc được sinh ra tự động từ Metadata của lớp. Stream này mô tả chính xác điểm lưu trữ con trỏ trong từng đối tượng, giúp GC có thể phân tích đồ thị tham chiếu một cách **an toàn kiểu tĩnh (type-safe) và hiệu năng cao**. Bằng cách đánh dấu (mark) các đối tượng còn sống và quét (sweep) những đối tượng không được tham chiếu, Unreal Engine đảm bảo rằng chương trình không gặp hiện tượng memory leak hay dangling pointer, vốn là các vấn đề phổ biến trong C++.

Reflection trong Unreal Engine không phải là reflection của C++ (vốn gần như không tồn tại), mà được xây dựng như một **hệ thống ngôn ngữ siêu mô tả (meta-language system)**. Khi lập trình viên viết UCLASS, UPROPERTY, UFUNCTION, trình tiền xử lý của Unreal (Unreal Header Tool – UHT) sẽ phân tích file header và sinh ra một tập hợp metadata ở dạng code C++ mở rộng. Metadata này được tích hợp vào lớp final, cho phép engine thực hiện:

- Tạo đối tượng động (runtime class instantiation)

- Tìm kiếm thuộc tính và hàm theo tên
- Gọi hàm động (dynamic invocation)
- Phản chiếu cấu trúc dữ liệu của asset
- Tích hợp với Blueprint, Editor và Serialization

Từ góc nhìn học thuật, đây là một **Reflection System thuộc loại Hybrid Static–Runtime**, trong đó siêu dữ liệu được tạo ở compile-time nhưng được truy cập đầy đủ ở runtime thông qua hệ thống UClass và UProperty.

Mỗi lớp kế thừa từ UObject có một đối tượng UClass tương ứng, đóng vai trò như **bản mô tả kiểu (type descriptor)** trong hệ thống phản chiếu. UClass chứa danh sách UProperty,dữ liệu có thể được mô tả, kiểm tra và thay đổi một cách động, một khả năng vốn khó đạt được chỉ bằng C++ thuần túy.

Reflection là nền tảng để Unreal Engine xây dựng **Blueprint Virtual Machine**, vì toàn bộ các nút (nodes), biến, hàm trong Blueprint được tạo ra và ràng buộc từ Metadata của C++ thông quangôn ngữ biến dịch tĩnh (C++) và **ngôn ngữ kịch bản động (Blueprint)**—một mô hình kết hợp hiếm gặp trong các engine truyền thống.

Hệ thống UObject sử dụng Reflection để thực hiện tuần tự hóa (serialization) không theo kiểu nhị phân thuần túy, mà theo dạng **property-by-property** dựa vào metadata. Điều này có ba lợi thế học thuật quan trọng:

1. **Tương thích phiên bản (Version Resilience):** vì serialization dựa vào tên và metadata, thay đổi layout bộ nhớ không gây lỗi.

2. **Editor Integration:** editor có thể trực tiếp render UI cho từng thuộc tính dựa vào UProperty.
3. **Tối ưu hóa dữ liệu trò chơi:** cho phép cook/uncook data linh hoạt theo platform.

Unreal Engine cũng có khả năng thực hiện Custom Serialization dựa trên việc override Serialize(), nhưng mọi quá trình vẫn phải tuân thủ hệ thống metadata của UObject để giữ tính nhất quán.

UObject System và Reflection của Unreal Engine là một ví dụ điển hình về cách một engine AAA giải quyết hạn chế của C++ bằng cách tự xây dựng một **hệ thống ngôn ngữ siêu dữ liệu (metaprogramming ecosystem)**. Đây không chỉ là một kỹ thuật phần mềm mà còn là **một cơ sở hạ tầng tri thức (knowledge infrastructure)** giúp engine hỗ trợ: quản lý tài nguyên, an toàn bộ nhớ, serialization, scripting, editor tooling và networking. Trong ngữ cảnh kiến trúc phần mềm, có thể xem hệ thống này như **một lớp ngôn ngữ siêu hình (meta-layer)** đặt lên trên C++, giúp C++ trở nên mềm dẻo như một ngôn ngữ động nhưng vẫn giữ được hiệu năng và tính chặt chẽ kiểu tĩnh.

2.1.2 Actor – Component Architecture

Actor–Component Architecture của Unreal Engine là một ví dụ điển hình và trưởng thành của mô hình thiết kế Composition over Inheritance trong các engine hiện đại. Thay vì tạo ra một hệ thống phân cấp lớp (class hierarchy) sâu và cứng nhắc, Unreal Engine lựa chọn cách xây dựng hành vi trò chơi bằng cách ghép các thành phần (components) vào một đối tượng trung tâm gọi là Actor. Điều này giúp tăng tính mô-đun, khả năng tái sử dụng, khả năng mở rộng, và giảm thiểu hiện tượng “ké thừa kim tự tháp” (inheritance explosion) vốn thường xuất hiện trong các trò chơi có logic phức tạp.

Trong Unreal Engine, Actor là một **thực thể có mặt trong thế giới 3D** và được quản lý bởi World và Level System. Về bản chất, Actor không chứa nhiều logic phức tạp; bản thân nó đóng vai trò như một **container ngữ nghĩa** và một **điểm neo (anchor point)** cho các thành phần chức năng. Actor cung cấp transform cơ bản,

vòng đời (BeginPlay, Tick, EndPlay), khả năng tham gia vào networking (replication), và được phản chiếu đầy đủ qua hệ thống UObject Reflection. Vai trò quan trọng nhất của Actor là tạo ra **khung cấu trúc (structural context)** để các Component có thể được gắn vào và phối hợp với nhau.

Component trong Unreal Engine hiện diện như **khối mô-đun hành vi**. Các thành phần này có thể cung cấp khả năng vật lý (Physics Component), render (Mesh Component), âm thanh (Audio Component), chuyển động (Movement Component), hoặc logic gameplay (Custom Actor Component). Các Component có thể được gắn tĩnh (Static Attachment) hoặc gắn động (Runtime Attachment). Nhờ đó, Actor trở thành một đối tượng mở rộng linh hoạt, cho phép lập trình viên xây dựng hành vi phức tạp bằng cách **kết hợp** nhiều Component thay vì phải tạo ra nhiều lớp Actor riêng biệt.

Về mặt học thuật, Component đóng vai trò như **Functional Modules**, tương đương các “behavior nodes” trong mô hình Entity–Component Systems (ECS), nhưng Unreal vẫn giữ tính hướng đối tượng qua Actor Framework.

2.1.3 Pawn – Character – Controller

Kiến trúc **Pawn – Character – Controller** trong Unreal Engine thể hiện một mô hình phân tách nhiệm vụ mang tính hệ thống giữa **thực thể vật lý trong thế giới** và **logic điều khiển trùu tượng**. Thay vì gộp chuyển động, đầu vào và logic AI vào một đối tượng duy nhất, Unreal Engine triển khai mô hình **ba tầng** nhằm đạt sự linh hoạt, tính tái sử dụng và khả năng hỗ trợ multiplayer hiệu quả. Tư tưởng cốt lõi nằm ở việc **tách vật thể có thể chiếm quyền điều khiển (Possessable Entity)** ra khỏi **thực thể điều khiển (Controller)**, đồng thời cung cấp một lớp trung gian tối ưu cho chuyển động nhân vật (Character). Đây là kiến trúc rất trưởng thành, được sử dụng xuyên suốt trong cả Single-player lẫn Multiplayer Networking.

Pawn là lớp trùu tượng đại diện cho bất kỳ thực thể nào có thể được điều khiển trong thế giới trò chơi. Về mặt bản chất, Pawn **không bắt buộc phải là nhân vật người**; nó có thể là drone, xe, robot, camera bay, hoặc bất kỳ đối tượng nào có logic nhận đầu vào và phản hồi lại thế giới. Pawn cung cấp các hàm cơ bản cho input,

movement, possession và replication. Trong hệ thống Actor–Component đã trình bày trước, Pawn là một Actor có năng lực tương tác cao hơn, có thể được "điều khiển" thông qua một Controller. Điều này tạo ra một sự phân tách quan trọng: **Pawn thể hiện hình dạng và hành vi vật lý**, trong khi **Controller định nghĩa ý chí** của thực thể đó.

Character là một lớp con của Pawn được tối ưu hóa cho mô hình nhân vật người hoặc sinh vật hai chân–hai tay. Nó đi kèm với hệ thống **Character Movement Component**, cho phép xử lý các hành vi di chuyển phức tạp như đi, chạy, nhảy, rơi, leo dốc, phanh, ma sát, và các trạng thái động lực học khác. Ngoài ra, Character còn tích hợp sẵn **Capsule Collider**, hệ thống hoạt ảnh (Animation Blueprint), và cơ chế Root Motion. Bằng cách đóng gói nhiều tính năng đặc thù vào Character, Unreal Engine tạo ra một đối tượng có khả năng tái sử dụng rất cao cho đa số trò chơi có nhân vật người chơi.

Controller là một lớp độc lập, **tồn tại không phụ thuộc vào Pawn**. Một Controller có thể điều khiển một Pawn thông qua cơ chế **Possession**. Controller được chia thành hai loại:

- **PlayerController** – đại diện cho người chơi thật, kết nối với input từ bàn phím, chuột, gamepad hoặc thiết bị VR.
- **AIController** – điều khiển Pawn dựa trên thuật toán AI, Behavior Tree hoặc logic tùy chỉnh.

Bản thân Controller không có hình dạng, không có collider, không tham gia vật lý. Nó chỉ đại diện cho “bộ não” quyết định Pawn sẽ làm gì. Khi chuyển Pawn (ví dụ: đi vào xe, cưỡi ngựa), Controller có thể “unpossess” Pawn hiện tại rồi “possess” Pawn khác. Đây là cơ chế linh hoạt hỗ trợ gameplay phong phú.

Mối quan hệ giữa Controller và Pawn là dạng **1–1 tại một thời điểm**, như người điều khiển và phương tiện:

- Controller **ra lệnh**
- Pawn **thực thi**

Pawn cung cấp API để Controller di chuyển nó (AddMovementInput, AddControllerYawInput, ...), còn Controller tập trung xử lý quyết định (input mapping hoặc AI logic). Khi một Pawn bị hủy hoặc trở nên không hợp lệ, Controller sẽ unpossess và trở lại trạng thái rãnh. Kiến trúc này giúp Unreal hỗ trợ hoàn hảo các cơ chế đặc trưng của game: điều khiển nhân vật, điều khiển phương tiện, nhập vai AI, chuyển host trong multiplayer, hoặc thay đổi nhân vật giữa các màn chơi.

2.1.4 World, Level, World Partition

Trong Unreal Engine, các khái niệm **World**, **Level**, và **World Partition** hợp thành một kiến trúc tổ chức dữ liệu không gian quy mô lớn (spatial data organization). Đây là nền tảng cho việc mô phỏng thế giới ảo từ quy mô nhỏ (single map) cho đến những thế giới mở rộng lớn (open world). Unreal Engine phân tách rõ ràng giữa **không gian chạy thực tế** (runtime world), **dữ liệu nội dung** (level assets), và **cơ chế streaming thông minh** (partition system). Cách tổ chức này giúp giảm chi phí bộ nhớ, tối ưu hiệu suất, và nâng cao khả năng làm việc nhóm trong sản xuất game AAA.

World là đối tượng trung tâm đại diện cho toàn bộ môi trường mô phỏng trong thời gian chạy. Mỗi World chứa:

- Tập các **Level** đang được tải vào bộ nhớ
- Hệ thống Actor đang sống (spawn, destroy, tick)
- Thế giới vật lý (Physics Scene)
- Hệ thống AI (Navigation, Perception, EQS)
- Hệ thống ánh sáng, âm thanh, và thời gian (Time Management)
- Các Subsystem quy mô toàn thế giới (World Subsystems)

Về mặt học thuật, World đóng vai trò giống như **Scene Graph Root** kết hợp **Simulation Manager**, chịu trách nhiệm đồng bộ và điều phối tất cả đối tượng trong cảnh. Mỗi phiên chơi (play session) thường có đúng một World chính (Persistent World), nhưng engine có thể tạo World phụ cho Editor, PIE, hoặc simulations tạm thời.

Level trong Unreal Engine là một đơn vị lưu trữ nội dung bao gồm Actors, ánh sáng, mesh, volumes, và các asset có vị trí cố định trong không gian. Level không phải là một thế giới độc lập mà là **tập hợp dữ liệu được World tải vào và quản lý**.

Có hai loại Level chính:

- **Persistent Level:** Level gốc của World; luôn được nạp.
- **Sublevels:** Các Level bổ sung có thể được stream vào/ra tùy điều kiện.

Trong quy trình sản xuất, Level cho phép:

- Chia nhỏ game thành nhiều phần logic
- Hỗ trợ nhiều nhóm nghệ sĩ làm việc song song
- Giảm dung lượng bộ nhớ bằng streaming
- Tổ chức gameplay theo khu vực

Ở cấp kiến trúc, Level có thể xem như các **module không gian** (spatial modules) mà World ghép lại để tạo thành cảnh hoàn chỉnh.

2.1.5 Subsystem Architecture

Subsystem Architecture trong Unreal Engine là một cơ chế nền tảng được thiết kế nhằm tách rời các chức năng hệ thống khỏi vòng đời của Actor hay Component, đồng thời cung cấp một kiến trúc mô-đun hóa, dễ mở rộng và nhất quán cho các dịch vụ cấp engine, cấp game instance, cấp world hoặc cấp level. Subsystem đóng vai trò như các “service modules” chạy song song với gameplay nhưng không phụ thuộc vào sự tồn tại của các đối tượng trong cảnh, qua đó mang lại sự ổn định và khả năng tái sử dụng cao. Về bản chất, subsystem khắc phục hạn chế của các singleton truyền thống bằng một khung quản lý vòng đời nhất quán, gắn chặt với lifecycle tự nhiên của Unreal và được tích hợp sâu vào hệ thống reflection.

Unreal định nghĩa một hệ phân cấp subsystem rõ ràng, mỗi loại tương ứng với một tầng kiến trúc khác nhau. Engine Subsystem (UEngineSubsystem) cung cấp các dịch vụ tồn tại xuyên suốt vòng đời engine, không phụ thuộc vào bất kỳ world hay level nào. Nó là nơi thích hợp cho các hệ thống mang tính toàn cục như quản lý logging tùy chỉnh, hệ thống asset pipeline, hoặc layer tích hợp với third-party SDK.

Vòng đời của Engine Subsystem bắt đầu khi engine khởi tạo và chỉ kết thúc khi engine tắt, đảm bảo sự ổn định tối đa.

Editor Subsystem (UEditorSubsystem và UEditorEngineSubsystem) mở rộng kiến trúc subsystem cho môi trường biên tập, cho phép các plugin hoặc công cụ mở rộng tích hợp chức năng trực tiếp vào Editor mà không ảnh hưởng tới build runtime. Đây là cơ chế nền tảng cho các workflow như tự động hóa build asset, quản lý cảnh, hay hệ thống tool hỗ trợ nhà thiết kế. Nó hiện thân cho triết lý “editor as a runtime environment” của Unreal — coi editor như một runtime thực thụ để vận hành các dịch vụ dài hạn.

Đối với gameplay, Unreal cung cấp hai lớp subsystem quan trọng: Game Instance Subsystem (UGameInstanceSubsystem) và World Subsystem (UWorldSubsystem). Game Instance Subsystem tồn tại xuyên suốt vòng đời của phiên chơi, kể từ lúc game instance được khởi tạo cho đến khi toàn bộ trò chơi kết thúc. Do không bị reset khi chuyển level, nó phù hợp để quản lý những dịch vụ cần tính liên tục như kết nối mạng, hệ thống matchmaking, quản lý user session, hoặc lưu trữ dữ liệu persistent của người chơi.

Trong khi đó, World Subsystem tương ứng trực tiếp với từng world đang hoạt động. Vì mỗi world trong Unreal có thể đại diện cho một level loaded, một persistent world, hoặc thậm chí một simulation context, World Subsystem là nơi lý tưởng cho các hệ thống cần bám sát môi trường gameplay cụ thể, như quản lý AI navigation, hệ thống thời tiết, quản lý spawning cho một level, hoặc cơ chế streaming nội bộ. Khi world bị unload, subsystem đi kèm cũng được giải phóng theo, đảm bảo vòng đời đồng bộ và sạch sẽ.

Ở cấp độ thấp nhất, Local Player Subsystem (ULocalPlayerSubsystem) được gắn liền với từng người chơi cục bộ — đặc biệt quan trọng trong bối cảnh split-screen và multiplayer cục bộ. Mỗi người chơi sở hữu hệ thống subsystem riêng, cho phép quản lý input, UI state, hoặc dữ liệu người chơi cục bộ một cách độc lập. Điều này tạo ra một lớp abstraction để gameplay có thể quản lý trạng thái theo từng user mà không phải dựa vào Pawn hoặc Controller vốn thay đổi trong suốt trận.

Điểm mạnh cốt lõi của subsystem architecture nằm ở vòng đời được engine quản lý hoàn toàn, khả năng khởi tạo tự động, cũng như khả năng truy cập thống nhất thông qua các API như `GetSubsystem<T>()`. Điều này loại bỏ sự phụ thuộc vào global singleton tự viết, nâng cao độ an toàn bộ nhớ, và đảm bảo subsystem tương thích với hệ thống hot reload, multithreading, cùng reflection của Unreal. Nhờ đó, subsystem trở thành một mô hình tổ chức “service-oriented” mang tính hiện đại, cho phép các nhà phát triển xây dựng gameplay hoặc toolchain một cách modular, testable và có thể mở rộng trong các dự án quy mô lớn.

2.1.6 Module System

Module System trong Unreal Engine là nền tảng tổ chức và đóng gói mã nguồn cốt lõi, cho phép engine và các dự án game đạt được tính mô-đun, khả năng mở rộng, khả năng biên dịch chọn lọc và tính linh hoạt trong quản lý phụ thuộc. Module không chỉ là đơn vị biên dịch (compilation unit) mà còn là đơn vị triển khai logic và tài nguyên, được tích hợp chặt chẽ vào hệ thống build của Unreal (Unreal Build Tool – UBT) cũng như hệ thống reflection và runtime loading của engine.

Về bản chất, mỗi module là một gói chức năng độc lập, có thể được nạp (load) và gỡ (unload) tại runtime đối với các module hỗ trợ dynamic loading. Cơ chế này cho phép Unreal Engine tồn tại với cấu trúc phân lớp mềm dẻo, giảm thiểu sự phụ thuộc giữa các phần của engine và tạo điều kiện cho bên thứ ba mở rộng mà không cần sửa vào mã nguồn lõi. Mỗi module xác định rõ các phạm vi visibility — bao gồm Public, Private, và Protected — được thể hiện qua các thư mục “Public/Private” và khai báo trong file `.Build.cs`. Điều này tạo nên ranh giới rõ ràng giữa API được phép sử dụng bên ngoài module và implementation nội bộ.

Về phân loại, Unreal định nghĩa nhiều dạng module khác nhau tương ứng với ngữ cảnh sử dụng. Loại phổ biến nhất là Runtime Module, chứa các thành phần gameplay, engine subsystem, hoặc cơ chế vận hành game chạy trực tiếp khi trò chơi hoạt động. Các Editor Module cung cấp chức năng cho môi trường biên tập, bao gồm các công cụ, widget tùy chỉnh, và hệ thống xử lý asset; chúng tồn tại song song nhưng

tách biệt với Runtime Modules để đảm bảo build shipping không bao gồm code của editor. Ngoài ra, Unreal hỗ trợ Developer Modules (phục vụ testing, profiling, debug utilities) và Third-party Modules (đóng gói thư viện ngoài). Sự phân biệt này giúp tối ưu thời gian biên dịch, giảm kích thước build, và tăng tính bảo trì.

Một đặc trưng quan trọng của Module System là cơ chế lifecycle được quản lý qua các hàm StartupModule() và ShutdownModule() được cài đặt trong lớp triển khai IModuleInterface. Khi engine khởi động, UBT và runtime loader sẽ xác định các module cần thiết dựa trên dependency graph được mô tả trong .uproject hoặc .uplugin và .Build.cs, sau đó thực hiện khởi tạo tuần tự. Điều này cho phép module tự cấu hình dịch vụ, đăng ký subsystem, thêm extension point hoặc hook vào editor ngay trong giai đoạn startup. Module System do đó đóng vai trò như một "dependency injection primitive" ở cấp engine — mặc dù Unreal không sử dụng DI framework truyền thống, nó cung cấp khả năng mở rộng tương tự dựa trên lifecycle hooks.

Hệ thống phụ thuộc (dependency resolution) trong Unreal mang tính tĩnh trong build-time nhưng linh hoạt trong runtime. UBT phân tích các dependency Public vs Private để quyết định đường dẫn include, thứ tự biên dịch và liên kết. Ở runtime, module có thể được load thông qua FModuleManager::LoadModuleChecked hoặc các macro tiện ích, cho phép hot-reload hoặc plugin-based extension. Đặc biệt, đối với các dự án lớn, Module System cho phép tổ chức codebase thành nhiều module nhỏ — ví dụ: GameplayModule, AI, Inventory, Interaction, UI — giúp rút ngắn build incremental, tăng tốc độ iteration và giảm độ coupling giữa các hệ thống.

Đáng chú ý, Module System còn liên kết chặt chẽ với plugin architecture của Unreal. Mỗi plugin thực chất là một container của một tập module, có thể bao gồm cả runtime lẫn editor module. Điều này tạo điều kiện để phân phối chức năng dưới dạng các module độc lập, tái sử dụng được qua nhiều dự án khác nhau. Plugin và module được mô tả bằng metadata rõ ràng (như phiên bản, loại, dependency), hỗ trợ cơ chế marketplace và đảm bảo an toàn tương thích.

Tổng thể, Module System trong Unreal Engine không chỉ là một cơ chế tổ chức mã nguồn, mà còn là một kiến trúc nền móng cho phép engine vận hành như

một hệ sinh thái mở, linh hoạt và có khả năng mở rộng cao. Nó tạo ra cơ sở hạ tầng cần thiết để quản lý độ phức tạp của các dự án AAA, đồng thời cung cấp cho nhà phát triển khả năng tổ chức và mở rộng logic một cách an toàn, có cấu trúc và tối ưu cho cả build-time lẫn runtime.

2.1.7 Unreal Build Tool (UBT) và Build Modules

Unreal Build Tool (UBT) là hệ thống build chuyên dụng của Unreal Engine, đóng vai trò như một bộ biên dịch siêu cấu hình (meta-build system), thay thế cho các công cụ build truyền thống như Make, CMake hoặc MSBuild. UBT không chỉ chịu trách nhiệm biên dịch và liên kết mã nguồn, mà còn quản lý cấu trúc module, phụ thuộc, cấu hình target và tối ưu hóa quá trình build ở quy mô rất lớn. Trong nhiều khía cạnh, UBT có thể được xem như một trong những thành phần cốt lõi tạo nên khả năng phát triển dự án AAA đa nền tảng của Unreal.

UBT vận hành dựa trên một mô hình mô-đun được định nghĩa thông qua các file .Build.cs, trong đó mỗi module mô tả rõ ràng phạm vi API, dependencies và các tham số build cần thiết. Thay vì sử dụng hệ thống configure/make truyền thống, UBT sử dụng metadata được embed trong chính mã C#, cho phép developer mô tả hành vi build bằng ngôn ngữ lập trình bậc cao. Cơ chế này đem lại tính biểu đạt mạnh mẽ và khả năng kiểm tra logic build ngay trong thời gian phân tích, đồng thời cho phép plugin hoặc module tùy chỉnh mở rộng build system mà không cần chỉnh sửa phần lõi của Unreal.

Một đặc trưng quan trọng của UBT là kiến trúc Target Rules, nơi mỗi target—như Editor, Client, Server, hoặc Game—được định nghĩa bởi một lớp C# mở rộng TargetRules. Mỗi target xác định các tham số như loại build (Development, Shipping, Debug), nền tảng (Win64, Linux, Android, iOS), chính sách tối ưu hóa, hỗ trợ reflection, và workflow cần thiết để biên dịch một executable cụ thể. Hệ thống này cho phép Unreal có thể tạo các bản build khác nhau cho cùng một project mà không cần thay đổi cấu trúc mã nguồn. Chẳng hạn, Editor build bao gồm Editor Modules và các công cụ biên tập, trong khi Shipping build loại bỏ hoàn toàn code editor để tối ưu kích thước và hiệu năng.

Trong ngữ cảnh Build Modules, UBT quản lý biên dịch có điều kiện thông qua các dependency Public/Private mô tả trong module. Public Dependencies cho phép module khác include header và liên kết mã, trong khi Private Dependencies chỉ áp dụng cho nội bộ module. Mô hình phân tách như vậy giúp giảm lượng mã phải build lại, tránh việc truyền phụ thuộc không cần thiết, và đảm bảo biên giới module rõ ràng. Điều này đặc biệt quan trọng trong các dự án lớn có hàng nghìn file header, nơi việc thay đổi một file có thể gây ra một lượng rebuild lớn nếu phạm vi phụ thuộc không được kiểm soát.

UBT cũng triển khai một cơ chế include order enforcement, đảm bảo quá trình include header diễn ra theo thứ tự nghiêm ngặt để tránh lỗi build do phụ thuộc vòng hoặc header không deterministic. Kết hợp với precompiled headers (PCH) được tạo tự động, UBT đảm bảo tốc độ build nhanh và ổn định trên nhiều nền tảng khác nhau. Hệ thống build của Unreal còn bao gồm cơ chế Unity Build (gộp nhiều file C++ thành một compilation unit) để rút ngắn thời gian compile, dù developer có thể vô hiệu hóa khi cần debugging chính xác hơn.

Một yếu tố quan trọng khác là hỗ trợ đa nền tảng. UBT chứa các lớp abstraction cho từng nền tảng, mô tả compiler toolchain, linker, kiến trúc CPU, hệ điều hành và cấu hình optimization. Điều này cho phép Unreal build trên Windows, Linux, macOS, Android, iOS, PlayStation, Xbox... mà không cần developer tự xử lý build script phức tạp. Với mỗi nền tảng, UBT tự động tạo ra build graph tối ưu, kiểm tra điều kiện phụ thuộc, và cấu hình environment phù hợp.

UBT còn tích hợp sâu với hệ sinh thái plugin và module của Unreal. Khi một plugin được thêm vào project, UBT tự động nhận diện các module bên trong plugin, phân tích dependency, và đưa chúng vào build graph nếu cần thiết. Điều này tạo nên luồng mở rộng rất mạnh mẽ: bất cứ ai cũng có thể viết plugin chứa module runtime, module editor hoặc third-party library, và toàn bộ sẽ được UBT quản lý mà không cần thêm công cụ build ngoài.

Cuối cùng, UBT không chỉ là công cụ build mà còn là một orchestrator của toàn bộ pipeline compile-link-cook-package. Nó phối hợp với Unreal Header Tool

(UHT) để quét và tạo mã phản chiếu (reflection), sau đó chuyển sang phase biên dịch C++, rồi liên kết, và cuối cùng tạo ra executable hoặc các gói đóng gói phân phối. Điều này biến UBT trở thành trái tim của workflow phát triển Unreal ở quy mô chuyên nghiệp: linh hoạt, mô-đun, và tối ưu hóa toàn diện.

2.1.8 Garbage Collection và Memory Model

Hệ thống quản lý bộ nhớ của Unreal Engine — bao gồm Garbage Collection (GC), cơ chế tracking object và memory model tổng thể — là một trong những thành phần cốt lõi đảm bảo sự ổn định và an toàn của toàn bộ gameplay framework. Trong một môi trường phức tạp, gồm hàng triệu đối tượng có vòng đời khác nhau (actors, components, assets, subsystem objects), Unreal cần một phương pháp quản lý bộ nhớ vừa linh hoạt, vừa có thể kiểm soát được trong thời gian chạy. Hệ thống GC của Unreal vì thế được thiết kế như một giải pháp bán tự động (semi-managed), kết hợp ưu điểm của C++ hiệu năng cao với khả năng quản lý rác có cấu trúc.

Về bản chất, Unreal sử dụng một mô hình reference tracking-based garbage collection, trong đó mọi đối tượng thuộc họ UObject đều nằm dưới sự kiểm soát của GC. Khác với các ngôn ngữ có GC toàn phần như Java hoặc C#, Unreal không theo mô hình tracing continuous GC mà áp dụng mark-and-sweep theo chu kỳ. Mỗi đối tượng UObject khi được tạo ra thông qua cơ chế `NewObject<T>()` hoặc `ConstructObject` sẽ được đăng ký vào một global object array — trung tâm của cơ chế quản lý bộ nhớ. GC sau đó định kỳ quét qua đồ thị tham chiếu để xác định đối tượng nào còn được sử dụng và đối tượng nào có thể thu gom.

Cơ chế tracking phụ thuộc chặt chẽ vào hệ thống reflection. Mỗi lớp UCLASS, mỗi thuộc tính UPROPERTY, và mỗi container được đánh dấu (như TArray, TMap, TSet chứa UObject*) đều sinh ra metadata mô tả cách GC truy cập vào các reference. Điều này tạo nên một graph tham chiếu được engine hiểu hoàn toàn, cho phép GC có thể lèn theo các ràng buộc (root → reachable objects → leaves). Bất kỳ thuộc tính nào không nằm trong UPROPERTY hoặc không được đánh dấu đều không được GC theo dõi, dẫn đến nguy cơ thu gom nhầm hoặc rò rỉ bộ nhớ, tùy cách sử dụng. Đây

cũng là lý do Unreal buộc lập trình viên tuân thủ nghiêm ngặt quy tắc “mọi con trỏ UObject phải được quản lý qua UPROPERTY”.

Hệ thống GC của Unreal làm việc dựa trên Root Set, tập hợp các đối tượng được coi là luôn tồn tại hoặc bắt đầu của một chuỗi tham chiếu. Root Set bao gồm: các singleton nội bộ engine, các object được đánh dấu RF_RootSet, object thuộc world hiện tại, hoặc các đối tượng được giữ bởi hệ thống subsystem. GC bắt đầu từ các root này, đánh dấu tất cả đối tượng có thể truy cập, rồi quét phần còn lại để thu gom. Điều này cung cấp một cơ chế an toàn, cho phép Unreal kiểm soát vòng đời object ngay cả trong các trường hợp streaming level phức tạp hoặc chuyển cảnh real-time.

Về mặt vận hành, GC hoạt động theo chu kỳ và thường được kích hoạt dựa trên số lượng allocations hoặc thời gian tích lũy giữa các khung hình. Đây là điểm cân bằng giữa hiệu năng và độ ổn định: Unreal tránh chạy GC liên tục để không ảnh hưởng đến performance, đồng thời đảm bảo không để rác tăng quá nồng. Ngoài GC thường kỳ, engine còn có các cơ chế như Incremental GC, nơi quá trình sweep được chia nhỏ qua nhiều frame nhằm giảm spike thời gian xử lý — đặc biệt quan trọng với game console và VR nơi độ trễ frame là yếu tố then chốt.

Ngoài GC dành cho UObject, Unreal vẫn sử dụng C++ manual memory management cho các đối tượng không kế thừa từ UObject. Điều này tạo ra một mô hình bộ nhớ kép: một phần được quản lý bởi GC, phần còn lại tuân theo quy tắc RAII của C++. Các class không thuộc hệ thống UObject (như FVector, TSharedPtr, TUniquePtr, hoặc các struct thuần C++) không nằm trong hệ thống GC và phải được quản lý thủ công hoặc sử dụng smart pointer. Mô hình kép này, mặc dù phức tạp, mang lại linh hoạt: các cấu trúc nhẹ, không cần reflection, có thể là thiết kế hiệu năng cao trong gameplay mà không cần phải tham gia vào cơ chế GC.

Cơ chế GC gắn chặt với lifecycle của World và Level Streaming. Khi một level bị unload, tất cả các UObject không còn thuộc đồ thị tham chiếu của world sẽ bị đánh dấu để thu gom. Điều này đảm bảo memory footprint được kiểm soát chặt chẽ khi chuyển cảnh hoặc load/unload sublevels, tránh tích lũy memory không cần

thiết. Sự tích hợp này cũng cho thấy vai trò trung tâm của GC trong quy trình runtime của Unreal — không chỉ là một tính năng quản lý bộ nhớ, mà là một phần của pipeline vận hành của toàn bộ engine.

Một điểm đặc biệt khác là cơ chế Reference Collector và Garbage Collection Cluster. Clustering cho phép Unreal gom các đối tượng có liên hệ chặt chẽ vào một nhóm, giảm số lượng thao tác mark-and-sweep và tăng đáng kể hiệu năng cho các project lớn (như AAA open-world games). Đây là một minh chứng cho tính thực dụng của thiết kế memory model trong Unreal: tối ưu cho các trường hợp sử dụng thực tế, thay vì theo đuổi lý thuyết thuần túy.

Tổng kết lại, hệ thống Garbage Collection và Memory Model của Unreal Engine đại diện cho một kiến trúc quản lý bộ nhớ lai (hybrid) vừa an toàn, vừa hiệu năng cao. Nó kết hợp sự chính xác của tracking C++ static với sự tiện lợi của garbage collection. Nhờ tích hợp sâu với reflection, metadata, world lifecycle và gameplay framework, cơ chế GC của Unreal đảm bảo rằng các dự án có độ phức tạp lớn — từ game quy mô nhỏ đến open-world AAA — đều có thể vận hành ổn định, tối ưu và dễ bảo trì.

2.2 Unreal Editor 5 – Giao diện và Công cụ chính

2.2.1 Level Editor

Level Editor là trung tâm của môi trường phát triển trong Unreal Engine, đóng vai trò như một không gian tích hợp cho việc xây dựng, tổ chức và thao tác thế giới ảo (virtual world). Nó là công cụ mà nhà thiết kế, kỹ thuật viên, và nghệ sĩ sử dụng thường xuyên nhất, cho phép kết hợp tài sản (assets), logic gameplay và các hệ thống không gian vào một cấu trúc nhất quán. Với tư cách là giao diện chủ đạo của Unreal Editor, Level Editor không chỉ cung cấp cơ chế chỉnh sửa trực quan mà còn biểu hiện triết lý thiết kế “WYSIWYG – What You See Is What You Get” của Unreal: mọi thay đổi trong editor phản ánh ngay lập tức trên thế giới trong game.

Về mặt cấu trúc, Level Editor là một workspace đa thanh (multi-panel workspace) bao gồm Viewport, Outliner, Details Panel, Content Browser và các công cụ phụ trợ. Viewport cung cấp góc nhìn 3D với khả năng điều hướng theo mô hình camera tự do, cho phép tương tác trực tiếp với Actors thông qua các gizmo dịch chuyển, xoay và tỉ lệ. Outliner thể hiện cấu trúc phân cấp của toàn bộ level theo dạng cây, giúp người dùng theo dõi mối quan hệ và vòng đời của các đối tượng trong không gian. Details Panel cung cấp khả năng tùy chỉnh thuộc tính đối tượng dựa trên hệ thống reflection, liên kết chặt chẽ giữa giao diện và metadata của class C++. Ba thành phần này tạo nên một pipeline thao tác trực quan, từ việc lựa chọn, quan sát đến chỉnh sửa Actor.

Level Editor cũng là nơi diễn ra quy trình xây dựng không gian (world-building). Người dùng có thể tạo và quản lý các Level, Sublevel và World Partition Cells thông qua công cụ World Settings và Level Window. Ngoài việc đặt Actor, người dùng còn thao tác với ánh sáng, camera, foliage, landscape và các loại geometry. Unreal cung cấp một tập công cụ chuyên biệt như Landscape Tool, Foliage Tool, Geometry Editing Mode và Modeling Tools, cho phép tạo ra bề mặt địa hình, scatter thảm thực vật, chỉnh sửa mesh đơn giản hoặc tạo khói BSP/geometry blockout. Nhờ đó, Level Editor là môi trường thống nhất để vừa prototype gameplay vừa xây dựng thế giới ở chất lượng cuối.

Một điểm nổi bật là tính real-time feedback. Mọi thay đổi trong level — từ ánh sáng, vật liệu, animation đến gameplay logic gắn bằng Blueprint — đều được cập nhật ngay trong editor, cho phép người phát triển đánh giá thiết kế trong thời gian thực. Khi kết hợp với các cơ chế như Play-In-Editor (PIE), người dùng có thể kiểm tra hành vi gameplay trực tiếp trong bối cảnh level đang phát triển mà không cần build lại project. Điều này tạo nên một vòng lặp thiết kế nhanh, phù hợp với quy trình agile trong sản xuất game hiện đại.

Ở mức kiến trúc, Level Editor vận hành dựa trên sự kết hợp giữa Actor–Component Architecture và hệ thống Editor Modules. Các Editor Module mở rộng Level Editor bằng cách cung cấp chế độ chỉnh sửa (Editor Modes), toolbar, widget

và hành vi custom cho viewport. Nhờ Module System, nhà phát triển có thể tạo công cụ riêng, từ editor utilities đơn giản đến các hệ thống chỉnh sửa cho pipeline phức tạp, như công cụ layout UI, AI annotation, mass entity management hoặc world generation. Điều này biến Level Editor thành một nền tảng mở, có khả năng thích ứng với quy mô dự án.

Về mặt quản lý dự án lớn, Level Editor hỗ trợ hệ thống phân chia thế giới (World Partition), cho phép streaming và subdivision không gian tự động. Người dùng có thể làm việc trên một phần của thế giới mà không cần tải toàn bộ landscape khổng lồ. Khi làm việc theo nhóm, Level Editor hỗ trợ Multi-User Editing, cho phép nhiều người chỉnh sửa cùng một level trong thời gian thực, dựa trên hệ thống tracking transaction của Unreal. Điều này đặc biệt quan trọng trong các dự án AAA, nơi đội ngũ đồng đảo cần phối hợp trên cùng một bản đồ.

Một thành tố quan trọng khác là khả năng tích hợp nội dung. Thông qua Content Browser, người dùng drag-and-drop assets vào Level Editor, từ static mesh, animation sequence, sound cue đến blueprint class. Mỗi liên kết giữa Editor và hệ thống asset được quản lý bởi Asset Registry và các Editor Subsystem, đảm bảo rằng tất cả thay đổi về asset đều được phản ánh nhất quán trong level. Các engine system như Lighting, Navigation Mesh, Volumes, AI dữ liệu, physics simulation được kích hoạt hoặc bake trực tiếp trong Level Editor, biến nó thành môi trường tổng hợp kết nối mọi hệ thống gameplay.

Tổng thể, Level Editor là giao diện trực quan hóa toàn bộ kiến trúc Unreal Engine trong không gian ba chiều. Nó không chỉ là công cụ để đặt đối tượng mà còn là nền tảng hợp nhất: kết nối asset, code, blueprint và hệ thống engine vào một workflow liên tục. Sự kết hợp giữa trực quan (visual editing), cấu trúc hóa (hierarchical world representation) và extensibility (editor modules) khiến Level Editor trở thành trung tâm của trải nghiệm phát triển game trong Unreal Engine, nơi thế giới ảo được hiện thực hóa từ ý tưởng đến sản phẩm hoàn chỉnh.

2.2.2 World Outliner

World Outliner là thành phần trung tâm trong Level Editor chịu trách nhiệm biểu diễn cấu trúc phân cấp (hierarchical representation) của toàn bộ các Actor tồn tại trong một Level hoặc trong World đang được mở. Khác với Viewport, nơi người dùng quan sát thế giới ở dạng trực quan 3D, World Outliner cung cấp một hình thức trừu tượng hoá ở dạng danh sách cấu trúc cây, cho phép quản lý, tìm kiếm và thao tác Actor một cách có hệ thống. Đây là công cụ quan trọng để duy trì tổ chức của level, đặc biệt trong các dự án trung bình đến lớn, nơi số lượng Actor có thể lên tới hàng chục nghìn.

Về mặt chức năng, World Outliner hoạt động như một bản đồ cấu trúc logic của thế giới, liệt kê mọi Actor theo quan hệ cha-con hoặc theo grouping tùy chỉnh. Nó cho phép người dùng quan sát không chỉ sự tồn tại của từng Actor mà còn context hoạt động của chúng trong cấu trúc level. Với mỗi Actor, Outliner hiển thị tên, loại, icon, trạng thái bật/tắt, trạng thái hidden, và các tag phân loại, giúp người phát triển nhanh chóng định vị đối tượng trong không gian phức tạp. Cơ chế tìm kiếm theo tên, lớp và tag cũng được tích hợp, hỗ trợ mạnh mẽ cho việc điều hướng trong project lớn.

Một yếu tố quan trọng trong hoạt động của World Outliner là cách nó phản ánh Actor–Component Architecture. Mặc dù các Component không được hiển thị như đối tượng độc lập, World Outliner cho phép truy cập nhanh đến các Component thông qua Details Panel khi một Actor được chọn. Điều này tạo ra một pipeline chỉnh sửa tự nhiên: người dùng chọn Actor trong Outliner → xem cấu trúc component của Actor → chỉnh thuộc tính bằng metadata reflection. Nhờ vậy, Outliner đóng vai trò như trung tâm điều phối giữa Level Hierarchy và các công cụ chỉnh sửa thuộc tính.

Ở cấp độ tổ chức, World Outliner hỗ trợ Folder Structure nhằm phân nhóm Actor theo logic thiết kế hoặc chức năng. Các folder không ảnh hưởng đến logic runtime của game mà chỉ tồn tại trong editor, giúp giảm tải nhận thức (cognitive load) cho người phát triển. Chẳng hạn, một level có thể được tổ chức thành folder như "Environment", "Lighting", "NPC", "Triggers", "Cinematics", hoặc "Audio". Cơ chế này tương tự một hệ thống namespace, giúp duy trì tính nhất quán và dễ theo dõi với đội ngũ làm việc theo nhóm.

Khi kết hợp với các kỹ thuật hiện đại như World Partition, Outliner mở rộng chức năng để hiển thị thêm thông tin về cell streaming và trạng thái của từng phần không gian của world. Người dùng có thể quan sát cell nào đang loaded, unloaded hoặc locked để chỉnh sửa, đảm bảo rằng chỉnh sửa không gây xung đột khi nhiều người cộng tác trên cùng một bản đồ. Trong chế độ Multi-User Editing, World Outliner còn hiển thị trạng thái “đang chỉnh sửa bởi người khác”, giúp tránh collision trong chỉnh sửa và tăng tính minh bạch.

Về mặt tương tác, Outliner hỗ trợ thao tác kéo – thả để thay đổi quan hệ cha-con giữa các Actor. Điều này cực kỳ quan trọng khi làm việc với các đối tượng phức tạp như cinematic rigs, hierarchical meshes hoặc bộ trigger liên kết. Khi một Actor được set làm con của Actor khác, chúng chia sẻ một hệ tọa độ tương đối và một vòng đời in-editor, tiện cho việc tổ chức các nhóm logic. Tuy nhiên, Outliner chỉ thể hiện hierarchy trong editor; trong runtime hierarchy chỉ được công nhận nếu Component-based hoặc code tạo ra mối quan hệ rõ ràng.

Một khía cạnh quan trọng khác của World Outliner là vai trò của nó trong quản lý performance và debugging. Người dùng có thể nhanh chóng khoanh vùng những Actor ảnh hưởng đến hiệu năng như các nguồn sáng động (dynamic light), particle system nặng, blueprint nhiều logic, hoặc các Actor runtime-spawned. Kết hợp với các công cụ như Actor Filter, Search và Layer System, Outliner giúp xác định đối tượng gây vấn đề một cách nhanh chóng. Việc bật/tắt Actor tại đây cũng cho phép đánh giá tác động của từng đối tượng lên cảnh trong thời gian thực.

Ở mức kiến trúc editor, World Outliner được triển khai như một Editor Module mở rộng dựa trên Slate, sử dụng mô hình MVC: một phần quản lý dữ liệu Actor, một phần hiển thị UI dạng cây, và một phần xử lý tương tác. Hệ thống này đảm bảo rằng mọi thay đổi trong world (spawn, destroy, rename, attach) đều được phản ánh ngay lập tức lên Outliner nhờ cơ chế event-driven thông qua các delegate của engine. Cách vận hành này làm cho Outliner trở thành một công cụ không thể tách rời khỏi vòng đời và pipeline của Level Editor.

Tóm lại, World Outliner không chỉ là bảng danh sách đơn giản liệt kê Actor mà là một khung phân tích thế giới (world analytical framework). Nó cung cấp cái nhìn có cấu trúc, khả năng điều hướng hiệu quả, công cụ tổ chức dự án lớn, hỗ trợ streaming hiện đại, và đóng vai trò nền tảng trong việc kiểm soát tính ổn định và sự phức tạp của level. Trong workflow phát triển game, World Outliner là công cụ định hình cách người phát triển tư duy về cấu trúc không gian, mối quan hệ và vòng đời của đối tượng, làm cầu nối giữa thiết kế trực quan và logic nội tại của engine.

2.2.3 Details Panel

Details Panel là một thành phần cốt lõi trong Level Editor của Unreal Engine, chịu trách nhiệm hiển thị và cho phép chỉnh sửa chi tiết các thuộc tính (properties) của đối tượng được chọn. Trong workflow phát triển game, Details Panel đóng vai trò như cầu nối giữa metadata của UObject/Actor/Component và tương tác trực quan của người dùng, giúp nhà phát triển thao tác các thông số của thế giới ảo một cách trực quan, chính xác và có kiểm soát.

Về mặt cấu trúc, Details Panel dựa trên hệ thống Reflection và Property System của Unreal. Mọi class kế thừa từ UObject có thể được khai báo bằng macro UCLASS, trong khi các thuộc tính muốn hiển thị trong editor đều được đánh dấu bằng UPROPERTY với các specifier phù hợp (EditAnywhere, EditDefaultsOnly, VisibleAnywhere, v.v.). Khi một Actor hoặc Component được chọn trong Level Editor, Details Panel truy xuất metadata này, tạo ra bảng giao diện động, hiển thị từng thuộc tính kèm tooltip, kiểu dữ liệu, giá trị hiện tại và các ràng buộc (range, enum, flags). Cơ chế này cho phép editor luôn đồng bộ với vòng đời object, đảm bảo mọi thay đổi đều được áp dụng trực tiếp lên runtime instance hoặc blueprint defaults.

Một tính năng quan trọng là Categorization và Grouping. Details Panel tự động phân loại thuộc tính theo category (ví dụ: Transform, Rendering, Physics, AI, Animation), giúp người dùng dễ dàng định hướng và tìm kiếm thông số cần chỉnh sửa. Các category có thể mở rộng hoặc tùy chỉnh thông qua metadata specifiers (Category, DisplayName) trong code, tạo ra sự linh hoạt tối đa cho workflow editor. Nhờ vậy, panel không chỉ hiển thị thông số mà còn tạo ra hierarchy logic, tương tự

cách World Outliner tổ chức Actor, giúp nhà phát triển định hình mối quan hệ thuộc tính theo ngũ cành.

Details Panel cũng hỗ trợ tương tác động với dữ liệu runtime. Khi người dùng thay đổi giá trị của một property, engine sẽ tự động cập nhật object trong memory, thực hiện validation dựa trên các ràng buộc được khai báo, và kích hoạt event hoặc callback (như OnPropertyChanged) nếu cần. Điều này tạo nên một vòng lặp realtime feedback, cho phép preview ngay lập tức ảnh hưởng của các thay đổi trên thế giới, từ transform của Actor, thông số vật lý, cường độ ánh sáng, đến tham số trong blueprint logic. Cơ chế này đặc biệt quan trọng cho iterative design, khi designer cần thử nghiệm và tinh chỉnh các thông số liên tục mà không cần biên dịch lại code.

Tích hợp với Blueprint System, Details Panel còn hiển thị và chỉnh sửa các exposed variables của blueprint instance, cho phép thay đổi logic runtime mà không cần code mới. Nhờ kết nối giữa Blueprint, UPROPERTY và Details Panel, người dùng có thể thao tác trực tiếp trên instance của Actor trong level, đồng thời preserve giá trị defaults cho reusable blueprint. Cơ chế này nhấn mạnh triết lý “live editing” của Unreal: mọi chỉnh sửa có thể áp dụng ngay, đồng bộ với asset metadata và runtime instance.

Ở cấp độ workflow nâng cao, Details Panel hỗ trợ customization thông qua Editor Module và Detail Customization. Developer có thể tạo các panel riêng cho từng loại object, triển khai custom widgets (color picker, curve editor, asset reference picker, slider với range validation) và logic validation đặc thù. Điều này biến Details Panel từ một bảng thuộc tính tĩnh thành framework UI tương tác, cho phép mở rộng tính năng editor mà không phá vỡ quy trình core.

Ngoài ra, Details Panel còn tương tác với Property Editing System của Unreal, bao gồm Undo/Redo, transaction system và multi-object editing. Khi nhiều Actor được chọn, panel hiển thị các thuộc tính chung, cho phép thay đổi đồng loạt, đồng thời bảo đảm consistency và rollback an toàn. Hệ thống này được tích hợp chặt với Editor Subsystem, đảm bảo tính toàn vẹn dữ liệu và khả năng cộng tác multi-user.

Tổng kết, Details Panel là trung tâm thao tác thuộc tính trong Level Editor, nơi metadata của Unreal Engine được trực quan hóa và quản lý một cách linh hoạt. Nó không chỉ hiển thị giá trị, mà còn cung cấp môi trường để kiểm soát lifecycle, validate dữ liệu, thực hiện preview realtime, và mở rộng thông qua editor customization. Nhờ đó, Details Panel đóng vai trò chiến lược trong workflow phát triển game, kết nối sâu sắc giữa code, blueprint, và thiết kế trực quan trong editor, tạo điều kiện cho iterative design, multi-user collaboration và modular workflow.

2.2.4 Content Browser

Content Browser là công cụ quản lý tài nguyên (assets) trung tâm của Unreal Engine, đóng vai trò như trung tâm điều phối nội dung số trong toàn bộ workflow phát triển game. Nó cung cấp một giao diện trực quan cho phép người dùng khám phá, tổ chức, tạo mới, chỉnh sửa, và gắn kết các asset vào level hoặc blueprint. Content Browser không chỉ là một file manager truyền thống mà là một framework quản lý metadata, dependency và asset lifecycle gắn chặt với engine runtime.

Về mặt cấu trúc, Content Browser hiển thị asset theo các folder và collection, đồng thời cung cấp các chế độ xem dạng danh sách, lưới (grid view), hoặc tree view. Mỗi asset trong Unreal — từ Static Mesh, Skeletal Mesh, Material, Texture, Sound Cue, Blueprint, Animation đến Particle System — đều được đăng ký trong Asset Registry, nơi lưu trữ metadata chi tiết như loại asset, đường dẫn, tags, class, version và dependency. Khi người dùng tương tác với asset trong Content Browser, engine truy vấn Asset Registry để hiển thị đầy đủ thông tin và cung cấp các thao tác hợp lệ như drag-and-drop vào Viewport, level Outliner, hoặc blueprint graph.

Một chức năng trọng yếu là quản lý và tổ chức asset. Content Browser hỗ trợ folder hierarchy để phân nhóm asset theo logic dự án, đồng thời cung cấp Collections và Smart Collections cho phép tạo các tập hợp động dựa trên tags, type, hoặc property. Điều này đặc biệt hữu ích trong các dự án quy mô lớn, nơi số lượng asset có thể lên tới hàng chục nghìn file. Nhờ vậy, designer và artist có thể truy cập nhanh các tài nguyên cần thiết mà không mất thời gian tìm kiếm thủ công.

Content Browser cũng tích hợp workflow creation và import asset. Người dùng có thể import trực tiếp từ file system, kéo-thả asset vào engine, hoặc tạo asset mới từ template có sẵn (ví dụ: Material Instance, Blueprint Class). Khi import, Unreal tự động generate metadata và thực hiện validation, đảm bảo rằng asset tuân thủ chuẩn định dạng và có thể sử dụng trực tiếp trong editor. Đồng thời, Content Browser quản lý redirectors và reference fixup, giữ cho các reference giữa asset luôn nhất quán khi có refactor hoặc di chuyển file, điều này rất quan trọng trong việc bảo trì dự án lớn.

Một yếu tố nổi bật khác là tương tác runtime với asset. Content Browser không chỉ hiển thị asset mà còn cho phép thực thi các hành động như edit, duplicate, migrate, cook, hoặc reimport. Khi kết hợp với hệ thống Blueprint, người dùng có thể kéo asset vào graph để tạo node tương ứng, tạo nên một luồng liên kết trực tiếp giữa asset, logic gameplay và level. Cơ chế này nhấn mạnh triết lý “live content workflow” của Unreal, nơi mọi tài nguyên đều có thể được thao tác, kiểm thử, và preview ngay lập tức.

Ở cấp độ kiến trúc, Content Browser là Editor Module, được triển khai dựa trên framework Slate và gắn với Editor Subsystems. Điều này cho phép mở rộng thông qua custom asset types, asset actions, và asset editor (ví dụ: Material Editor, Animation Editor, Niagara Editor). Mỗi extension đều tích hợp trực tiếp vào Content Browser thông qua API, hỗ trợ developer hoặc studio mở rộng khả năng quản lý và thao tác asset theo nhu cầu riêng. Ngoài ra, Content Browser tích hợp chặt với Source Control, giúp quản lý versioning, locking, và collaboration trong môi trường multi-user, đảm bảo tính toàn vẹn dữ liệu trong các dự án team.

Một khía cạnh quan trọng là dependency và reference visualization. Content Browser cho phép truy vết reference, hiển thị asset nào được sử dụng bởi level hoặc blueprint nào, từ đó giúp tối ưu memory footprint, giảm rủi ro leak hoặc asset orphaned. Kết hợp với Editor Utility Widgets, người dùng có thể tạo công cụ custom để batch rename, batch fix references, hoặc batch migrate asset, tăng hiệu quả workflow trong dự án lớn.

Tóm lại, Content Browser là trung tâm quản lý và vận hành tài nguyên của Unreal Engine, nơi asset metadata, dependency, workflow creation và runtime integration được tổ chức một cách thống nhất. Nó cung cấp khả năng điều hướng nhanh, tổ chức có cấu trúc, kiểm soát dependency, và mở rộng thông qua editor module. Trong workflow phát triển game, Content Browser không chỉ là công cụ quản lý file mà còn là nền tảng để gắn kết asset, level, blueprint và gameplay logic, đóng vai trò then chốt trong việc duy trì tính ổn định, hiệu quả và khả năng mở rộng của dự án.

2.2.5 Material Editor

Material Editor trong Unreal Engine là một công cụ chuyên dụng, cho phép nhà phát triển và artist xây dựng, chỉnh sửa và quản lý vật liệu (materials) sử dụng trong rendering pipeline của engine. Nó đóng vai trò như một graph-based shader editor, nơi logic hiển thị bề mặt (surface shading), ánh sáng, và các hiệu ứng vật liệu được thiết kế bằng các node trực quan thay vì viết code shader thủ công. Material Editor là cầu nối giữa artistic workflow và hệ thống rendering hiện đại, cho phép tạo ra vật liệu với độ chân thực cao, hiệu ứng động, và tối ưu cho cả realtime lẫn cinematic rendering.

Về mặt cấu trúc, Material Editor triển khai một node-based graph system, trong đó mỗi node đại diện cho một toán tử, texture sample, parameter, hoặc hàm tính toán shading. Các node này có thể được kết nối với nhau để xác định cách ánh sáng tương tác với bề mặt của mesh, bao gồm các thông số như Base Color, Metallic, Specular, Roughness, Normal, Emissive Color, Opacity, và nhiều thông số nâng cao khác. Cách tiếp cận đồ thị này vừa trực quan cho designer, vừa linh hoạt cho kỹ sư shader, vì toàn bộ logic rendering được biểu diễn bằng đồ thị có thể mở rộng, reusable, và parameter-driven.

Một tính năng quan trọng của Material Editor là Material Parameters và Instance System. Thông qua việc khai báo các parameter (Scalar, Vector, Texture), người dùng có thể tạo Material Instance từ một Material gốc (Parent Material). Material Instance cho phép thay đổi các thông số mà không cần biên dịch lại toàn bộ

shader, giảm đáng kể thời gian iteration và tăng hiệu quả workflow. Điều này đặc biệt hữu ích trong các dự án lớn, nơi nhiều asset hoặc actor share cùng một base material nhưng yêu cầu biến thể về màu sắc, độ nhám, hay texture.

Material Editor còn tích hợp real-time preview. Người dùng có thể xem trước vật liệu trên các mesh mẫu (Preview Mesh) hoặc trên mesh thực trong level, với khả năng thay đổi lighting environment, exposure, reflection capture và post-process effect. Preview này được render bằng same rendering pipeline như runtime, đảm bảo mọi thay đổi trong editor phản ánh chính xác kết quả cuối cùng. Sự tích hợp này cung cấp feedback tức thì, giúp iterative design trở nên hiệu quả và giảm thiểu lỗi giữa preview và runtime.

Ở cấp độ kiến trúc, Material Editor là Editor Module tích hợp sâu với Shader Compiler và Asset System. Khi người dùng tạo hoặc chỉnh sửa material, engine sẽ sinh mã HLSL tương ứng, biên dịch và nạp shader vào GPU runtime. Unreal quản lý dependency giữa material, texture, blueprint và mesh assets, đảm bảo rằng mọi thay đổi đều propagate đúng cách trong asset registry và level. Hệ thống này kết hợp với Content Browser, cho phép kéo-thả textures, samplers, hoặc functions trực tiếp vào graph, tạo nên một workflow thống nhất giữa asset management và material authoring.

Material Editor còn hỗ trợ Material Functions và Layered Materials, cho phép tái sử dụng các module shading phức tạp và kết hợp nhiều vật liệu thành một thể thống nhất. Material Function là một dạng subgraph reusable, giúp designer tạo các hiệu ứng phức tạp như wet surface, dirt, wear, hoặc procedural blending mà không lặp lại logic. Layered Materials kết hợp nhiều material instance với blending rules, hỗ trợ các hiệu ứng composite phong phú, như terrain blending, decal overlay, hoặc multi-material mesh.

Một khía cạnh quan trọng khác là optimization và scalability. Material Editor cung cấp các công cụ như Shader Complexity View, LOD-based optimization, và static switch parameters, giúp nhà phát triển cân bằng giữa visual fidelity và

performance. Các thông số này rất quan trọng cho các dự án AAA hoặc open-world, nơi shader efficiency ảnh hưởng trực tiếp đến framerate và memory footprint.

Tóm lại, Material Editor là trung tâm thiết kế và quản lý vật liệu trong Unreal Engine, kết hợp trực quan hóa, node-based logic, parameter-driven instances, real-time preview, và tight integration với shader compiler và asset management. Nó không chỉ phục vụ nhu cầu mỹ thuật mà còn liên kết chặt chẽ với kỹ thuật rendering và workflow game development, cho phép iterative design nhanh chóng, reusable workflow, và tối ưu hiệu năng. Material Editor là minh chứng cho triết lý của Unreal: kết hợp trực quan, linh hoạt và mạnh mẽ, đáp ứng cả designer và programmer trong quy trình phát triển hiện đại.

2.2.6 Animation Editor

Animation Editor là một trong những công cụ trung tâm của Unreal Engine, chuyên dụng cho việc tạo, chỉnh sửa và quản lý các animation asset, bao gồm Skeletal Mesh Animation, Blend Spaces, Animation Montages, và Control Rig. Nó đóng vai trò như cầu nối giữa rigged character data, skeletal system, và gameplay framework, cho phép nhà phát triển hoặc animator thiết kế chuyển động sinh động và tích hợp trực tiếp vào gameplay hoặc cinematic sequence.

Về mặt cấu trúc, Animation Editor triển khai một graph-based và timeline-based interface, kết hợp giữa việc thao tác trực quan trên keyframes, curve editor, và node graph. Khi một Skeletal Mesh được load vào editor, Animation Editor hiển thị skeleton hierarchy và bones, cho phép người dùng thao tác các khớp (joint) để chỉnh pose, thêm keyframe, hoặc áp dụng animation sequence. Timeline cung cấp khả năng quản lý các keyframe theo thời gian, hỗ trợ in-between interpolation, easing, và snapping, giúp tạo ra các chuyển động mượt mà và kiểm soát chính xác timing.

Một điểm quan trọng là tích hợp với Animation Blueprint (AnimBP). Animation Editor không chỉ chỉnh sửa animation asset đơn lẻ mà còn liên kết trực tiếp với các blueprint logic điều khiển skeletal mesh. Thông qua AnimGraph và State Machines, người dùng có thể thiết lập các state transition, blend logic, và procedural control, cho phép characters phản ứng linh hoạt theo gameplay input, physics

simulation, hoặc AI behavior. Điều này nhấn mạnh triết lý data-driven animation, nơi animation logic được cấu trúc như một hệ thống modular, có thể tái sử dụng và mở rộng.

Animation Editor cũng hỗ trợ Blend Space và Animation Montages, cho phép kết hợp nhiều animation sequence dựa trên parameter input hoặc gameplay events. Blend Space được sử dụng để nội suy giữa các animation, ví dụ như chạy, đi, và sprint dựa trên tốc độ và hướng của character. Animation Montages hỗ trợ trigger các animation cụ thể trong gameplay, như attack combo, ability animation, hoặc cinematic actions, với khả năng blend và sync timing cao. Cơ chế này giúp tạo ra trải nghiệm động và liền mạch cho nhân vật trong game.

Về mặt kỹ thuật, Animation Editor tích hợp sâu với Skeletal Mesh, Physics Asset, và Control Rig. Nó hỗ trợ chỉnh sửa skeleton hierarchy, retarget animation giữa skeleton khác nhau, preview ragdoll physics, và kiểm tra constraints của physics asset. Control Rig cung cấp hệ thống procedural rigging trực tiếp trong editor, cho phép tạo animation procedural, procedural IK/FK switching, hoặc pose authoring mà không cần xuất sang DCC tool bên ngoài. Điều này biến Animation Editor thành một hub toàn diện cho tất cả quy trình animation, từ thiết kế, chỉnh sửa, đến tích hợp runtime.

Một yếu tố quan trọng khác là real-time preview và interactive feedback. Khi chỉnh sửa animation, editor hiển thị chuyển động ngay trên Skeletal Mesh trong context của level hoặc scene, bao gồm lighting, camera perspective, và collision. Người dùng có thể tương tác trực tiếp với bone, spline control, hoặc mesh attachment, giúp iterative design nhanh chóng, giảm thiểu lỗi khi triển khai vào runtime. Hệ thống này cũng hỗ trợ multi-layered animation, additive animation, và animation masking, cho phép composite animation phức tạp mà vẫn giữ hiệu năng tối ưu.

Ở cấp độ kiến trúc, Animation Editor là một Editor Module dựa trên framework Slate và Editor Subsystem. Nó kết hợp với Asset Registry, Content Browser, và Level Editor, cho phép drag-and-drop animation vào blueprint, mesh, hoặc level instance. Mọi thay đổi trong asset đều được quản lý thông qua transaction

system, hỗ trợ undo/redo, version control, và multi-user collaboration. Điều này đảm bảo workflow animation đồng bộ, reproducible, và scalable cho các dự án lớn.

Tóm lại, Animation Editor là trung tâm thiết kế và quản lý chuyển động trong Unreal Engine, kết hợp trực quan hóa skeleton, timeline-based editing, graph-driven logic, real-time preview, và tight integration với physics, control rig, blueprint, và asset system. Nó cho phép animator và developer tạo ra chuyển động nhân vật mượt mà, modular và data-driven, đồng thời tích hợp trực tiếp với gameplay và cinematic, làm cho Animation Editor trở thành công cụ không thể thiếu trong workflow phát triển game hiện đại.

2.2.7 Niagara Editor

Niagara Editor là công cụ chính của Unreal Engine để tạo, chỉnh sửa và quản lý hệ thống particle và visual effects (VFX), thay thế Cascade truyền thống với khả năng lập trình procedural mạnh mẽ hơn. Nó được thiết kế như một node-based VFX graph editor, kết hợp giữa artistic workflow và khả năng lập trình data-driven, giúp nhà phát triển tạo ra các hiệu ứng hình ảnh phức tạp, từ hạt đơn giản, khói, lửa, láp lánh, đến các hệ thống particle vật lý động, tích hợp trực tiếp vào gameplay hoặc cinematic sequence.

Về mặt cấu trúc, Niagara Editor sử dụng multi-graph architecture, bao gồm các graph như Emitter Graph, Particle Update Graph, Particle Spawn Graph, và System Graph. Mỗi graph cho phép lập trình hành vi particle theo giai đoạn: spawn (khởi tạo), update (cập nhật), và render (hiển thị). Thông qua các node, người dùng có thể điều khiển thuộc tính particle như vị trí, velocity, acceleration, lifetime, color, scale, rotation, và texture UV, cũng như các tương tác vật lý với môi trường, forces, collisions hoặc attractors.

Một tính năng nổi bật là Niagara Parameter và Data Interface System. Người dùng có thể khai báo các parameter (Scalar, Vector, Color, Texture, Actor, Curve) để điều khiển particle system từ level, blueprint, hoặc gameplay logic. Data Interfaces cho phép Niagara truy xuất dữ liệu runtime từ các nguồn khác nhau, ví dụ như skeletal mesh, collision volume, audio, hoặc runtime particle array, giúp tạo ra VFX

procedural, dynamic, và interactive. Cách tiếp cận này thể hiện triết lý data-driven VFX, nơi hiệu ứng được điều khiển bởi input từ gameplay hoặc environment.

Niagara Editor tích hợp real-time preview và GPU simulation. Người dùng có thể xem trực tiếp particle system trên preview mesh, trong editor viewport hoặc trong context của level, với lighting, shadow và post-process effects. Hỗ trợ GPU particles giúp Niagara quản lý hàng triệu particle với hiệu năng cao, giảm thiểu bottleneck CPU và đảm bảo khả năng scale cho game AAA hoặc open-world. Đồng thời, editor cung cấp các công cụ debug, visualization, và performance profiling như particle count, update rate, và render complexity, hỗ trợ tối ưu hóa VFX trong dự án lớn.

Ở cấp độ kiến trúc, Niagara Editor là Editor Module tích hợp sâu với Content Browser, Level Editor, Blueprint Editor, và Sequencer. Người dùng có thể drag-and-drop particle system vào level hoặc blueprint, bind parameter tới gameplay events, hoặc trigger Niagara System từ Sequencer để tạo cinematic effects. Mọi thay đổi được quản lý thông qua transaction system, hỗ trợ undo/redo, multi-user collaboration và version control, đảm bảo workflow procedural VFX đồng bộ và reproducible.

Niagara còn hỗ trợ modular design và reusable templates. Người dùng có thể tạo Emitter templates, System templates, và Function scripts, cho phép tái sử dụng logic particle trong nhiều hệ thống khác nhau mà không cần lặp lại từ đầu. Điều này đặc biệt hữu ích cho pipeline production, nơi các team artist và programmer cần phối hợp để duy trì tính nhất quán của hiệu ứng trên nhiều level và project.

Tóm lại, Niagara Editor là trung tâm tạo và quản lý visual effects trong Unreal Engine, kết hợp node-based procedural workflow, parameter-driven design, multi-graph architecture, GPU simulation, real-time preview và tight integration với editor system khác. Nó cho phép designer và programmer tạo ra VFX phức tạp, interactive, scalable và dễ bảo trì, trở thành một công cụ then chốt trong pipeline phát triển game và cinematic hiện đại.

2.2.8 Sequencer

Sequencer trong Unreal Engine là công cụ chính để tạo, chỉnh sửa và quản lý cinematic sequences, cutscenes, và in-game scripted events, đóng vai trò như một

non-linear editor (NLE) tích hợp trực tiếp vào engine. Nó kết hợp mạnh mẽ giữa timeline-based editing, keyframe animation, và data-driven control, cho phép nhà phát triển và artist lập trình các sự kiện diễn ra theo thời gian trong level, từ chuyển động camera, animation của nhân vật, đến trigger particle system hoặc sound cue.

Về mặt cấu trúc, Sequencer triển khai timeline-based track system, nơi mỗi actor, camera, hoặc object được gắn các tracks biểu diễn các thuộc tính cần thay đổi theo thời gian. Track có thể là transform track (vị trí, rotation, scale), animation track (skeletal animation, blend space, montage), event track (trigger custom events), hoặc property track (chỉnh thuộc tính blueprint, material parameters). Thông qua keyframe, user có thể thiết lập giá trị cụ thể tại thời điểm xác định, và engine sẽ nội suy các giá trị giữa các keyframe để tạo chuyển động mượt mà, đảm bảo tính real-time và cinematic fidelity.

Một điểm nổi bật của Sequencer là tight integration với Animation System, Niagara, và Material System. Các tracks có thể liên kết trực tiếp đến animation blueprint, Niagara System hoặc material parameter, cho phép designer tạo cinematic effects đồng bộ và interactive. Ví dụ, một explosion có thể trigger particle system, thay đổi ánh sáng, và tạo shake camera đồng thời, tất cả đều được lập trình qua Sequencer mà không cần viết code runtime thủ công.

Sequencer cũng hỗ trợ nested sequences và sub-sequences, cho phép chia một cinematic dài thành nhiều đoạn nhỏ, reusable và modular. Điều này rất quan trọng trong các dự án AAA hoặc cinematic dài, nơi cần quản lý nội dung phức tạp, đồng thời tăng khả năng phối hợp nhóm và tái sử dụng sequences. Nested sequences có thể chứa animation, camera cuts, hoặc logic event, và tất cả được đồng bộ theo master timeline, giúp workflow cinematic trở nên linh hoạt và scalable.

Về tương tác, Sequencer cung cấp real-time preview và scrubbing, nơi user có thể xem cảnh quay trong viewport, tương tác với camera, hoặc kiểm tra animation và VFX đồng thời. Các công cụ như cinematic camera, camera cuts, cinematic tracks, easing curves, và keyframe tangents giúp tạo chuyển động chính xác, cinematic quality, đồng thời hỗ trợ iterative design mà không cần render offline. Sequencer

cũng tích hợp với timeline snapping, curve editor, và keyframe manipulation tools, giúp người dùng tinh chỉnh timing và pacing một cách chính xác.

Ở cấp độ kiến trúc, Sequencer là một Editor Module hoạt động dựa trên framework Slate, tích hợp với Level Editor, Content Browser, Animation Editor, Niagara Editor, và Blueprint System. Các sequencer asset được quản lý như tài nguyên (asset) trong Content Browser, cho phép drag-and-drop vào level, blueprint, hoặc runtime system. Sequencer cũng hỗ trợ undo/redo, multi-user collaboration, và version control, đảm bảo workflow cinematic đồng bộ và reproducible cho cả đội ngũ phát triển.

Một khía cạnh quan trọng khác là event-driven and data-driven control. Sequencer có khả năng trigger blueprint events, call functions, hoặc set parameter values dựa trên timeline. Điều này giúp tạo cinematic logic hoặc gameplay scripting phức tạp mà vẫn trực quan, giúp designer và developer kiểm soát hành vi của world và actor mà không cần viết code runtime thủ công.

Tóm lại, Sequencer là trung tâm quản lý cinematic và timeline-based events trong Unreal Engine, kết hợp timeline editing, keyframe animation, real-time preview, modular nested sequences, tight integration với animation, VFX, material, và blueprint. Nó cho phép nhà phát triển tạo cinematic sequences, cutscenes, và gameplay-driven events chất lượng cao, interactive, scalable, và dễ bảo trì, trở thành công cụ không thể thiếu trong pipeline production của Unreal Engine.

2.2.9 MetaHuman tools

MetaHuman Tools là tập hợp công cụ tiên tiến của Unreal Engine, phục vụ việc tạo, tùy chỉnh, và triển khai digital human (con người số) với độ chân thực cao, từ ngoại hình đến animation biểu cảm, đồng thời tích hợp trực tiếp vào pipeline game hoặc cinematic. MetaHuman Tools đại diện cho sự kết hợp giữa procedural generation, high-fidelity rendering, rigging automation, và animation workflow, cung cấp giải pháp toàn diện cho việc xây dựng nhân vật chất lượng AAA mà không yêu cầu quá nhiều thời gian thủ công trong 3D modeling hay rigging.

Về mặt cấu trúc, MetaHuman Tools bao gồm MetaHuman Creator, Quixel Bridge integration, và các plugin trong Unreal Editor. MetaHuman Creator là ứng dụng web-based cho phép designer tùy chỉnh chi tiết nhân vật: shape khuôn mặt, tông da, màu tóc, mắt, răng, và các chi tiết phụ như lông mày, lông mi. Sau khi thiết kế, nhân vật có thể được xuất sang Unreal Engine thông qua Quixel Bridge, tạo asset MetaHuman hoàn chỉnh, bao gồm Skeletal Mesh, LODs, Material, Hair & Fur System (Groom), và rig biểu cảm mặt (Facial Rig).

Một điểm quan trọng của MetaHuman Tools là High-Fidelity Rigging and Facial Animation Integration. Mỗi MetaHuman được tích hợp với control rig chi tiết cho cả cơ thể và khuôn mặt, hỗ trợ animation procedural, mocap retargeting, và live performance capture. Facial Rig được tối ưu cho các biểu cảm phức tạp, với control attributes cho jaw, lips, eyes, cheeks, và micro-expressions, giúp tạo chuyển động tự nhiên và sinh động. Tích hợp với Live Link Face hoặc motion capture devices cho phép stream animation trực tiếp vào nhân vật trong Unreal Editor, phục vụ cinematic hoặc gameplay.

MetaHuman Tools còn kết hợp chặt chẽ với Material System và Hair/Fur System. Nhân vật được trang bị vật liệu chuẩn PBR, với textures phân giải cao, subsurface scattering cho da, và shaders tối ưu cho mắt, răng, và môi. Hair và fur sử dụng Groom System, hỗ trợ simulation, physics, và real-time lighting, đảm bảo realism trong mọi môi trường ánh sáng. Tất cả đều có thể chỉnh sửa và preview trực tiếp trong Level Editor hoặc Sequencer, tạo khả năng iterative design và integration vào cinematic hoặc gameplay sequence.

Về mặt workflow, MetaHuman Tools hỗ trợ scalability và modularity. Một MetaHuman asset có thể reuse skeleton, control rig, material instance, hoặc animation blueprint cho nhiều nhân vật khác nhau, giảm đáng kể thời gian sản xuất khi tạo cast nhân vật lớn cho game hoặc cinematic. Integration với Content Browser, Animation Editor, Sequencer, và Blueprint Editor cho phép drag-and-drop nhân vật, gán animation, trigger events, hoặc setup interactive gameplay một cách liền mạch.

Một yếu tố nổi bật khác là real-time preview và performance optimization. MetaHuman asset được tối ưu hóa cho cả runtime và cinematic, với LOD management, GPU skinning, hair simulation, và material optimization. Điều này giúp đảm bảo nhân vật chất lượng cao có thể chạy mượt trong môi trường game, cinematic, hoặc VR/AR, mà vẫn giữ được fidelity chi tiết trên da, tóc, và mắt.

Tóm lại, MetaHuman Tools là nền tảng toàn diện để tạo và quản lý nhân vật số chất lượng cao trong Unreal Engine, kết hợp procedural creation, high-fidelity rigging, material & hair system, animation integration, và real-time preview. Nó rút ngắn thời gian sản xuất, tăng khả năng reuse, đồng thời cho phép các team designer, animator và developer tạo ra nhân vật chân thực, interactive, và cinematic-ready, trở thành một công cụ then chốt trong pipeline phát triển game AAA và cinematic hiện đại.

2.3 Hệ thống Gameplay Framework

2.3.1 GameMode

GameMode là thành phần trung tâm của **Gameplay Framework** trong Unreal Engine, chịu trách nhiệm xác định các quy tắc, logic, và flow của một game hoặc level. Nó đóng vai trò như **giám sát toàn cục (global overseer)** cho gameplay, định nghĩa cách game bắt đầu, cách actor tương tác, cách người chơi và AI được quản lý, cũng như các điều kiện chiến thắng, thất bại hoặc trạng thái trò chơi. GameMode tồn tại duy nhất trên server trong các dự án multiplayer và không replicated sang client, nhấn mạnh vai trò quản lý logic **server-authoritative gameplay**.

Về mặt kiến trúc, GameMode kế thừa từ AGameModeBase hoặc AGameMode, cung cấp các **extension points** để cấu hình:

1. **Default Pawn Class:** Xác định loại Pawn mà người chơi sẽ sở hữu khi spawn. GameMode có thể lựa chọn default Pawn dựa trên team, chế độ chơi, hoặc custom logic spawn.
2. **PlayerController Class:** Liên kết controller để quản lý input và tương tác của người chơi, định nghĩa cách người chơi điều khiển Pawn.

3. **HUD Class:** Xác định giao diện hiển thị, như health bar, score, mini-map, hoặc crosshair.
4. **GameState Class:** Liên kết trạng thái chung (replicated) giữa server và client, bao gồm score, timer, và trạng thái gameplay khác.

GameMode là **nguồn quyết định luồng gameplay**, bao gồm các sự kiện như StartPlay(), HandleMatchHasStarted(), RestartPlayer(), và EndMatch(). Các hàm này được gọi trong vòng đời level, đảm bảo toàn bộ logic game được khởi tạo, chạy, và kết thúc theo trình tự xác định. Bên cạnh đó, GameMode có thể override các phương thức spawn, respawn, và team assignment để custom gameplay behavior.

Một điểm quan trọng là GameMode hoạt động song song với **PlayerController** và **Pawn**, tạo thành mô hình **Player-Pawn-Controller**. GameMode quyết định Pawn nào spawn, PlayerController nào sở hữu Pawn, và luật chơi áp dụng cho actor đó. Đồng thời, GameMode cũng phối hợp với AIController cho các NPC, thiết lập behavior tree, perception system, và navigation system, đảm bảo logic của AI phù hợp với gameplay rules.

GameMode còn tích hợp **matchmaking, scoring, và timer**. Nó có thể quản lý số lượng người chơi tối đa, phân team, theo dõi kills/deaths, tính toán thắng-thua, và điều khiển round hoặc match duration. Trong các dự án multiplayer, GameMode là **server-authoritative**, chịu trách nhiệm điều phối state và replication sang GameState để client nhận được thông tin đồng bộ về gameplay.

Ở cấp độ mở rộng, GameMode cho phép **blueprint-based customization**, enabling rapid prototyping. Designer có thể tạo các blueprint GameMode để thiết lập rules, event hooks, spawn logic, và scoring system mà không cần viết code C++ từ đầu. Điều này nhấn mạnh triết lý **data-driven gameplay framework**, nơi logic cao cấp có thể cấu hình thông qua blueprint, vẫn bảo toàn kiến trúc C++ core.

Tóm lại, **GameMode là trụ cột quản lý gameplay trong Unreal Engine**, chịu trách nhiệm xác định luật chơi, quản lý flow, quyết định các actor liên quan, và phối hợp với các thành phần như PlayerController, Pawn, GameState, và HUD. Nó cung cấp môi trường để triển khai server-authoritative, modular, và data-driven

gameplay, đóng vai trò then chốt trong việc đảm bảo tính nhất quán, linh hoạt, và khả năng mở rộng của dự án game.

2.3.2 GameState và PlayerState

Trong **Gameplay Framework** của Unreal Engine, **GameState** và **PlayerState** đóng vai trò quan trọng trong việc **quản lý và đồng bộ trạng thái gameplay** giữa server và client, đặc biệt trong môi trường multiplayer. Hai lớp này đảm bảo rằng thông tin về game và người chơi được lưu trữ, cập nhật và replicated một cách nhất quán, đồng thời tách biệt các dữ liệu server-authoritative và dữ liệu client-aware để tăng tính modular và scalability.

GameState (kế thừa từ AGameStateBase) là lớp chịu trách nhiệm lưu trữ trạng thái **toàn cục của trò chơi**, bao gồm các thông tin như thời gian match, score tổng, team assignment, round state, và các event chung. Khác với GameMode, GameState **được replicated sang client**, cho phép mọi client quan sát trạng thái game một cách nhất quán mà không thay đổi logic server-authoritative.

Các chức năng chính của GameState bao gồm:

1. **Tracking Global State:** Quản lý thông tin chung như match timer, score, countdown, và trạng thái round (ví dụ: pre-match, in-progress, post-match).
2. **Replication và Synchronization:** Dữ liệu trong GameState được replicate sang tất cả client, giúp HUD, UI, và các hệ thống khác hiển thị thông tin chính xác.
3. **Team Management:** GameState lưu trữ thông tin về team assignment, danh sách người chơi trong từng team, và điểm số của team.
4. **Integration với GameMode:** GameMode quyết định rules và logic, nhưng GameState giữ dữ liệu runtime để client có thể đồng bộ hiển thị, tạo nên separation-of-concerns giữa **logic server** và **data client**.

Nhờ GameState, các client có thể theo dõi trạng thái game mà không cần quyền can thiệp vào logic server, đảm bảo **fair gameplay** trong các dự án multiplayer.

PlayerState (kế thừa từ APlayerState) lưu trữ **thông tin cá nhân của từng người chơi**, bao gồm tên, score, kills, deaths, team affiliation, ping, và các thuộc tính

custom do developer định nghĩa. PlayerState là **replicated object**, tồn tại cả trên server và client, cho phép các hệ thống UI, HUD hoặc leaderboard hiển thị thông tin người chơi một cách chính xác.

Các điểm quan trọng của PlayerState bao gồm:

1. **Persistent Player Data**: Thông tin được giữ nguyên kể cả khi người chơi respawn hoặc Pawn của họ bị hủy. Điều này tách biệt giữa **actor đại diện trong world (Pawn)** và **dữ liệu lâu dài của người chơi (PlayerState)**.
2. **Replication và Client Awareness**: PlayerState được replicate sang client, cung cấp dữ liệu realtime cho UI, leaderboards, hoặc scoreboards.
3. **Integration với GameState và GameMode**: PlayerState được quản lý bởi GameMode khi spawn, phân team, hoặc tính điểm, và dữ liệu được lưu trong GameState để phục vụ cho toàn bộ session.
4. **Extensibility**: Developer có thể mở rộng PlayerState bằng cách thêm các thuộc tính custom như health bonus, power-ups, hoặc các statistics phức tạp, phục vụ gameplay và analytics.

Mối quan hệ giữa GameState và PlayerState

- **GameState ↔ PlayerState**: GameState giữ thông tin toàn cục, trong khi PlayerState lưu trữ dữ liệu từng người chơi. GameState thường duy trì danh sách tất cả PlayerState hiện có trong session, từ đó tổng hợp điểm số, team, hoặc trạng thái trận đấu.
- **Separation-of-Concerns**: GameMode chịu trách nhiệm logic server-authoritative, GameState chịu trách nhiệm đồng bộ dữ liệu toàn cục, và PlayerState đảm bảo thông tin cá nhân được preserve qua các Pawn hoặc respawn cycles.
- **Replication Hierarchy**: PlayerState được replicate sang client thông qua GameState, hoặc trực tiếp từ server, giúp UI hiển thị chính xác thông tin cá nhân mà không làm ảnh hưởng logic gameplay server.

Tóm lại, **GameState và PlayerState là nền tảng quản lý dữ liệu trong Unreal Engine**, đặc biệt quan trọng cho các dự án multiplayer. GameState cung cấp

dữ liệu toàn cục cho client, trong khi PlayerState lưu trữ thông tin cá nhân lâu dài của người chơi. Sự phân tách này đảm bảo **modularity, scalability, và fair gameplay**, đồng thời tạo tiền đề cho việc mở rộng các tính năng như leaderboard, team logic, scoring, và analytics.

2.3.3 Actor Lifecycle

Trong Unreal Engine, Actor là đơn vị cơ bản đại diện cho tất cả đối tượng trong thế giới 3D, từ nhân vật, vật thể tương tác, đến camera, light hay trigger volume. Hiểu rõ Actor Lifecycle là điều kiện tiên quyết để phát triển gameplay, quản lý memory, và tích hợp các subsystem như physics, rendering, AI, và input một cách hiệu quả. Actor Lifecycle định nghĩa các trạng thái, các callback, và vòng đời từ khi spawn đến khi destroy, đảm bảo engine có thể quản lý memory, replication, và interaction đúng cách.

2.3.3.1 Spawn và Initialization

- **Construction:** Khi một Actor được spawn, Unreal Engine gọi Constructor để khởi tạo tất cả các thành phần (Components) được attach. Đây là giai đoạn khởi tạo object C++ cơ bản, thiết lập các giá trị default và tạo các reference cần thiết.
- **PostInitializeComponents():** Sau khi tất cả components được khởi tạo, Unreal thực hiện hook này để thiết lập dependency giữa components, chuẩn bị cho các logic runtime như physics, collision, và rendering.
- **OnConstruction(const FTransform& Transform):** Trong editor, hàm này được gọi khi Actor được đặt vào level hoặc giá trị property được chỉnh sửa. Nó cho phép designer thực hiện procedural setup, ví dụ như tự động tạo mesh hoặc particle effect dựa trên property.

2.3.3.2 BeginPlay

- **BeginPlay()** là callback quan trọng nhất trong lifecycle của Actor khi level bắt đầu hoặc khi Actor được spawn runtime. Tại thời điểm này,

tất cả Actor trong level đã được fully initialized và có thể tương tác với world, input, physics, hoặc các actor khác.

- Đây là nơi developer thường khởi tạo gameplay logic như bắt đầu AI behavior, trigger timeline animation, spawn additional actors, hoặc thiết lập dynamic properties dựa trên trạng thái level.

2.3.3.3 Tick và Runtime Update

- Tick(float DeltaTime) là hàm được gọi mỗi frame cho tất cả Actor có bCanEverTick = true. Đây là cơ chế cập nhật liên tục, cho phép actor thực hiện logic runtime như di chuyển, kiểm tra collision, update animation, hoặc kiểm soát particle system.
- Unreal Engine tối ưu hóa tick thông qua tick groups và tick dependencies, giúp đảm bảo actor chỉ được tick khi cần thiết, giảm overhead, và hỗ trợ scaling cho world lớn.

2.3.3.4 EndPlay và Cleanup

- Khi actor bị destroy hoặc level kết thúc, Unreal Engine gọi EndPlay(const EEndPlayReason::Type EndPlayReason). Đây là nơi Actor giải phóng resources, unregister delegates, dừng timers, hoặc terminate AI behavior.
- Các EndPlayReason phổ biến bao gồm Destroyed, LevelTransition, hoặc Quit, cung cấp ngữ cảnh để developer thực hiện cleanup chính xác, đảm bảo không leak memory hay dangling references.

2.3.3.5 Destroy và Garbage Collection

- Destroy(): Khi Actor không còn cần thiết, phương thức Destroy() được gọi, đánh dấu actor để engine xóa ở frame tiếp theo.
- Unreal Engine sử dụng Garbage Collection (GC) để quản lý memory của các UObject, bao gồm Actor và Components. Actor chỉ thực sự

được deallocate khi không còn reference nào trỏ đến nó, đảm bảo tránh crash hoặc memory leak.

Components của actor cũng tuân theo lifecycle tương tự, với RegisterComponent(), InitializeComponent(), BeginPlay(), và DestroyComponent(), đồng bộ với lifecycle của parent actor.

2.3.4 Input System (Enhanced Input)

Unreal Engine cung cấp một hệ thống Enhanced Input hiện đại, mở rộng khả năng quản lý input so với hệ thống truyền thống, hỗ trợ mapping đa dạng, trigger events linh hoạt, và runtime modification. Enhanced Input được thiết kế để tách biệt dữ liệu input khỏi logic gameplay, giúp designer và developer dễ dàng định nghĩa hành vi người chơi, quản lý các control scheme phức tạp, và mở rộng cho nhiều thiết bị (keyboard, mouse, gamepad, touch, VR controllers) mà không cần viết code low-level.

2.3.4.1 Input Mapping Context

Input Mapping Context (IMC) là trung tâm quản lý mapping giữa các device input và gameplay action. Nó cho phép gán nhiều input device hoặc phím nháy vào cùng một hành động (action) và ưu tiên giữa các context khác nhau. Ví dụ: một nhân vật có thể có GameplayContext cho chơi game bình thường và UIContext khi mở menu, trong đó các phím được remap tạm thời mà không ảnh hưởng đến logic gameplay.

IMC hỗ trợ layered contexts, nơi nhiều context có thể active đồng thời và resolve priority dựa trên thiết lập developer. Điều này rất quan trọng trong các dự án phức tạp, ví dụ khi player vừa di chuyển vừa mở menu, hoặc khi cần temporary remap input cho cutscene hay cinematic sequence.

2.3.4.2 Input Actions

Input Action (IA) là abstraction cho một hành động gameplay cụ thể, ví dụ “Jump”, “Fire”, “MoveForward”, hoặc “Interact”. Mỗi Input Action xác định kiểu dữ

liệu input (Boolean, Axis, Vector2D, Vector3D), mức sensitivity, và các callback event (Started, Triggered, Completed, Canceled).

Input Action tách biệt input event với logic gameplay, giúp:

- Reusable: cùng một action có thể được bind cho nhiều device hoặc key.
- Data-driven: designer có thể thay đổi key binding trong editor hoặc runtime mà không cần chỉnh code.
- Modular: action có thể được attach vào PlayerController hoặc Pawn thông qua Enhanced Input Component, cho phép xử lý input độc lập với actor logic.

2.3.4.3 Triggers và Modifiers

Triggers là cơ chế để xác định khi nào một input action được kích hoạt, thay vì chỉ dựa trên nhấn/nối phím đơn giản. Một trigger có thể là:

- Pressed/Released: kích hoạt khi phím nhấn hoặc thả.
- Hold: chỉ trigger nếu phím giữ đủ thời gian.
- Tap: kích hoạt khi nhấn và thả nhanh.
- Chord: kích hoạt khi nhiều phím được nhấn đồng thời.

Modifiers cho phép biến đổi giá trị input trước khi gửi đến action, bao gồm:

- Scale: nhân giá trị input với một hệ số (ví dụ sensitivity của analog stick).
- Deadzone: loại bỏ giá trị nhỏ của analog input để tránh jitter.
- Inversion: đảo chiều trực (ví dụ camera pitch).
- Custom logic: developer có thể viết modifier riêng để tạo các hiệu ứng phức tạp như input curve, smoothing, hay procedural input.

2.3.4.4 Workflow và Integration

Enhanced Input kết hợp chặt chẽ với PlayerController và Pawn, nơi Enhanced Input Component attach vào actor để xử lý input runtime. Workflow tiêu biểu:

- Tạo Input Mapping Context và Input Actions trong Content Browser hoặc blueprint.
- Bind IMC vào PlayerController hoặc Pawn khi BeginPlay.
- Định nghĩa Triggers và Modifiers cho mỗi Input Action.
- Gọi callback events trong PlayerController hoặc Pawn để triển khai gameplay logic (ví dụ di chuyển, nhảy, bắn).

Enhanced Input cũng hỗ trợ dynamic context switching, rất hữu ích trong các game có nhiều trạng thái (combat, exploration, UI) và multi-device input, đồng thời giữ khả năng mở rộng cho VR, AR, hoặc custom controllers.

2.3.4.5 Ưu điểm của Enhanced Input

- Modular và Data-driven: designer có thể cấu hình input mà không cần thiệp code.
- Scalable: dễ dàng mở rộng cho nhiều devices, nhiều contexts, hoặc nhiều người chơi.
- Flexible: hỗ trợ triggers phức tạp và modifiers để kiểm soát chính xác hành vi input.
- Integrated: seamless binding với PlayerController, Pawn, Blueprint, và Gameplay Framework.

2.3.5 *UI Framework*

Trong Unreal Engine, UI Framework đóng vai trò thiết yếu trong việc hiển thị và quản lý giao diện người dùng, cung cấp các công cụ trực quan và lập trình để tạo HUD, menu, dialogue, và các thành phần tương tác trong gameplay. Hệ thống UI của Unreal Engine chủ yếu dựa trên UMG (Unreal Motion Graphics), kết hợp với các tiện ích và widgets chuẩn hóa, tạo nên một pipeline interactive, modular, và data-driven UI.

2.3.5.1 Unreal Motion Graphics (UMG)

UMG là hệ thống widget-based UI editor trong Unreal Engine, cho phép designer và developer tạo giao diện bằng cách kéo-thả components trực quan. Mỗi widget là một đối tượng UUserWidget, có thể chứa text, image, button, progress bar, slider, hoặc custom widget. UMG kết hợp blueprint scripting, hỗ trợ dynamic behavior, event handling, và binding data runtime.

Các đặc điểm quan trọng của UMG:

- Hierarchy and Layout: Widgets có thể được tổ chức theo cây (hierarchy), với khả năng nesting, canvas panel, overlay, vertical/horizontal box, hoặc grid. Điều này cho phép xây dựng giao diện phức tạp, responsive và scalable cho nhiều độ phân giải màn hình.
- Data Binding: UMG hỗ trợ bind properties trực tiếp với gameplay data, ví dụ health, score, ammo, hoặc inventory, đảm bảo UI luôn phản ánh trạng thái game real-time.
- Animations and Transitions: UMG cung cấp timeline-based animations cho widgets, cho phép fade-in/out, slide, scale, hoặc complex transitions, giúp UI trở nên sinh động và cinematic-ready.
- Event Handling: UMG hỗ trợ mouse, keyboard, touch input, và gamepad navigation, cho phép designer tạo UI interactive mà không cần viết code C++ phức tạp.

2.3.5.2 Common UI (CUI)

Common UI là tập hợp các standardized, reusable UI components và framework được Epic Games cung cấp, nhằm rút ngắn thời gian phát triển các hệ thống HUD và menu. Common UI hỗ trợ:

- Widget Templates: bao gồm buttons, lists, grids, tooltips, dialog boxes, cho phép reuse across multiple levels hoặc projects.
- Input Integration: phối hợp chặt chẽ với Enhanced Input để xử lý navigation bằng gamepad, keyboard, hoặc touch.

- Layered UI System: hỗ trợ overlay, modal, focus management, và z-order control, đảm bảo các widgets hiển thị đúng hierarchy và tương tác mượt mà.
- Dynamic Management: Common UI hỗ trợ spawning, adding, removing, và stacking widgets runtime, giúp UI framework trở nên linh hoạt trong các tình huống gameplay phức tạp như inventory, pause menu, hoặc in-game HUD.

2.3.5.3 Workflow và Integration

UI Framework trong Unreal Engine được tích hợp chặt chẽ với Gameplay Framework:

- Widgets thường được attach vào PlayerController hoặc HUD, đảm bảo khả năng interaction với player input và game state.
- Blueprint hoặc C++ logic có thể cập nhật UI dựa trên GameState, PlayerState, hoặc Actor events, tạo UI reactive, real-time, và data-driven.
- Sequencer và cinematic cũng có thể trigger widget animations hoặc visibility changes, đồng bộ UI với gameplay hoặc cutscene.

2.3.5.4 Ưu điểm của UI Framework

- Modular và Reusable: Widget và template có thể reuse trong nhiều level hoặc dự án.
- Data-Driven: UI phản ánh trạng thái game real-time thông qua binding với GameState, PlayerState, hoặc Actor properties.
- Flexible và Interactive: Hỗ trợ đa dạng input devices, animation, dynamic spawning và hierarchical layout.
- Integrated với Gameplay Framework: Phối hợp với PlayerController, HUD, Enhanced Input, và Blueprint, đảm bảo tương tác liền mạch giữa gameplay và UI.

2.3.6 Gameplay Ability System (GAS)

Gameplay Ability System (GAS) là một framework mạnh mẽ trong Unreal Engine, được thiết kế để quản lý các khả năng, hiệu ứng, và trạng thái phức tạp của nhân vật, đặc biệt hữu ích trong các game RPG, MOBA, hoặc action với mechanics đa dạng. GAS cung cấp kiến trúc data-driven, modular, và network-ready, giúp developer dễ dàng triển khai abilities, buffs/debuffs, và attribute management mà vẫn đảm bảo tính nhất quán trên môi trường multiplayer.

2.3.6.1 Core Components của GAS

Gameplay Ability (UGameplayAbility)

- Là abstraction cho một action hoặc khả năng mà actor có thể thực hiện, ví dụ nhảy, bắn, cast spell, hoặc heal.
- Gameplay Ability có thể được trigger thông qua input, event, hoặc condition trong gameplay.
- Hỗ trợ activation policy, ví dụ OnInputPressed, OnSpawned, hoặc TriggeredByGameplayEvent, giúp định nghĩa khi nào ability bắt đầu.

Ability System Component (UAbilitySystemComponent)

- Là central hub để quản lý tất cả abilities, gameplay effects, và attribute của một actor.
- Actor muốn sử dụng GAS cần attach một Ability System Component, thường gắn vào Pawn hoặc Character.
- Hỗ trợ replication cho multiplayer, đảm bảo server-authoritative control nhưng vẫn đồng bộ trạng thái đến client.

Gameplay Effects (UGameplayEffect)

- Là các effect tác động lên attributes như health, mana, stamina, speed, hoặc bất kỳ giá trị custom nào.
- Có thể apply trực tiếp, over-time, hoặc instant, với duration, stacking rules, và conditional modifiers.

- Kết hợp với Attribute Set để GAS quản lý tất cả các thuộc tính actor theo cách modular và data-driven.

Attributes và Attribute Sets (UAttributeSet)

- AttributeSet định nghĩa tập hợp thuộc tính cho actor, ví dụ health, mana, attack power.
- Attribute được GAS quản lý, replicate, và trigger gameplay effects, đảm bảo consistency giữa client và server.

2.3.6.2 Activation, Triggers và Cooldown

- Ability có thể được trigger qua Enhanced Input, Gameplay Events, hoặc thời gian runtime, cho phép xây dựng các cơ chế như combo, chain abilities, hoặc reaction-based abilities.
- GAS hỗ trợ cooldowns, costs, và conditions, giúp kiểm soát khi nào ability có thể activate và các resource tiêu tốn (mana, stamina, item).
- Trigger và tag system cho phép abilities tương tác, ví dụ một buff có thể enable hoặc block một ability khác, giúp xây dựng gameplay depth.

2.3.6.3 Tag System và Conditional Logic

- GAS sử dụng Gameplay Tags để phân loại abilities, effects, hoặc actor state.
- Tags hỗ trợ filtering, blocking, hoặc chaining abilities, ví dụ Stunned có thể block Move ability, hoặc OnFire buff trigger thêm damage over time.
- Tag system giúp developer implement complex game rules mà không cần hardcode, nâng cao modularity và scalability.

2.3.6.4 Networking và Multiplayer Support

- GAS được thiết kế server-authoritative, mọi ability activation, effect application, và attribute modification đều được validate trên server.

- Replication được thực hiện tự động cho attributes, gameplay effects, và ability activation, đảm bảo mọi client nhìn thấy state consistent.
- Hệ thống prediction và replication conflict resolution giúp gameplay vẫn mượt mà ngay cả trong môi trường multiplayer.

2.3.6.5 Workflow và Data-Driven Design

- Ability, Effects, và Attribute Sets đều có thể được tạo và chỉnh sửa trong Data Asset, hỗ trợ data-driven workflow.
- Designer có thể cấu hình abilities, durations, costs, stacking rules mà không cần code.
- Blueprint và C++ có thể mở rộng ability logic để tạo custom activation, targeting, hoặc procedural effects, đảm bảo flexibility và reusability.

2.3.6.6 Ưu điểm của GAS

- Modular và Scalable: Quản lý abilities, effects, và attributes theo component, dễ mở rộng cho nhiều nhân vật hoặc class.
- Data-Driven: Designer có thể cấu hình gameplay effects và abilities mà không cần chỉnh code.
- Network-Ready: Server-authoritative, replicated, hỗ trợ prediction, ideal cho multiplayer games.
- Complex Interaction: Tag system, conditional activation, stacking rules, và duration mechanics cho phép xây dựng cơ chế gameplay phức tạp

2.3.7 Networking và Replication

Trong Unreal Engine, **Networking và Replication** là thành phần thiết yếu để xây dựng các trò chơi multiplayer, đảm bảo trạng thái trò chơi nhất quán, server-authoritative gameplay và đồng bộ hóa client. Hệ thống này cho phép Actor và các biến được replicate từ server sang client, đảm bảo mọi client nhìn thấy trạng thái của thế giới game một cách chính xác. Actor muốn replicate cần được đánh dấu

bReplicates = true, trong khi các biến có thể sử dụng UPROPERTY(Replicated) hoặc ReplicatedUsing để kích hoạt callback khi giá trị thay đổi. Hệ thống replication còn hỗ trợ các điều kiện và tần suất replicate, giúp tối ưu băng thông và performance, đồng thời giữ cho gameplay chính xác và mượt mà. Bên cạnh đó, Unreal Engine sử dụng **Remote Procedure Calls (RPC)** để client và server có thể gọi hàm trên actor từ xa. RPC được chia thành Server RPC, Client RPC và Multicast RPC, phục vụ cho các hành động cần server-authoritative, truyền thông tin đến client cụ thể, hoặc broadcast sự kiện đến tất cả client, đồng thời kết hợp validation để giảm nguy cơ cheating. **Movement replication** là một cơ chế quan trọng khác, đặc biệt cho Pawn và Character, nơi client gửi input lên server, server tính toán vị trí mới và broadcast lại, kết hợp với prediction và correction để giảm perception lag, cùng network smoothing nhằm đảm bảo di chuyển hiển thị mượt mà trên client. Hệ thống này cũng hỗ trợ physics-based replication, giúp đồng bộ vị trí, rotation và forces của các đối tượng vật lý. Khi triển khai, developer cần chú ý attach Actor vào replication, sử dụng RepNotify cho variables, validate RPC, và tích hợp CharacterMovementComponent để tối ưu trải nghiệm multiplayer. Nhờ thiết kế server-authoritative, data-driven, và tích hợp chặt chẽ với Gameplay Framework, PlayerController, PlayerState, và GAS, hệ thống Networking và Replication của Unreal Engine cho phép xây dựng các dự án multiplayer chính xác, interactive, scalable, đồng thời hỗ trợ prediction, smoothing và quản lý logic phức tạp một cách hiệu quả.

2.4 Blueprint Scripting và C++ Programming

2.4.1 Blueprint Visual Scripting

- Blueprint Editor: Blueprint Editor là công cụ trung tâm để tạo, chỉnh sửa và quản lý các blueprint asset trong Unreal Engine. Nó cung cấp một môi trường trực quan, nơi developer và designer có thể kéo-thả các node, kết nối logic, và quản lý các thuộc tính của actor, component, hoặc UI widget. Blueprint Editor hỗ trợ nhiều view khác nhau, bao gồm Graph Editor, Details Panel, và Components Panel, giúp người dùng

quản lý cả logic runtime và cấu trúc component của đối tượng một cách trực quan và hiệu quả.

- Event Graph: Event Graph là nơi diễn ra phần lớn logic runtime của blueprint thông qua hệ thống node-based. Tại đây, các sự kiện như BeginPlay, Tick, hoặc các input event được kết nối với các action, function call, hoặc flow control nodes. Event Graph giúp biểu diễn rõ ràng luồng logic, dễ debug và theo dõi, đồng thời tích hợp chặt chẽ với hệ thống gameplay framework để phản ứng với actor, pawn, hoặc game state.**Functions / Macros**
- Functions / Macros: Blueprint Functions và Macros cho phép đóng gói các logic lặp lại hoặc phức tạp thành các node reusable. Functions có thể nhận tham số input/output và được gọi nhiều lần trong cùng blueprint hoặc giữa các blueprint khác nhau, giúp modularity và maintainability. Macros tương tự, nhưng có khả năng chứa nhiều execution pins, hỗ trợ các luồng logic phức tạp hơn mà không tạo function mới, rất hữu ích cho repetitive behavior hoặc procedural logic.**Event Dispatchers**
- Blueprint Communication: Blueprint Communication là tập hợp các cơ chế để blueprint tương tác với nhau mà không tạo ra dependency chặt chẽ. Nó bao gồm việc tham chiếu trực tiếp, sử dụng casting, hoặc binding thông qua Event Dispatchers và Interfaces, giúp các blueprint trao đổi dữ liệu, trigger sự kiện, hoặc gọi các function một cách modular và maintainable.**Blueprintable Components**.
- Event Dispatchers: Event Dispatchers hoạt động như các callback trong blueprint. Chúng cho phép một blueprint gửi thông điệp đến blueprint khác khi một sự kiện xảy ra, ví dụ actor bị destroy, player đạt score, hoặc trigger animation. Event Dispatchers giúp tách biệt producer và consumer của sự kiện, tăng tính modular, giảm coupling, và đảm bảo

logic sự kiện có thể mở rộng hoặc thay đổi mà không ảnh hưởng trực tiếp đến blueprint khác.

- Interfaces: Blueprint Interfaces định nghĩa các contract cho blueprint, cho phép blueprint khác gọi function mà không cần biết implementation chi tiết. Điều này giúp tạo ra loose coupling, cho phép nhiều blueprint implement cùng một interface nhưng có logic khác nhau, rất hữu ích trong các hệ thống modular hoặc multiplayer khi nhiều actor có cùng hành vi nhưng triển khai khác nhau.
- Blueprintable Components: Blueprintable Components là các component có thể attach vào bất kỳ actor nào, chứa logic reusable mà không cần duplicate code. Chúng có thể chứa Event Graph, Functions, và sử dụng blueprint communication để phản ứng với gameplay events. Việc sử dụng Blueprintable Components giúp tăng modularity, maintainability, và khả năng reuse logic trong nhiều actor khác nhau, đồng thời tích hợp liền mạch với hệ thống Actor và Component lifecycle.

2.4.2 C++ Programming

Unreal Engine mở rộng C++ bằng các macro như UCLASS, UPROPERTY, và UFUNCTION, cho phép các lớp, biến và hàm C++ tích hợp trực tiếp với hệ thống Reflection, Garbage Collection, và Blueprint. UCLASS định nghĩa một lớp có thể được quản lý bởi engine, cho phép spawn, replicate, và attach vào world. UPROPERTY đánh dấu biến để engine có thể serialize, replicate, hoặc expose cho editor/Blueprint, đồng thời tích hợp với Garbage Collection. UFUNCTION cho phép hàm được gọi từ Blueprint, replicate qua mạng, hoặc đăng ký với event system, tạo cầu nối giữa C++ và visual scripting. Reflection và Meta System

2.4.3 Khi nào dùng Blueprint – khi nào dùng C++

Trong Unreal Engine, việc lựa chọn giữa Blueprint Visual Scripting và C++ Programming phụ thuộc vào mục tiêu dự án, độ phức tạp của gameplay logic, hiệu

suất, và khả năng maintainability. Blueprint cung cấp môi trường trực quan, dễ tiếp cận cho designer và developer không chuyên C++, cho phép xây dựng nhanh các logic gameplay, prototyping, UI, và animation control. Nó hỗ trợ event-driven design, modular functions/macros, và tích hợp liền mạch với editor, giúp iterative development nhanh chóng. Nhờ khả năng hot reload và live debugging, designer có thể thay đổi logic mà không cần rebuild toàn bộ project. Tuy nhiên, Blueprint có nhược điểm là performance thấp hơn C++, đặc biệt khi xử lý các thuật toán phức tạp, AI, hoặc hệ thống vật lý nặng. Ngoài ra, quá nhiều nodes trong blueprint có thể dẫn đến khó maintain và debug trong các project lớn.

Ngược lại, C++ cung cấp control chi tiết, hiệu suất cao, và khả năng mở rộng. Nó thích hợp cho các hệ thống core của engine, gameplay mechanics phức tạp, hệ thống AI, physics simulation, network replication, hoặc các tính năng cần tối ưu runtime. C++ cho phép developer quản lý memory, pointer, lifecycle, và integration với Garbage Collection, đồng thời hỗ trợ modularity thông qua modules và plugins. Nhược điểm là steeper learning curve, thời gian iterative development lâu hơn, và không trực quan như Blueprint, đặc biệt với designer hoặc những người không quen lập trình.

Một chiến lược tối ưu thường là hybrid approach, kết hợp C++ cho core logic và performance-critical systems, trong khi Blueprint dùng cho gameplay logic, UI, prototyping và interaction. Ví dụ, nhân vật hoặc actor có thể được triển khai bằng C++ để kiểm soát physics, abilities, và replication, còn các sự kiện, UI update, hoặc simple movement input có thể xử lý bằng Blueprint. Cách tiếp cận này tận dụng được ưu điểm của cả hai công cụ, giảm nhược điểm riêng, và phù hợp với các dự án game hiện đại yêu cầu cả flexibility, modularity, và scalability.

2.5 Asset và Content System

2.5.1 Asset Types (*Mesh, Material, Texture, Animation, Audio...*)

Unreal Engine hỗ trợ nhiều loại asset khác nhau, mỗi loại phục vụ một vai trò cụ thể trong game. **Mesh** bao gồm Static Mesh (đối tượng tĩnh) và Skeletal Mesh

(nhân vật hoặc vật thể có rig animation), dùng để hiển thị hình học trong scene. **Material** định nghĩa bề mặt và shading của mesh, kết hợp với **Texture** để tạo chi tiết màu sắc, bump, hay specular. **Animation** bao gồm Animation Sequence, Blend Space, và Animation Montage, dùng cho chuyển động nhân vật hoặc props. **Audio** assets cung cấp sound effect, music, voice, và ambient sound. Các asset này được quản lý qua Content Browser và có thể được reuse trong nhiều level, blueprint, hoặc actor, tạo tính modular và tiết kiệm resource.

2.5.2 Level

Level là container của tất cả các actor, components, lighting, và environment trong một scene cụ thể. Nó cũng là một asset, có thể được lưu, load, hoặc instance runtime. Level thường bao gồm **sub-levels**, giúp quản lý world partition và streaming assets hiệu quả. Mỗi level asset lưu trữ thông tin về placement của actor, lighting settings, navigation mesh, và volume settings, đóng vai trò trung tâm trong workflow thiết kế game.

2.5.3 Blueprint Class và Blueprint Asset

Blueprint Class là asset chứa logic actor hoặc component, được tạo từ C++ class hoặc từ scratch. Blueprint Asset là instance hoặc data-only blueprint, chứa thông tin cấu hình, variables, hoặc event binding mà không chứa logic execution nặng. Sự phân tách này giúp **tái sử dụng logic**, tạo data-driven workflow, và giảm duplication giữa các actor trong project.

2.5.4 Asset Metadata

Mỗi asset trong Unreal Engine đều có **metadata**, bao gồm tên, tag, category, version, và dependencies. Metadata hỗ trợ việc **tìm kiếm, filtering, và quản lý asset** trong project lớn, đồng thời giúp engine và build tool xác định dependencies để optimize cooking và packaging. Metadata còn có thể được sử dụng runtime để query asset properties, ví dụ phân loại item, type effect, hoặc level-specific settings.

2.5.5 Soft/Hard References

Unreal Engine phân biệt giữa **hard reference** (strong reference) và **soft reference** (TSoftObjectPtr). Hard reference giữ asset luôn loaded trong memory khi object tồn tại, đảm bảo sẵn sàng sử dụng nhưng tăng memory usage. Soft reference chỉ lưu path tới asset và load runtime khi cần, giúp giảm memory footprint và hỗ trợ **lazy loading** cho large worlds hoặc streaming-heavy games.

2.5.6 Asset Cooking Pipeline

Asset Cooking là quá trình **chuyển đổi asset từ dạng source sang dạng runtime-ready**, tối ưu cho target platform. Pipeline này bao gồm compression, serialization, dependency resolution, và platform-specific optimization. Cooking đảm bảo assets nhỏ gọn, load nhanh, và chuẩn hóa format cho các platform khác nhau (PC, console, mobile). Đây là bước quan trọng để build final package và giảm thời gian load runtime.

2.5.7 Asset Streaming

Asset Streaming là cơ chế **dynamic load/unload assets** dựa trên proximity, visibility, hoặc gameplay context. Nó kết hợp với world partition, level streaming, hoặc distance-based loading để giảm memory usage và tăng performance. Ví dụ, texture, mesh, hoặc audio sẽ chỉ load khi player gần hoặc cần hiển thị, và unload khi không còn sử dụng. Streaming cho phép game thế giới mở lớn hoạt động mượt mà mà không làm nghẽn memory hoặc hạ fps.

2.6 Animation System

2.6.1 Skeleton

Skeleton là asset trung tâm trong hệ thống animation, định nghĩa cấu trúc xương (bones) của một Skeletal Mesh. Skeleton xác định hierarchy, parent-child relationships, và transform defaults của mỗi bone, từ đó animation sequences và animation blueprint có thể áp dụng cho mesh tương ứng. Skeleton giúp **tái sử dụng animation** giữa nhiều Skeletal Mesh có cùng rig, đồng thời là nền tảng cho retargeting và IK.

2.6.2 Animation Blueprint

Animation Blueprint (AnimBP) là asset chứa **logic điều khiển animation runtime** cho Skeletal Mesh. Nó kết hợp Event Graph, State Machines, Blend Nodes và procedural logic để xác định pose cuối cùng của nhân vật mỗi frame. AnimBP cho phép lập trình **dynamic, responsive animation**, ví dụ blend walk/run, trigger attack animation, hoặc phản ứng với gameplay variables như speed, direction, hoặc health.

2.6.3 Animation State Machine

Animation State Machine là hệ thống **quản lý transitions giữa các trạng thái animation**. Mỗi state đại diện cho một animation hoặc blend, ví dụ Idle, Walk, Run, Jump. Transitions được điều khiển bằng các boolean, float, hoặc gameplay variables, giúp animation flow **mượt mà và logic**. State Machine thường nằm trong Animation Blueprint và hỗ trợ nested states, blend in/out timing, và priority rules.

2.6.4 Blend Space

Blend Space là asset cho phép **blend giữa nhiều animation dựa trên input variables**, ví dụ movement speed và direction. Nó cung cấp interpolation giữa animations, cho phép nhân vật chuyển động mượt mà từ walk sang run, hoặc quay góc di chuyển theo joystick input. Blend Space giảm lượng state transitions cần thiết và hỗ trợ smooth procedural animation.

2.6.5 Control Rig

Control Rig là hệ thống rigging procedural trong Unreal Engine, cho phép **tạo và điều khiển pose trực tiếp trong engine**. Control Rig dùng để animate characters, props, hoặc camera, hỗ trợ forward/inverse kinematics, constraints, và procedural adjustments. Nó cũng có thể tích hợp với Sequencer để tạo cinematic animation, giúp designer hoặc animator trực tiếp manipulate skeleton mà không cần DCC tool bên ngoài.

2.6.6 IK System

IK System cho phép tính toán và điều chỉnh vị trí của bone cuối cùng dựa trên target, ví dụ foot placement trên uneven terrain, hand reaching objects, hoặc head tracking. IK giúp animation trở nên **realistic và interactive**, đồng thời có thể combine với FK (Forward Kinematics) để đạt sự chính xác cao trong movement và pose.

2.6.7 Retargeting

Retargeting là cơ chế apply animation từ một Skeleton sang Skeleton khác với hierarchy hoặc proportions khác nhau. Unreal Engine sử dụng retargeting để reuse animation assets, giảm workload tạo animation mới, và hỗ trợ modular character system, ví dụ một animation walk/run có thể dùng cho nhiều nhân vật với height, limb length khác nhau mà vẫn mượt mà.

Retargeting là cơ chế apply animation từ một Skeleton sang Skeleton khác với hierarchy hoặc proportions khác nhau. Unreal Engine sử dụng retargeting để reuse animation assets, giảm workload tạo animation mới, và hỗ trợ modular character system, ví dụ một animation walk/run có thể dùng cho nhiều nhân vật với height, limb length khác nhau mà vẫn mượt mà.

Retargeting là cơ chế **apply animation từ một Skeleton sang Skeleton khác** với hierarchy hoặc proportions khác nhau. Unreal Engine sử dụng retargeting để reuse animation assets, giảm workload tạo animation mới, và hỗ trợ modular character system, ví dụ một animation walk/run có thể dùng cho nhiều nhân vật với height, limb length khác nhau mà vẫn mượt mà.

2.7 Hệ thống đồ họa và VFX trong UE5

2.7.1 Công nghệ đồ họa UE5

2.7.1.1 Lumen Global Illumination

Lumen là hệ thống real-time global illumination (GI) thế hệ mới trong Unreal Engine, thiết kế để cung cấp ánh sáng gián tiếp, phản xạ, và môi trường động một cách chính xác mà không cần sử dụng pre-baked lightmaps. Truyền thống, GI được

tính toán offline thông qua light baking, khiến việc thay đổi lighting runtime trở nên khó khăn và tốn thời gian. Lumen giải quyết vấn đề này bằng cách sử dụng kết hợp screen-space ray tracing và software ray tracing approximation, từ đó tạo ra ánh sáng phản xạ và lan truyền tự nhiên trên các bề mặt trong scene, đồng thời phản ứng động với thay đổi geometry hoặc light source.

Một trong những ưu điểm nổi bật của Lumen là dynamic lighting support, cho phép scene phản ứng ngay lập tức khi ánh sáng hoặc objects di chuyển, ví dụ nhân vật đi qua bóng, đèn bật/tắt, hoặc vật thể thay đổi vị trí. Hệ thống còn hỗ trợ indirect lighting, phản chiếu màu sắc giữa các vật thể và tạo soft bounce light, giúp cảnh trở nên sống động và chân thực hơn. Lumen cũng tích hợp với Nanite virtualized geometry, cho phép ánh sáng gián tiếp chiếu lên các mesh độ phân giải cao mà vẫn tối ưu hiệu suất.

Tuy nhiên, Lumen vẫn là một giải pháp approximation và có thể giảm chất lượng chi tiết trong một số tình huống extreme, như các bề mặt nhỏ hoặc ánh sáng cường độ cao. Nó cung cấp nhiều tùy chọn cân bằng giữa chất lượng hình ảnh và performance, giúp developer điều chỉnh phù hợp với mục tiêu platform, từ console next-gen đến PC high-end. Nhìn chung, Lumen Global Illumination đại diện cho bước tiến quan trọng trong rendering dynamic lighting, kết hợp realism, flexibility, và real-time performance trong Unreal Engine.

2.7.1.2 Nanite Virtualized Geometry

Nanite là hệ thống virtualized geometry thế hệ mới trong Unreal Engine, được thiết kế để render các scene có độ phân giải cực cao với hàng tỷ polygons mà vẫn duy trì hiệu suất mượt mà. Truyền thông, artist phải tạo nhiều Level of Detail (LOD) cho mỗi mesh để giảm polygon count khi camera cách xa, dẫn đến quy trình asset phức tạp và hạn chế chi tiết. Nanite loại bỏ nhu cầu này bằng cách tự động quản lý, stream và cull các micro-polygons dựa trên distance, screen size và visibility, giúp developer import trực tiếp các high-poly assets từ ZBrush, Maya hoặc Photogrammetry mà không cần tối ưu thủ công.

Hệ thống hoạt động dựa trên cơ chế clustered triangle rendering, nơi mesh được chia thành các clusters và chỉ render những clusters có thể thấy trên màn hình, giảm đáng kể GPU workload và memory bandwidth. Nanite cũng hỗ trợ hierarchical LOD, đảm bảo các đối tượng hiển thị chi tiết phù hợp với khoảng cách và tầm nhìn của camera. Khi kết hợp với Lumen Global Illumination, ánh sáng và phản xạ trên mesh high-poly vẫn chính xác, tạo ra môi trường thực tế mà không ảnh hưởng đáng kể đến hiệu suất.

Mặc dù Nanite đem lại khả năng rendering chi tiết vượt trội, nó có một số giới hạn: các mesh cần phải là static (không deform nhiều), không thích hợp cho các skeletal mesh phức tạp hoặc dynamic destructible geometry. Hệ thống cũng yêu cầu GPU mạnh để khai thác tối đa hiệu quả, đặc biệt khi sử dụng kết hợp với các tính năng next-gen khác như Virtual Shadow Maps và Lumen.

2.7.1.3 Virtual Shadow Maps

Virtual Shadow Maps (VSM) là hệ thống shadow mapping thế hệ mới trong Unreal Engine, được thiết kế để cung cấp high-resolution, dynamic shadows cho các scene có độ phức tạp cao, đặc biệt khi kết hợp với Nanite virtualized geometry. Truyền thống, shadow mapping dựa trên fixed-resolution shadow map, dẫn đến hiện tượng aliasing, shadow acne hoặc thiếu chi tiết trên các bề mặt gần camera hoặc mesh chi tiết. VSM giải quyết vấn đề này bằng cách sử dụng tile-based virtualized representation, nơi chỉ những phần của shadow map được cài đặt cho view hiện tại mới được render và lưu trữ, tối ưu hóa memory usage và GPU bandwidth.

VSM hỗ trợ dynamic lighting và Nanite meshes, đảm bảo rằng shadow phản chiếu chính xác trên các mesh có độ phân giải cực cao, kể cả những chi tiết nhỏ mà traditional shadow map không thể capture. Hệ thống còn cung cấp soft shadows, contact shadows, và penumbra filtering, giúp shadow trở nên tự nhiên và liền mạch trong môi trường động. Virtualization cho phép scale shadow quality trong world lớn mà không làm tăng đáng kể memory footprint hay giảm framerate.

Một ưu điểm quan trọng của VSM là khả năng integration liền mạch với Lumen GI, giúp ánh sáng và shadow tương tác tự nhiên trong scene. Tuy nhiên, VSM yêu cầu GPU hiện đại để khai thác hiệu quả, và cần cân nhắc giữa resolution, tile size, và shadow distance để cân bằng chất lượng hình ảnh và performance.

Tóm lại, Virtual Shadow Maps là bước tiến quan trọng trong rendering pipeline, cung cấp shadows chính xác, mượt mà, và scalable, đồng thời hỗ trợ các world lớn, high-poly assets, và dynamic lighting, góp phần nâng cao realism và immersive experience trong Unreal Engine.

2.7.1.4 Temporal Super Resolution (TSR)

Temporal Super Resolution (TSR) là hệ thống upscaling và anti-aliasing thế hệ mới trong Unreal Engine, được thiết kế để cải thiện chất lượng hình ảnh từ các frame render ở độ phân giải thấp hơn, đồng thời tối ưu hiệu suất GPU. TSR hoạt động dựa trên temporal accumulation, kết hợp thông tin từ các frame trước và motion vectors để tái tạo chi tiết hình ảnh, giảm hiện tượng aliasing, shimmering hoặc jagged edges khi camera hoặc objects chuyển động.

Hệ thống TSR còn sử dụng edge reconstruction và confidence weighting để xác định những vùng cần khôi phục chi tiết, từ đó đạt chất lượng gần tương đương native resolution nhưng giảm đáng kể computational cost. Điều này cho phép Unreal Engine render scene ở resolution thấp hơn, sau đó upscale bằng TSR để đạt hình ảnh sắc nét và mượt mà, đặc biệt hữu ích cho high framerate gaming trên console next-gen hoặc PC high-end.

Một ưu điểm nổi bật của TSR là khả năng tích hợp liền mạch với Nanite, Lumen và post-processing pipeline, giúp cảnh vật, ánh sáng, và shading vẫn giữ chi tiết và ổn định khi camera di chuyển nhanh hoặc scene thay đổi động. Tuy nhiên, TSR vẫn là một giải pháp approximation; trong các tình huống extreme với motion rất nhanh hoặc frame-to-frame changes lớn, đôi khi có thể xuất hiện ghosting hoặc slight blur. Việc cân bằng quality vs. performance là cần thiết để tối ưu trải nghiệm người chơi.

Tóm lại, Temporal Super Resolution là công cụ quan trọng để nâng cao visual fidelity và performance trong Unreal Engine, cho phép các game thế hệ mới chạy mượt mà ở high framerate mà vẫn duy trì hình ảnh sắc nét, ổn định và chân thực.

2.7.2 Niagara VFX System

2.7.2.1 GPU/CPU Particle

Trong Unreal Engine, hệ thống particle là thành phần chủ chốt để tạo các hiệu ứng visual effects như smoke, fire, sparks, dust, hay magic spells. Particle system có thể được tính toán trên CPU hoặc GPU, tùy thuộc vào quy mô, độ phức tạp và yêu cầu performance của scene.

CPU-based particle systems sử dụng bộ xử lý trung tâm để tính toán vị trí, velocity, rotation, size, và life của từng particle. CPU particle cung cấp flexibility cao, cho phép interactions phức tạp với physics, collisions, hoặc gameplay events, và dễ dàng debug trực quan. Tuy nhiên, khi số lượng particle tăng lên hàng chục nghìn, CPU particle có thể trở thành bottleneck, ảnh hưởng đến framerate và performance tổng thể.

GPU-based particle systems, như Niagara GPU Particles, sử dụng bộ xử lý đồ họa để thực hiện các tính toán particle parallel, cho phép render hàng triệu particle đồng thời mà không gây áp lực lên CPU. GPU particles tối ưu cho các hiệu ứng large-scale visual phenomena, ví dụ clouds, waterfalls, hoặc explosion clouds, nhưng giới hạn tương tác trực tiếp với game logic và physics của CPU, do dữ liệu particle chủ yếu tồn tại trên GPU memory.

Unreal Engine cho phép hybrid approach, kết hợp CPU và GPU particles trong cùng một hệ thống Niagara để tận dụng cả flexibility và performance. Ví dụ, particle chính tương tác gameplay có thể xử lý trên CPU, trong khi smoke, sparks hay debris không tương tác có thể xử lý trên GPU. Việc lựa chọn GPU hay CPU particle phụ thuộc vào mục tiêu visual fidelity, interaction complexity, và performance target, giúp developer tối ưu hóa cả chất lượng hiệu ứng và tốc độ khung hình.

2.7.2.2 Niagara Graph

Niagara Graph là thành phần trung tâm của hệ thống Niagara trong Unreal Engine, nơi developer và technical artist tạo và điều khiển particle behavior thông qua node-based visual scripting. Graph này xác định cách mỗi particle được spawn, cập nhật, và render, đồng thời quản lý tất cả các thuộc tính như position, velocity, color, size, lifetime, và rotation.

Niagara Graph được cấu trúc thành nhiều stages, bao gồm Spawn, Update, và Render, cho phép kiểm soát toàn bộ lifecycle của particle. Spawn stage xác định số lượng particle được tạo ra và các thông số khởi tạo. Update stage tính toán các thuộc tính particle theo thời gian, ví dụ movement, forces, collisions, hoặc procedural animation. Render stage quyết định cách particle xuất hiện trên màn hình, liên kết với material, mesh, sprite, hoặc ribbon.

Một điểm mạnh của Niagara Graph là modularity thông qua Modules, nơi các logic particle có thể được đóng gói, reuse, và share giữa các emitter hoặc system khác nhau. Ngoài ra, Niagara Graph hỗ trợ parameter binding, cho phép particle tương tác với gameplay variables, camera, environment, hoặc user input, từ đó tạo ra hiệu ứng động, responsive và interactive.

Niagara Graph cũng tích hợp với GPU simulation, giúp thực hiện tính toán particle song song, hiệu quả, đặc biệt với số lượng particle lớn mà không ảnh hưởng đến CPU. Điều này kết hợp với event-driven particle logic cho phép tạo các hiệu ứng phức tạp như collision response, trail generation, hoặc dynamic color changes dựa trên gameplay events.

2.7.3 Chaos Physics

2.7.3.1 Rigid Body

Trong Unreal Engine, Rigid Body là thành phần cơ bản của physics simulation, đại diện cho các vật thể cứng, không biến dạng khi chịu lực hoặc va chạm. Mỗi rigid body được gắn vào Actor hoặc Component, có khối lượng (mass), inertia tensor, velocity, và các thuộc tính vật lý khác để tính toán motion và interaction với môi trường.

Rigid Body sử dụng PhysX hoặc Chaos Physics Engine (tùy phiên bản Unreal Engine) để tính toán collision response, forces, torques, và constraints theo thời gian thực. Khi áp dụng lực, như gravity, impulses, hoặc collisions, rigid body sẽ phản ứng dựa trên Newtonian mechanics, tạo chuyển động tự nhiên, bám sát thực tế vật lý. Nó hỗ trợ kinematic, dynamic, và static states, cho phép developer kiểm soát vật thể có di chuyển tự do, theo kịch bản hoặc cố định trong world.

Rigid Body còn tích hợp với collision shapes (box, sphere, capsule, convex mesh) để tối ưu tính toán va chạm, và hỗ trợ constraints như hinge, spring, và physics joint để tạo cơ cấu phức tạp như cửa quay, búa đậm, hoặc cầu treo. Khi kết hợp với ragdoll physics, rigid body trở thành nền tảng để simulate skeletal mesh dynamics, từ các nhân vật chết, bị va đập, đến props interactive trong gameplay.

Việc sử dụng rigid body đúng cách giúp game đạt realism, interactivity, và predictable physics behavior, đồng thời cân bằng performance nhờ việc giới hạn số lượng dynamic rigid body hoặc sử dụng sleeping/activation states.

2.7.3.2 Cloth

Trong Unreal Engine, Cloth là hệ thống mô phỏng các vật thể mềm hoặc vải, cho phép nhân vật, props, hoặc environment tương tác với lực, chuyển động và collision một cách realistic. Cloth simulation dựa trên particle-based physics hoặc mass-spring system, nơi mỗi vertex của mesh vải được coi là một particle, kết nối với nhau bằng các spring constraints để duy trì hình dạng tổng thể trong quá trình chuyển động.

Cloth được tích hợp với Skeletal Mesh, cho phép các phần vải như áo, váy, hoặc khăn choàng follow chuyển động của nhân vật, đồng thời phản ứng với gravity, wind, collision, và character motion. Hệ thống hỗ trợ collision với cả rigid bodies và capsule shapes của nhân vật, đảm bảo vải không xuyên qua cơ thể hoặc môi trường.

Ngoài ra, Unreal Engine cung cấp nhiều tuning parameters như stiffness, damping, friction, và bending resistance, cho phép artist điều chỉnh hành vi vải từ mềm mại, bay bổng đến cứng cáp, bám sát nhân vật hoặc môi trường. Cloth

simulation cũng có thể kết hợp với LOD và GPU acceleration để tối ưu performance, đặc biệt trong các scene với nhiều nhân vật hoặc vật lý phức tạp.

Việc sử dụng Cloth simulation đúng cách giúp nâng cao visual realism, tạo cảm giác interactive và immersive trong gameplay hoặc cinematic, đồng thời tăng tính chi tiết và sống động cho nhân vật và môi trường trong Unreal Engine.

2.7.3.3 Destruction

Trong Unreal Engine, Destruction System là cơ chế cho phép vật thể trong game bị phá hủy một cách động và realistic, bao gồm vỡ nát, sụp đổ, hoặc tan rã khi chịu lực, va chạm, hoặc sự kiện gameplay. Hệ thống này tích hợp chặt chẽ với Rigid Body Physics, Chaos Physics Engine, và các physics constraints để đảm bảo các mảnh vụn (fracture pieces) phản ứng chính xác theo cơ học Newton.

Destruction trong Unreal Engine thường sử dụng Fracture Meshes, nơi một mesh được chia thành nhiều phần nhỏ (chunks) bằng Voronoi or other procedural fracturing algorithms. Khi mesh bị tác động, engine tính toán forces propagation, collision response, và momentum transfer giữa các chunks, tạo ra hiệu ứng phá hủy tự nhiên, bao gồm cả debris scattering và dust generation.

Hệ thống cũng hỗ trợ hybrid destruction, kết hợp dynamic destruction với pre-simulated animation để cân bằng performance và realism, đặc biệt trong các scene phức tạp hoặc thế giới mở. Thông qua việc điều chỉnh các tham số như mass, damage threshold, constraint strength, và impact propagation, developer có thể kiểm soát mức độ phá hủy từ nhẹ nhàng đến toàn bộ kết cấu sụp đổ.

Destruction System giúp tăng interactivity và immersion cho game, cho phép môi trường trở nên sống động và phản ứng trực tiếp với hành động của người chơi, đồng thời mở ra các cơ chế gameplay sáng tạo như phá hủy cover, hủy công trình, hoặc tạo cinematic effects trong realtime.

2.7.3.4 Vehicles

Trong Unreal Engine, Vehicles là hệ thống chuyên dụng để mô phỏng các phương tiện di chuyển, bao gồm xe hơi, xe máy, xe tải, hoặc các phương tiện cơ giới

khác, với động lực học (physics-based simulation) chân thực. Hệ thống vehicle dựa trên Rigid Body Physics, wheel constraints, suspension, và tire friction models để đảm bảo chuyển động, va chạm và phản ứng với địa hình tự nhiên.

Mỗi vehicle trong Unreal Engine thường được cấu hình bằng Vehicle Pawn hoặc WheeledVehicle class, kết hợp với Vehicle Movement Component, nơi quản lý các thông số quan trọng như engine torque, brake force, steering, differential setup, và suspension stiffness. Wheel Collisions và Tire Friction được mô phỏng dựa trên PhysX hoặc Chaos Vehicle System, giúp vehicle tương tác chính xác với các bề mặt khác nhau, từ đường nhựa, cát, đến đất đá.

Hệ thống còn hỗ trợ network replication cho multiplayer, cho phép vehicle movement và state được đồng bộ giữa các client và server, đồng thời kết hợp với animation hoặc camera systems để tạo cảm giác immersive khi lái xe. Ngoài ra, Unreal Engine cho phép tùy chỉnh các tính năng nâng cao như manual gear shifting, skid effects, wheel spin particles, và sound cues, giúp tạo ra trải nghiệm lái xe sống động và chân thực.

Việc sử dụng hệ thống Vehicles đúng cách giúp game đạt realism, responsive controls, và dynamic interaction với environment, đồng thời mở ra cơ hội gameplay đa dạng như racing, open-world exploration, hoặc vehicular combat.

2.8 Audio System

2.8.1.1 Unreal Audio Engine

Unreal Audio Engine là hệ thống âm thanh tích hợp trong Unreal Engine, cung cấp rendering, playback, và spatialization cho audio assets trong game. Hệ thống này được thiết kế để hỗ trợ high-fidelity sound, real-time mixing, và interactive audio, từ các sound effect đơn giản đến môi trường 3D phức tạp.

Audio Engine quản lý nhiều loại audio assets, bao gồm Wave, Sound Cue, Sound Wave, Dialogue Wave, và Sound Submix, cho phép kết hợp, filter, hoặc apply effects theo thời gian thực. Hệ thống hỗ trợ 3D spatialization, giúp xác định vị trí âm

thanh trong không gian dựa trên listener và source, tạo trải nghiệm immersive và realistic, đặc biệt trong game first-person hoặc VR.

Một tính năng quan trọng của Unreal Audio Engine là real-time modulation và parameter control, cho phép developer thay đổi volume, pitch, filter, hoặc reverb dựa trên gameplay variables hoặc environment changes. Hệ thống còn hỗ trợ attenuation, occlusion, và reflection, giúp âm thanh phản ứng với vật thể, distance, và bối cảnh xung quanh.

Unreal Audio Engine cũng tích hợp với submixes và effect chains, cho phép áp dụng reverb, EQ, compression, hoặc convolution reverb cho nhóm audio, giúp tạo cohesive soundscape cho game. Ngoài ra, engine hỗ trợ dynamic loading/unloading audio assets và platform-specific optimization, đảm bảo performance ổn định trên console, PC, và mobile.

Tóm lại, Unreal Audio Engine cung cấp nền tảng mạnh mẽ cho việc tạo, quản lý, và render âm thanh interactive, immersive và chất lượng cao, đóng vai trò quan trọng trong việc nâng cao trải nghiệm người chơi và realism trong Unreal Engine.

2.8.1.2 Sound Cue

Sound Cue là asset trong Unreal Engine dùng để tạo logic âm thanh phức tạp bằng hệ thống node-based, cho phép kết hợp, điều chỉnh, và biến đổi các audio assets một cách trực quan. Sound Cue không chỉ đơn giản phát một file âm thanh mà còn cung cấp interactive, procedural, và dynamic audio dựa trên gameplay events.

Mỗi Sound Cue được xây dựng từ các nodes, bao gồm các loại như Wave Player (phát sound wave), Modulator (điều chỉnh pitch/volume), Mixer (kết hợp nhiều âm thanh), Delay, Randomizer, và Looping nodes. Nodes này có thể kết hợp để tạo ra các hiệu ứng âm thanh phức tạp, ví dụ: footsteps có random pitch và volume, ambient sounds với layer echo và reverb, hoặc weapon fire với procedural variation.

Sound Cue còn hỗ trợ parameter binding, cho phép giá trị pitch, volume, filter hoặc effect được điều chỉnh runtime dựa trên gameplay variables như distance, character speed, hoặc environment context. Đồng thời, Sound Cue tích hợp với Sound

Classes và Submixes, giúp quản lý grouping, volume control, và apply effect chain cho nhiều Sound Cue cùng lúc, tạo ra cohesive audio mix cho scene hoặc level.

Một ưu điểm quan trọng của Sound Cue là khả năng visual scripting, giúp designer hoặc sound engineer xây dựng audio logic mà không cần lập trình C++. Điều này tăng tốc workflow, hỗ trợ iterative testing, và tạo điều kiện cho audio trở nên responsive và immersive trong gameplay.

2.8.1.3 Attenuation và Spatialization

Trong Unreal Engine, Attenuation và Spatialization là hai cơ chế quan trọng để tạo trải nghiệm âm thanh 3D realistic và immersive. Chúng định hình cách âm thanh được nghe bởi người chơi dựa trên vị trí của listener và source, cũng như môi trường xung quanh.

Attenuation là quá trình giảm cường độ âm thanh dựa trên khoảng cách giữa nguồn và listener. Unreal Engine cho phép định nghĩa các attenuation settings như falloff curves, min/max distance, và rolloff function, từ đó kiểm soát cách âm thanh yếu dần khi người chơi di chuyển ra xa nguồn. Attenuation cũng có thể kết hợp với các parameter khác như cone settings, giúp âm thanh phát ra hướng cụ thể, tăng tính hướng không gian và định hướng nghe.

Spatialization là cơ chế tái tạo vị trí âm thanh trong không gian 3D, bao gồm cả panning, HRTF (Head-Related Transfer Function), và directionality cues. Spatialization cho phép người chơi cảm nhận âm thanh đến từ hướng và khoảng cách chính xác, ví dụ tiếng脚步声 phía sau, tiếng xe ở bên phải, hoặc tiếng nước chảy từ xa. Unreal Engine hỗ trợ nhiều spatialization method, từ stereo panning đơn giản đến binaural rendering cho VR, giúp nâng cao immersion.

Kết hợp Attenuation và Spatialization, Unreal Engine có thể mô phỏng dynamic audio environment, nơi âm thanh phản ứng với chuyển động của nhân vật, vật thể, và các điều kiện môi trường như walls, obstacles, hoặc reverb zones. Điều này tạo ra trải nghiệm âm thanh sống động, đồng thời cho phép developer tối ưu performance bằng cách giảm volume hoặc stream các audio assets ở xa mà vẫn giữ realism.

2.8.1.4 Submix / DSP Effects

Trong Unreal Engine, Submix và DSP (Digital Signal Processing) Effects là các thành phần quan trọng trong audio pipeline, cho phép quản lý, mix, và xử lý âm thanh theo nhóm nhằm tạo ra môi trường âm thanh chất lượng cao và immersive.

Submix là một kênh trung gian trong audio engine, nơi nhiều Sound Cue hoặc Sound Wave có thể gửi tín hiệu âm thanh để mix và xử lý chung trước khi output đến listener. Submix cho phép quản lý volume, priority, routing, và effect chains cho nhiều nguồn âm thanh cùng lúc. Ví dụ, toàn bộ âm thanh ambient của một level có thể đi qua một submix để áp dụng chung reverb, EQ, hoặc compression, giúp cohesive soundscape và dễ dàng điều chỉnh tổng thể mà không cần thay đổi từng sound asset riêng lẻ.

DSP Effects là các effect xử lý tín hiệu âm thanh theo thời gian thực, bao gồm reverb, delay, EQ, compression, modulation, và convolution reverb. Unreal Engine cho phép áp dụng DSP effects trực tiếp lên submix hoặc individual sound, và hỗ trợ parameter automation, cho phép biến đổi các thuộc tính effect dựa trên gameplay variables. Ví dụ, reverb có thể thay đổi dựa trên size của room, distance của listener, hoặc environmental condition, tạo ra âm thanh động, responsive và sống động.

Sự kết hợp giữa Submix và DSP Effects tạo điều kiện cho mixing hierarchical, interactive audio, giảm thiểu load CPU bằng cách xử lý nhóm audio chung, đồng thời giữ tính modular, flexible và dễ quản lý trong các project lớn. Hệ thống này cũng hỗ trợ side-chaining, ducking, và multi-effect chains, giúp composer hoặc sound designer kiểm soát chính xác mức độ ảnh hưởng của từng âm thanh trong gameplay hoặc cinematic.

2.8.1.5 Audio Mixer

Audio Mixer là thành phần cốt lõi trong Unreal Audio Engine, chịu trách nhiệm tích hợp, mix, và render tất cả tín hiệu âm thanh trước khi output tới speaker hoặc headphones. Nó cho phép kết hợp nhiều audio sources, submixes, và DSP effects để tạo ra âm thanh chất lượng cao, interactive, và real-time.

Audio Mixer quản lý toàn bộ audio pipeline, bao gồm routing, volume management, effect processing, và spatialization. Các sound sources như Sound Cue, Sound Wave, hoặc dialogue assets được gửi tới Audio Mixer, nơi chúng có thể được apply effects, attenuated, spatialized, và grouped vào submixes. Điều này đảm bảo rằng tất cả âm thanh trong scene được phối hợp một cách consistent và balanced, đồng thời giảm thiểu artefacts như clipping hoặc phasing.

Một tính năng quan trọng của Audio Mixer là real-time processing, cho phép thay đổi parameters, automation, và effect chaining dựa trên gameplay events. Ví dụ, volume của background music có thể giảm khi dialogue xuất hiện, reverb tăng khi nhân vật di chuyển vào không gian lớn, hoặc low-pass filter áp dụng khi nhân vật đi vào môi trường underwater. Audio Mixer cũng hỗ trợ multi-threaded processing và GPU acceleration để tối ưu hiệu suất trên console, PC và VR platforms.

Ngoài ra, Audio Mixer tích hợp với Submixes và RTPC (Real-Time Parameter Control), giúp composer và sound designer tạo dynamic, responsive audio, đồng thời giữ workflow modular và scalable cho dự án lớn. Nhờ Audio Mixer, Unreal Engine có thể xử lý complex audio environments với hàng trăm sound sources mà vẫn duy trì high-fidelity, low-latency output.

CHƯƠNG 3. QUY TRÌNH XÂY DỰNG, LẬP TRÌNH VÀ TRIỂN KHAI DEMO

3.1 Quy trình xây dựng hệ thống

3.1.1 Môi trường phát triển

Trong quá trình xây dựng demo giữa kỳ môn Phát triển trò chơi, nhóm sử dụng môi trường phát triển dựa trên Unreal Engine. Việc lựa chọn môi trường này giúp đảm bảo đáp ứng đúng yêu cầu của bài tiểu luận, bao gồm phần Blueprint, AI, Physics, UI và các thành phần cốt lõi của một game engine theo thang điểm đã được đề cập trong yêu cầu đề bài.

3.1.1.1 Unreal Engine (Version 5.5)

Engine chính được sử dụng để xây dựng toàn bộ demo. Unreal đáp ứng đầy đủ các yêu cầu của bài giữa kỳ như Blueprint, AI Behavior, Collision, Animation, UI và Particle System.

Những tính năng chính của Unreal được dùng trong demo:

- **Blueprint Visual Scripting** – xây dựng toàn bộ logic nhân vật, enemy, UI, xử lý sự kiện mà không dùng code C++.
- **Collision & Overlap** – dùng cho va chạm Player–Enemy, phát hiện đòn tấn công, tính sát thương.
- **AI System** – sử dụng *AIController*, *Behavior Tree*, *NavMesh* để xây dựng hành vi tuần tra, đuổi theo, và tấn công người chơi.
- **Animation Blueprint** – quản lý trạng thái Idle, Walk, Run, Attack của nhân vật và enemy.
- **Niagara VFX** – sử dụng tạo hiệu ứng máu, va chạm đạn, và hiệu ứng khi enemy bị tiêu diệt.
- **UMG UI Designer** – xây dựng UI HP bar, số kill, text thông báo.
- **Lighting & Environment Tools** – thiết kế ánh sáng, bóng đổ, skybox và bố cục map.

Unreal Engine là công cụ quan trọng nhất trong hệ thống vì toàn bộ demo được phát triển và chạy trực tiếp trong môi trường này.

3.1.1.2 Visual Studio

Dùng làm IDE để xem hoặc mở rộng code C++ (nếu cần). Trong demo này, phần lớn logic được xây bằng Blueprint nhưng IDE vẫn được dùng để:

- Kiểm tra cấu trúc project
- Debug lỗi compile khi thêm thành phần mới
- Tạo class kế thừa AIController / Character khi cần

3.1.1.3 Fab Assets

Sử dụng để lấy:

- Model môi trường (đá, cây, chi tiết trang trí...)
- Vật liệu PBR (Metal, Rock, Ground, Sand...)
- Một số mesh dùng cho Level Design

Các asset này được nhập thông qua Fab tích hợp trong Unreal.

3.1.1.4 Git / GitLab

- Lưu trữ mã nguồn
- Theo dõi thay đổi
- Tránh mất dữ liệu khi phát triển demo Unreal

3.1.1.5 OBS Studio

Công cụ dùng để quay video phần demo theo đúng yêu cầu Video Demo trong đề bài (HD 720p, thuyết trình bằng giọng nói)

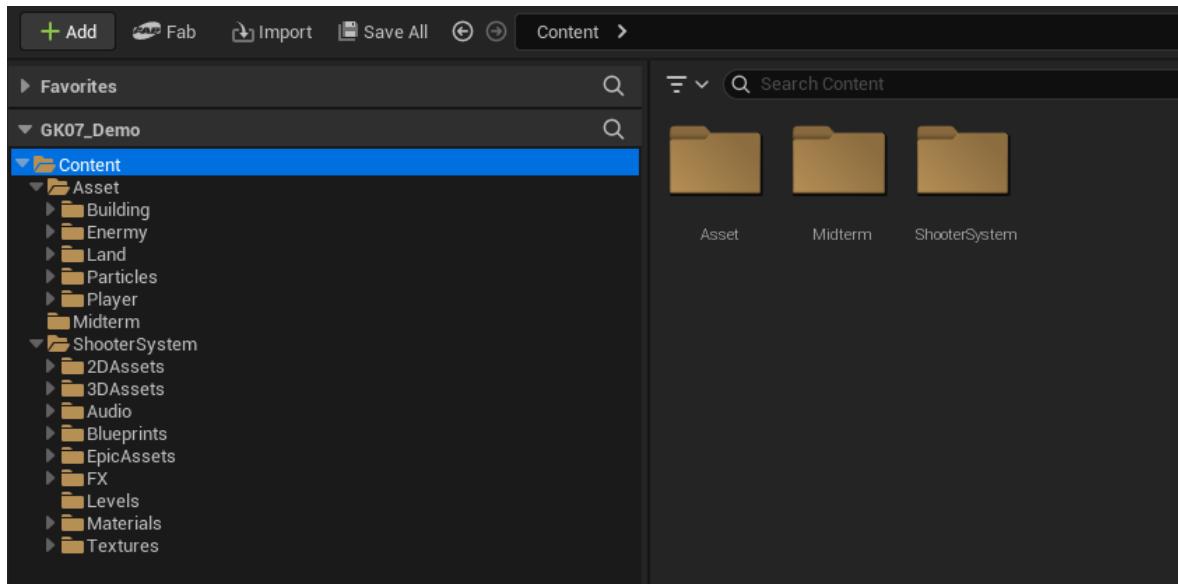
3.1.2 Cấu trúc project

Dự án giữa kỳ được tổ chức thành hai nhóm thư mục lớn:

- Asset/ (tài nguyên lấy từ Fab, Marketplace)
- ShooterSystem/ (module gameplay chính).

- Midterm/ chứa map demo chính.

Cách tổ chức này giúp tách biệt rõ giữa tài nguyên dùng cho map và logic gameplay, đồng thời đảm bảo việc phát triển, mở rộng và bảo trì dễ dàng.

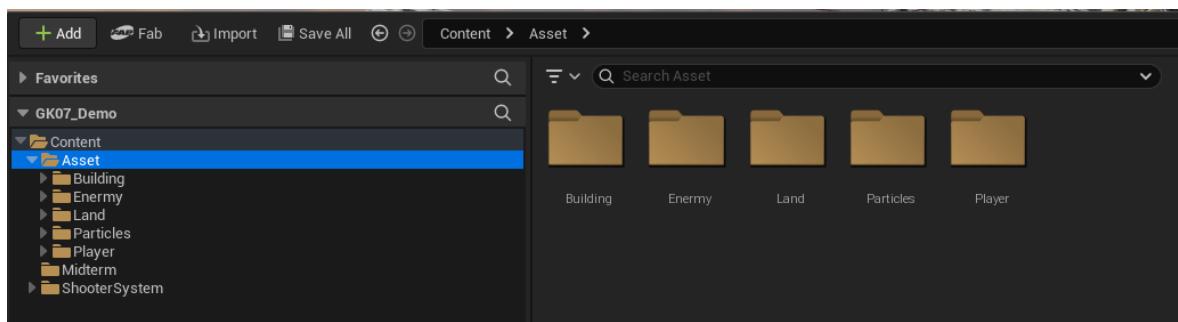


Hình 3.1 Cấu trúc tổ chức project

3.1.3 Các asset và tài nguyên được sử dụng

3.1.3.1 Thư mục Asset/

Đây là **thư mục Asset** chứa toàn bộ tài nguyên môi trường được lấy từ Fab, Quixel, hoặc Marketplace. Các tài nguyên này chủ yếu phục vụ cho Level Design và xây dựng map Midterm.



Hình 3.2 Thư mục asset

- **Building/:**

- Chứa các model dùng để dựng công trình, tường, nhà cửa, vật chắn.

- Tạo cấu trúc địa hình cho gameplay.
- Link: Modular Old Power Plant - 65 Assets
<https://www.fab.com/listings/006c8ac4-81fa-4835-819cb080f37e14ef>

- **Enemy/:**

- Mesh model của Enemy lấy từ Fab
- Link: Quantum Modular Character Free Sample
<https://www.fab.com/listings/8e200050-3158-4762-b297-f785b5b1533d>

- **Land/:**

- Chứa các mesh/texture/vật liệu dùng cho mặt đất, landscape hoặc địa hình.
- Dùng để xây dựng môi trường tự nhiên cho map.
- Link: MW Landscape Auto Material
<https://www.fab.com/listings/6602874e-ef24-48c9-9055-a7ac07384696>

- **Particles/:**

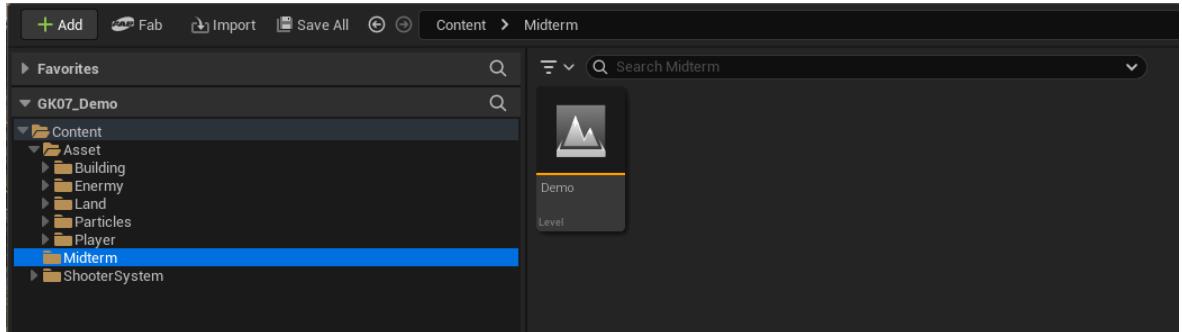
- Hiệu ứng lửa trại, khói: M5 VFX Vol2. Fire and Flames (Niagara)
<https://www.fab.com/listings/c5b0270a-a295-4644-a4be-42cb1e56a197>
- Hiệu ứng tuyết rơi: Particles and Wind Control System
<https://www.fab.com/listings/f673ef70-1c66-4c7f-8751-9f84ddb8b083>

- **Player/:**

- Mesh nhân vật Player.
- Dùng làm Skeleton Mesh cho BP_Player
- Link: Survival Character FREE
<https://www.fab.com/listings/11d20d01-b764-4936-8163-cb20d05c369e>

3.1.3.2 Thư mục Midterm/

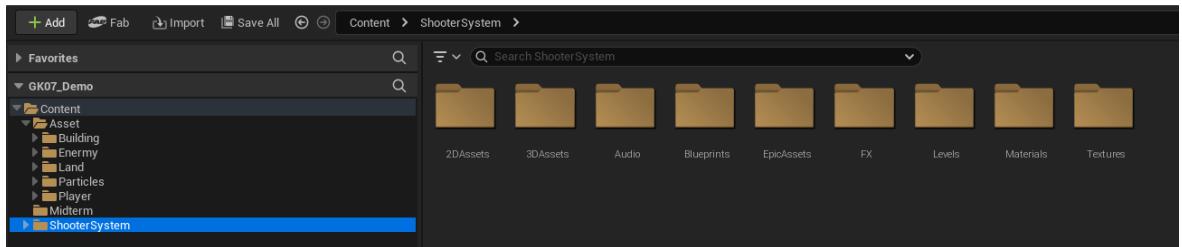
Đây là nơi bạn đặt map demo giữa kỳ. Đã được chuẩn bị tất cả để demo gameplay.



3.1.3.3 Thư mục ShooterSystem/

Đây là module được lấy từ phát triển từ nhiều nguồn, sau đó nhóm tự chỉnh sửa – mở rộng – lược bỏ phần dư để phục vụ đúng yêu cầu bài giữa kỳ. Mọi logic liên quan đến nhân vật, kẻ địch, AI, vũ khí, âm thanh, hiệu ứng và UI đều nằm trong thư mục này.

Cấu trúc thực tế gồm các thư mục con như hình:



- **2DAssets/:** Thư mục chứa toàn bộ tài nguyên hình ảnh 2D sử dụng cho hệ thống HUD và giao diện.
- **3DAssets/:** Chứa các mô hình 3D phục vụ cho hệ thống bắn súng và tương tác gameplay.
- **Audio/:** Chứa toàn bộ âm thanh phục vụ gameplay.
- **Blueprints/:** Đây là trái tim của hệ thống ShooterSystem, nơi chứa toàn bộ logic gameplay. Bao gồm:
 - **BP_PlayerCharacter**
 - Xử lý di chuyển, nhảy, chạy
 - Nhận input, điều khiển animation

- Logic bắn súng, spawn đạn
- **BP_EnemyCharacter**
 - Chứa logic cơ bản của enemy
 - Kết hợp AI Controller
- **BP_AIController**
 - Điều khiển enemy theo Behavior Tree
- **Behavior Tree (BT_Enemy)**
 - Tuần tra → Phát hiện → Đuổi theo → Tấn công
- **Blackboard (BB_Enemy)**
 - Lưu trữ thông tin target, state
 - ...
- **EpicAssets/**: Dùng để bổ sung chi tiết môi trường hoặc element phụ mà không cần tự tạo lại.
- **FX/**: Thư mục chứa các Niagara Effects dùng cho hiệu ứng đặc biệt.
- **Levels/**: Chứa các level thử nghiệm của ShooterSystem của nhóm.
- **Materials/**: Chứa toàn bộ vật liệu (Material) dành cho vũ khí, player, enemy và hiệu ứng.
- **Textures/**: Tạo độ chi tiết và chất lượng cho vật thể trong game.

CHƯƠNG 4. HỆ THỐNG NHÂN VẬT NGƯỜI CHƠI

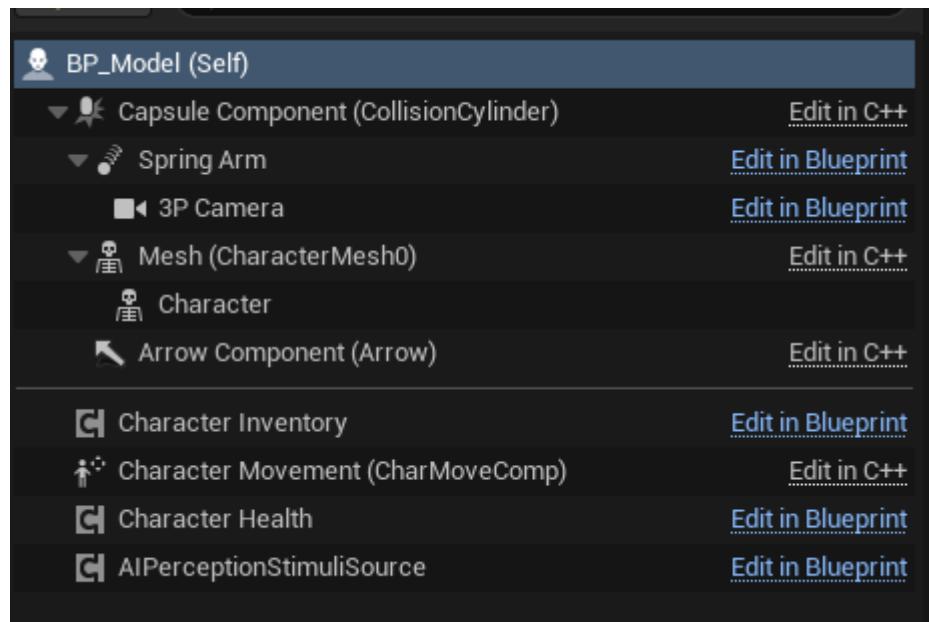
4.1 Player

Hệ thống nhân vật người chơi (Player System) được xây dựng dựa trên **Unreal Engine 5 Enhanced Input System**, tích hợp nhiều module: điều khiển di chuyển, điều khiển camera, hệ thống chiến đấu (shooting – aiming – reload), quản lý trang bị (equipment), tương tác môi trường (pickup – enter/exit vehicle), hệ thống sức khỏe (health), và cơ chế chuyển đổi giữa người chơi và phương tiện (vehicle driving).

Cấu trúc Player được phát triển từ **Template ThirdPerson** lấy từ Fab Unreal, sau đó mở rộng thành một nhân vật đa chức năng trong môi trường open-world.

4.1.1 Cấu trúc tổng thể của Player

Player Blueprint (BP_Model) bao gồm các thành phần chính:



Hình 4.1 Cấu trúc tổng thể của Player

- **CapsuleComponent**: collision chính giúp Player va chạm với môi trường và enemy.
- **Spring Arm + 3P Camera**: điều khiển camera góc nhìn TPS, theo sau nhân vật.
- **Mesh (CharacterMesh0)**: model nhân vật mannequin UE5

- **Character:** model nhân vật thật copy các animation từ Mesh.
- **Character Movement Component:** xử lý di chuyển: tốc độ, nhảy, trọng lực.
- **Character Health Component:** quản lý HP, bị trúng đạn, bị enemy đánh hoặc bị cán bởi xe.
- **Character Inventory Component:** lưu trữ, trang bị và chuyển đổi vũ khí.
- **AI Perception Stimuli Source:** phát tín hiệu giúp enemy AI có thể phát hiện Player.

Cách tổ chức này cho phép Player dễ dàng mở rộng, tách biệt các module để tái sử dụng hoặc nâng cấp.

4.1.2 Góc nhìn và điều khiển di chuyển

Player sử dụng góc nhìn **Third-Person** (TPS), camera bám phía sau lưng nhân vật, cho phép bao quát môi trường.

Hệ thống điều khiển sử dụng Enhanced Input, với các phím:

- **WASD** – di chuyển
- **Space** – nhảy
- **Chuột** – xoay camera
- **C** – crouch khi có hỗ trợ
- **LMB** – bắn
- **RMB** – aim
- **E** – tương tác
- **X/Z** – đổi vũ khí
- **R** – reload
- **Q** – mở inventory & đóng inventory



Hình 4.2 Góc nhìn TPS của nhân vật trong môi trường demo

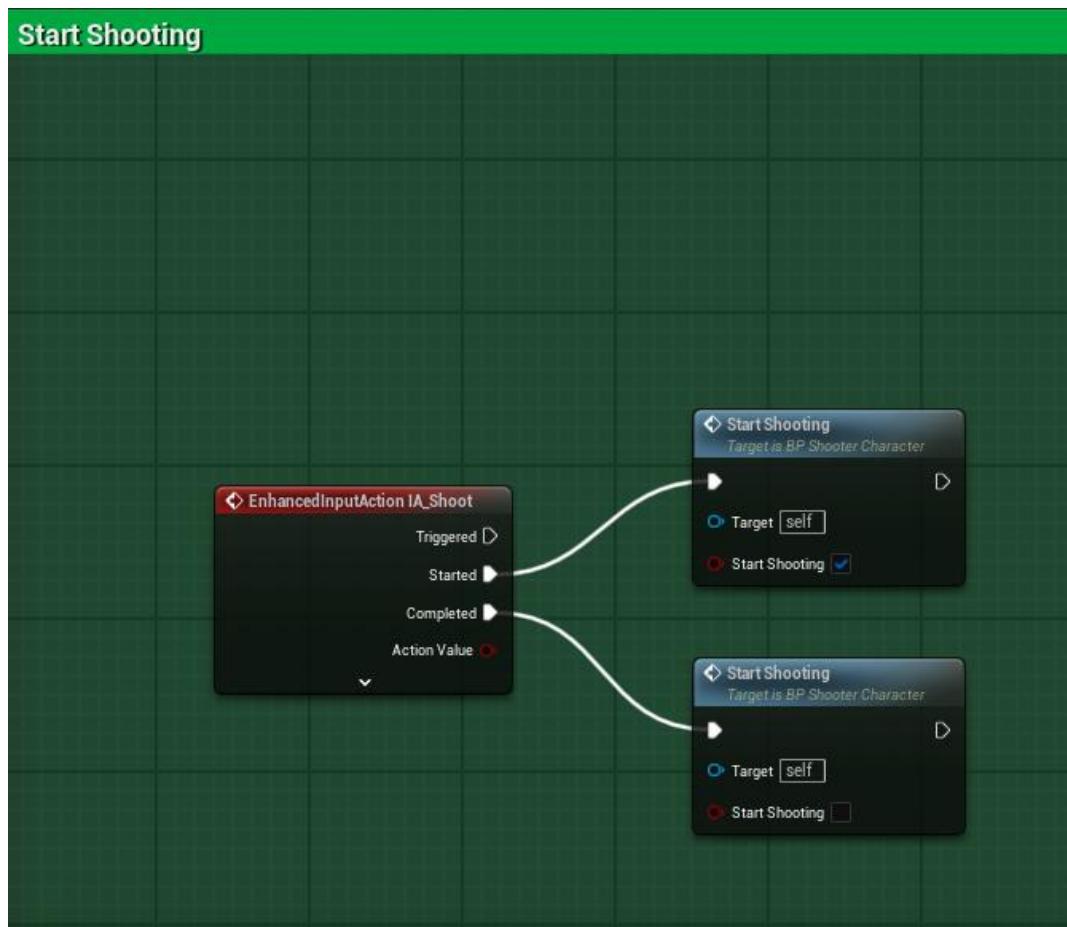
4.1.3 Hệ thống chiến đấu (Combat System)

4.1.3.1 Bắn súng – Start Shooting

Cơ chế bắn súng sử dụng IA_Shoot với ba trạng thái Triggered / Started / Completed.

Khi nhấn giữ chuột trái:

- StartShooting(true) được gọi → bắt đầu bắn
- Khi nhả → StartShooting(false) → dừng bắn

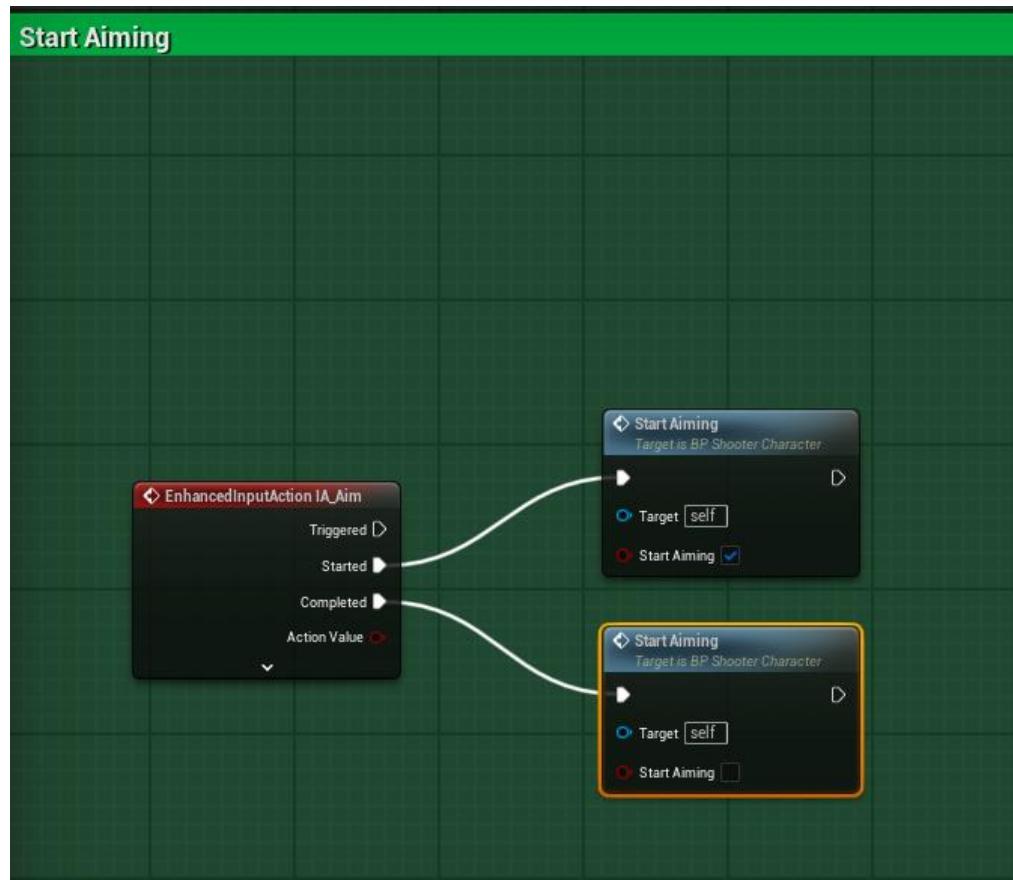


Hình 4.3 Blueprint xử lý bắn súng (Start Shooting)

4.1.3.2 Nhắm bắn – Start Aiming

Nhấn chuột phải kích hoạt chế độ ngắm bắn:

- Thu camera lại
- Giảm FOV
- Điều chỉnh chuyển động upper-body
- Tăng độ chính xác

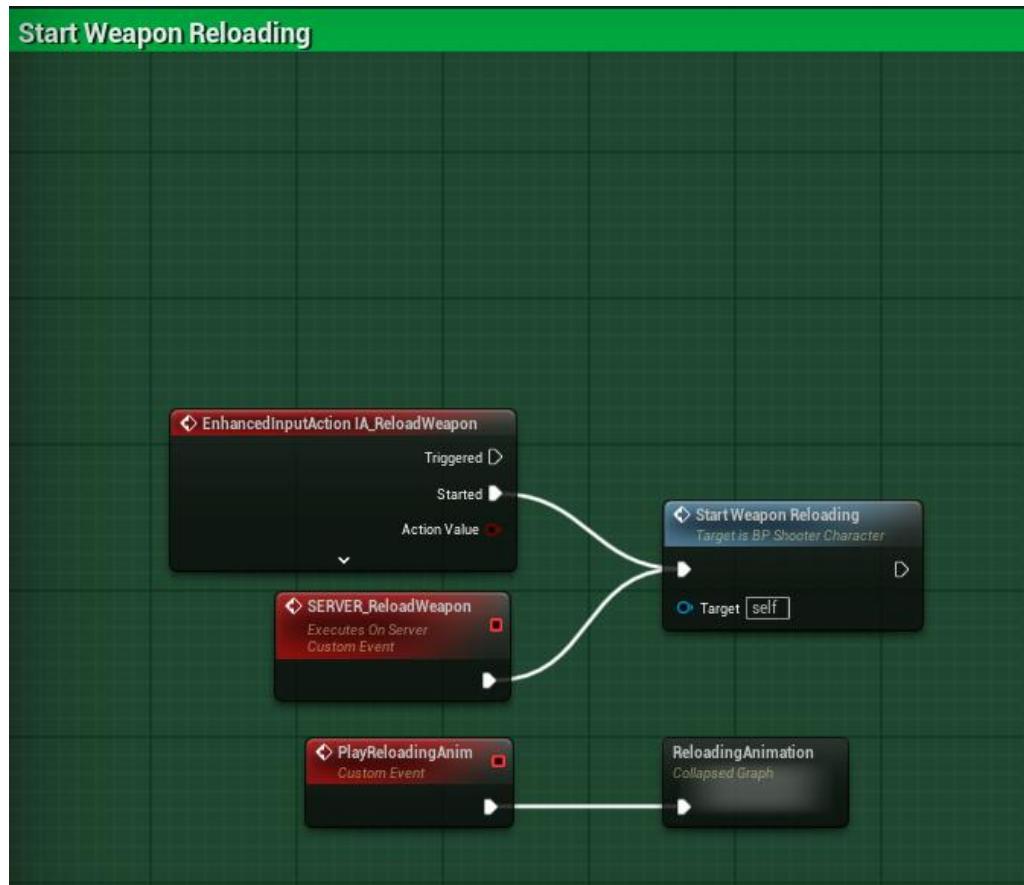


Hình 4.4 Blueprint xử lý nhắm bắn (Start Aiming)

4.1.3.3 Nạp đạn – Start Weapon Reloading

Nút **R** gọi **IA_ReloadWeapon**, bao gồm:

- StartWeaponReloading(self)
- SERVER_ReloadWeapon (event hỗ trợ network)
- PlayReloadingAnim (animation reload)



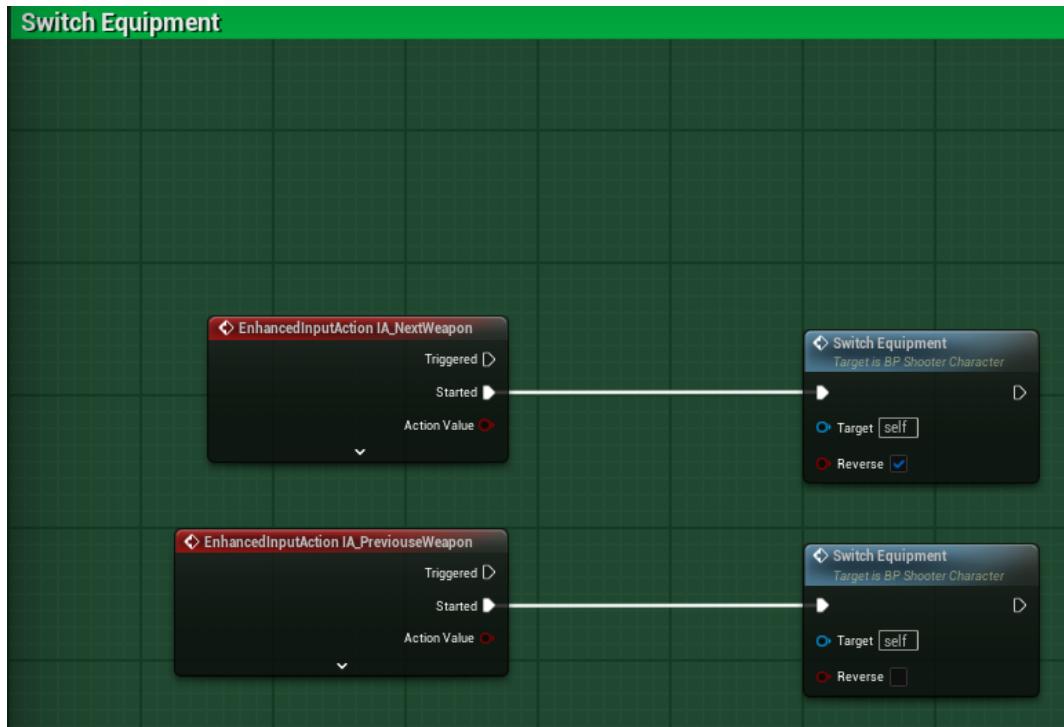
Hình 4.5 Blueprint nạp đạn và animation (Reloading)

4.1.4 Hệ thống trang bị (Equipment System)

4.1.4.1 Chuyển đổi vũ khí – Switch Equipment

Player có thể chuyển qua lại giữa nhiều loại vũ khí:

- IA_NextWeapon → SwitchEquipment (Reverse = false)
- IA_PreviousWeapon → SwitchEquipment (Reverse = true)

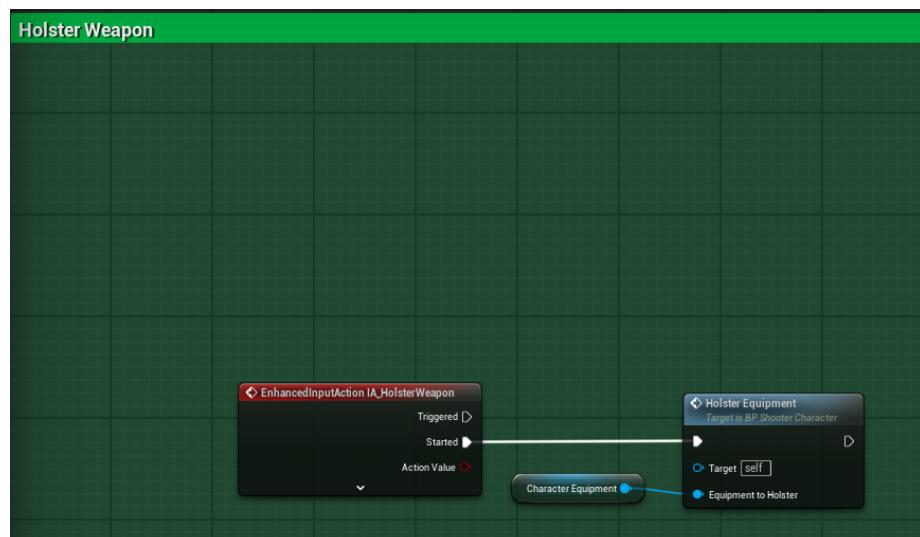


Hình 4.6 Blueprint chuyển đổi vũ khí (Next/Previous Weapon)

4.1.4.2 Cất vũ khí – Holster Weapon

Nhấn **H** để thu vũ khí:

- Animation hạ súng
- Tắt trạng thái “armed”
- Trở về Idle pose



Hình 4.7 Blueprint cất vũ khí (Holster Weapon)

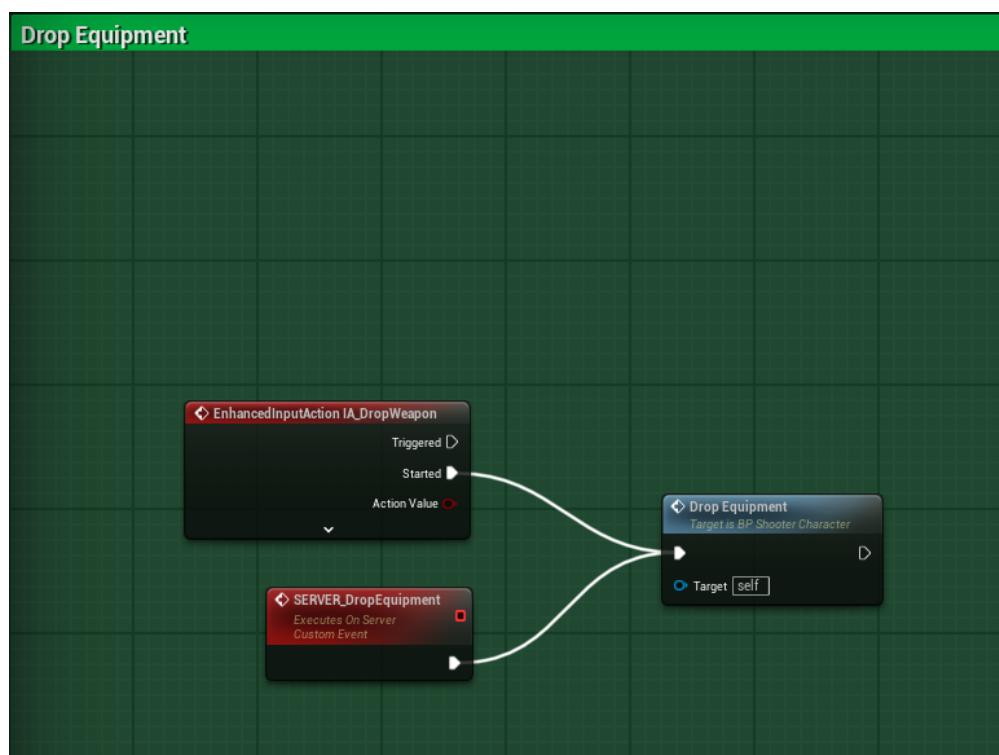
4.1.5 Vứt vũ khí – Drop Equipment

IA_DropWeapon thực thi:

- DropEquipment (self)
- SERVER_DropEquipment ()

Kết quả:

- Vũ khí rơi xuống đất
- Người chơi có thể nhặt lại hoặc đổi vũ khí khác



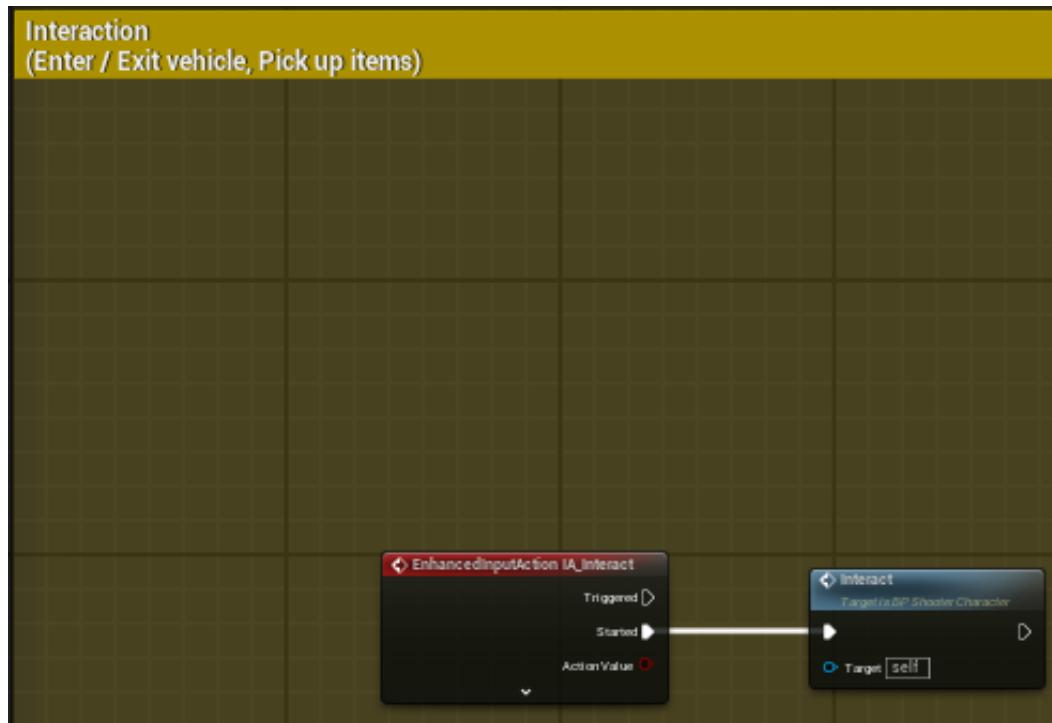
Hình 4.8 Blueprint vứt vũ khí (Drop Weapon)

4.1.6 Hệ thống tương tác (Interaction System)

Tương tác dùng IA_Interact và bao gồm:

- Nhặt vật phẩm (Rocket Ammo, Weapon)
- Mở/dóng inventory
- Vào xe / Ra xe

Logic sử dụng detection bằng Overlap → Get Interact Target.



Hình 4.9 Blueprint tương tác (Enter Vehicle / Pickup Items)

4.1.7 Hệ thống sức khỏe – Health System

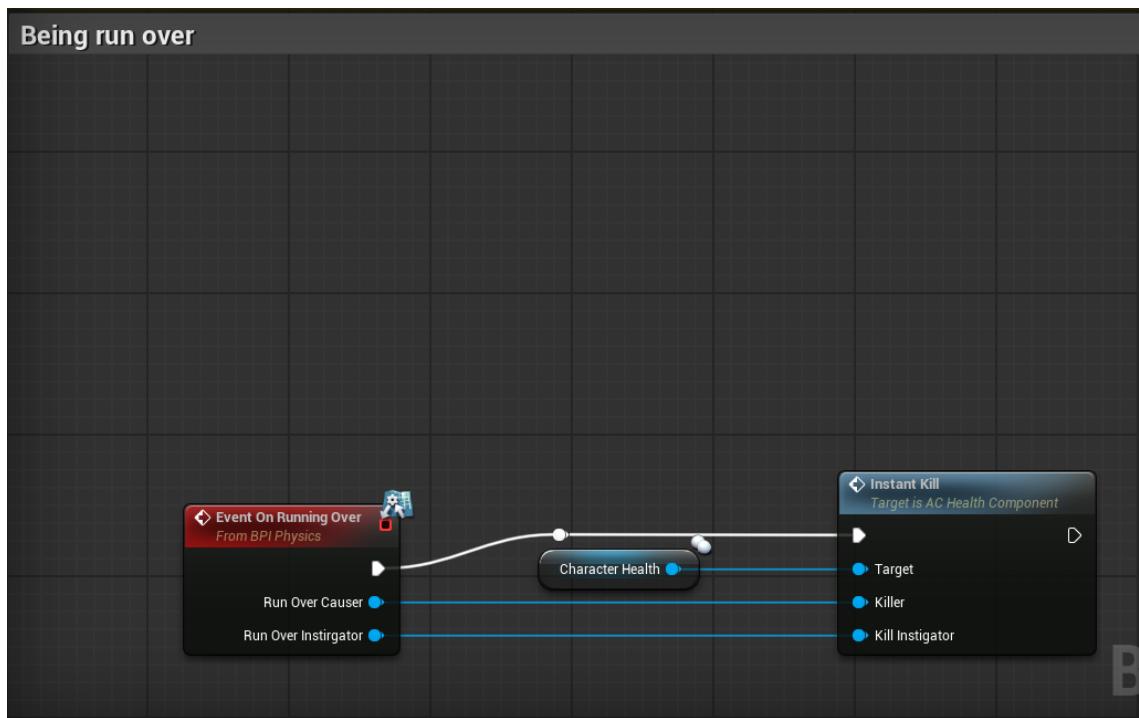
Player có thanh HP và giảm máu khi:

- Bị enemy tấn công
- Bị bắn
- Bị xe cán

Event đặc biệt:

Being Run Over

- Nhận event OnRunningOver từ Vehicle
- Gọi InstantKill () trong HealthComponent
→ Player/Enemy chết ngay lập tức khi bị xe cán.



Hình 4.10 Blueprint xử lý xe cán người chơi (Instant Kill)

4.1.8 Hệ thống lái xe (Vehicle Control System)

Player có thể điều khiển xe qua Interaction.

4.1.8.1 Vào xe – Get In

- Player bị disable movement
- Camera chuyển sang camera của xe
- Input chuyển sang BP_Vehi_Car

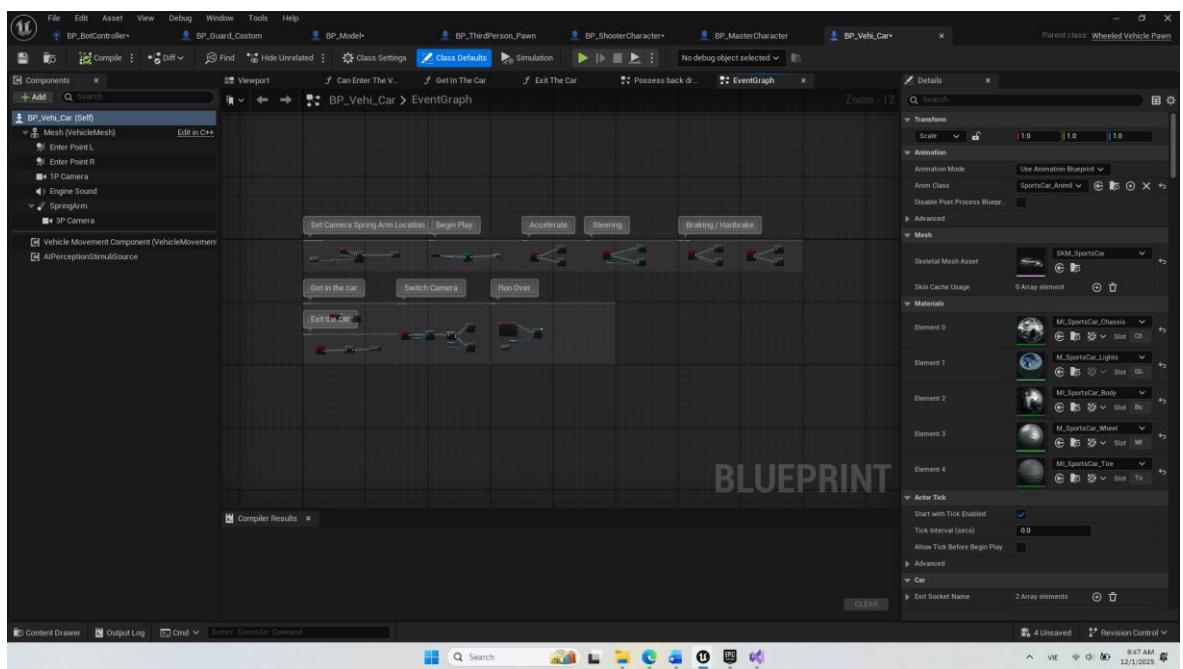
4.1.8.2 Điều khiển xe

Input:

- **W** → Accelerate
- **S** → Reverse
- **A/D** → Steering
- **Space** → Handbrake

4.1.8.3 Ra xe – Exit

- Camera trả về Player
- Trả quyền điều khiển
- Ra khỏi vị trí ghé lái



Hình 4.11 Blueprint hệ thống xe (Accelerate, Steering, Get in/Exit Vehicle)

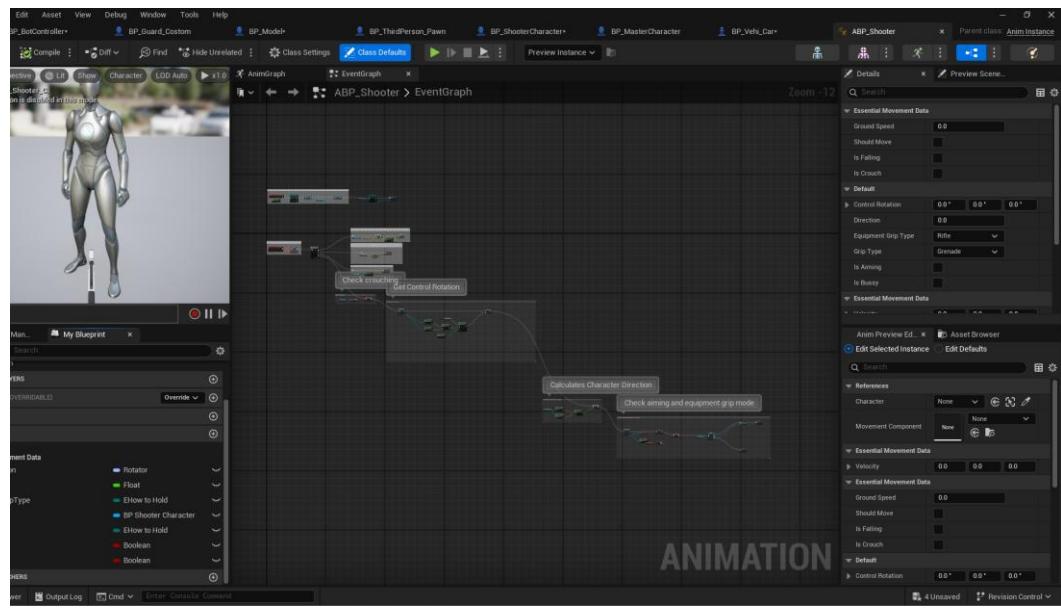
4.1.9 Animation Blueprint

Player sử dụng animation retarget từ mannequin:

- Idle
- Walk
- Run
- Jump

State Machine chuyển đổi dựa theo các tham số:

- Speed
- IsInAir
- IsArmed / IsAiming



Hình 4.12 Animation State Machine của Player

4.1.10 Giao diện người chơi (Player UI)

Các UI chính gồm:

- HP Bar
- Ammo
- Menu / Inventory
- Hướng dẫn điều khiển (Control Guide)

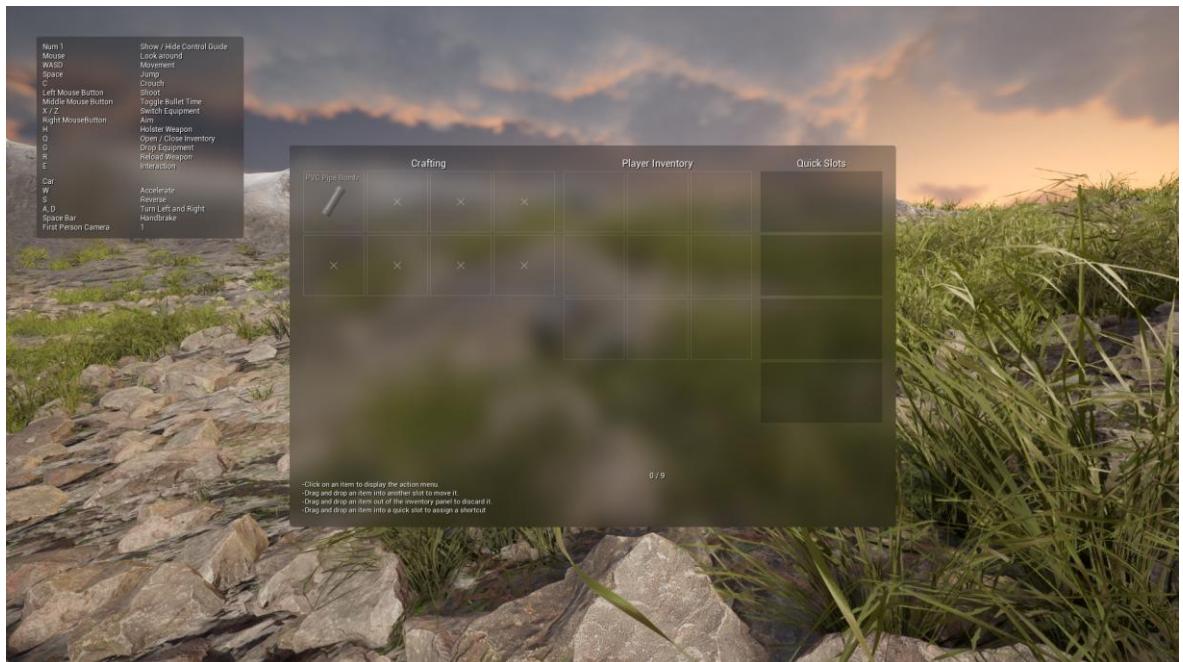
UI liên tục được cập nhật bằng Event Dispatcher và giá trị trong Health / Inventory component.



Hình 4.13 UI hiển thị bảng hướng dẫn điều khiển



Hình 4.14 UI hiển thị HP Bar



Hình 4.15 UI hiển thị inventory

4.1.11 Collision & tương tác gameplay

Các collision quan trọng:

- **CapsuleComponent** để va chạm với môi trường và enemy.
- **Hit / Overlap Event** để trigger damage, pickup items, hoặc kích hoạt UI.

4.2 Enemy AI

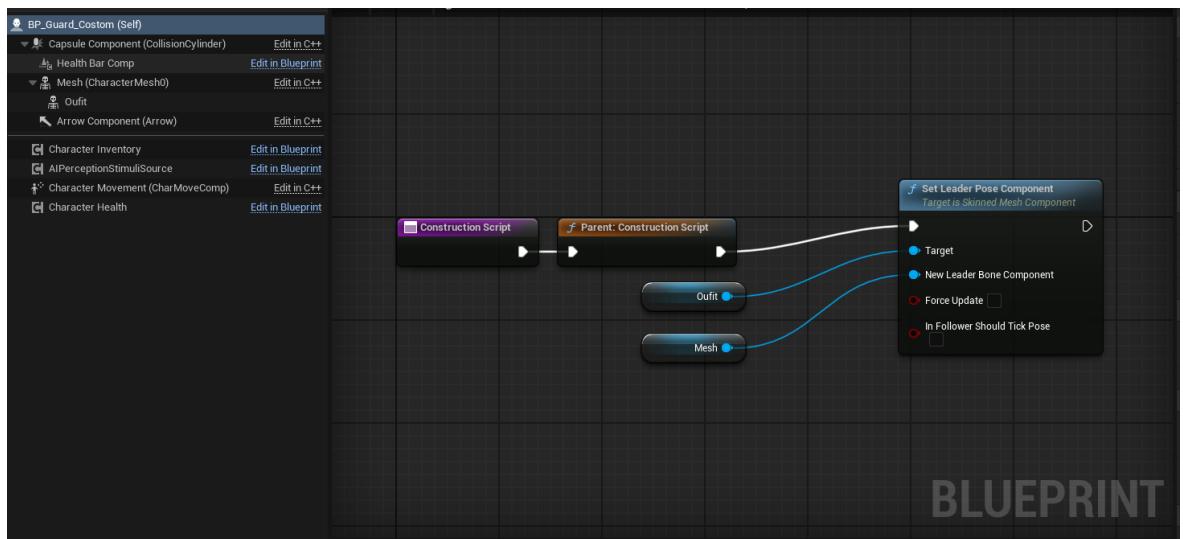
Enemy được xây dựng trong Blueprint **BP_Guard_Custom**, kế thừa từ **BP_ShooterCharacter**. Do đó, AI sở hữu toàn bộ khả năng của người chơi:

- Di chuyển
- Nhắm bắn
- Bắn súng bằng line trace
- Nạp đạn
- Animation Idle / Walk / Shoot / Aim / Reload
- Health Component

BP_Guard_Custom bao gồm các component chính:

- Capsule Collision

- Mesh (nhân vật người model mặc định)
 - Outfit (model nhân vật cho enemy)
- Character Movement Component
- Character Health
- Character Inventory (quản lý súng đạn)
- AI Perception Stimuli Source (để enemy khác có thể detect nó nếu cần)



Hình 4.16 Components của BP_Guard_Custom

4.2.1 Hệ thống cảm biến – AI Perception (Sight)

AI sử dụng **AI Perception với Sense Sight**, cho phép:

- Phát hiện Player trong tầm nhìn
- Kiểm tra line of sight (không xuyên tường)
- Kích hoạt các Task trong Behavior Tree dựa trên biến Blackboard “Enemy Spotted”

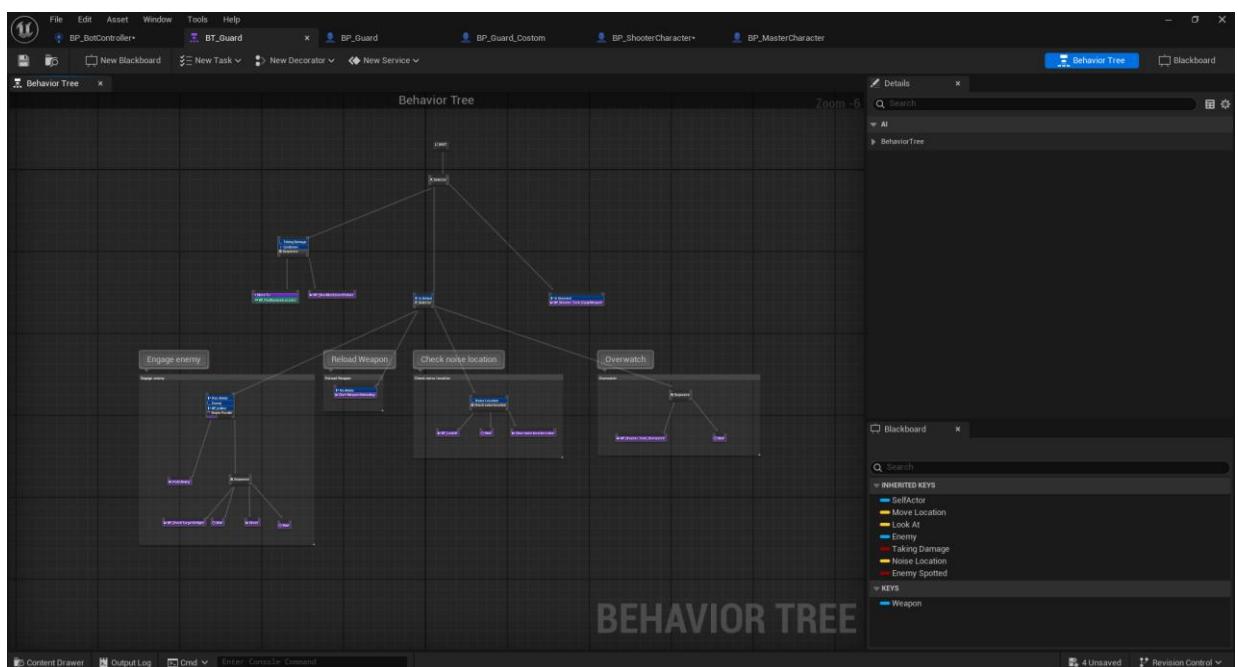
Khi Player lọt vào Sight:

- AI cập nhật Blackboard Key **Enemy**
- Hành vi chuyển sang **Engage Enemy**
- AI bắt đầu aim + bắn

4.2.2 Behavior Tree (BT_Guard)

Behavior Tree điều khiển toàn bộ logic AI. Struct chính:

- **Selector gốc**
- Kiểm tra trạng thái bị damage
- Kiểm tra có vũ khí hay không (Is Armed / Unarmed)
- Engage Enemy
- Reload Weapon
- Check Noise Location
- Overwatch (Quan sát môi trường khi rảnh)

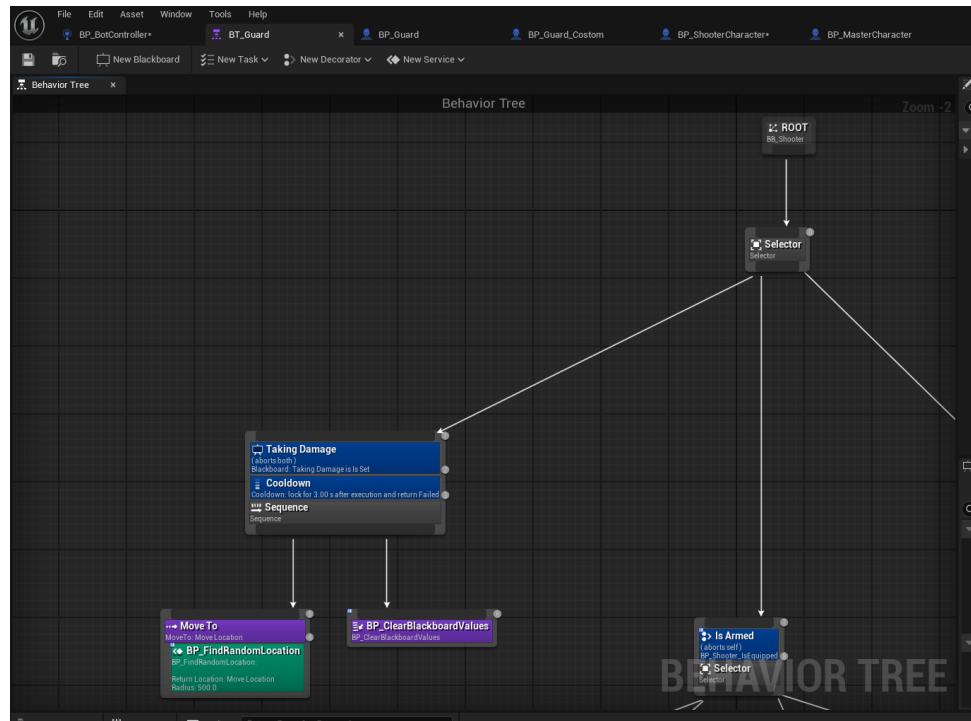


Hình 4.17 Behavior Tree tổng quan (BT_Guard)

4.2.2.1 Xử lý khi bị tấn công – Taking Damage

Khi AI nhận damage:

- Blackboard Setting: TakingDamage = true
- AI tạm dừng các hoạt động
- Di chuyển đến vị trí ngẫu nhiên (BP_FindRandomLocation)
- Xoá các key Blackboard cũ để reset trạng thái

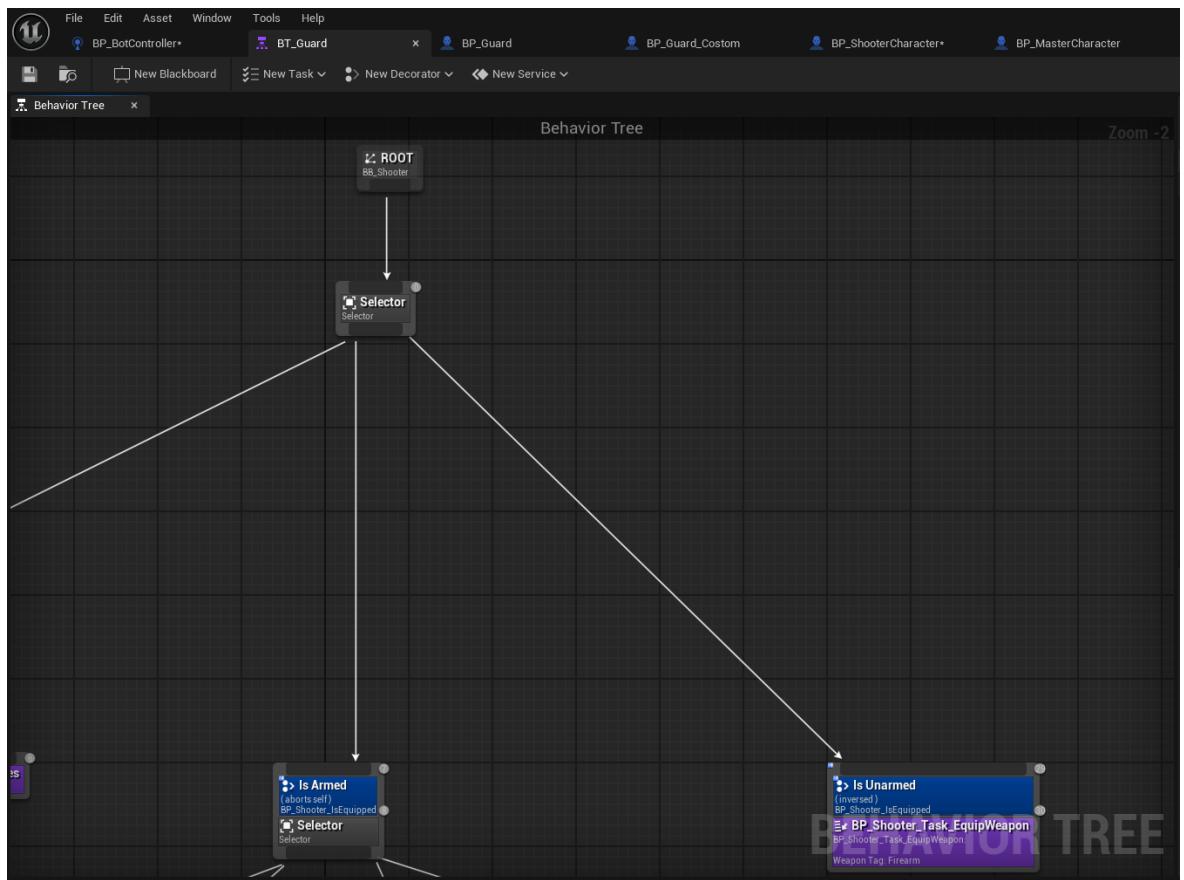


Hình 4.18 Nhánh Taking Damage

4.2.2.2 Kiểm tra trạng thái vũ khí – Is Armed / Is Unarmed

Nếu AI chưa cầm vũ khí → gọi Task EquipWeapon:
 BP_Shooter_Task_EquipWeapon(WeaponTag = Firearm)

Nếu đã có vũ khí → chuyển đến nhánh Engage Enemy hoặc Overwatch.



Hình 4.19 Nhánh Is Armed / Is Unarmed

4.2.2.3 Engage Enemy – Hành vi tấn công người chơi

Đây là nhánh quan trọng nhất. Hành vi bao gồm:

1. Has Ammo?

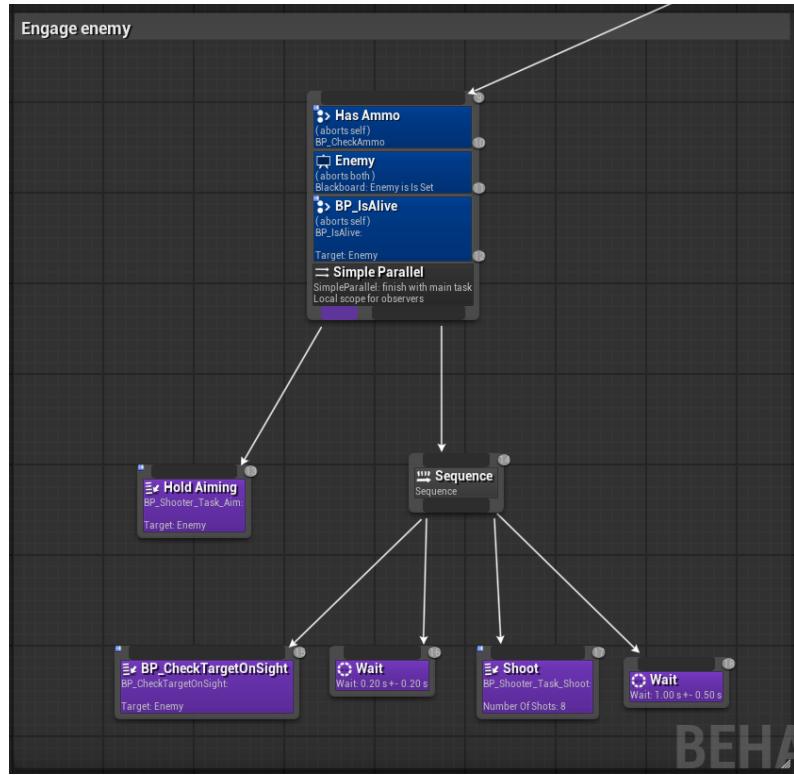
- Nếu có đạn → tiếp tục
- Nếu hết đạn → chuyển sang “Reload Weapon”

2. Enemy is Set?

- Kiểm tra Player có trong tầm nhìn

3. Simple Parallel Task:

- **Hold Aiming** (Task giữ tư thế aim)
- **CheckTargetOnSight** (kiểm tra lại line of sight)
- **Shoot** (bắn bằng line trace)
- **Wait** nhằm pacing tốc độ bắn

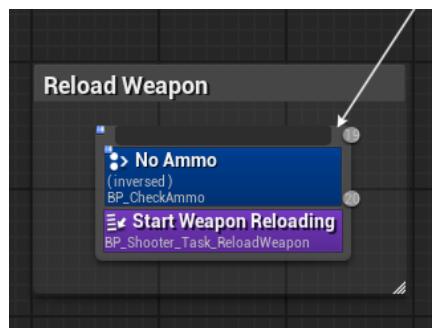


Hình 4.20 Nhánh Engage Enemy (AI bắn Player)

4.2.2.4 Reload Weapon – Nạp đạn khi hết ammo

Nếu ammo = 0:

- Decorator “No Ammo” kích hoạt
- Task “StartWeaponReloading”
- Gọi animation reload tương tự Player

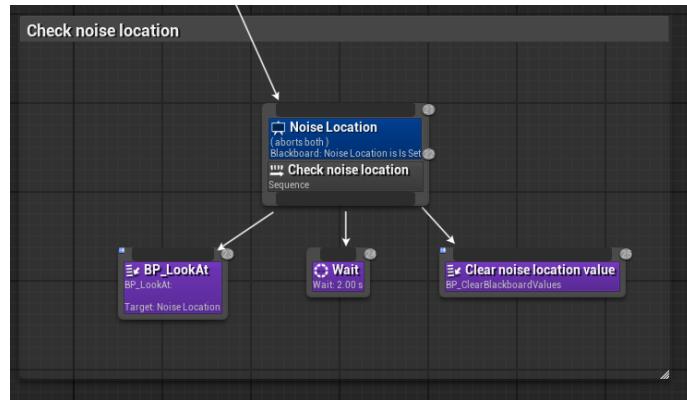


Hình 4.21 Nhánh Reload Weapon

4.2.2.5 Check Noise Location – Điều tra tiếng động

Khi AI nghe tiếng súng hoặc tiếng di chuyển:

- Blackboard cập nhật **NoiseLocation**
- AI quay mặt về noise (LookAt)
- Chờ 2 giây
- Xoá NoiseLocation để reset

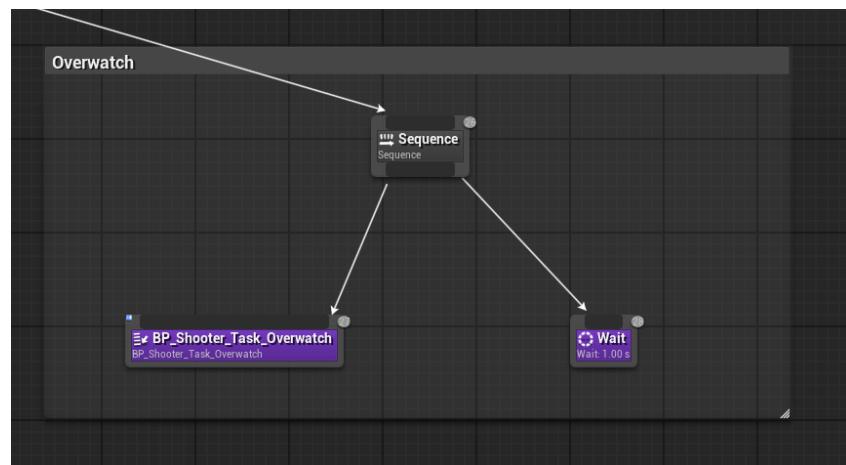


Hình 4.22 Kiểm tra tiếng động (Noise Investigation)

4.2.2.6 Overwatch – Quan sát khu vực khi rảnh

Nếu không có kẻ thù và không nghe tiếng động:

- AI quay đầu qua lại
- Đứng canh trong 1 giây
- Lặp lại



Hình 4.23 Nhánh Overwatch (AI quan sát môi trường)

4.2.3 Cơ chế tấn công – Line Trace Shooting

Enemy sử dụng cùng logic như Player, Đây là dạng bắn chính xác cao, dùng cho AI chiến đấu tầm xa.

- Khi trong Engage Enemy → Task “Shoot” thực hiện line trace
- Kiểm tra hướng bắn
- Gây damage lên Player
- Trigger animation bắn

4.2.4 Cơ chế di chuyển & chết

AI di chuyển bằng:

- MoveTo (Task tiêu chuẩn của UE)
- BP_FindRandomLocation khi bị bắn
- Di chuyển nhẹ sang vị trí mới để né tránh
- Không có patrol waypoint cố định, mà AI có kiểu “patrol tại chỗ” bằng Overwatch.

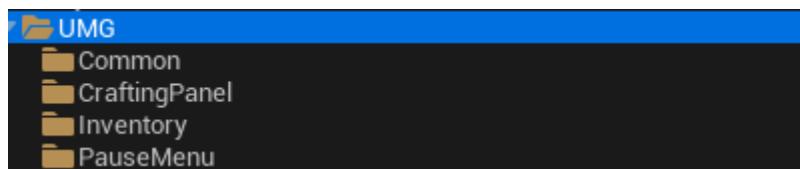
Enemy có Health Component. Khi HP = 0:

- Chạy animation chết: gục xuống, mất xương
- Xoá khỏi thế giới sau khoảng thời gian ngắn

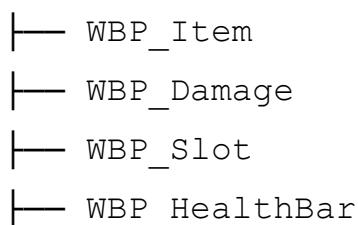
4.3 UI

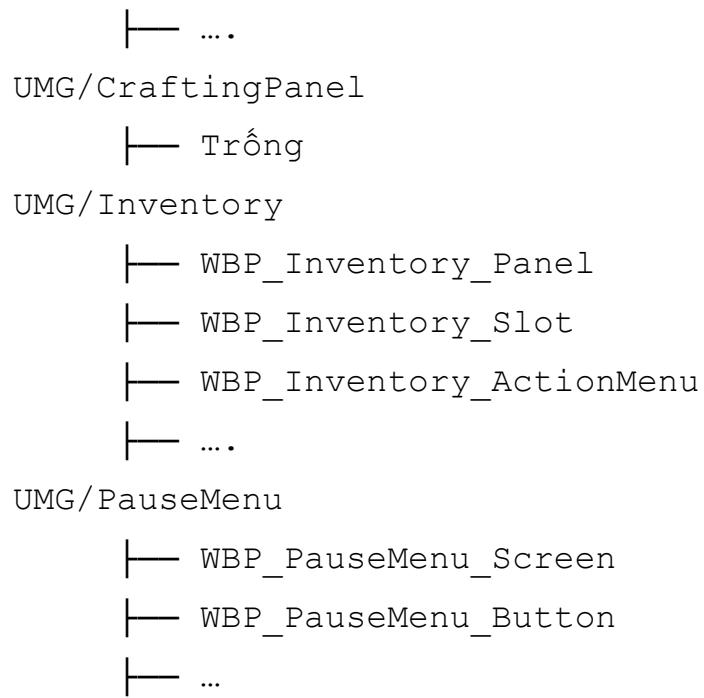
4.3.1 Kiến trúc tổ chức UI

UI được chia thành module rõ ràng:



UMG/Common



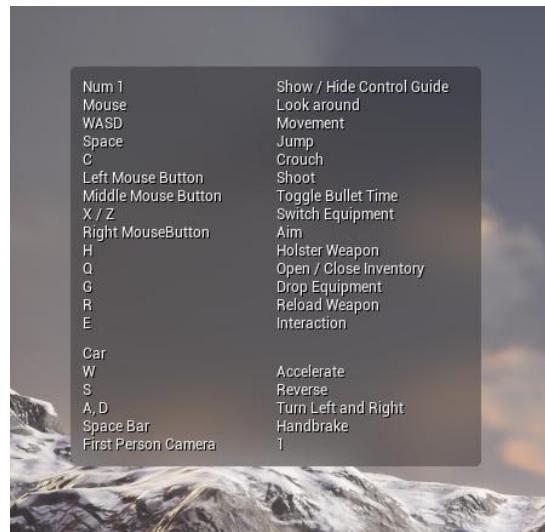


Cách tổ chức này đảm bảo asset dễ tái sử dụng và dễ bảo trì.

4.3.2 UI điều khiển (Control Guide)

Control Guide là bảng hướng dẫn phím tắt hiển thị ở góc trái màn hình. Người chơi có thể bật/tắt bằng phím **Num 1**.

- Là widget dạng overlay semi-transparent
- Hiển thị toàn bộ phím điều khiển: Movement, Shooting, Aim, Inventory, Car Driving...
- Không gây ảnh hưởng đến gameplay khi bật



Hình 4.24 Giao diện Control Guide



Hình 4.25 UI fullscreen của giao diện Control Guide

4.3.3 UI hiển thị vũ khí & đạn (Weapon HUD)

Khi người chơi trang bị vũ khí, UI sẽ hiển thị:

- Icon vũ khí
- Tên vũ khí
- Số lượng đạn hiện tại / tối đa (ví dụ: **1 / 4**)

UI này chỉ xuất hiện khi người chơi đang cầm vũ khí.

Cơ chế cập nhật:

- Khi bắn → gửi event update ammo
- Khi reload → cập nhật lại full ammo
- Khi đổi vũ khí → cập nhật weapon info mới



Hình 4.26 UI fullscreen khi trang bị Rocket Launcher

4.3.4 Hệ thống Inventory

Inventory là một phần UI phức tạp nhất trong hệ thống, bao gồm 3 khu vực:

- **Crafting Panel:** Dành cho các vật phẩm chế tạo (phát triển trong tương lai, phần này đang trống tại hiện tại)
- **Player Inventory**
 - Lưới nhiều ô chứa item (đạn, súng, vật phẩm craft...).

Cho phép:

- Kéo – thả để di chuyển slot
- Kéo ra ngoài để vứt item
- Chọn item để xem menu hành động (Equip / Use / Discard)

- **Quick Slots:** Các ô bên phải dùng để gán vật phẩm làm phím tắt.

Cách mở/tắt: Nhấn **G** để hiện hoặc đóng Inventory.

Logic cập nhật: Khi nhặt item hoặc dùng item → InventoryComponent trigger event → Inventory UI cập nhật lại danh sách.



Hình 4.27 Giao diện Inventory (Crafting + Inventory + Quick Slots)

4.3.5 UI Pause Menu (ESC)

Khi người chơi nhấn **ESC**, Pause Menu xuất hiện.

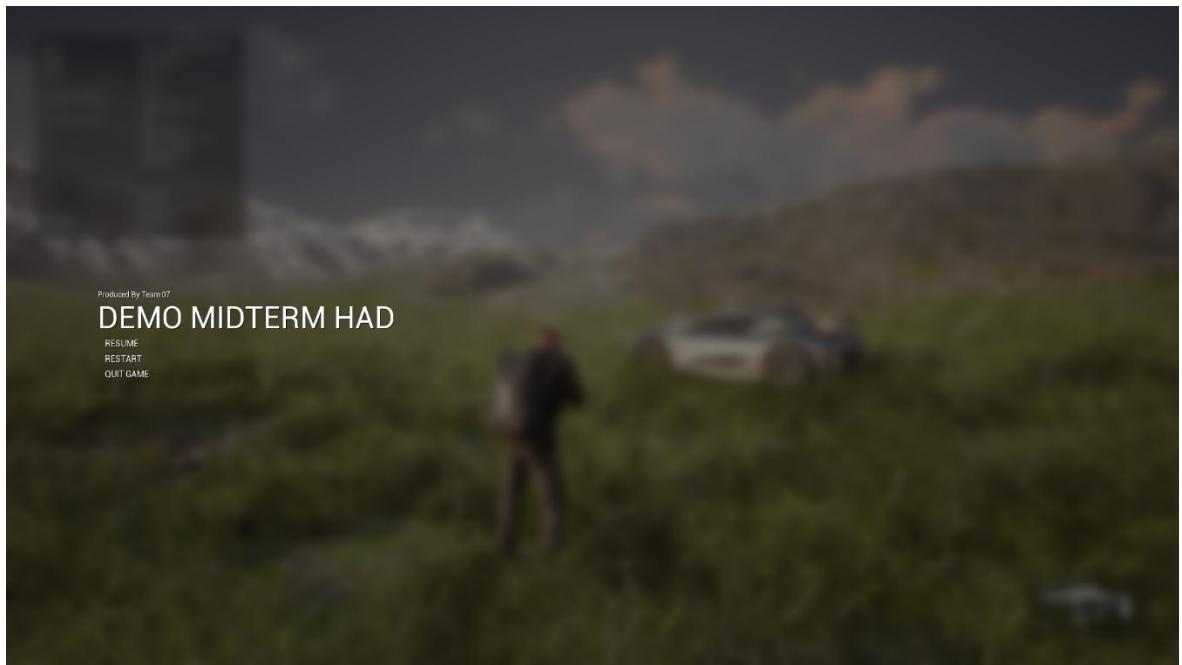
Menu bao gồm:

- **Resume**
- **Restart**
- **Quit Game**

Đồng thời màn hình nền được làm mờ (Gaussian Blur), tạo hiệu ứng tập trung vào menu.

Cơ chế:

- Mở menu → SetGamePaused (true)
- Resume → SetGamePaused (false)
- Các nút được tạo trong widget **WBP_PauseMenu_Screen**



Hình 4.28 Giao diện Pause Menu

4.3.6 HP Bar & Health UI

Trong mục Common, hệ thống có các widget:

- WBP_HealthBar
- WBP_HealthGauge
- WBP_Damage

HP Bar hoạt động như sau:

- Nhận dữ liệu từ CharacterHealthComponent
- Cập nhật % HP qua SetPercent
- Xuất hiện khi Player bị tấn công



Hình 4.29 UI hiển thị máu và tương tác khi bị bắn

4.3.7 Pickup Prompt UI (tự động)

Khi Player đứng gần vật phẩm, vũ khí (ví dụ Rocket Ammo), hệ thống hiển thị text prompt nhỏ bên cạnh item:



Hình 4.30 UI hiển thị vật phẩm

CHƯƠNG 5. DEMO GAMEPLAY

Chương này trình bày quá trình trải nghiệm của người chơi trong bản demo, mô tả các hệ thống gameplay chính đã được xây dựng: Player, Combat, Inventory, Enemy AI, Vehicle và UI. Các nội dung được trích xuất trực tiếp từ bản dựng thực tế của dự án.

5.1 Mục tiêu của bài demo

Bản demo được thiết kế để trình bày các tính năng cốt lõi của hệ thống game bắn súng góc nhìn thứ ba đủ để thể hiện các thành phần Engine, bao gồm:

- Điều khiển nhân vật (TPS Movement & Camera)
- Chiến đấu tầm xa (Shooting – Aiming – Reloading)
- Tương tác vật phẩm và hệ thống Inventory
- Đối thủ AI có khả năng chiến đấu thực tế
- Điều khiển xe và cơ chế cán chết (Run Over)
- Hệ thống UI đầy đủ: HUD, Inventory, Pause Menu, Control Guide

Các tính năng trên được tích hợp thành một vòng lặp gameplay hoàn chỉnh.

5.2 Bối cảnh bản đồ (Map Context)

Bản đồ demo đặt nhân vật trong một khu vực ngoài trời rộng lớn, gồm:

- Địa hình núi tuyết và thảm cỏ
- Một chiếc xe thể thao phục vụ cơ chế lái xe
- Một số vật phẩm như Rocket Ammo
- Enemy AI được đặt rải rác trong map



Hình 5.1 Toàn cảnh bản đồ và nhân vật khi bắt đầu demo



Hình 5.2 Toàn cảnh bản đồ và nhân vật khi bắt đầu demo

5.3 Luồng trải nghiệm gameplay tổng quát

Người chơi sẽ trải nghiệm demo theo thứ tự tự nhiên như sau:

- Người chơi xuất hiện trong map, nhìn thấy Control Guide hướng dẫn phím bấm.
- Di chuyển, xoay camera, nhảy và thử các thao tác cơ bản.
- Nhặt vật phẩm xuất hiện gần nhân vật.
- Mở Inventory để xem item, quản lý slot, kéo– thả vật phẩm.
- Trang bị vũ khí và thực hiện bắn thử.
- Chạm trán Enemy AI ở gần hoặc ở tòa Building → AI phát hiện và tấn công người chơi.
- Người chơi bắn hạ AI hoặc né tránh.
- Người chơi vào xe, điều khiển xe chạy quanh bản đồ.
- Nhấn ESC để mở Pause Menu.

5.4 Trình diễn từng hệ thống Gameplay

5.4.1 Di chuyển nhân vật & Góc nhìn

Người chơi điều khiển nhân vật theo góc nhìn thứ ba (Third-Person Shooter):

- WASD: di chuyển
- Chuột: xoay camera
- Space: nhảy
- C: crouch

Hệ thống animation gồm: Idle → Walk → Jump (nếu có).



Hình 5.3 Gameplay nhân vật idle ở góc nhìn TPS



Hình 5.4 Gameplay nhân vật walk ở góc nhìn TPS



Hình 5.5 Gameplay nhân vật jump ở góc nhìn TPS



Hình 5.6 Nhân vật chết

5.4.2 Chiến đấu: Bắn – Nhắm – Nạp đạn

Hệ thống chiến đấu sử dụng **line trace shooting**, tương tự nhiều game TPS/FPS:

- LMB: bắn
- RMB: aim

- R: reload
- Vũ khí hiển thị ammo trên HUD

UI vũ khí hiển thị:

- Icon súng
- Tên vũ khí
- Đạn hiện tại / tối đa



Hình 5.7 UI vũ khí khi Player aim



Hình 5.8 UI vũ khí khi Player bắn



Hình 5.9 UI vũ khí khi Player nạp đạn

5.4.3 Hệ thống Inventory

Nhấn **G** để mở Inventory.

Inventory được chia thành 3 phần:

- **Crafting Panel** – vật phẩm chế tạo (chưa phát triển)

- **Player Inventory** – danh sách vật phẩm
- **Quick Slots** – gán vật phẩm vào phím tắt

Tính năng:

- Drag & Drop vật phẩm
- Kéo item ra ngoài để vứt
- Menu hành động cho item: Equip, Use, Discard
- Gán vào Quick Slot bằng kéo thả



Hình 5.10 Giao diện Inventory với Crafting – Inventory – Quick Slots

5.4.4 Enemy AI Combat

Enemy AI có khả năng:

- Phát hiện Player bằng **AI Perception (Sight)**
- Quan sát khu vực bằng **Overwatch Behavior**
- Điều tra tiếng động bằng **Check Noise Location**
- Bắn Player bằng line trace như nhân vật người chơi
- Reload vũ khí khi hết đạn
- Phản ứng: di chuyển né khi bị bắn

AI tham chiến thông qua Behavior Tree với các nhánh chính:

- Engage Enemy
- Reload Weapon

- Check Noise
- Overwatch
- Taking Damage



Hình 5.11 Enemy AI trong lúc giao tranh với Player đang trong Car



Hình 5.12 Enemy AI chết

5.4.5 Hệ thống điều khiển xe (Vehicle Driving)

Người chơi có thể tương tác với xe:

- E: vào xe
- W/S: ga / lùi
- A/D: rẽ trái / phải
- Space: phanh tay

Hệ thống xe bao gồm:

- Chuyển camera sang CameraVehicle
- Chuyển input sang VehicleController
- Cơ chế **run over**: xe cán trúng Player hoặc Enemy gây chết ngay lập tức



Hình 5.13 Người chơi điều khiển xe trong demo

5.4.6 Hệ thống UI

Demo có 4 UI chính:

5.4.6.1 Control Guide

Hiển thị phím bấm các thao tác trong game.



Hình 5.14 Control Guide hiển thị trong gameplay

5.4.6.2 Weapon HUD

Hiển thị đạn / vũ khí đang sử dụng.



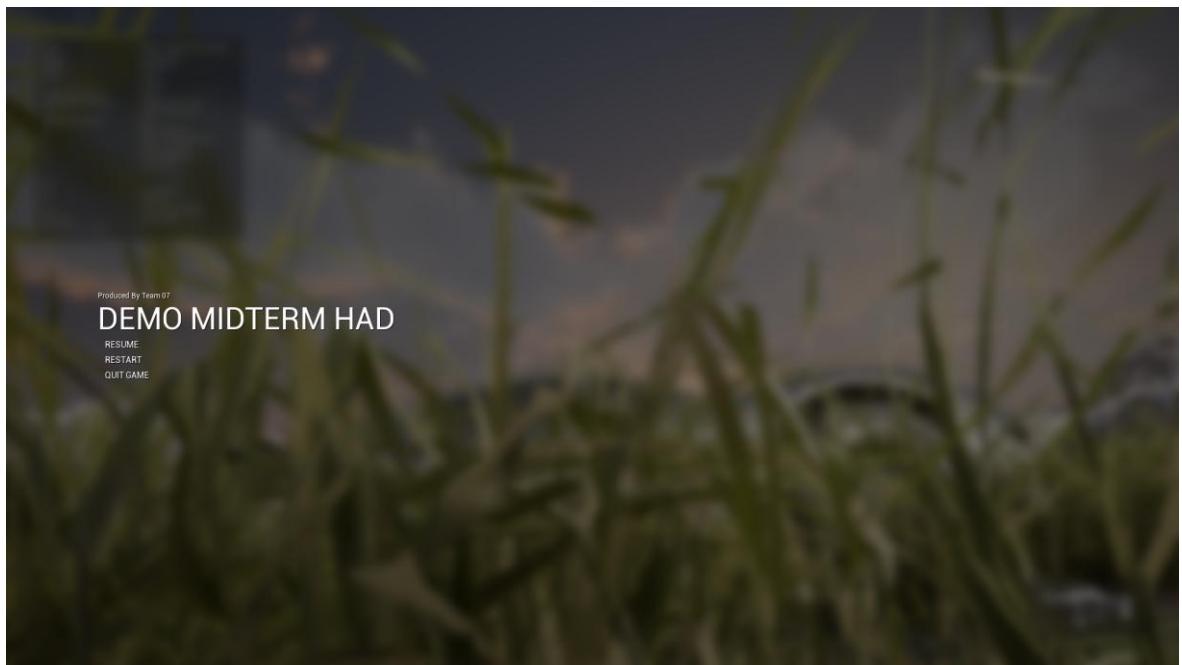
Hình 5.15 HUD vũ khí

5.4.6.3 Inventory UI



Hình 5.16 Inventory

5.4.6.4 Pause Menu



Hình 5.17 Menu Pause

CHƯƠNG 6. KẾT LUẬN

6.1 Kết luận

Qua quá trình tìm hiểu và xây dựng sản phẩm demo, nhóm đã hiểu rõ cấu trúc hoạt động của Unreal Engine 5, từ hệ thống lõi, workflow trong editor đến các công cụ hỗ trợ lập trình và thiết kế. Unreal Engine chứng tỏ khả năng mạnh mẽ trong việc tạo ra đồ họa chất lượng cao, hệ thống gameplay phức tạp và pipeline tối ưu cho các dự án lớn.

Demo được triển khai đã thể hiện được nhiều hệ thống gameplay thực tế: điều khiển nhân vật, chiến đấu, AI địch, hệ thống vật phẩm, UI và phương tiện. Điều này cho thấy Unreal Engine phù hợp không chỉ cho học tập mà còn cho các dự án game quy mô lớn, cinematic hoặc mô phỏng. Đề tài đã đạt được mục tiêu đề ra và giúp nhóm nâng cao kỹ năng lập trình game, thiết kế hệ thống và làm việc với một công nghệ chuyên nghiệp trong ngành.

6.2 Hướng phát triển

Với demo này, có thể mở rộng đề tài theo các hướng sau:

1. Mở rộng bản đồ và cơ chế thế giới mở (Open-world): Ứng dụng World Partition để quản lý world lớn.
2. Tích hợp hệ thống Multiplayer: Sử dụng Networking & Replication để hỗ trợ chơi online.
3. Xây dựng hệ thống nhiệm vụ (Quest System): Giúp gameplay phong phú hơn.
4. Cải thiện AI: AI thông minh hơn với EQS, Perception nâng cao và nhiều trạng thái chiến đấu.
5. Tối ưu hóa hiệu năng: Tối ưu ánh sáng, mesh, streaming và blueprint structure.
6. Bổ sung hiệu ứng VFX phức tạp: Kết hợp Niagara để tăng tính hấp dẫn cho gameplay.

TÀI LIỆU THAM KHẢO

- [1] E. Games, 2022. [Online]. Available: <https://www.unrealengine.com/en-US/blog/history-of-unreal-engine>.
- [2] W. contributors, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Unreal_Engine.
- [3] E. G. Documentation. [Online]. Available: <https://docs.unrealengine.com>.
- [4] E. Games, 2022. [Online]. Available: <https://www.unrealengine.com/en-US/unreal-engine-5>.
- [5] U. Technologies, 2025. [Online]. Available: <https://unity.com>.
- [6] Gamasutra, 2019. [Online]. Available: <https://www.gamasutra.com/>.