



UNIVERSIDAD DE MÁLAGA



Graduado en Ingeniería Informática

Desarrollo de un videojuego 3D de acción  
centrado en la IA enemiga en Unreal Engine 5.

Development of a 3D action video game centered  
on enemy AI in Unreal Engine 5.

Realizado por  
Carlos Quesada Cruz

Tutorizado por  
David Bueno Vallejo

Departamento  
Lenguajes y Ciencias de la Computación

MÁLAGA, septiembre de 2024



UNIVERSIDAD  
DE MÁLAGA



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA  
GRADUADO EN INGENIERÍA INFORMÁTICA

**Desarrollo de un videojuego 3D de acción centrado en la IA  
enemiga en Unreal Engine 5.**

**Development of a 3D action video game centered on enemy  
AI in Unreal Engine 5.**

Realizado por  
**Carlos Quesada Cruz**

Tutorizado por  
**David Bueno Vallejo**

Departamento  
**Lenguajes y Ciencias de la Computación**

UNIVERSIDAD DE MÁLAGA MÁLAGA,  
SEPTIEMBRE DE 2024

## **Resumen**

Este Trabajo de Fin de Grado se enfoca en el desarrollo de un videojuego perteneciente al género de acción. Se usará la herramienta de Unreal Engine. El documento presenta una breve introducción al género y se muestran ejemplos usados como referencias. Además, se hará mención al arte usado en el juego. Del mismo modo se mostrará cómo aplicar los procesos típicos del desarrollo software en el contexto de la creación de videojuegos. Además, se explorará la diversa documentación generada durante el proceso de diseño del juego. Por otra parte, se detallará cómo se ha realizado la implementación del software, destacando las decisiones de diseño y técnicas utilizadas para lograr una funcionalidad adecuada. Asimismo, se explicarán las herramientas usadas durante el proceso de desarrollo. Por último, se mostrará un análisis de las pruebas realizadas y sus resultados, exponiendo cómo se comporta el juego ante usuarios reales.

**Palabras clave:** Videojuego, Unreal Engine, Acción

## **Abstract**

This Final Degree Project focuses on the development of a video game belonging to the action genre. The Unreal Engine tool will be used. The document presents a brief introduction to the genre and shows examples used as references. In addition, the artwork used in the game will be mentioned. It will also show how to apply typical software development processes in the context of game creation. In addition, the various documentation generated during the game design process will be explored. On the other hand, it will detail how the implementation of the software has been carried out, highlighting the design decisions and techniques used to achieve adequate functionality. The tools used during the development process will also be explained. Finally, an analysis of the tests carried out and their results will be shown, explaining how the game behaves in front of real users.

**Keywords:** Videogames, Unreal Engine, Action.

# Índice

|                                                         |           |
|---------------------------------------------------------|-----------|
| <b>1 Introducción</b>                                   | <b>8</b>  |
| 1.1 Motivación                                          | 8         |
| 1.2 Objetivos                                           | 8         |
| 1.3 Estructura de la memoria                            | 8         |
| <b>2 Contexto</b>                                       | <b>10</b> |
| 2.1 Conceptos habituales                                | 10        |
| 2.2. Subgénero first-person action en los videojuegos   | 10        |
| 2.3. Referentes                                         | 11        |
| 2.3.1. The Elder Scrolls V: Skyrim                      | 11        |
| 2.3.2. Dishonored                                       | 12        |
| 2.3.3. Call of Duty: Modern Warfare II                  | 13        |
| 2.4. Metodologías de trabajo                            | 13        |
| 2.5. Tecnologías utilizadas                             | 14        |
| 2.5.1. Unreal Engine                                    | 14        |
| 2.5.2. Convai                                           | 14        |
| 2.6. Arte del juego                                     | 15        |
| 2.6.1. Assets de uso libre                              | 15        |
| 2.6.2. Música                                           | 16        |
| 2.6.3. Imágenes                                         | 16        |
| <b>3 Análisis, diseño y solución</b>                    | <b>22</b> |
| 3.1. Primer concepto                                    | 22        |
| 3.2. Aplicación metodología SCRUM                       | 22        |
| 3.3. Captura de requisitos                              | 24        |
| 3.3.1. Requisitos funcionales                           | 24        |
| 3.3.2. Requisitos no funcionales                        | 25        |
| 3.4. Diagrama de casos de uso                           | 26        |
| 3.5. Diagrama de flujo del comportamiento de mi enemigo | 28        |

|                                                   |           |
|---------------------------------------------------|-----------|
| 3.5.1 Sword Enemy.....                            | 28        |
| 3.6. Bocetos de interfaces de usuario .....       | 29        |
| 3.7. Mapas de Navegación.....                     | 32        |
| 3.8. Diseño de mecánicas de juego .....           | 32        |
| 3.9. Diseño de niveles .....                      | 33        |
| 3.10. Historia del juego.....                     | 34        |
| <b>4 Implementación .....</b>                     | <b>36</b> |
| 4.1. Menú e interfaces.....                       | 36        |
| 4.1.1. Menú Principal .....                       | 36        |
| 4.1.2 Menú de Nueva Partida .....                 | 39        |
| 4.1.3 Menú de Pausa.....                          | 40        |
| 4.1.4 Menú de Guardar Partida .....               | 42        |
| 4.1.5 Menú de Cargar Partida.....                 | 44        |
| 4.2. Sistema de guardado .....                    | 46        |
| 4.3 Cámara .....                                  | 49        |
| 4.4 Construcción del mundo.....                   | 50        |
| 4.4.1 Iluminación del mapa .....                  | 51        |
| 4.4.2 Creación Base de la isla .....              | 51        |
| 4.4.3 Creación océano.....                        | 53        |
| 4.4.4 Creación de la isla .....                   | 53        |
| 4.4.5 Vegetación y rocas .....                    | 55        |
| 4.4.6 Incorporación de assets adicionales .....   | 56        |
| 4.5 Personaje jugable.....                        | 57        |
| 4.5.1 Input Mapping Context e Input Actions ..... | 57        |
| 4.5.2 HUD (Head-Up Display).....                  | 58        |
| 4.5.3 Actor Component .....                       | 59        |
| 4.5.4 Interfaces .....                            | 60        |
| 4.5.5 GameMode .....                              | 62        |
| 4.5.6 Blueprint del personaje .....               | 63        |

|                                                                    |           |
|--------------------------------------------------------------------|-----------|
| 4.6 Enemigo .....                                                  | 65        |
| 4.6.1 AI Controller (AIController_BaseEnemy) .....                 | 65        |
| 4.6.2 Behavior Tree (BT_SwordEnemy) .....                          | 66        |
| 4.6.3 Blackboard (BB_SwordEnemy).....                              | 67        |
| 4.6.4 Tareas del Behavior Tree .....                               | 68        |
| 4.6.5 Blueprints de Enemigos: BP_Base_Enemy y BP_Sword_Enemy ..... | 70        |
| 4.7 NPC Convai .....                                               | 71        |
| 4.8 Creación de los ataques del jugador .....                      | 74        |
| 4.8.1 Espada.....                                                  | 74        |
| 4.8.2 Rayo .....                                                   | 78        |
| 4.9 Efectos visuales .....                                         | 82        |
| 4.9.1 Texturas .....                                               | 82        |
| 4.9.2 Materiales .....                                             | 83        |
| 4.9.3 Sistemas Niagara.....                                        | 85        |
| 4.9.4 Blueprints de Efectos Visuales.....                          | 86        |
| 4.10 Animaciones.....                                              | 88        |
| 4.10.1 Animación personaje .....                                   | 88        |
| 4.10.1.1 Animaciones (Main Character y Sword Enemy).....           | 88        |
| 4.10.2 Animación enemigo .....                                     | 92        |
| 4.10.2.1 Animación de Ataque (AM_SwordEnemy_Attack) .....          | 92        |
| 4.10.2.2 Blend Space de Locomoción (BS_Locomotion) .....           | 93        |
| 4.10.2.3 Animation Blueprint (ABP_Sword_Enemy).....                | 94        |
| <b>5 Pruebas .....</b>                                             | <b>95</b> |
| 5.1 Primera prueba (Alpha) .....                                   | 95        |
| 5.1.1 Versión 1.0.1.....                                           | 96        |
| 5.1.2 Versión 1.0.2.....                                           | 96        |
| 5.2 Segunda prueba (Beta) .....                                    | 96        |
| 5.2.1 Perfil del jugador .....                                     | 97        |
| 5.2.2 Experiencia de juego .....                                   | 98        |

|                                             |            |
|---------------------------------------------|------------|
| <b>6 Conclusiones y líneas futuras.....</b> | <b>101</b> |
| 6.1 Conclusiones.....                       | 101        |
| 6.2 Líneas futuras .....                    | 102        |
| <b>Referencias.....</b>                     | <b>103</b> |
| <b>Anexo I Game Design Document .....</b>   | <b>103</b> |
| <b>Anexo II Manual de usuario .....</b>     | <b>107</b> |

# Índice de ilustraciones

|                                                                                   |    |
|-----------------------------------------------------------------------------------|----|
| Ilustración 1: Captura de pantalla de The Elder Scrolls V: Skyrim .....           | 5  |
| Ilustración 2: Captura de pantalla del videojuego Dishonored .....                | 6  |
| Ilustración 3: Captura de pantalla del videojuego COD Modern Warfare II .....     | 7  |
| Ilustración 4: Pantalla de carga generada mediante IA generativa .....            | 11 |
| Ilustración 5:: Pantalla del menu principal generada mediante IA generativa ..... | 12 |
| Ilustración 6: Titulo de mi juego generado mediante IA generativa .....           | 13 |
| Ilustración 7: Logo Save Game generado mediante IA generativa .....               | 13 |
| Ilustración 8: Logo Load Game generado mediante IA generativa .....               | 14 |
| Ilustración 9: You win generado mediante IA generativa .....                      | 15 |
| Ilustración 10: Logo You Died generado mediante IA generativa .....               | 15 |
| Ilustración 11: Diagrama caso de uso jugabilidad .....                            | 22 |
| Ilustración 12: Diagrama caso de uso menú principal .....                         | 23 |
| Ilustración 13: Diagrama caso de uso menú pausa .....                             | 23 |
| Ilustración 14: Diagrama de flujo Sword Enemy .....                               | 25 |
| Ilustración 15: Boceto Vista usuario .....                                        | 26 |
| Ilustración 16: Boceto victoria .....                                             | 27 |
| Ilustración 17: Boceto derrota .....                                              | 27 |
| Ilustración 18: Boceto menú principal .....                                       | 28 |
| Ilustración 19: Boceto menú de pausa .....                                        | 28 |
| Ilustración 20: Mapa de navegación inicio .....                                   | 29 |
| Ilustración 21: Mapa de navegación juego .....                                    | 29 |
| Ilustración 22: Diseño nivel .....                                                | 31 |
| Ilustración 23: Interfaz menú principal .....                                     | 34 |
| Ilustración 24: Blueprint new game button .....                                   | 35 |
| Ilustración 25: Blueprint quit button .....                                       | 35 |
| Ilustración 26: Blueprint Load Button .....                                       | 36 |
| Ilustración 27: Blueprint New Game .....                                          | 36 |
| Ilustración 28: Interfaz carga del juego .....                                    | 37 |
| Ilustración 29: instrucciones del juego .....                                     | 37 |
| Ilustración 30: Interfaz menú de pausa .....                                      | 38 |
| Ilustración 31: Blueprint Resume Button de menú de pausa .....                    | 39 |
| Ilustración 32: Blueprint Quit Button de menú de pausa .....                      | 39 |
| Ilustración 33: Blueprint QuitButtonMainMenu de menú de pausa .....               | 39 |
| Ilustración 34: Blueprint Load Button de menú de pausa .....                      | 40 |
| Ilustración 35: Blueprint Save Button de menú de pausa .....                      | 40 |
| Ilustración 36: Blueprint Event Construct de Guardar partida .....                | 41 |
| Ilustración 37: Blueprint Back Button de Guardar partida .....                    | 42 |

|                                                                     |    |
|---------------------------------------------------------------------|----|
| Ilustración 38: Blueprint Save Button slot de Guardar partida ..... | 42 |
| Ilustración 39: Interfaz Guardar partida.....                       | 43 |
| Ilustración 40: Blueprint event construct de Cargar partida .....   | 43 |
| Ilustración 41: Blueprint Back Button de Cargar partida .....       | 44 |
| Ilustración 42: Blueprint de Load Button 1 de Cargar partida .....  | 44 |
| Ilustración 43: Blueprint Sistema de guardado .....                 | 47 |
| Ilustración 44: Blueprint load game .....                           | 47 |
| Ilustración 45: Blueprint game load .....                           | 48 |
| Ilustración 46: Blueprint event graph carga de partida.....         | 48 |
| Ilustración 47: ViewPort de mi jugador .....                        | 50 |
| Ilustración 48: Elementos de Iluminación de mi mundo .....          | 51 |
| Ilustración 49: Herramientas de modelaje del mundo .....            | 52 |
| Ilustración 50: Elementos de agua del mundo .....                   | 53 |
| Ilustración 51: Materiales de Quixel Bridge.....                    | 54 |
| Ilustración 52: Layer de materiales del mundo .....                 | 54 |
| Ilustración 53: Herramientas de capas del mundo .....               | 55 |
| Ilustración 54: Assets de vegetación de Quixel MegaScans .....      | 56 |
| Ilustración 55: Assets Rocas Quixel MegaScans .....                 | 56 |
| Ilustración 56: Herramienta de vegetación .....                     | 56 |
| Ilustración 57: Input Mapping Context .....                         | 57 |
| Ilustración 58: Input Actions .....                                 | 58 |
| Ilustración 59: Input Action Right Hand Attack.....                 | 58 |
| Ilustración 60: Blueprint HUD.....                                  | 59 |
| Ilustración 61: Blueprint Begin Play del Actor Component .....      | 60 |
| Ilustración 62: Blueprint OnTakeAnyDamage del Actor Component ..... | 60 |
| Ilustración 63: Interfaz Sistema de Combate .....                   | 62 |
| Ilustración 64: Interfaz interacción objetos .....                  | 63 |
| Ilustración 65: Blueprint Win Game de GameMode.....                 | 64 |
| Ilustración 66: Blueprint AI controller del Base Enemy .....        | 67 |
| Ilustración 67: Behavior Tree Sword Enemy .....                     | 69 |
| Ilustración 68: Blackboard Sword Enemy .....                        | 70 |
| Ilustración 69: Blueprint CheckIfPlayerInRange .....                | 71 |
| Ilustración 70: Blueprint Attack_Actor.....                         | 71 |
| Ilustración 71: Blueprint ClearBlackboardKey .....                  | 72 |
| Ilustración 72: Blueprint FindLocationToPatrol.....                 | 73 |
| Ilustración 73: Blueprint Event Graph de BP_Base_Enemy.....         | 73 |
| Ilustración 74: Blueprint Event Graph de BP_Sword_Enemy.....        | 74 |
| Ilustración 75: Creación NPC Convai .....                           | 75 |
| Ilustración 76: Blueprint Event Begin Play NPC Convai .....         | 77 |
| Ilustración 77: Blueprint Event Tick BP_Sword (Parte 1) .....       | 81 |
| Ilustración 78: Blueprint Event Tick BP_Sword (Parte 2) .....       | 81 |

|                                                                     |     |
|---------------------------------------------------------------------|-----|
| Ilustración 79: Blueprint Event Begin Play BP_Sword .....           | 81  |
| Ilustración 80: Blueprint Funciones BP_Sword .....                  | 81  |
| Ilustración 81: Blueprint BP_Projectile .....                       | 83  |
| Ilustración 82: Blueprint BP_Magic_Projectile.....                  | 84  |
| Ilustración 83: Blueprint BP_Spell.....                             | 86  |
| Ilustración 84: Texturas utilizadas para Efectos visuales .....     | 87  |
| Ilustración 85: Materiales creados para el proyecto .....           | 88  |
| Ilustración 86: Blueprint M_EnemyHitScreen .....                    | 89  |
| Ilustración 87: Niagara System creados para el proyecto .....       | 90  |
| Ilustración 88: NS_FireHit .....                                    | 91  |
| Ilustración 89: Blueprint VFX Trail .....                           | 92  |
| Ilustración 90: Blueprint Niagara Module Script.....                | 93  |
| Ilustración 91: Animation Blueprint PersonCharacter (Parte 1) ..... | 94  |
| Ilustración 92: Animation Blueprint PersonCharacter (Parte 2) ..... | 95  |
| Ilustración 93: Animation Montage MainCharacter_SwordAttack.....    | 95  |
| Ilustración 94: Level Sequence MainCharacterAnimation .....         | 96  |
| Ilustración 95:Animation Montage SwordEnemyAttack .....             | 97  |
| Ilustración 96: Blend Space Enemy.....                              | 98  |
| Ilustración 97: Animation Blueprint Sword Enemy.....                | 100 |
| Ilustración 98: Gráfico de cuánto juegan los usuarios .....         | 102 |
| Ilustración 99: Gráfico de plataforma de los usuarios .....         | 103 |
| Ilustración 100: Gráfico ataques usados por el usuario .....        | 104 |
| Ilustración 101: Gráfico vida suficiente .....                      | 104 |
| Ilustración 102: Gráfico interacción NPCs .....                     | 105 |
| Ilustración 103: Gráfico guardado usuarios.....                     | 105 |
| Ilustración 104: Gráfico finalización del juego .....               | 106 |

# Índice de tablas

|                                       |     |
|---------------------------------------|-----|
| Tabla 1: Conceptos habituales .....   | 3   |
| Tabla 2: Assets de uso libre .....    | 9   |
| Tabla 3: Assets de uso gratuito ..... | 10  |
| Tabla 4: Mecánicas de juego .....     | 116 |
| Tabla 5: Controles de juego .....     | 117 |

# 1.Introducción

## 1.1 Motivación

La ingeniería informática abarca una amplia variedad de disciplinas que pueden ser investigadas. Una de ellas es el desarrollo de videojuegos, una rama particularmente fascinante

Cuando consideramos la aplicación de los principios y procesos de la ingeniería informática. Esto incluye desde la creación de la documentación necesaria hasta la implementación de soluciones y la realización de pruebas para asegurar su funcionalidad.

Es importante también reconocer que el desarrollo de software no siempre tiene como objetivo la productividad o la utilidad práctica, sino también el entretenimiento y el enriquecimiento cultural. Los videojuegos no solo ofrecen diversión, sino que también pueden fomentar la creatividad, el pensamiento crítico y el trabajo en equipo, aspectos que contribuyen al desarrollo personal y social de quienes los juegan. Por ello, es fundamental darles el reconocimiento y la importancia que merecen dentro del campo de la ingeniería informática.

Otra razón significativa para elegir este tema es mi pasión por los videojuegos, que fue lo que inicialmente me motivó a comenzar a programar. Este Trabajo de Fin de Grado representa una oportunidad para explorar a fondo este campo y adquirir las habilidades y competencias necesarias para llevar a cabo el proyecto. Además, opté por esta área debido a mi curiosidad por la inteligencia artificial y su potencial aplicación en los videojuegos.

## 1.2 Objetivos

Los principales objetivos de este Trabajo de Fin de Grado son:

- El desarrollo de un videojuego de acción, siguiendo las características típicas de este género, las cuales serán descritas posteriormente.
- La utilización y puesta en práctica de todos los conocimientos adquiridos durante el grado en las distintas asignaturas. Por ejemplo, captura de

requisitos de la asignatura 'Ingeniería de requisitos' o el uso de sistemas inteligentes de la asignatura 'Sistemas Inteligentes I'

### 1.3 Estructura de la memoria

- Introducción: en este punto se explican las motivaciones y objetivos del trabajo, así como la estructura de este.
- Contexto: en este punto se tratará de situar al lector, tanto en lo referente al ámbito de los videojuegos como al de la ingeniería de software. Se explicará la metodología a seguir y también herramientas y recursos necesarios.
- Análisis, diseño y solución: en este punto se definirá el juego y se expondrán los requisitos necesarios para desarrollar el sistema. También se mostrará casos de uso, así como distintos diagramas sobre enemigos, bocetos de interfaz de usuario y mapas de navegación. Por último, se hablará de conceptos exclusivos del desarrollo de videojuegos, como el diseño de las mecánicas de juego, diseño de niveles e historia.
- Implementación: en este punto podrá verse como se ha llevado a cabo la implementación propuesta en el apartado uno, desglosada por apartados para su fácil comprensión.
- Pruebas: en este punto se explicará cómo se han realizados las pruebas al software desarrollado.
- Conclusiones y líneas futuras en este punto se expondrán las conclusiones tras el trabajo realizado.
- Anexo: en este punto se encontrará información que se ha considerado interesante incluir en el trabajo, pero no tenían cabida en la estructura general del documento.

## 2.Contexto

En este apartado estableceremos qué es un videojuego de acción, algunos conceptos que se usarán más adelante. También describiremos la metodología, las tecnologías y los recursos que se usarán en el proyecto.

### 2.1 Conceptos habituales

En el sector de los videojuegos existen muchos términos que se usan frecuentemente y no son de uso general. Por ello, se muestran los términos que se usarán durante el documento con su significado.

| Término          | Significado                                                                                                                                                                                                      |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Videojuego indie | Videojuego desarrollado por compañías pequeñas o incluso una sola persona                                                                                                                                        |
| <i>Blueprint</i> | Programa escrito en un lenguaje de programación compatible con Unreal Engine que se utiliza para controlar el comportamiento de los objetos y elementos del juego.                                               |
| <i>Assets</i>    | Un <i>asset</i> es una representación de cualquier ítem que puede ser utilizado en su juego o proyecto. Un <i>asset</i> podría ser un archivo de audio, una imagen, o cualquiera de los otros tipos de archivos. |

Tabla 1: Conceptos habituales

### 2.2. Subgénero first-person action en los videojuegos

El subgénero *first-person action* se caracteriza por situar al jugador en una perspectiva de primera persona, donde experimenta el juego desde los ojos del personaje controlado. Este enfoque proporciona una sensación inmersiva, haciendo que el jugador sienta que está directamente en el mundo del juego. A diferencia de otros subgéneros que se enfocan más en la exploración o la

narrativa, los juegos de acción en primera persona ponen un gran énfasis en el combate y la acción rápida.

Este subgénero se popularizó con la llegada de títulos icónicos como *Doom* (1993), que definió muchos de los principios básicos de los juegos de acción en primera persona, como el movimiento fluido, el combate intenso y la recolección de armas y recursos. A lo largo del tiempo, estos juegos han evolucionado para incluir una variedad de armas y habilidades, desde espadas y arcos hasta poderes sobrenaturales como rayos o hechizos. Este enfoque en la acción inmediata, combinado con la perspectiva inmersiva, crea una experiencia intensa para el jugador.

Un elemento distintivo de los juegos de *first-person action* es la mezcla de combate cuerpo a cuerpo y a distancia, donde la estrategia y la habilidad del jugador son esenciales para avanzar. Aunque algunos títulos pueden incluir elementos de exploración o puzzles, el núcleo del subgénero sigue siendo la acción constante y la lucha contra enemigos.

Este subgénero ha mantenido su relevancia gracias a la innovación en mecánicas de juego y gráficos, atrayendo tanto a veteranos como a nuevos jugadores. En los últimos años, hemos visto lanzamientos que combinan la acción en primera persona con otros géneros, enriqueciendo aún más la experiencia de juego. Analizaremos algunos de estos títulos más adelante.

## 2.3. Referentes

A continuación, se mencionarán los tres juegos que han servido de referencia para elaborar el proyecto.

### 2.3.1. The Elder Scrolls V: Skyrim

Este título fue lanzado por *Bethesda Game Studios* el 11 de noviembre de 2011 para múltiples plataformas, incluyendo PlayStation 3, Xbox 360 y PC. Más tarde, recibió varias ediciones remasterizadas y versiones para consolas más recientes, como la PlayStation 4, Xbox One y Nintendo Switch.

*Skyrim* es un juego de rol de acción en primera persona que se destaca por su vasto mundo abierto y la libertad que ofrece al jugador para explorar. Aunque se centra en la narrativa y la construcción del personaje, el combate es un aspecto crucial, donde el jugador puede usar una amplia variedad de armas, desde espadas hasta magia. Este juego ha servido de inspiración por su combinación de exploración y combate en primera persona, y por la sensación de inmersión que logra a través de su diseño de mundo abierto.

La idea que se quiere conservar en el proyecto es la libertad de acción y la capacidad de enfrentarse a los desafíos del juego utilizando diversas habilidades y armas. Además, el diseño del mundo en *Skyrim* permite al jugador descubrir nuevos lugares y secretos, lo que incentiva la exploración continua.



*Ilustración 1: Captura de pantalla de The Elder Scrolls V: Skyrim*

### 2.3.2. Dishonored

Este título fue lanzado por *Arkane Studios* y *Bethesda Softworks* el 9 de octubre de 2012 para PlayStation 3, Xbox 360 y PC. Posteriormente, fue adaptado para PlayStation 4 y Xbox One en su edición remasterizada. (Dishonored.com, s.f.)

*Dishonored* es un juego de acción en primera persona que combina elementos de sigilo, combate y poderes sobrenaturales. El jugador controla a Corvo Attano, un asesino con la capacidad de usar habilidades especiales, lo que le permite enfrentar situaciones de combate de manera creativa. La flexibilidad del juego para permitir diferentes enfoques en cada misión ya sea con sigilo o acción directa, ha sido una gran fuente de inspiración.

En el proyecto, se busca capturar esa sensación de poder y creatividad en el combate, permitiendo al jugador decidir cómo enfrentar a sus enemigos, ya sea utilizando ataques directos con la espada o empleando poderes especiales como rayos. Esta combinación de combate y estrategia es fundamental para la experiencia de juego que queremos ofrecer



*Ilustración 2: Captura de pantalla del videojuego Dishonored*

### 2.3.3. Call of Duty: Modern Warfare II

Este título fue lanzado por *Infinity Ward* y *Activision* el 28 de octubre de 2022 para PlayStation 4, PlayStation 5, Xbox One, Xbox Series X/S y PC. (CallofDuty.com, 2022)

*Call of Duty: Modern Warfare II* es un juego de disparos en primera persona que destaca por su combate intenso y su narrativa cinematográfica. El jugador se sumerge en una serie de misiones globales con un equipo de élite, utilizando una amplia variedad de armas y tácticas para superar los desafíos. El juego combina acción rápida con una jugabilidad estratégica, y ofrece un diseño de niveles detallado que va desde ciudades en ruinas hasta paisajes exóticos.

Se busca capturar la dinámica y el realismo del combate de *Modern Warfare II*, permitiendo al jugador experimentar una acción fluida y emocionante. La atención al detalle en el diseño de armas y entornos, junto con una narrativa envolvente, son aspectos clave que queremos integrar para ofrecer una experiencia inmersiva y espectacular en primera persona



*Ilustración 3: Captura de pantalla del videojuego COD Modern Warfare II*

## 2.4. Metodologías de trabajo

La metodología que se ha usado en este proyecto es una metodología Ágil basada en Scrum. Este tipo de enfoque para el desarrollo de software se basa en un proceso iterativo en el que se divide el trabajo en periodos cortos llamados *sprints*. El proceso comienza con una reunión para planificar el sprint, donde se establecen los objetivos y las tareas a realizar. Al final del sprint se realiza una revisión para identificar problemas y proponer mejoras. Este proceso se repite en ciclos sucesivos hasta que se completa el proyecto. (Scrum.org, s.f.) En puntos posteriores se explicará cómo se ha adaptado esta metodología al proyecto.

## 2.5. Tecnologías utilizadas

A continuación, se describirán las tecnologías que se han usado para llevar a cabo el proyecto.

### 2.5.1. Unreal Engine

*Unreal Engine* (Epic Games, s.f.) es un motor de videojuegos potente que permite el desarrollo de juegos en tres dimensiones. Ofrece un extenso conjunto de herramientas y funcionalidades para el diseño de entornos, la creación de personajes, la programación de la lógica del juego, la animación y el procesamiento de audio, entre otros aspectos.

Una de las principales ventajas de utilizar *Unreal Engine* es su capacidad para generar gráficos de alta calidad y efectos visuales avanzados, lo que lo hace ideal para desarrollos AAA y experiencias inmersivas. Además, *Unreal*

*Engine* permite publicar juegos en diversas plataformas, incluyendo PC, consolas, dispositivos móviles y realidad virtual. Otra ventaja significativa es que ofrece una versión gratuita con acceso a todas las herramientas necesarias para comenzar el desarrollo, y su modelo de licencia se basa en regalías, lo que puede ser más accesible para desarrolladores independientes.

Asimismo, cuenta con una comunidad extensa y activa, así como una rica colección de recursos en línea, como tutoriales, foros y documentación oficial (*Epic Games, s.f.*), que facilitan el aprendizaje y el intercambio de conocimientos entre desarrolladores.

### 2.5.2. Convai

*Convai* (*Convai, s.f.*) es una herramienta especializada en la generación de conversaciones inteligentes con personajes no jugables (*NPCs*). Esta plataforma permite desarrollar diálogos dinámicos y naturales mediante el uso de inteligencia artificial, facilitando la creación de interacciones realistas dentro de los videojuegos

Una de las principales ventajas de utilizar *Convai* es su capacidad para generar respuestas contextuales y adaptativas, lo que mejora significativamente la inmersión y la interacción del jugador con los *NPCs*. La herramienta ofrece un entorno intuitivo para diseñar y ajustar las conversaciones, así como un sistema de análisis que ayuda a refinar las interacciones basadas en el *feedback* y los datos recolectados durante el juego.

Además, *Convai* proporciona una integración sencilla con otros motores de juego y plataformas, lo que facilita su implementación en proyectos existentes. Su documentación y recursos en línea (*Convai, s.f.*) permiten a los desarrolladores acceder a guías, tutoriales y soporte técnico, contribuyendo a una curva de aprendizaje más fluida y a una rápida adopción de la herramienta en el proceso de desarrollo de videojuegos.

## 2.6. Arte del juego

Un videojuego no se limita solo al código; es el resultado de la colaboración entre diversas disciplinas. La creación de ilustraciones, música y efectos de sonido no

ha sido parte de este proyecto. Por lo tanto, los recursos utilizados en *Agora Island* han sido adquiridos de artistas que proporcionan material gratuito.

A continuación, se detallarán los recursos empleados y los artistas responsables de su creación.

### 2.6.1. Assets de uso libre

Se han utilizado varias librerías adicionales, las cuales están disponibles para su descarga a través del *marketplace* de *Epic Games*. A continuación, se presenta una tabla que detalla las características principales de cada una de estas librerías, incluyendo su nombre, precio y autor:

| Nombre                 | Precio   | Autor            |
|------------------------|----------|------------------|
| Control Rig UE5        | Gratuito | Epic Games       |
| Infinity Blade Weapons | Gratuito | Epic Games       |
| MegaScans              | Gratuito | Quixel Megascans |
| ParagonMinions         | Gratuito | Epic Games       |
| Pirate                 | Gratuito | Bugrimov Maksim  |

Tabla 2: Assets de uso libre

Se han utilizado diversos assets de uso libre y gratuito para complementar el diseño y desarrollo del proyecto, los cuales fueron extraídos de dos plataformas especializadas: *TurboSquid* y *Sketchfab*. A continuación, se presenta una tabla que detalla el nombre de cada asset, el autor que lo ha creado, y el precio (en caso de ser gratuito o tener algún costo), para ofrecer una visión clara de los recursos utilizados y su procedencia.

| Nombre del paquete                  | Precio   | Referencia                                                            | Autor    |
|-------------------------------------|----------|-----------------------------------------------------------------------|----------|
| Aleksandr-class-archipelago-frigate | Gratuito | <a href="https://www.turbosquid.com/">https://www.turbosquid.com/</a> | Zoan Oy  |
| Chest                               | Gratuito | <a href="https://www.turbosquid.com/">https://www.turbosquid.com/</a> | 3DMish   |
| Crate                               | Gratuito | <a href="https://www.turbosquid.com/">https://www.turbosquid.com/</a> | DJMaesen |
| Dock-pier                           | Gratuito | <a href="https://www.turbosquid.com/">https://www.turbosquid.com/</a> | Pixel3d  |

|                                    |          |                                                                       |              |
|------------------------------------|----------|-----------------------------------------------------------------------|--------------|
| Empty-ship                         | Gratuito | <a href="https://www.turbosquid.com/">https://www.turbosquid.com/</a> | Liaval       |
| Jungle-cabin                       | Gratuito | <a href="https://www.turbosquid.com/">https://www.turbosquid.com/</a> | Thunder      |
| Light-house                        | Gratuito | <a href="https://www.turbosquid.com/">https://www.turbosquid.com/</a> | CoolBean     |
| Palm                               | Gratuito | <a href="https://sketchfab.com/feed">https://sketchfab.com/feed</a>   | Farukjoy     |
| Pirate-sailing-ship                | Gratuito | <a href="https://www.turbosquid.com/">https://www.turbosquid.com/</a> | BConnolly    |
| Ship-pinnace                       | Gratuito | <a href="https://www.turbosquid.com/">https://www.turbosquid.com/</a> | Ship Pinnace |
| Ship-vv                            | Gratuito | <a href="https://www.turbosquid.com/">https://www.turbosquid.com/</a> | Gogiart      |
| Skeleton                           | Gratuito | <a href="https://www.turbosquid.com/">https://www.turbosquid.com/</a> | Soidev       |
| spherical-ship                     | Gratuito | <a href="https://www.turbosquid.com/">https://www.turbosquid.com/</a> | IrisProcess  |
| starship-troopers-fast-attack-ship | Gratuito | <a href="https://www.turbosquid.com/">https://www.turbosquid.com/</a> | 78887        |

Tabla 3: Assets de uso gratuito

### 2.6.2. Música

El sonido que se utilizó para el mapa de Agora Island proviene de un video de YouTube del canal Foto Video Aéreo, titulado SONIDOS DEL MAR Y GAVIOTAS HD. Este audio se seleccionó para aportar una ambientación realista y envolvente al juego, capturando los sonidos del mar y las gaviotas que complementan la atmósfera de la isla.

### 2.6.3. Imágenes

En el desarrollo visual del proyecto, se ha utilizado inteligencia artificial para generar diversas imágenes que complementan el diseño y la estética del juego. Estas imágenes fueron creadas utilizando la herramienta Leonardo.ai, una

plataforma de generación de arte mediante IA, que nos permitió obtener representaciones gráficas de alta calidad de manera eficiente.

La utilización de *Leonardo.ai* fue clave para la generación de las siguientes imágenes, permitiendo explorar rápidamente múltiples conceptos y estilos artísticos. Cada imagen fue ajustada para reflejar los escenarios, personajes y elementos visuales del juego Agora Island.

- Interfaces: Las imágenes del fondo de cada uno de los menús del proyecto han sido generadas mediante inteligencia artificial.
- Botones y logos: Los logotipos y botones que aparecen en el videojuego han sido creados mediante inteligencia artificial.

El uso de inteligencia artificial ha permitido una optimización significativa del flujo de trabajo en la creación de imágenes, reduciendo tiempos de producción y proporcionando resultados innovadores que complementan la visión artística del juego. *Leonardo.ai* demostró ser una herramienta flexible para generar imágenes acordes a los requisitos del proyecto, manteniendo la consistencia con la temática y el estilo deseado.

Cada imagen fue generada a partir de descripciones textuales, conocidas como *prompts*, que fueron introducidas en la IA para guiar el proceso de creación. A continuación, se presentan las imágenes generadas junto con los *prompts* utilizados.



*Ilustración 4: Pantalla de carga generada mediante IA generativa*

**Prompt:** A dramatic, cinematic photograph set against the backdrop of a lush, tropical island, where a majestic mountain rises from

the dense foliage, its peak shrouded in a whispers of mist. In the distance, a menacing pirate ship with tattered sails and a black hull cuts through the turquoise waters, leaving a trail of frothy white wake behind it. The sky above is a deep shade of cerulean during the golden hour, with the sun's warm rays casting a warm, honeytoned light on the entire scene. The atmospheric haze adds a sense of depth, and the waves gently lap against the shore, creating a soothing melody. The composition is balanced, with the mountain and pirate ship placed diagonally, drawing the viewer's eye to the horizon. The photograph is infused with a sense of adventure, mystery, and a touch of danger, inviting the viewer to step into this exotic, high-seas world.



*Ilustración 5:: Pantalla del menu principal generada mediante IA generativa*

**Prompt:** A cinematic photograph featuring a serene island environment with a majestic mountain range rising in the background, surrounded by lush green foliage and a few wispy clouds scattered across a bright blue sky. In the distance, a pirate ship sails across the calm turquoise waters, its sails billowing in the gentle breeze. On the sandy beach, a lone pirate figure stands, minuscule in the vast landscape, his weathered skin and tattered clothing blending into the surroundings, making him almost imperceptible. The pirate's facial features are obscured, adding to the mystique, with a hint of his tricorne hat visible above the sandy dunes. The warm golden light of the setting sun

casts a gentle glow on the entire scene, with the mountain's rugged terrain and the pirate ship's intricate details rendered in exquisite depth and texture, inviting the viewer to explore the hidden treasures of this idyllic island sanctuary.



*Ilustración 6: Título de mi juego generado mediante IA generativa*

**Prompt:** a dramatic, high-contrast cinematic photograph of the video game title .Agora Island suspended in mid-air, surrounded by a subtle mist, evoking the mystical aura of a tropical isle, with a distressed, vintage pirate map-inspired aesthetic, weathered and torn at the edges, displayed on a worn, wooden sign, adorned with rusty old-fashioned lanterns and bits of frayed rope, set against a soft, gradient blue sky with hints of warm golden light, as if captured at dawn or dusk, with bold, cursive, gold-embossed lettering and subtle, ornate illustrations of palm trees, anchors, and other nautical elements, exuding a sense of adventure and mystery.



*Ilustración 7: Logo Save Game generado mediante IA generativa*

**Prompt:** a dramatic, high-contrast cinematic photograph of the video game title "Save Game"suspended in mid-air, surrounded by a subtle mist, evoking the mystical aura of a tropical isle, with a distressed, vintage pirate map-inspired aesthetic, weathered and torn at the edges, displayed on a worn, wooden sign, adorned with rusty old-fashioned lanterns and bits of frayed rope, set against a soft, gradient blue sky with hints of warm golden light, as if captured at dawn or dusk, with bold, cursive, gold-embossed lettering and subtle, ornate illustrations of palm trees, anchors, and other nautical elements, exuding a sense of adventure and mystery.



*Ilustración 8: Logo Load Game generado mediante IA generativa*

**Prompt:** a dramatic, high-contrast cinematic photograph of the video game title "Load Game"suspended in mid-air, surrounded by a subtle mist, evoking the mystical aura of a tropical isle, with a distressed, vintage pirate map-inspired aesthetic, weathered and torn at the edges, displayed on a worn, wooden sign, adorned with rusty old-fashioned lanterns and bits of frayed rope, set against a soft, gradient blue sky with hints of warm golden light, as if captured at dawn or dusk, with bold, cursive, gold-embossed lettering and subtle, ornate illustrations of palm trees, anchors, and other nautical elements, exuding a sense of adventure and mystery.



*Ilustración 9: You win generado mediante IA generativa*

**Prompt:** A cinematic, high-contrast photograph with a shallow depth of field, featuring a worn, vintage-inspired wooden sign with the bold, golden title "You Win" emblazoned across it, centered against a rustic, misty island backdrop with swaying palm trees and a subtle, sunkissed horizon. The sign's weathered wood grain and ornate metal fixtures evoke a sense of adventure and aged treasure. In the foreground, a scattering of sandy footprints, a few scattered seashells, and a coiled rope hint at a pirate's lair, with the island's lush foliage and twisted driftwood limbs curling around the edges of the frame. The warm, golden light of the setting sun casts a mystical glow, evoking a sense of triumph and excitement. The overall atmosphere is one of swashbuckling adventure, mystery, and celebration, with the title "You Win" taking center stage as the ultimate treasure.



*Ilustración 10: Logo You Died generado mediante IA generativa*

**Prompt:** A dramatic, high-contrast, cinematic photograph with a distressed woodgrain texture, displaying a worn, weathered title plate with the haunting phrase "You Died.etched in bold, golden serif font, set against a misty, moonlit island backdrop, complete with swaying palm trees, jagged rocks, and a subtle, eerie glow, as if lit by a lantern, evoking a sense of mystery and foreboding, with hints of pirate folklore and adventure, the woodgrain surface bearing scratches, scuffs, and subtle cracks, further emphasizing the sense of age and decay, the overall atmosphere dark, moody, and introspective, inviting the viewer to step into the eerie, mystical world of the island, where danger lurks beneath the surface.

## 3. Análisis, diseño y solución

Una vez establecido el contexto de trabajo, se procede a explicar el proceso que se ha seguido para diseñar y especificar el juego.

### 3.1. Primer concepto

Al inicio, la idea del juego no estaba completamente definida, más allá de saber que se desarrollaría en una isla con temática de piratas. Esta falta de definición representaba un desafío, ya que, como se ha mencionado anteriormente, juegos con una ambientación similar pueden tomar enfoques muy diferentes. No tener claro el alcance del proyecto desde el principio puede ser riesgoso. Aunque es cierto que el primer concepto es la etapa más temprana del desarrollo, es fundamental delimitar correctamente lo que se quiere lograr para poder avanzar. Es necesario contar con una idea clara para poder establecer los requisitos del sistema de manera adecuada. Por lo tanto, la solución fue comenzar el proyecto con una lluvia de ideas y, a partir de ahí, seleccionar aquellas que mejor se ajustaban al proyecto.

Los primeros bocetos del juego revelaban que se trataría de una aventura en la que el jugador ayudaría a un pirata a recuperar su cofre, el cual había sido robado en una isla. Se planteó una jugabilidad básica en la que el personaje controlado por el jugador pudiera moverse, atacar y explorar la isla en busca del cofre.

Por otro lado, se decidió que se incluirían distintos tipos de enemigos que estarían dispersos por la isla y te atacarían al entrar en un rango específico. Los enemigos deben mostrar un comportamiento inteligente, como perseguir al jugador o tomar decisiones basadas en la posición del jugador en relación con ellos.

Finalmente, se destacó la necesidad de implementar un sistema que permita guardar el progreso del jugador, ya que es importante que el usuario pueda disfrutar de la experiencia a su propio ritmo. Si en algún momento se siente cansado y necesita tomar un descanso, debería poder continuar desde el punto en el que dejó el juego.

Una vez definidos los puntos más importantes, se puede comenzar con el desarrollo propiamente dicho. Primero se explicará cómo se adaptó la metodología de trabajo al proyecto y, a continuación, se especificarán los requisitos.

## 3.2. Aplicación metodología SCRUM

Como se mencionó anteriormente, se optó por una metodología Ágil basada en Scrum para llevar a cabo el proyecto. Una vez tomada esta decisión, el siguiente paso fue determinar cómo implementarla en el desarrollo. Se comenzó estableciendo la duración de los *sprints*, los cuales tendrían una duración de dos semanas, comenzando los lunes y finalizando los domingos de la semana siguiente. Al inicio de cada sprint, se destinaba tiempo a la planificación.

Las actividades clave para organizar cada *sprint* incluían:

- Asignar las tareas a realizar durante esas dos semanas.
- Evaluar el sprint anterior: si había tareas pendientes, se reasignaban al nuevo sprint, analizando las razones por las cuales no se completaron en el sprint correspondiente. Por ejemplo, si una tarea requería más tiempo del previsto, se ajustaba en la nueva planificación.
- Verificar que las tareas del sprint anterior se hubieran completado correctamente.
- Investigar nuevas herramientas y *assets* que pudieran ser útiles para mejorar o ampliar las funcionalidades del proyecto, evaluando su viabilidad e integración.

Para facilitar la planificación de los *sprints*, se utilizó la herramienta en línea Jira. Esta herramienta cuenta con plantillas que ayudan a organizar el trabajo, además de permitir mantener un seguimiento detallado de la gestión de tareas y sus registros.

En total, se llevaron a cabo seis *sprints*. El número de tareas realizadas en cada sprint varió, ya que no todas las tareas requerían el mismo tiempo.

El sprint más relevante fue el primero, ya que durante este se llevaron a cabo actividades fundamentales para el éxito del proyecto. En este sprint, se realizó una captura exhaustiva de los requisitos, que sirvieron como base para crear las

tareas que se abordarían en los *sprints* siguientes. Este proceso fue crucial, ya que permitió establecer una visión clara de lo que el proyecto necesitaba y cómo se organizaría el trabajo a lo largo del desarrollo.

Además de la captura de requisitos, en este sprint también se dedicó tiempo a investigar nuevas herramientas y *assets* que podrían ser útiles para el proyecto. Esta investigación fue esencial para identificar recursos que mejorarán la calidad y eficiencia del desarrollo. La información obtenida permitió tomar decisiones informadas sobre qué herramientas y *assets* integrar, asegurando que el proyecto contara con las mejores opciones disponibles.

Este sprint marcó el rumbo del proyecto, definiendo las bases sobre las cuales se construirían las funcionalidades principales y estableciendo un enfoque estratégico para los siguientes *sprints*. La planificación cuidadosa y la investigación realizada en esta etapa sentaron las bases para un desarrollo más organizado y efectivo en los *sprints* posteriores.

### 3.3 Captura de requisitos

El punto de partida de cualquier proyecto de software debe ser una visión clara y bien definida de lo que se pretende lograr con el nuevo sistema. Esta visión debe incluir tanto los objetivos generales del sistema como las limitaciones y condiciones que puedan afectar su desarrollo. Los requisitos son esenciales para describir de manera precisa las funcionalidades y características que el sistema debe incorporar. Un requisito se puede entender como una necesidad o funcionalidad que el sistema debe ofrecer para resolver un problema o satisfacer una demanda del usuario final (IEEE, 1990).

Es fundamental que estos requisitos sean identificados y comprendidos al inicio del proyecto para evitar malentendidos y asegurar que el desarrollo siga el rumbo correcto. Este proceso de recopilación de requisitos no solo define qué es lo que el sistema debe hacer, sino que también ayuda a identificar posibles desafíos y establecer prioridades en el desarrollo. A continuación, se desglosarán los requisitos del proyecto, organizados en diferentes categorías para una mejor comprensión y manejo.

### 3.3.1 Requisitos funcionales

Un requisito funcional es aquel que especifica que un sistema o componente debe ser capaz de realizar. (IEEE, 1990) De esta forma, se capturan los siguientes requisitos funcionales para Agora Island:

#### **Personaje jugable:**

- El personaje podrá saltar.
- El personaje podrá moverse tanto de frente como horizontalmente.
- El personaje podrá lanzar un rayo al frente.
- El personaje podrá realizar daño a los enemigos con su arma.
- El personaje contará con un sistema de vida.
- El personaje podrá recibir daño y morir.
- El personaje podrá recoger el objeto final y finalizar el juego.
- El personaje contará con un mini mapa donde podrá observar el mapa desde una vista aérea.

#### **Partida**

- La partida podrá ser guardada y cargada en cualquier momento y zona del mapa.

Los datos que se guardan son:

- Salud del jugador
- salud de los enemigos
- la localización del jugador en el mapa.

#### **Historia**

La historia se contará a través de varios *NPCs* que encuentras en el mapa.

#### **Mapa**

- El mapa consistirá en una única isla.
- El mapa contendrá enemigos repartidos por la isla.
- El mapa contará con puntos de referencia para saber hacia dónde dirigirte para completar el juego.

- El mapa está diseñado para incitar al jugador a explorar gracias a los assets incluidos.

## **Enemigos**

- Se encuentran repartidos por toda la isla.
- Una vez que un enemigo ha sido derrotado no volverá a aparecer hasta que comience una nueva partida o mueras y/o reinicies.
- Todos los enemigos contarán con un atributo de vida máxima.
- Los enemigos contarán con dos tipos de ataques con una espada.
- Los enemigos contarán con un atributo de daño fijo.

## **Menú**

- El menú principal del juego contará con tres opciones: Nueva Partida, Cargar Partida y Salir del juego.
- La opción Nueva Partida comenzará una nueva sesión del juego.
- La opción Cargar Partida nos dejará elegir una de las partidas que tenemos ya guardadas previamente.
- La opción Guardar Partida nos permitirá guardar localización del personaje, vida actual y toda la información de los enemigos en uno de los espacios habilitados para guardar partida.

## **NPC**

- Los *NPCs* estarán repartidos alrededor del mapa.
- Tienen un sistema de diálogo mediante chat a través de una interfaz en la pantalla del usuario.
- Tienen un sistema de diálogo mediante micrófono en el cual puedes interactuar con ellos.
- Los *NPCs* han sido entrenados para tener una historia propia y características propias de ellos.

### **3.3.2 Requisitos no funcionales**

Por otra parte, los requisitos no funcionales describen las características que deben cumplir los requisitos funcionales para asegurar la calidad del sistema. Estos incluyen aspectos como precisión, rendimiento, seguridad, y

mantenimiento, entre otros. Para Agora Island, se han definido los siguientes requisitos no funcionales:

El software será desarrollado utilizando el motor de videojuegos *Unreal Engine 5.3* utilizando su herramienta conocida como *blueprints*.

La interfaz del usuario debe ser simple e intuitiva para facilitar su uso por parte del usuario.

- El juego debe ejecutarse en un ordenador.
- El juego no debe experimentar bloqueos ni comportamientos inestables bajo condiciones normales de uso.
- Los tiempos de carga no deben ser superiores a los 15 segundos.

### 3.4 Diagrama de casos de uso

Un diagrama de casos de uso es una representación gráfica que muestra la interacción entre actores, casos de uso y las relaciones entre estos elementos. Cada caso de uso describe un requisito funcional del sistema. A continuación, se presentan varios diagramas de casos de uso en diferentes escenarios:

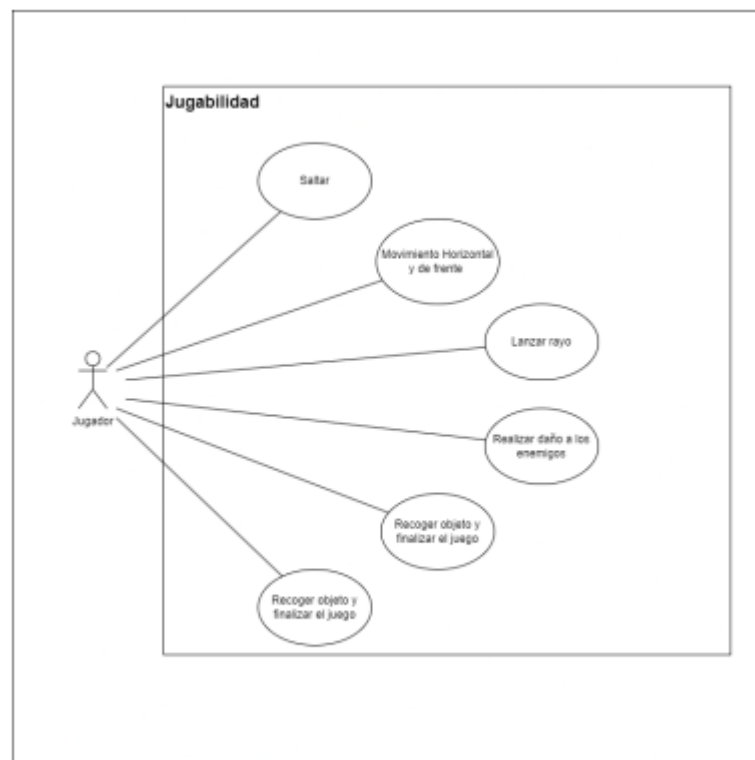


Ilustración 11: Diagrama caso de uso jugabilidad

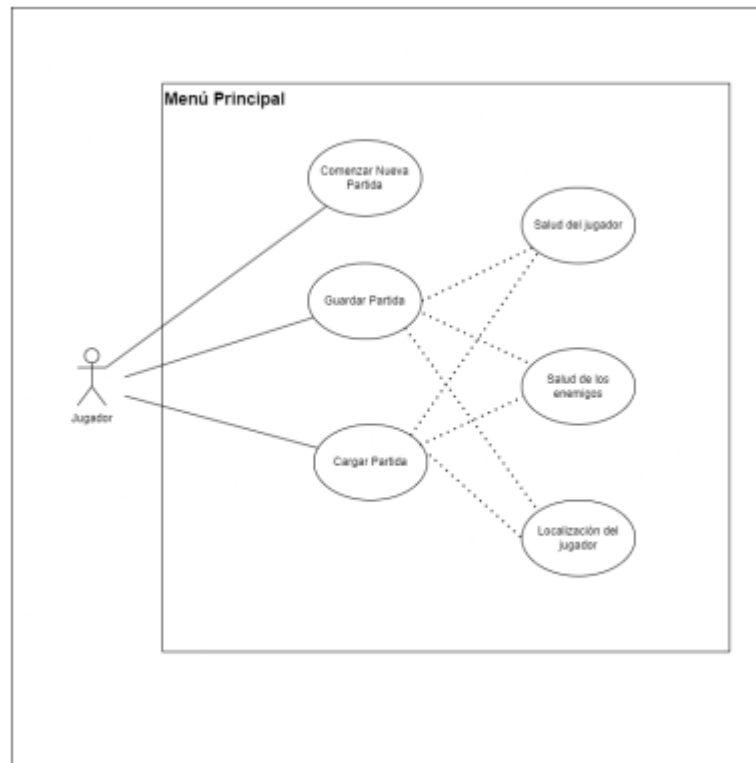


Ilustración 12: Diagrama caso de uso menú principal

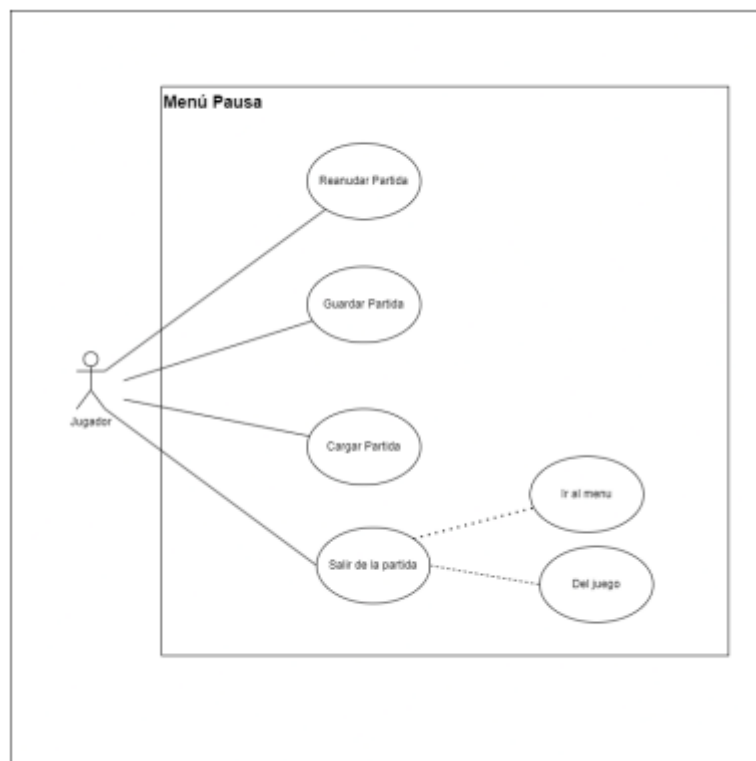


Ilustración 13: Diagrama caso de uso menú pausa

## 3.5 Diagrama de flujo del comportamiento de mi enemigo

En el presente proyecto se ha definido un sistema básico de inteligencia artificial para los enemigos basado en un conjunto de reglas de comportamiento predefinidas. Para ilustrar el funcionamiento de este sistema, se ha desarrollado un diagrama de flujo que detalla las acciones del enemigo en diferentes situaciones en función de la proximidad del jugador.

### 3.5.1 *Sword Enemy*

El comportamiento del enemigo sigue el siguiente patrón:

- **Patrullaje:** El enemigo comienza en un estado de patrullaje, en el cual se mueve por un área predefinida de manera aleatoria o en un patrón específico. Durante este estado, el enemigo se encuentra inactivo en cuanto a acciones ofensivas y está pendiente de detectar al jugador.
- **Rango de detección:** Cuando el jugador entra dentro del rango de detección del enemigo, este cambia de estado, pasando de patrullar a perseguir al jugador. El rango de detección se define como un área circular alrededor del enemigo dentro de la cual el jugador puede ser detectado visualmente.
- **Persecución:** Una vez que el jugador es detectado, el enemigo inicia una persecución en la que intenta acortar la distancia que lo separa del jugador. Durante este estado, el enemigo sigue al jugador hasta que este salga de su rango de detección.
- **Rango de ataque:** Si el jugador, mientras es perseguido, entra dentro del rango de ataque del enemigo, este comenzará a realizar ataques físicos o a distancia, dependiendo del tipo de enemigo. Los ataques continuarán mientras el jugador se mantenga dentro del rango de ataque.
- **Reversión a patrullaje:** Si el jugador consigue salir del rango de detección del enemigo (por ejemplo, al correr lejos o esconderse), el enemigo detiene la persecución y regresa a su estado original de patrullaje.

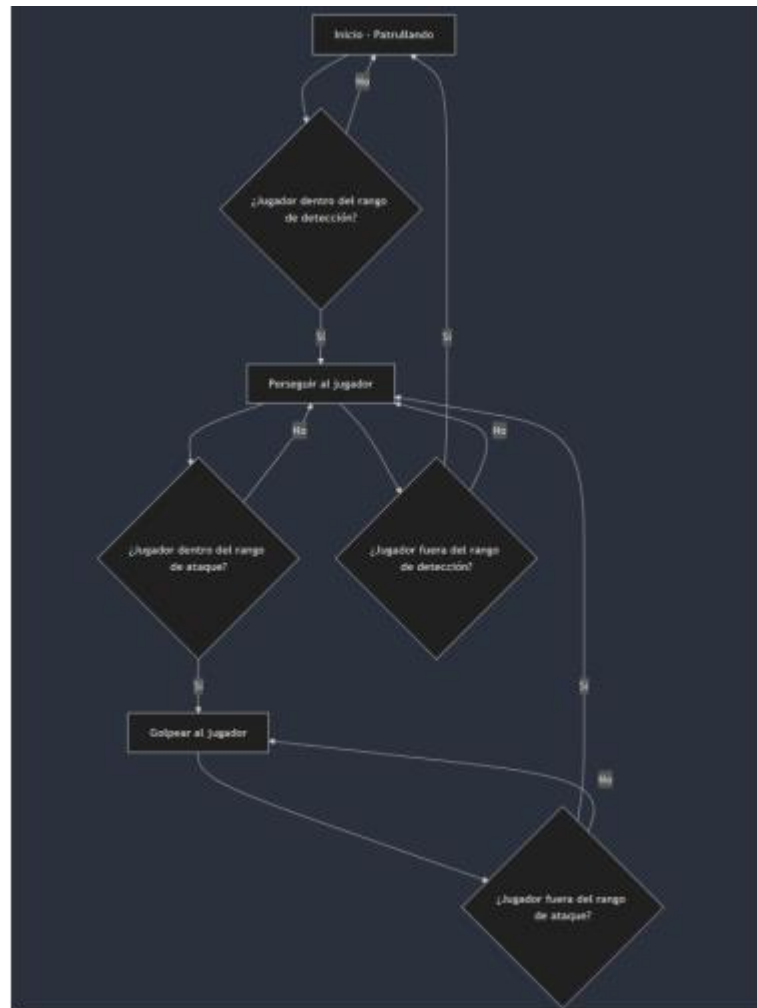


Ilustración 14: Diagrama de flujo Sword Enemy

Este diagrama de flujo es fundamental para simular el comportamiento dinámico de los enemigos en el juego, brindando una sensación de interacción reactiva y realista entre el jugador y el entorno. El ciclo de patrullaje, detección, persecución y ataque añade una capa estratégica para el jugador, ya que este deberá gestionar sus acciones en función del comportamiento predecible del enemigo.

### 3.6 Bocetos de interfaces de usuario

Los bocetos de interfaces son una herramienta crucial en el proceso de diseño de software, ya que proporcionan una representación esquemática de la apariencia y la funcionalidad esperada de la interfaz. En el contexto de este proyecto, se han desarrollado diversos bocetos de interfaces que ilustran diferentes pantallas del juego. Estos bocetos, creados utilizando la herramienta

Canvas (Canvas, s.f.), servirán como referencia durante la fase de implementación de los menús del juego.

Canvas es una herramienta en línea gratuita que facilita la creación de *wireframes* y prototipos interactivos, permitiendo una visualización clara de la interacción del usuario y la navegación. Los bocetos proporcionan una primera evaluación de la usabilidad y la experiencia de navegación, facilitando la identificación de problemas potenciales y permitiendo ajustes proactivos antes de la implementación definitiva. De este modo, se garantiza un diseño de interfaz que sea intuitivo y accesible para los usuarios finales. Los bocetos elaborados incluyen las siguientes pantallas:

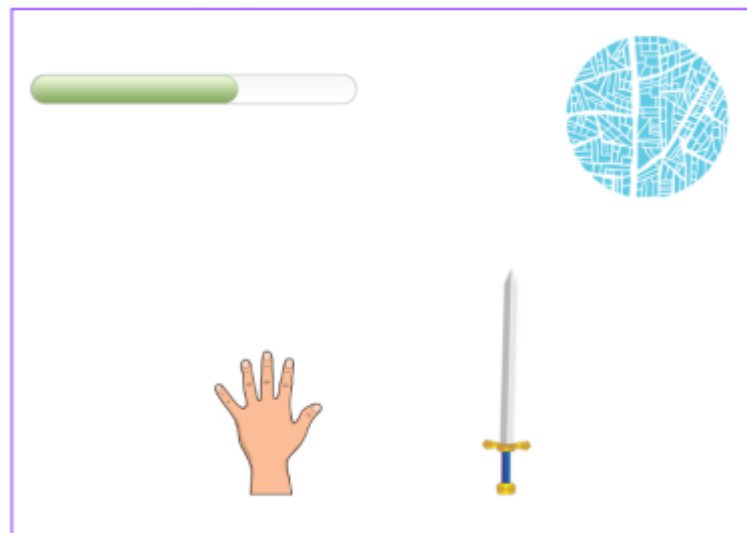


*Ilustración 15: Boceto Vista usuario*



*Ilustración 16: Boceto victoria*



*Ilustración 17: Boceto derrota*



*Ilustración 18: Boceto menú principal*

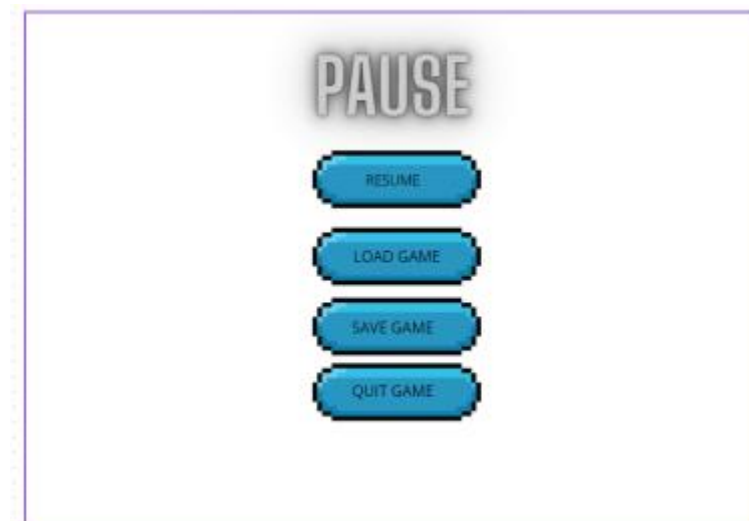


*Ilustración 19: Boceto menú de pausa*

### 3.7. Mapas de Navegación

Para completar los bocetos de interfaces, también se han desarrollado mapas de navegación de los menús del juego. Un mapa de navegación de menús es una herramienta que facilita la visualización de la estructura y las relaciones entre los distintos elementos de un sistema de menús o de navegación de una interfaz. Esta representación visual muestra cómo se organizan los elementos, como botones, pestañas, secciones y páginas, y cómo se interrelacionan entre sí.

Para Agora Island, se han creado los siguientes mapas de navegación:

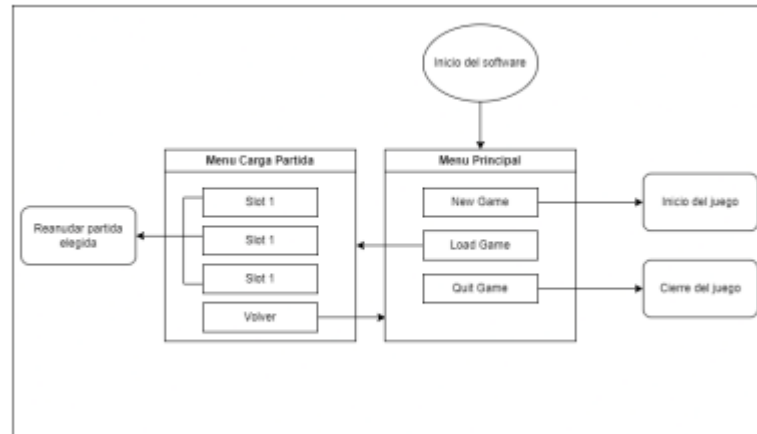


Ilustración 20: Mapa de navegación inicio

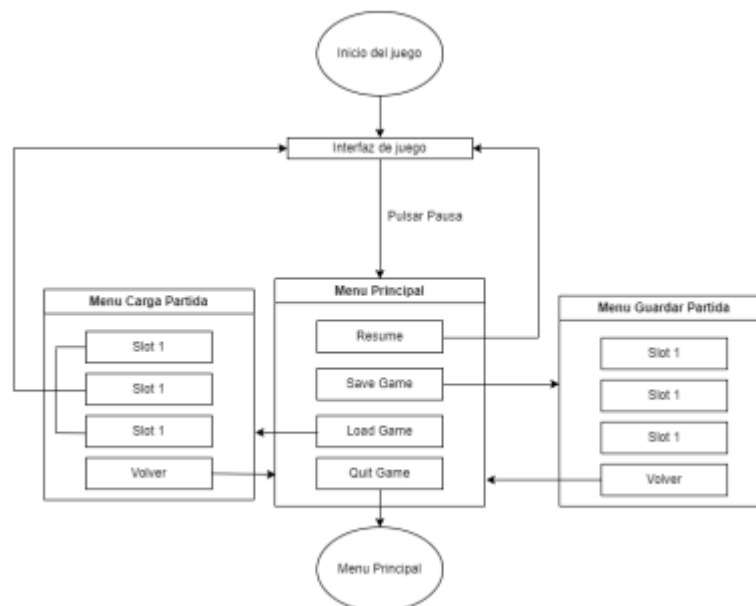


Ilustración 21: Mapa de navegación juego

### 3.8. Diseño de mecánicas de juego

Las mecánicas de juego se refieren a las reglas que establecen lo que el jugador puede o no puede hacer, y cómo estas acciones influyen en su interacción con el entorno del juego. Tal como se mencionó en los requisitos funcionales, una de las principales mecánicas definidas es la capacidad del personaje jugable para disparar. Del mismo modo, el resto de las acciones disponibles para el jugador deben considerarse como parte del conjunto de mecánicas de juego.

Sin embargo, el diseño de mecánicas no se limita simplemente a enumerar acciones o habilidades del personaje. Es fundamental asegurarse de

que estas mecánicas estén bien equilibradas entre sí y que mantengan coherencia con el entorno en el que se desarrollan. Un desequilibrio en las mecánicas puede afectar la experiencia del jugador, volviendo el juego demasiado difícil o muy fácil, o incluso frustrante si las mecánicas no son claras o no se integran bien con el diseño general.

Para garantizar que las habilidades y mecánicas del juego estén adecuadamente balanceadas, se propone lo siguiente:

- Limitar la cadencia de disparo a un disparo cada 2 segundos.
- Impedir que el personaje pueda lanzar un rayo al mismo tiempo que golpea con la espada.
- Cada golpe de espada debe completar su animación antes de que pueda iniciarse un nuevo golpe.
- Los ataques del jugador no afectan a los *NPCs* con los que puedes dialogar.

Estas reglas contribuyen a un equilibrio en la jugabilidad, logrando que el juego no sea ni demasiado fácil ni excesivamente difícil. Al limitar la cadencia de disparo y evitar la combinación simultánea de ataques a distancia y cuerpo a cuerpo, se fomenta la toma de decisiones estratégicas por parte del jugador. Además, al exigir que la animación del golpe de espada se complete antes de iniciar otro, se añade un elemento de tiempo y precisión en el combate, lo que aumenta el desafío sin hacerlo frustrante. En conjunto, estas restricciones promueven una jugabilidad más táctica y equilibrada, donde el jugador debe gestionar sus habilidades de manera eficiente para superar los retos del entorno.

### 3.9. Diseño de niveles

En los juegos de acción en primera persona, el diseño de niveles está intrínsecamente ligado a las mecánicas de juego, tal y como se ha comentado en el apartado anterior. Por ello, se dedicará una sección específica para explicar cómo se plantea el diseño del mapa en *Agora Island*.

El diseño del mapa para *Agora Island* presenta una estructura bien definida que integra tanto elementos visuales como jugables. El mapa se desarrolla en una isla, la cual incluye diversos entornos clave como una playa,

casas, y barcos en el mar. Los robots enemigos estarán distribuidos estratégicamente por toda la isla, añadiendo tensión y desafío al jugador.

Una parte importante del mapa es el pirata, quien se convierte en un personaje central al solicitar la ayuda del jugador. Además, en las casas habrá un *NPC* que también pedirá colaboración, introduciendo más interacción y narrativa en el mundo. En el cielo, el jugador podrá observar una nave, mientras que, en el extremo de la isla, cuya forma se asemeja a una luna, se encuentra el tesoro que el pirata busca, acompañado de otra nave espacial.

El entorno dinámico del mapa incluye elementos visuales como las olas del mar en movimiento, vegetación y rocas, lo que aporta vida y detalle a la ambientación. Este diseño invita al jugador a explorar y visitar zonas de la isla, mientras completa misiones y enfrenta a los enemigos.

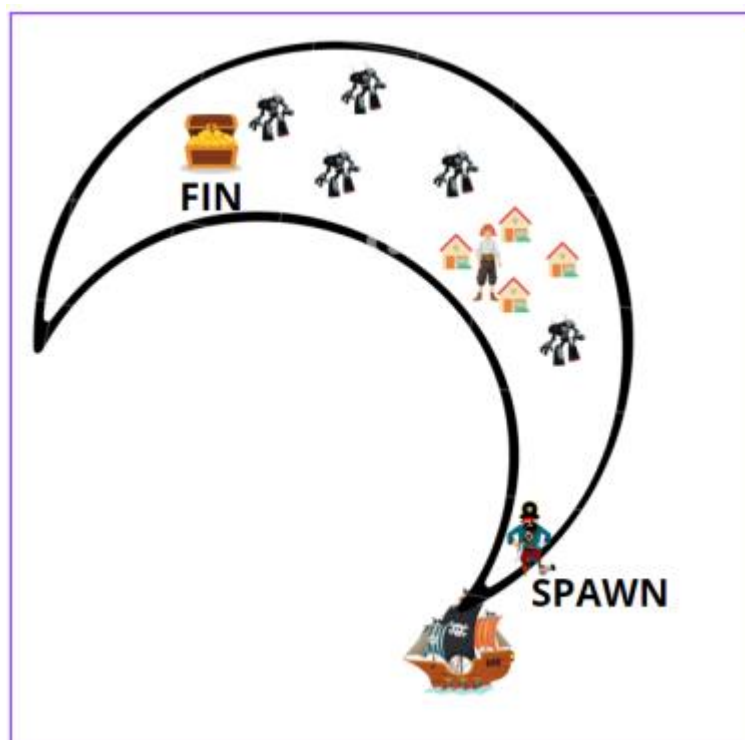


*Ilustración 22: Diseño nivel*

### 3.10. Historia del juego

Para cerrar este apartado, es crucial mencionar la historia de Agora Island, ya que es el elemento que conecta y da sentido a todo lo expuesto previamente. El jugador toma el control de un personaje que debe enfrentarse a enemigos

robóticos, explorar un entorno vibrante, y cumplir con misiones clave. Sin embargo, la historia no se trata solo de completar objetivos y derrotar enemigos; se trata de descubrir el propósito detrás de todo lo que sucede en la isla.

El jugador se encontrará con un pirata que le pedirá ayuda para encontrar un tesoro oculto en la isla, que tiene forma de luna. A lo largo de la historia, también interactuará con un *NPC* en las casas que solicitará su apoyo, añadiendo profundidad a la narrativa y ofreciendo más capas de desafíos y recompensas.

## 4.Implementación

El diseño del juego ya ha sido completamente definido y acotado, lo que permite avanzar a la fase de implementación. En este capítulo se describe cómo se han implementado los distintos requisitos del sistema. Para evitar sobrecargar al lector con demasiados detalles técnicos, solo se incluirán los fragmentos de código que resulten más relevantes. No obstante, se detalla el proceso de trabajo con *Unreal Engine*, y se explicarán algunos scripts y funciones clave en lenguaje natural, para facilitar la comprensión del desarrollo y las decisiones tomadas durante la implementación

### 4.1 Menú e interfaces

En *Unreal Engine*, la creación de menús e interfaces de usuario se realiza mediante el sistema UMG (*Unreal Motion Graphics*), que ofrece un conjunto de herramientas avanzadas para diseñar y gestionar interfaces interactivas. El diseño se basa en la construcción de widgets, que permiten organizar y personalizar los elementos de la interfaz de manera modular. A continuación, se explica cómo se han implementado las distintas interfaces utilizando este sistema.

#### 4.1.1 Menú Principal

El menú tiene un diseño visual acorde al estilo del juego, con botones que parecen de madera y un fondo que muestra una playa tropical, un barco pirata y un personaje observando el mar. Esto ayuda a que el jugador se sumerja en el ambiente del juego desde el primer momento. El menú se maneja fácilmente con teclado, ratón o controlador, y está implementado utilizando los *Widget Blueprints* de *Unreal Engine*.



Ilustración 23: Interfaz menú principal

A continuación, se describirán los distintos botones y como se han implementado:

- **New Game Button:** Este bloque maneja el evento cuando el jugador selecciona "Nuevo Juego" en el menú principal.
  - Evento "On Clicked (NewGameButton)": Se activa cuando el botón "New Game" es presionado.
  - "Create WBP Loading Screen Widget": Al presionar el botón, se crea un *widget* de pantalla de carga (WBP Loading Screen Widget), que se añade al *viewport* con el nodo "Add to Viewport". Esto permite mostrar una pantalla de carga mientras el nivel se está preparando.
  - "Get Player Pawn" y "Get Controller": Se obtiene el *Pawn* del jugador y su controlador, necesario para interactuar con el juego y controlar la cámara o las entradas del jugador.
  - "Delay": Introduce una pausa de 2 segundos (0.2 en el valor de duración) antes de cargar el nivel. Esta pausa puede estar diseñada para permitir la visualización de la pantalla de carga o para gestionar cualquier inicialización adicional.
  - "Open Level": Una vez finalizado el *delay*, se utiliza este nodo para abrir el nivel deseado (*IslandMap*). Esto lleva al jugador al mapa del juego, cargando el entorno inicial del juego.

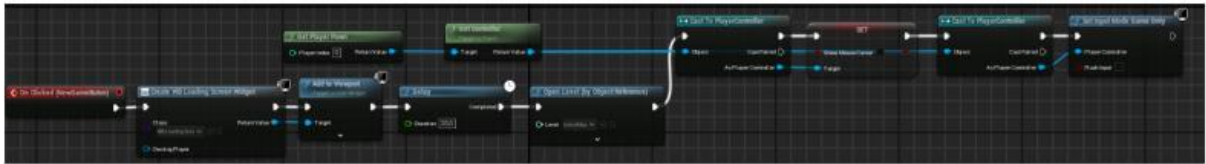


Ilustración 24: Blueprint new game button

- **Quit Button:** Este bloque maneja el evento cuando el jugador selecciona el botón "Salir" en el menú principal.
  - Evento "on clicked (QuitButton)": Se activa cuando el jugador presiona el botón de salida.
  - "Quit Game": Al ser activado, cierra el juego. El nodo tiene la opción de "Ignore Platform Restrictions" desactivada, lo que indica que se desea respetar las restricciones de las plataformas sobre cómo salir del juego.

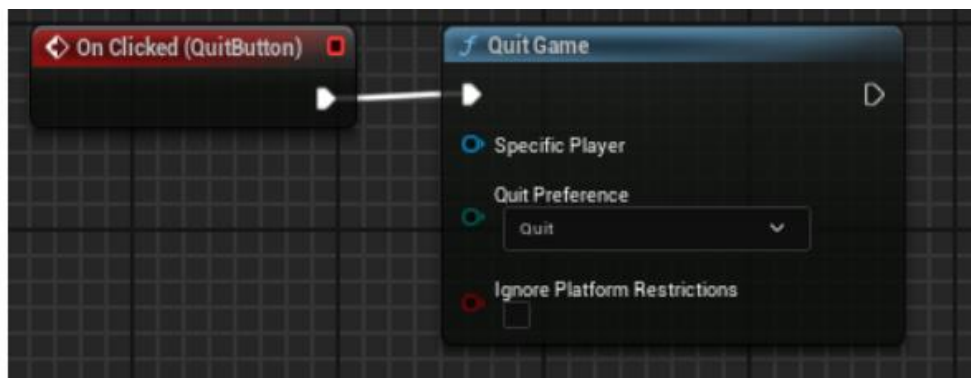


Ilustración 25: Blueprint quit button

- **Load Button:** Este bloque maneja la funcionalidad para cargar un menú de selección de partidas guardadas.
  - Evento "on clicked (LoadButton)": Se activa cuando el jugador presiona el botón de cargar partida.
  - "Remove from Parent": Elimina el menú actual de la vista, de forma que no interfiera con el nuevo menú que se va a cargar.
  - "Create WBP Load Menu Widget": Crea un *widget* correspondiente al menú de carga de partidas guardadas (*WBP Load Menu Widget*).

- "Add to Viewport": Se añade el *widget* de carga de partidas al *viewport*, permitiendo que el jugador vea y seleccione las partidas guardadas.



Ilustración 26: Blueprint Load Button

- **New Game:** En el caso del botón de "New Game", una vez que el nivel es cargado, se realizan ajustes adicionales para el modo de entrada del jugador y la visibilidad del cursor del ratón.
  - "Get Controller": Obtiene el controlador del jugador para cambiar el modo de interacción.
  - "Cast to PlayerController": Se realiza un "cast" para confirmar que el controlador es del tipo `PlayerController`, que es necesario para modificar la visibilidad del cursor y el modo de entrada.
  - "Set Show Mouse Cursor": Este nodo se utiliza para ocultar el cursor cuando se está en el juego, estableciendo la propiedad `Show Mouse Cursor` a `false`.
  - "Set Input Mode Game Only": Cambia el modo de entrada del jugador para que el control esté enfocado solo en el juego, sin interfaces de usuario. Esto permite que las entradas del teclado y del *mouse* controlen directamente al personaje o las mecánicas del juego sin interferencias.



Ilustración 27: Blueprint New Game

#### 4.1.2 Menú de Nueva Partida

La pantalla de carga de nueva partida se mantendrá durante 20 segundos al seleccionar el botón *New Game*, dando así la sensación de tener una pantalla de carga de nivel. Adicionalmente he añadido un widget con las instrucciones sobre cómo jugar al juego.



Ilustración 28: Interfaz carga del juego

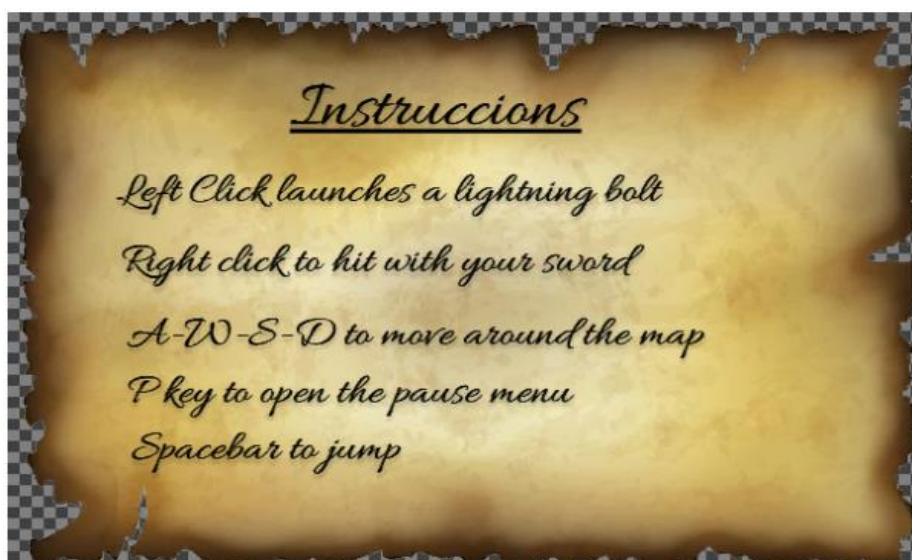


Ilustración 29: instrucciones del juego

### 4.1.3 Menú de Pausa

Este menú de pausa está diseñado con el mismo estilo visual que el menú principal, con botones de madera y un fondo limpio para asegurar coherencia en la interfaz de usuario. Las funciones del menú de pausa están implementadas a través de *Widget Blueprints* de *Unreal Engine*, permitiendo una experiencia fluida y fácilmente accesible para los jugadores.



Ilustración 30: Interfaz menú de pausa

A continuación, se explicará la implementación de los distintos botones:

- **Resume Button:** Se activa cuando el jugador hace clic en el botón de "Resume" para continuar jugando.
  - El nodo "Remove from Parent" elimina el menú de pausa del *viewport*.
  - El nodo "Set Game Paused" quita el juego del estado de pausa, permitiendo que la acción continúe.
  - "Set Show Mouse Cursor" oculta el cursor del ratón, devolviendo el control completo al jugador.
  - "Set Input Mode Game Only" asegura que la entrada del jugador solo se dirija al juego, eliminando la interacción con el HUD o cualquier *widget* en pantalla.

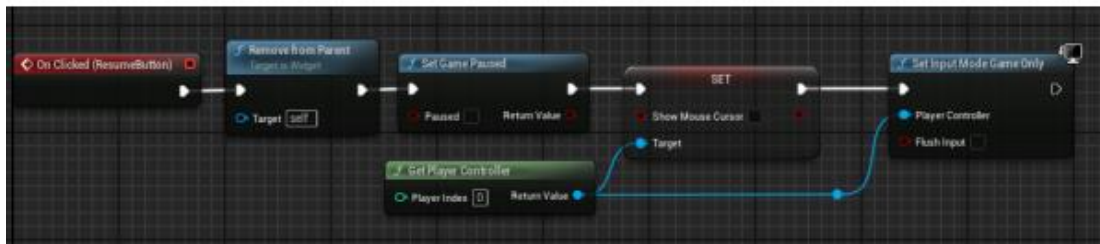


Ilustración 31: Blueprint Resume Button de menú de pausa

- **Quit Button:** Se activa cuando el jugador selecciona el botón para salir del menú de pausa, pero permanecer en el nivel actual.
  - El nodo "Set visibility" cambia la visibilidad de ciertos elementos del menú, haciendo que los botones u opciones específicas se oculten o muestren.

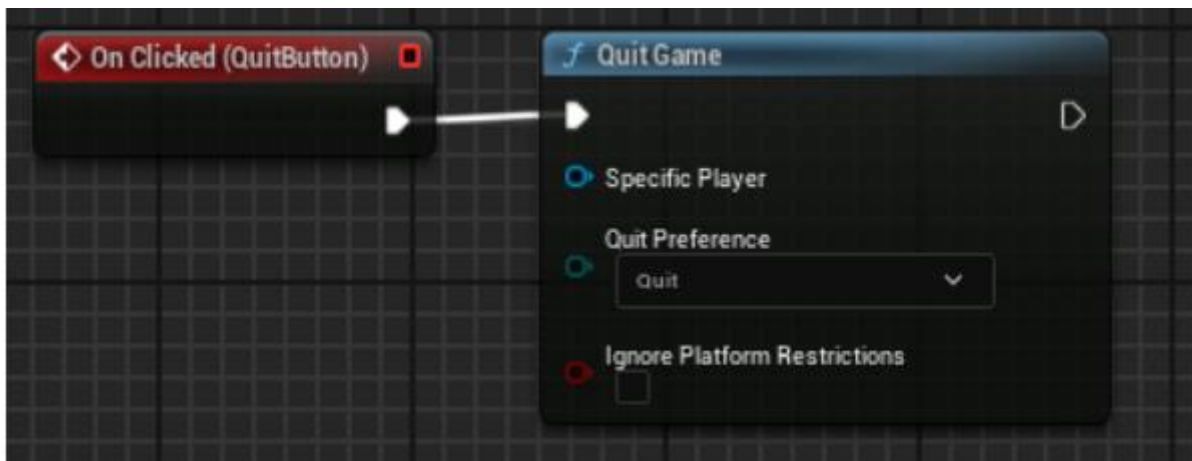


Ilustración 32: Blueprint Quit Button de menú de pausa

- **Quit Button Main Menu y Quit Button Desktop:**
  - Para "QuitButtonMainMenu", el nodo "Open Level by Object Reference" carga el nivel del menú principal, permitiendo al jugador salir de la partida actual y regresar al menú principal del juego.
  - Para "QuitButtonDesktop", el nodo "Quit Game" cierra el juego por completo, finalizando la aplicación.



Ilustración 33: Blueprint QuitButtonMainMenu de menú de pausa

- **Load Button:** Este evento se activa cuando el jugador presiona el botón de cargar partida en el menú de pausa.
  - El nodo "Is valid" comprueba si el *widget* de carga ya ha sido creado para evitar duplicados.
  - Si el *widget* de carga no es válido, el nodo "Create WBP Load Menu Widget" crea un nuevo *widget* de menú de carga y lo añade al *viewport* con "Add to Viewport".

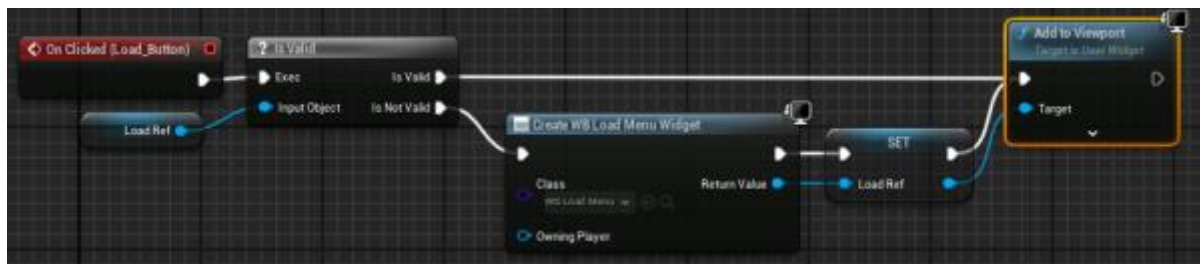


Ilustración 34: Blueprint Load Button de menú de pausa

- **Save Button:** Se activa cuando el jugador decide guardar su progreso.
  - Similar al botón de cargar, el nodo "Is valid" verifica si el *widget* de guardado ya existe.
  - Si no existe, el nodo "Create WBP Save Menu Widget" crea el *widget* de guardado, y lo añade al *viewport* usando "Add to Viewport".



Ilustración 35: Blueprint Save Button de menú de pausa

#### 4.1.4 Menú de Guardar Partida

Este bloque de *Blueprints* maneja las acciones del menú de guardar partida en *Unreal Engine*, permitiendo navegar entre menús, gestionar instancias del juego y guardar partidas.

- **Event Construct:** Se activa cuando el *widget* de menú de guardado de partida es construido (es decir, cuando aparece en la pantalla).

- El nodo "Get Game Instance" obtiene una referencia a la instancia del juego actual. Esta instancia contiene información global del juego, como los datos del progreso del jugador.
- El nodo "Cast To WB\_MyGameInstance" intenta convertir esa referencia a un tipo específico de instancia de juego, en este caso, "WB\_MyGameInstance", que es una clase personalizada creada para gestionar las mecánicas específicas del proyecto.
- Si la conversión es exitosa, se almacena la referencia a esta instancia del juego en una variable con el nodo "SET Game Instance Ref", lo que permite acceder a sus funciones y variables más adelante. Esto es fundamental para manejar acciones como el guardado de partidas, ya que la instancia del juego es donde se almacenan los datos necesarios para esa funcionalidad.

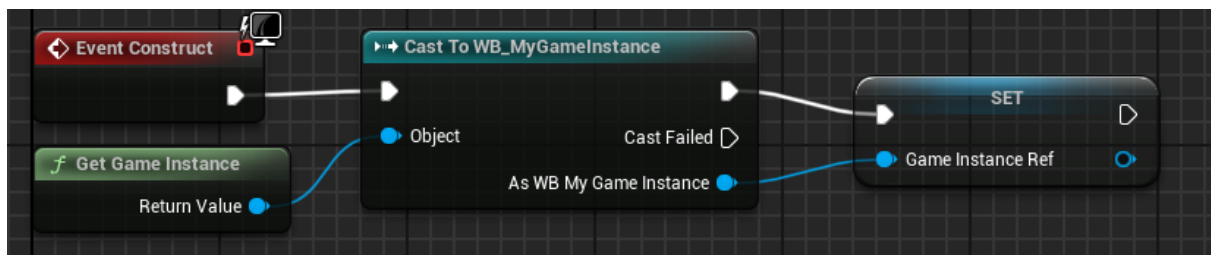


Ilustración 36: Blueprint Event Construct de Guardar partida

- **Back Button:** Este evento ocurre cuando el jugador presiona el botón de "Volver" dentro del menú.
  - **Remove from Parent:** Este nodo elimina el widget actual del viewport, es decir, cierra el menú de guardado y lo saca de la pantalla.
  - **Branch (Is Game Paused):** Verifica si el juego está en pausa. Dependiendo del resultado, se toman distintas acciones.
  - **Create WB Main Menu Widget:** Si el juego no está pausado, este nodo crea el *widget* correspondiente al menú principal.
  - **Add to Viewport:** Añade el widget del menú principal al *viewport*, lo que permite que el jugador pueda interactuar con él.

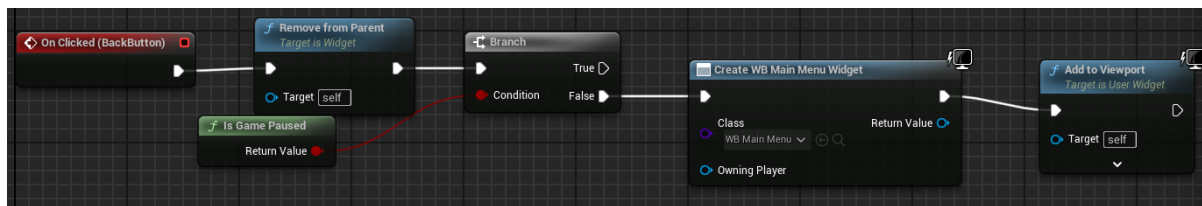


Ilustración 37: Blueprint Back Button de Guardar partida

- **Save Button 1:** Este evento se activa cuando el jugador hace clic en el botón de "Guardar".
  - **SET (Slot Name Save):** Establece el nombre del slot en el que se va a guardar la partida. En este caso, se asigna un valor de "slot1", lo que permite identificar el espacio de guardado.
  - **Save Game (Game Instance Ref):** Este nodo realiza la acción de guardar la partida utilizando la referencia a la instancia de juego (Game Instance). Es crucial para que los datos persistentes sean almacenados correctamente.

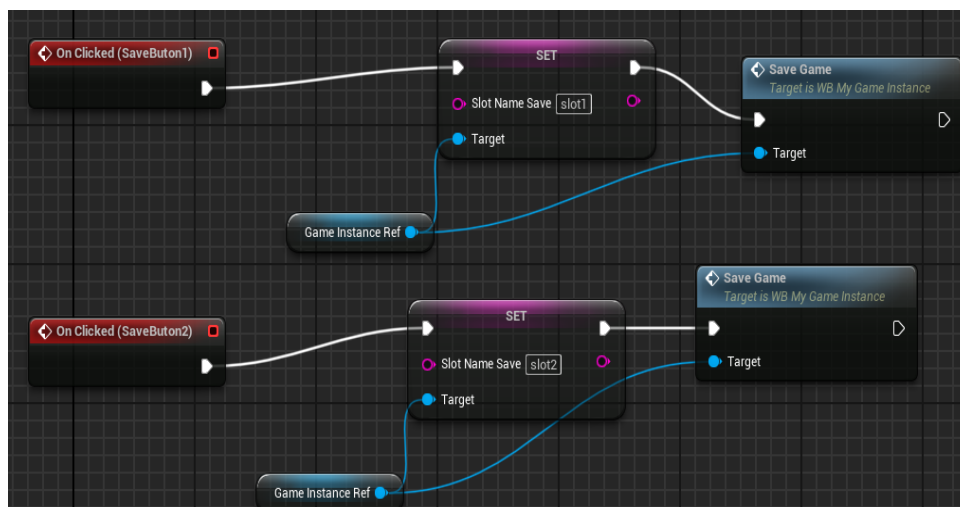


Ilustración 38: Blueprint Save Button slot de Guardar partida

#### 4.1.5 Menú de Cargar Partida

Este menú gestiona el proceso de carga de partidas, incluyendo las opciones para navegar por el menú de carga y restaurar una partida guardada.



Ilustración 39: Interfaz Guardar partida

- **Event Construct:** Se activa cuando el *widget* de menú de carga de partida es construido (es decir, cuando aparece en la pantalla).
  - El nodo "Get Game Instance" obtiene una referencia a la instancia del juego actual.
  - Luego, el nodo "Cast To WB\_MyGameInstance" intenta convertir esa referencia a un tipo específico de instancia de juego, en este caso "WB\_MyGameInstance".
  - Si la conversión es exitosa, se almacena la referencia a esta instancia del juego en una variable con el nodo "SET Game Instance Ref", lo que permite acceder a ella más adelante.

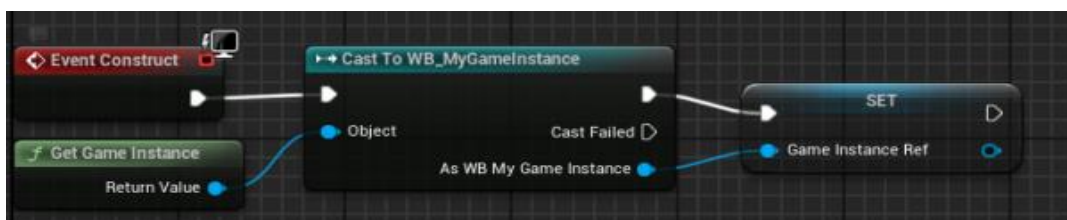


Ilustración 40: Blueprint event construct de Cargar partida

- **Back Button:** Este evento se activa cuando el jugador hace clic en el botón de "Back" para salir del menú de cargar partida y regresar al menú principal.

- El nodo "Remove from Parent" elimina el *widget* actual (menú de carga) del viewport, lo que significa que deja de ser visible en la pantalla.
- El nodo "Is Game Paused" verifica si el juego está en pausa.
- "Branch" usa el resultado de esta verificación. Si el juego está pausado, el nodo crea un nuevo widget de menú principal mediante "Create WB Main Menu Widget".
- Luego, el *widget* del menú principal se añade al *viewport* usando el nodo "Add to Viewport", haciendo que aparezca en la pantalla.

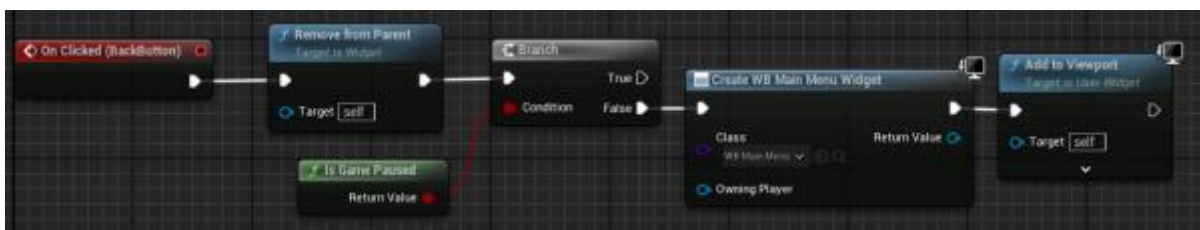


Ilustración 41: Blueprint Back Button de Cargar partida

- **Load Button 1:** Se activa cuando el jugador hace clic en el botón para cargar la partida desde una ranura específica (en este caso, "slot1").
  - El nodo "SET Slot Name Save" define el nombre de la ranura de guardado ("slot1"), indicando qué partida debe cargarse.
  - El nodo "Menu Load Game" llama a una función dentro de la instancia del juego para ejecutar la carga de la partida guardada desde la ranura especificada.

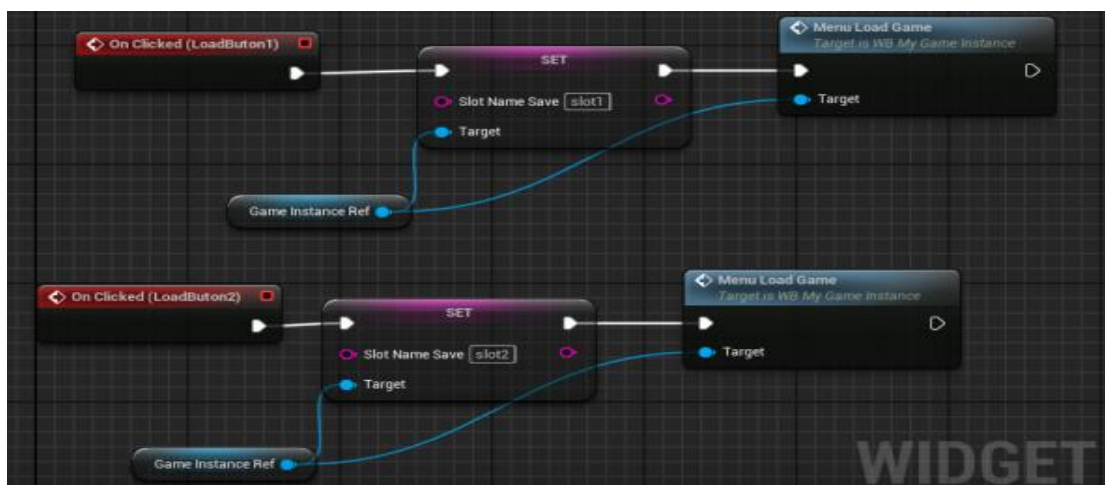


Ilustración 42: Blueprint de Load Button 1 de Cargar partida

## 4.2. Sistema de guardado

El sistema de guardado implementado en el proyecto se basa en dos archivos clave: `WB_MyGameInstance` y `WB_SaveGame`. `WB_MyGameInstance` contiene la lógica principal para gestionar las funciones de guardado y carga de partidas, incluyendo la manipulación de datos del jugador y el nivel actual. Por otro lado, `WB_SaveGame` actúa como el contenedor de datos, almacenando información esencial como la posición del jugador (mediante un *transform*) y el nombre del nivel en el que se encuentra. Este sistema permite que el jugador pueda guardar su progreso en cualquier momento y retomar la partida desde el mismo punto y nivel en el que dejó el juego. A continuación, se desarrollarán los distintos archivos que se utilizan:

`WB_MyGameInstance`: actúa como el núcleo central de la gestión de datos durante el juego y contiene varias variables importantes que permiten almacenar y manipular información crítica relacionada con el progreso del jugador. Estas variables se utilizan para facilitar el guardado y la carga de partidas, entre otras funciones. A continuación, se detallan las principales variables presentes y funciones:

- **Save Game**: Esta función se encarga de almacenar el estado actual del juego en un archivo de guardado.
  - **Create Save Game Object (WB\_SaveGame)**: Crea una instancia del archivo de guardado, utilizando la clase `WB_SaveGame`, que contiene dos variables clave:
    - *Transform* (para almacenar la posición y rotación del jugador)
    - *savedLevel* (para guardar el nombre del nivel actual).
  - **Get Player Character**: Obtiene el personaje controlado por el jugador en el juego.
  - **Get Actor Transform**: Captura las coordenadas actuales del personaje (su ubicación, rotación y escala) y las asigna a la variable *Transform* dentro de `WB_SaveGame`.
  - **Get Current Level Name**: Obtiene el nombre del nivel actual y lo almacena en la variable *savedLevel* dentro de `WB_SaveGame`.

- **Save Game to Slot:** Finalmente, guarda los datos en un "slot" o espacio de guardado, permitiendo que el jugador retome la partida desde ese punto.
- **Get Player Character:** Obtiene el personaje controlado por el jugador en el juego.
- **Get Actor Transform:** Captura las coordenadas actuales del personaje (su ubicación, rotación y escala) y las asigna a la variable *Transform* dentro de *WB\_SaveGame*.
- **Get Current Level Name:** Obtiene el nombre del nivel actual y lo almacena en la variable *savedLevel* dentro de *WB\_SaveGame*.
- **Save Game to Slot:** Finalmente, guarda los datos en un "slot" o espacio de guardado, permitiendo que el jugador retome la partida desde ese punto.
- **SaveSlotName (String):** Esta variable almacena el nombre del archivo de guardado (slot) en el que se registran los datos del jugador. Permite al sistema identificar y acceder a la partida guardada correcta cuando se carga el juego. Los jugadores pueden tener múltiples archivos de guardado, y esta variable define cuál se está usando en cada momento.
- **GameInstanceRef (Referencia):** Esta variable guarda una referencia a la instancia de juego actual. Es crucial para que otras partes del sistema de guardado/carga puedan acceder a la información del juego, como los datos de jugador y nivel. A través de esta referencia, se puede acceder y modificar la información del juego de forma global.
- **SavedPlayerTransform (Transform):** Esta variable almacena la posición, rotación y escala del jugador en el mundo del juego. Se utiliza durante las operaciones de guardado y carga para restaurar al jugador en la misma posición en la que estaba cuando se guardó la partida.
- **SavedPlayerLevel (String):** Almacena el nombre del nivel en el que se encontraba el jugador al momento de guardar. Esta variable asegura que el nivel correcto se cargue cuando el jugador decide reanudar su partida.

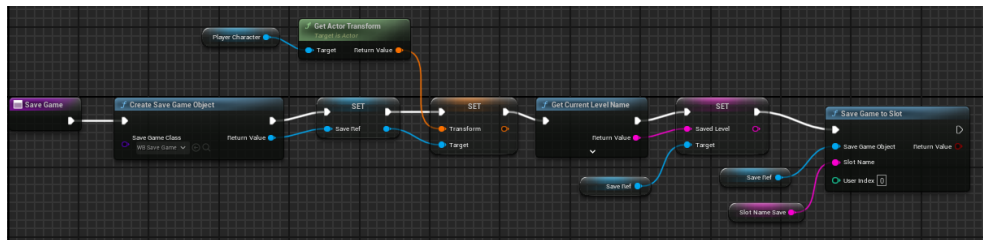


Ilustración 43: Blueprint Sistema de guardado

- **Load Game:** Esta función se encarga de recuperar el estado previamente guardado.
  - **Load Game from Slot:** Carga el archivo de guardado desde el slot especificado, accediendo a los datos almacenados en la instancia de `WB_SaveGame`.
  - **Get Player Character:** Recupera el personaje controlado por el jugador.
  - **Teleport:** El personaje se teletransporta a las coordenadas almacenadas en la variable `Transform` del archivo de guardado, asegurando que reaparezca en la misma ubicación en la que estaba al guardar.
  - **open Level1:** Cambia al nivel almacenado en la variable `savedLevel1`, permitiendo que el jugador vuelva al nivel donde se realizó la última acción de guardado.

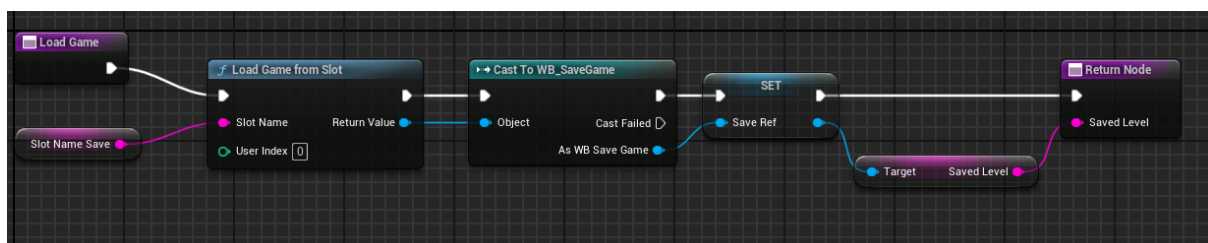


Ilustración 44: Blueprint load game

- **Game Load:** Esta función inicializa la partida cargando los datos guardados y colocando al jugador en su posición.
  - **Event Load:** Es el evento que inicia el proceso de carga cuando el juego se abre o se accede a la opción de cargar una partida.

- **Teleport:** Similar a la función de carga, teletransporta al jugador a la ubicación guardada en la variable **Transform** y lo coloca en el nivel adecuado según lo almacenado en **savedLevel1**.

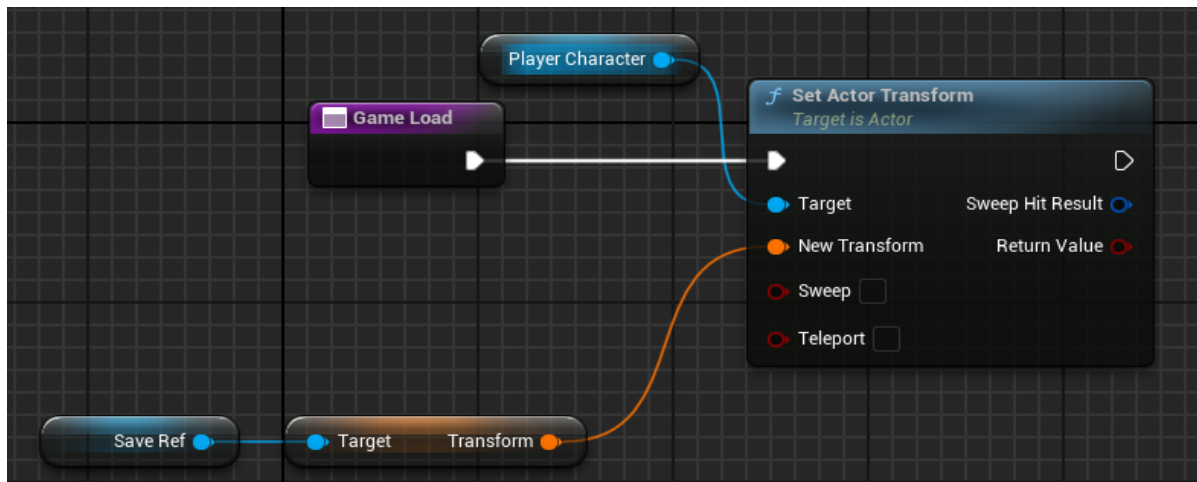


Ilustración 45: Blueprint game load

- **Event Graph:** conecta las funciones clave de guardado y carga con eventos dentro del juego. En este caso, controla cuándo y cómo se ejecutan las acciones de guardado y carga.

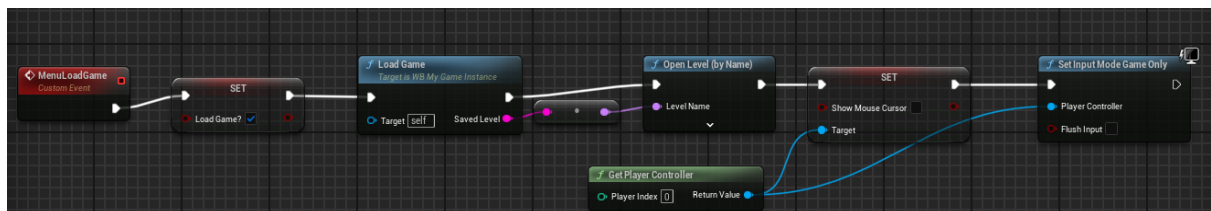


Ilustración 46: Blueprint event graph carga de partida

**WB\_SaveGame:** cumple la función de almacenar los datos esenciales del jugador y del juego en un formato persistente que puede ser recuperado en cualquier momento. Este archivo es clave en el sistema de guardado, ya que guarda la información necesaria para restaurar el estado del juego cuando se carga una partida.

- **Transform:** Esta variable almacena la ubicación, rotación y escala del jugador al momento de guardar la partida. Es de tipo **Transform**, lo que permite guardar de manera precisa la posición 3D del personaje dentro del nivel.

- **savedLevel**: Esta variable de tipo *String* guarda el nombre del nivel actual en el que se encuentra el jugador. Esto asegura que cuando se carga la partida, no solo se restaure la posición del personaje, sino que también se cargue el nivel adecuado.

## 4.3 Cámara

El sistema de cámara implementado en este proyecto utiliza una configuración en primera persona, ya que se trata de un juego de acción en el que el jugador experimenta el entorno desde la perspectiva del personaje controlado. La cámara está posicionada justo detrás del esqueleto del personaje, que en este caso está conformado únicamente por dos brazos visibles, dado que el enfoque está en la inmersión y simplicidad visual.

La cámara, denominada *FirstPersonCamera*, está integrada dentro del *blueprint* del personaje principal y está vinculada directamente al esqueleto. De esta forma, la vista del jugador se ajusta automáticamente a los movimientos del personaje, ya sea al caminar, girar o realizar acciones específicas como atacar o interactuar con el entorno.

En cuanto a la configuración técnica, la cámara está configurada en modo de proyección *Perspective* con un campo de visión (FOV) de 90 grados. Este ajuste proporciona un ángulo de visión amplio, sin generar distorsiones notables. Además, se ha activado la opción **Use Pawn Control Rotation**, permitiendo que la cámara siga la rotación del jugador en tiempo real, lo que asegura una experiencia visual coherente y fluida durante el juego.

Esta implementación busca ofrecer una visión clara y directa desde la perspectiva del personaje, optimizando la interacción con el entorno y los elementos de combate en el juego.

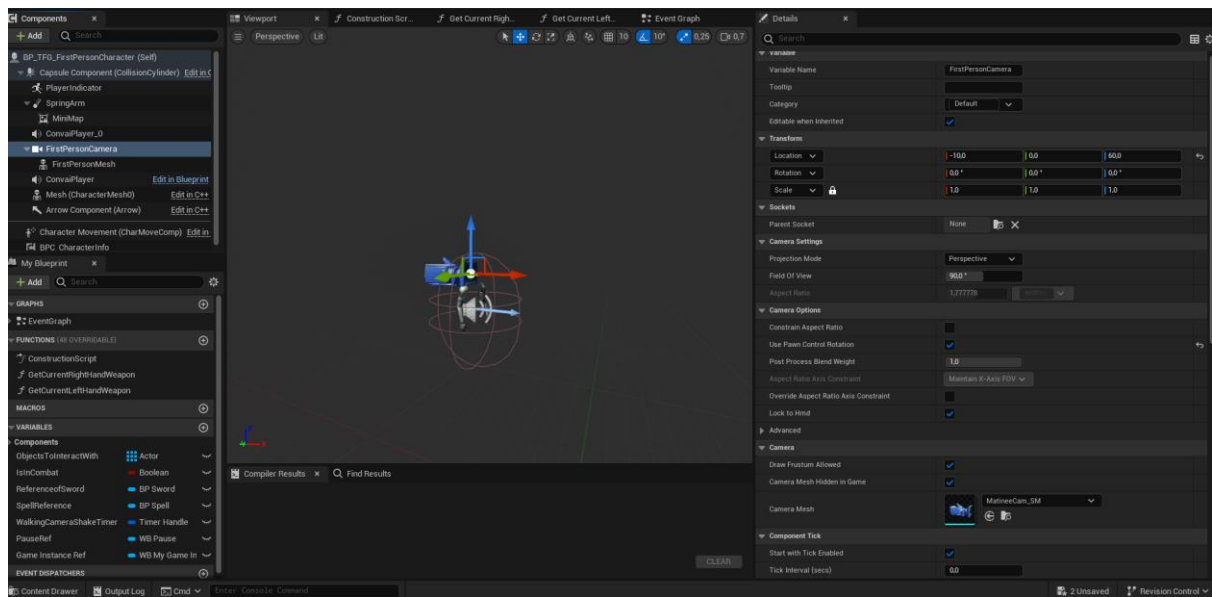


Ilustración 47: ViewPort de mi jugador

## 4.4 Construcción del mundo

La construcción del mundo para "Agora Island" se llevó a cabo utilizando *Unreal Engine 5.3*, comenzando desde un proyecto en blanco y desarrollando cada componente del entorno virtual con meticulosidad. El proceso de creación del mapa comenzó con la configuración de la iluminación, fundamental para establecer la atmósfera y realismo del juego. A continuación, se construyó la base de la isla, creando una estructura sólida sobre la que se añadirían los detalles. La creación del océano fue crucial para definir la interacción del jugador con el entorno marino, mientras que la construcción detallada de la isla permitió formar un espacio interactivo y dinámico. Finalmente, se incorporaron elementos de vegetación y rocas para enriquecer el paisaje y ofrecer una experiencia de juego más inmersiva. Cada etapa del desarrollo se realizó con las herramientas avanzadas de *Unreal Engine 5.3*, garantizando un entorno visualmente atractivo y funcional. A continuación, se desarrollarán cada una de las tareas anteriormente mencionadas:

### 4.4.1 Iluminación del mapa

La iluminación del entorno fue una de las primeras tareas críticas al construir "Agora Island". Se emplearon múltiples componentes de iluminación, incluyendo la *DirectionalLight*, que actúa como la fuente de luz principal, simulando la luz

solar; y el *SkyLight*, que permite capturar y reflejar la luz del cielo para iluminar las sombras de manera realista. Además, se configuró la *ExponentialHeightFog*, la cual crea un efecto de niebla atmosférica gradual, logrando una mayor profundidad y volumen en la escena. La incorporación de la *SkyAtmosphere* fue fundamental para generar una atmósfera de cielo realista, interactuando con los otros elementos de iluminación. Finalmente, se añadieron nubes volumétricas para proporcionar mayor realismo y variación en el entorno. Todo este sistema de iluminación fue crucial para definir el tono y la atmósfera del juego, asegurando un ambiente inmersivo y visualmente atractivo tanto en escenarios diurnos como nocturnos.

| Item Label ▲         | Type                 |
|----------------------|----------------------|
| IslandMap (Editor)   | World                |
| Lighting             | Folder               |
| DirectionalLight     | DirectionalLight     |
| ExponentialHeightFog | ExponentialHeightFog |
| SkyAtmosphere        | SkyAtmosphere        |
| SkyLight             | SkyLight             |
| SM_SkySphere         | StaticMeshActor      |
| VolumetricCloud      | VolumetricCloud      |

Ilustración 48: Elementos de Iluminación de mi mundo

#### 4.4.2 Creación Base de la isla

La base de la isla en "Agora Island" fue construida utilizando las potentes herramientas de modelado de terrenos que ofrece el *Landscape Mode* de *Unreal Engine* 5.3. Este modo permite esculpir y dar forma a terrenos complejos, ajustando la topografía de la isla para que refleje un entorno natural y dinámico. Entre las principales herramientas utilizadas para este proceso destacan *Sculpt*, *Erase*, *Smooth*, *Flatten*, y *Ramp*, cada una desempeñando un papel clave en la creación de una estructura de terreno realista.

- **Sculpt:** Esta herramienta fue fundamental en las primeras fases del diseño del terreno. Permitted modelar de manera libre las elevaciones y depresiones en el terreno, creando montañas y colinas. A través de diferentes ajustes en la intensidad y el tamaño del pincel, se pudo dar forma al relieve de la isla con una gran variedad de detalles.

- **Erase:** A medida que se definían las características del terreno, la herramienta *Erase* se utilizó para corregir áreas no deseadas o borrar partes del relieve, proporcionando la flexibilidad necesaria para ajustar la topografía de manera precisa y mantener un diseño limpio.
- **Smooth:** Una vez modelado el terreno, la herramienta *Smooth* fue esencial para suavizar las transiciones entre las áreas elevadas y las depresiones, creando una apariencia más natural y evitando bordes o picos abruptos en las formaciones del terreno. Esto garantizó que el terreno fuera visualmente coherente y agradable de explorar.
- **Flatten:** En zonas donde se requerían superficies planas, como para la ubicación de construcciones o espacios abiertos, se utilizó la herramienta *Flatten*. Esta permitió establecer áreas niveladas con precisión, necesarias para futuras interacciones del jugador o para la implementación de estructuras adicionales en el entorno.
- **Ramp:** Por último, la herramienta *Ramp* fue utilizada para crear pendientes controladas entre diferentes elevaciones del terreno, lo que permitió la creación de caminos suaves o transiciones entre colinas y valles. Esta herramienta facilitó el diseño de zonas accesibles para los jugadores, asegurando que el terreno fuera funcional y estéticamente consistente.

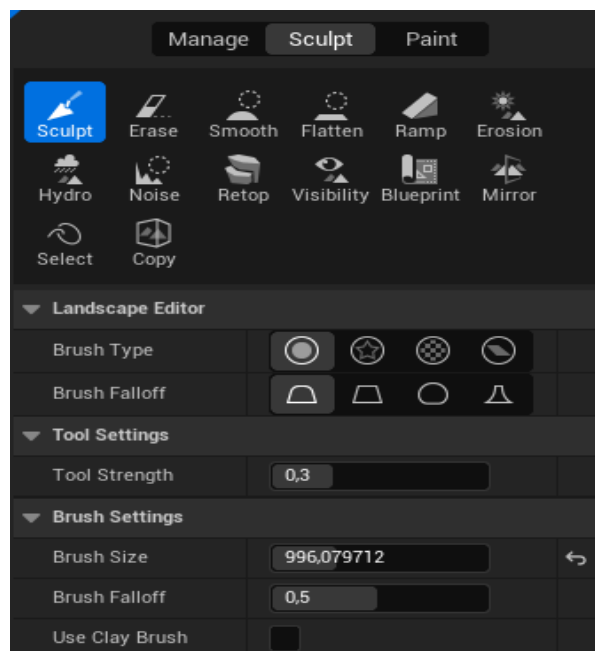


Ilustración 49: Herramientas de modelaje del mundo

La combinación de estas herramientas permitió construir la base de la isla de manera eficiente y con un alto grado de control sobre la topografía, asegurando un terreno diverso y realista que sirve como el núcleo de la experiencia de juego en "Agora Island".

#### 4.4.3 Creación océano

El océano que rodea "Agora Island" fue creado usando varios sistemas relacionados con el manejo del agua en *Unreal Engine*, destacando el *WaterBodyOcean*. Este componente permite no solo la generación del cuerpo de agua, sino que también regula la física y la interacción del agua con otros elementos del entorno, como las olas y las corrientes. Asimismo, se utilizó el *WaterBrushManager* para gestionar la creación y modificación de áreas de agua, lo que permitió ajustar con precisión los límites y la profundidad del océano. La implementación de la *WaterZone* fue fundamental para crear las zonas de interacción acuática, asegurando una experiencia fluida para los jugadores al navegar o interactuar con el agua. Este conjunto de herramientas ofreció la capacidad de personalizar el comportamiento del océano y mejorar el realismo visual.

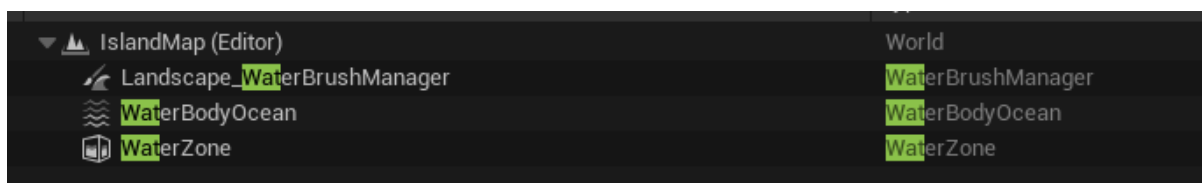


Ilustración 50: Elementos de agua del mundo

#### 4.4.4 Creación de la isla

La creación de la isla en "Agora Island" se realizó utilizando el sistema de *Landscape Mode* de *Unreal Engine 5.3*, que permite modelar y texturizar terrenos con gran precisión. En primer lugar, se creó un material personalizado combinando diversas texturas obtenidas a través de *Quixel Bridge*, garantizando un alto nivel de realismo en las superficies del terreno, como la arena, el césped y las rocas. Este material se diseñó para adaptarse de manera dinámica a las distintas áreas del mapa, permitiendo una transición fluida entre diferentes tipos de superficies y elevaciones.



*Ilustración 51: Materiales de Quixel Bridge*

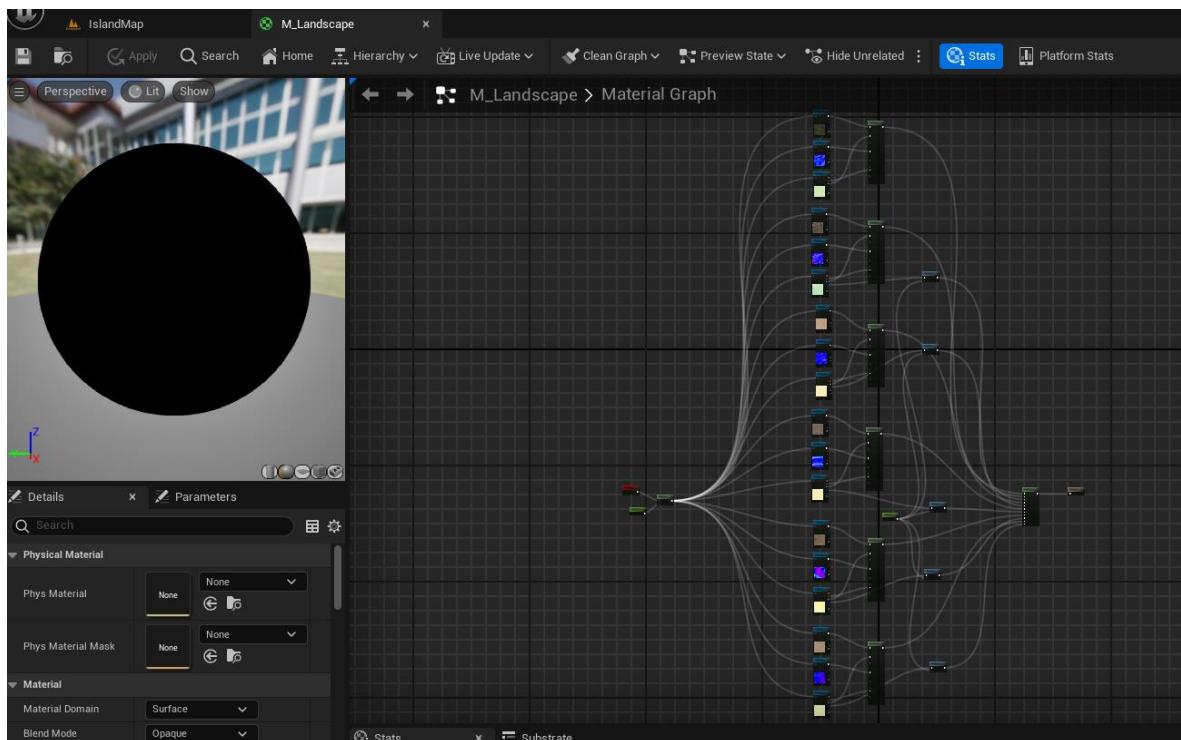


*Ilustración 52: Layer de materiales del mundo*

Una vez preparado el material, se procedió a pintar el mapa utilizando la herramienta Paint dentro del *Landscape Mode*, permitiendo aplicar las texturas en distintas partes de la isla de forma precisa y personalizada. Con la ayuda de un pincel de gran tamaño, se cubrieron áreas extensas rápidamente, mientras que se emplearon pinceles más pequeños para los detalles en zonas específicas como senderos, montañas y playas. Este proceso permitió definir claramente los distintos biomas de la isla, desde las zonas costeras hasta las áreas más elevadas, ofreciendo un entorno visualmente coherente y atractivo para los jugadores.

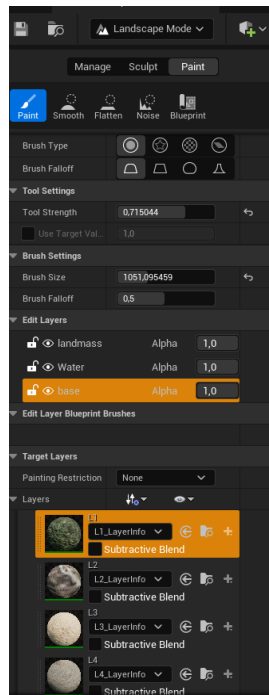


Ilustración 53: Herramientas de capas del mundo

El uso de texturas de alta calidad de *Quixel Megascans*, junto con la versatilidad de las herramientas de *Landscape Mode*, fue fundamental para lograr un paisaje natural y realista, que no solo enriquece la estética del juego, sino que también ofrece una experiencia inmersiva al jugador.

#### 4.4.5 Vegetación y rocas

Para dar vida al entorno natural de "Agora Island", se utilizaron los *assets* de *Quixel Megascans* dentro de *Unreal Engine 5.3*, específicamente modelos de alta calidad de vegetación y rocas. Estos *assets* proporcionan un nivel de detalle realista que contribuye a crear un entorno visualmente inmersivo. La herramienta *Foliage Mode* fue clave para distribuir de manera eficiente los elementos naturales a lo largo de la isla. Utilizando la opción *Paint* con un pincel de gran tamaño y un nivel de densidad ajustado, se aplicaron diferentes tipos de plantas, árboles y rocas sobre el terreno, asegurando una distribución natural y variada en todo el mapa. Gracias a las avanzadas opciones de filtros y selección de capas, fue posible colocar la vegetación únicamente en áreas de interés, como paisajes y estructuras estáticas, garantizando un entorno coherente y atractivo.

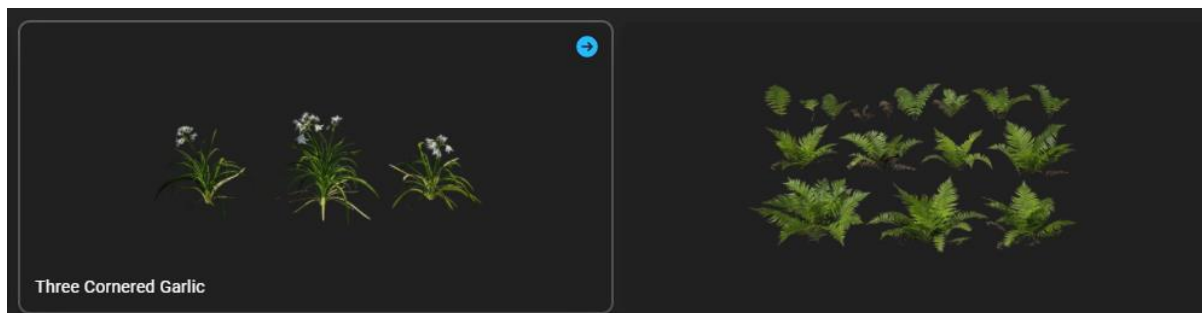


*Ilustración 54: Assets de vegetación de Quixel MegaScans*

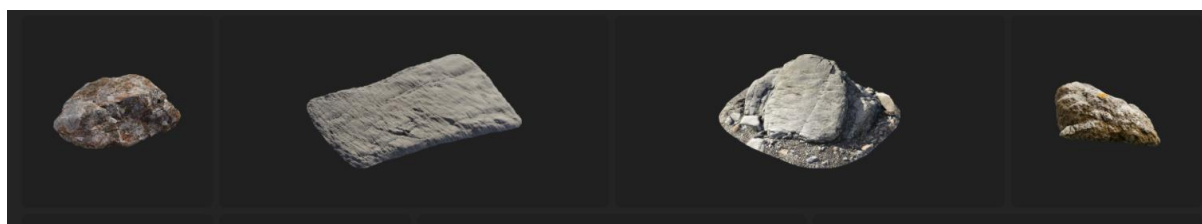


*Ilustración 55: Assets Rocas Quixel MegaScans*

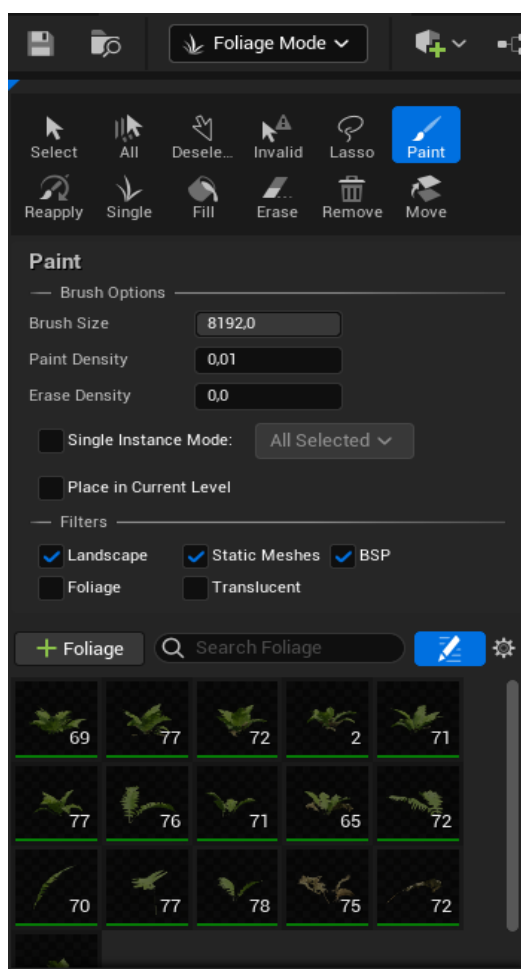


*Ilustración 56: Herramienta de vegetación*

#### 4.4.6 Incorporación de assets adicionales

Para enriquecer aún más el entorno de "Agora Island" y ofrecer una experiencia más inmersiva a los jugadores, se añadieron diversos assets cuidadosamente seleccionados que contribuyen a crear un ambiente único y lleno de detalles. Entre los elementos incluidos se encuentran casas, un faro que domina el horizonte, una nave abandonada y un barco pirata, cada uno de ellos aportando una historia visual al entorno. Además, se integraron detalles como una barca, troncos de palmeras, huesos, cofres y un muelle, que ayudan a construir una atmósfera envolvente y rica en narrativa.

Estos elementos no solo ofrecen un mayor nivel de detalle y complejidad al entorno visual, sino que también mejoran la interactividad del mundo, invitando a los jugadores a explorar cada rincón de la isla. Cada asset fue cuidadosamente colocado para complementar la topografía y la estética general de la isla, utilizando las herramientas de *Placement* y *Alignment* de *Unreal Engine* para asegurar que todo esté perfectamente integrado en el paisaje.

### 4.5 Personaje jugable

#### 4.5.1 Input Mapping Context e Input Actions

El **Input Mapping Context** (IMC) es el sistema que define cómo se interpretan las entradas del jugador. Es un mapa que vincula las teclas físicas o entradas del controlador a las acciones específicas dentro del juego. Cada *Input Mapping Context* puede ser personalizado según el estado del juego (combate, no combate, etc.).

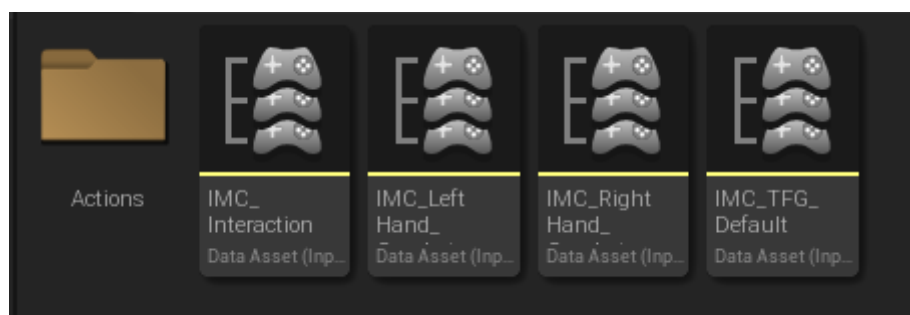


Ilustración 57: Input Mapping Context

Este contexto se asigna en el *Event Begin Play*, asegurándose de que el sistema de entradas esté correctamente vinculado con las acciones del personaje desde el inicio. Esto incluye las entradas clave como movimiento, ataques, y saltos, que están vinculadas a los *Input Actions*.

Los ***Input Actions*** son las acciones que el personaje puede ejecutar en respuesta a las entradas del jugador. Existen varias acciones definidas:

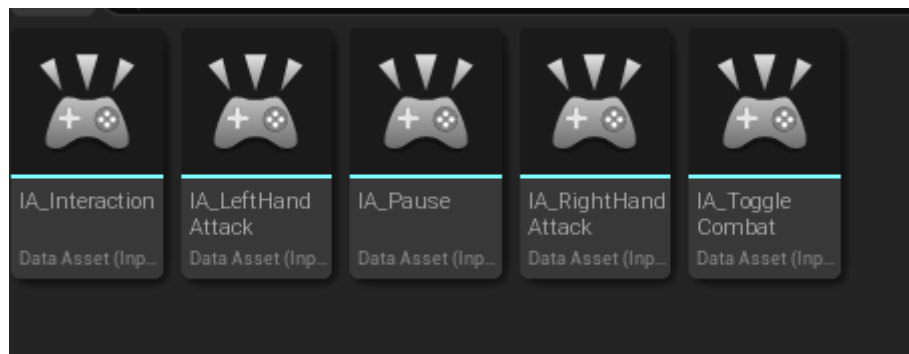


*Ilustración 58: Input Actions*

- **IA\_LeftHandAttack** y **IA\_RightHandAttack**: Controlan los ataques con las manos izquierda y derecha.
- **IA\_Move**: Manejando los movimientos hacia adelante/atrás y los laterales.
- **IA\_Jump**: Para la acción de saltar.

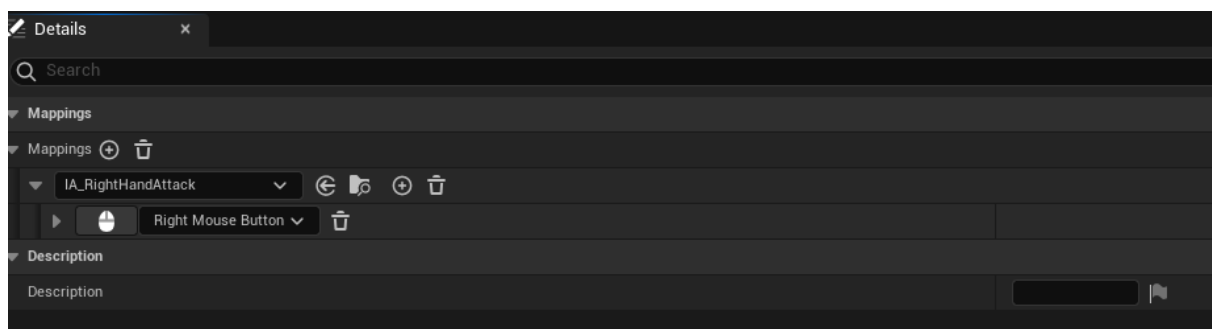


*Ilustración 59: Input Action Right Hand Attack*

Estos *Input Actions* están directamente vinculados a las funciones dentro del *Character Blueprint*, donde cada acción desencadena eventos como la reproducción de animaciones, movimientos o efectos visuales.

## 4.5.2 HUD (Head-Up Display)

El **HUD** en el proyecto tiene un papel importante, ya que presenta información crítica del personaje al jugador en tiempo real. En el *Event Begin Play*, vinculas el HUD con las variables clave del personaje, como la salud o el estado de combate.

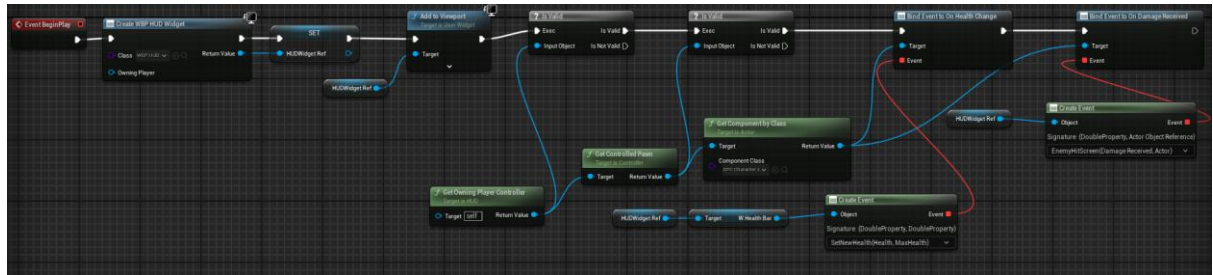


Ilustración 60: Blueprint HUD

### Barra de Salud y Vinculación al Personaje

La barra de salud, por ejemplo, está directamente conectada con la variable de salud dentro del *Character Blueprint*. Cada vez que el personaje recibe daño o se cura, el HUD se actualiza automáticamente para reflejar la cantidad correcta de salud.

El *Event Begin Play* asegura que el HUD esté correctamente sincronizado con el estado inicial del personaje, para que desde el momento en que comience el juego, se muestre la información correcta en pantalla.

## 4.5.3 Actor Component

En el *Character Info Blueprint*, gestionas atributos clave del personaje, como la salud, la fuerza y las habilidades. Estas variables se inician en *Event Begin Play* y son esenciales para definir el comportamiento del personaje.

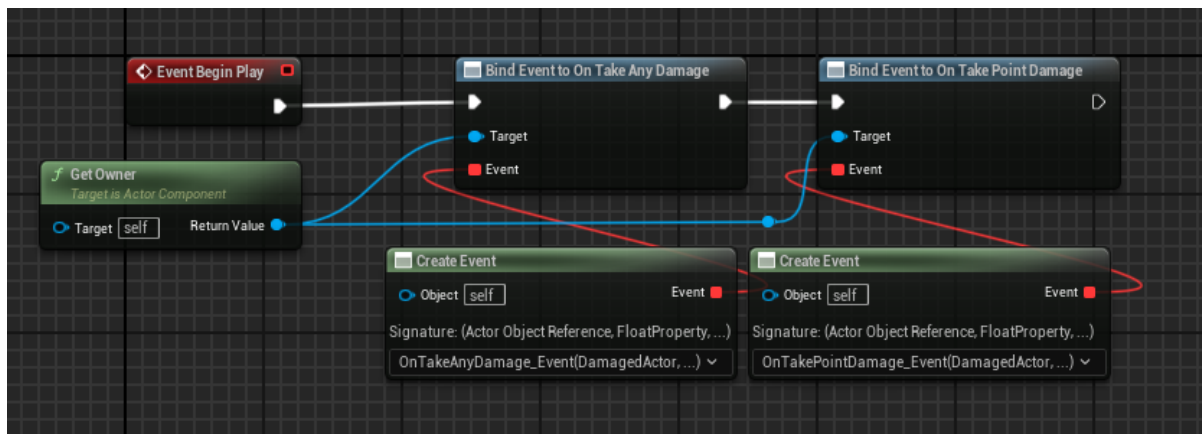


Ilustración 61: Blueprint Begin Play del Actor Component

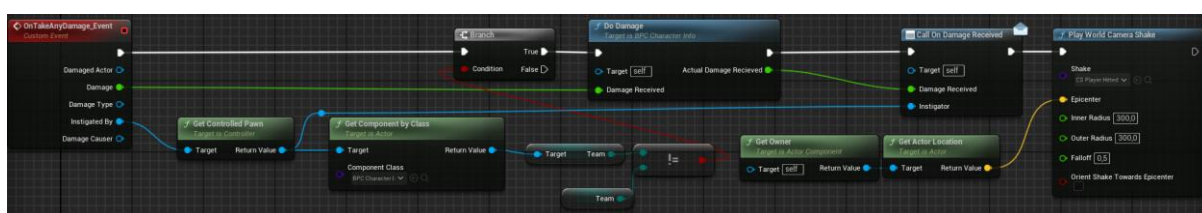


Ilustración 62: Blueprint OnTakeAnyDamage del Actor Component

## Salud y Daño

Las variables de salud se inicializan en *Begin Play*, lo que asegura que el personaje comience el juego con la cantidad correcta de vida. También se conectan al HUD para que, cuando la salud cambie (por daño o curación), el HUD se actualice de inmediato.

## Vinculación con el HUD

Aquí encontramos funciones que se utilizan para actualizar el HUD con la salud y otros atributos importantes del personaje. En el *Event Begin Play*, estas funciones se llaman para asegurarse de que el HUD esté mostrando la información correcta desde el inicio del juego.

### 4.5.4 Interfaces

Las **Interfaces** en *Unreal Engine* son fundamentales para la comunicación entre diferentes *blueprints* sin crear dependencias directas. Esto te permite crear un diseño más flexible y modular.

En este proyecto se han desarrollado dos interfaces fundamentales para gestionar tanto las interacciones del jugador con el entorno como el sistema de combate de los enemigos y el personaje principal. Las interfaces juegan un papel crucial en la modularidad y escalabilidad del juego, permitiendo que los objetos y personajes puedan implementar comportamientos comunes de forma eficiente y organizada. A continuación, se describen las dos interfaces creadas, detallando su funcionalidad y la integración dentro del proyecto.

### **Interfaz de Sistema de Combate:**

Esta interfaz está diseñada para gestionar los ataques realizados tanto por el personaje principal como por los enemigos. Implementa un sistema de combate modular, permitiendo que cualquier actor que implemente esta interfaz pueda ejecutar una serie de ataques predefinidos. Las funciones disponibles en esta interfaz son las siguientes:

- **SendEnemyAttack:**

Esta función permite a los enemigos enviar comandos de ataque hacia su objetivo, generalmente el jugador o algún otro actor del entorno. Al implementar esta función, cualquier actor puede desencadenar su animación o evento de ataque, lo que facilita la integración del sistema de combate en diferentes tipos de enemigos. El uso de esta función está vinculado al comportamiento del enemigo en su ciclo de vida dentro del juego, respondiendo cuando entra en combate o detecta al jugador.

- **SendLeftAttack:**

Esta función ejecuta un ataque desde el lado izquierdo del actor que implementa la interfaz. Su implementación es útil para personajes o enemigos que realizan ataques direccionales, lo que añade una capa de profundidad al sistema de combate. En el caso del jugador, este tipo de ataque puede estar vinculado a una entrada específica del controlador o a una decisión táctica durante el combate.

- **SendRightAttack:**

Similar a la función anterior, esta función permite al personaje realizar un ataque desde el lado derecho. Se utiliza en sistemas de combate avanzados, donde se busca ofrecer al jugador un control más preciso

sobre las acciones de ataque, permitiendo alternar entre ataques de diferentes direcciones. La implementación de ambas funciones (`SendLeftAttack` y `SendRightAttack`) da lugar a un sistema de combate dinámico y flexible, donde el jugador o enemigo puede ejecutar estrategias de ataque más complejas.

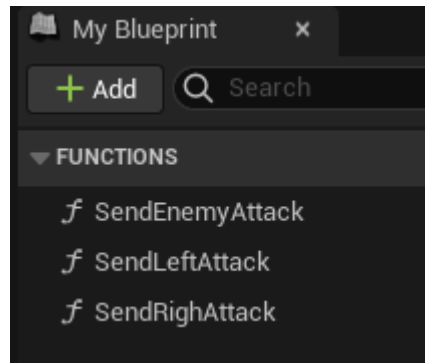


*Ilustración 63: Interfaz Sistema de Combate*

### **Interfaz de Interacción con Objetos:**

Esta segunda interfaz está enfocada en la interacción del jugador con los objetos del entorno. Implementa funciones que permiten manipular objetos interactivables, tales como recogerlos, activarlos o desactivarlos, según el contexto del juego. Las funciones disponibles en esta interfaz son las siguientes:

- **InteractWithObject:**

Esta función es utilizada para definir la interacción básica con cualquier objeto en el juego. Al implementar esta función, un objeto puede reaccionar a las acciones del jugador, como abrir puertas, recoger objetos o activar mecanismos. La versatilidad de esta función permite su uso en una amplia gama de situaciones de interacción en el juego.

- **ActivateObjectFunctionality:**

Esta función activa la funcionalidad de un objeto. Se utiliza en situaciones donde el jugador necesita activar un mecanismo específico, como encender una luz, activar una trampa o abrir una puerta. Su implementación es sencilla, pero muy útil en el desarrollo de puzzles o mecánicas que requieran interacción con el entorno.

- **DeactivateObjectFunctionality:**

De manera similar a la anterior, esta función desactiva la funcionalidad de un objeto. Es útil para apagar mecanismos, cerrar puertas o desactivar trampas, lo que añade una capa de control adicional sobre los objetos interactivos del entorno. Al tener tanto las funciones de activación como de desactivación en la misma interfaz, se asegura una flexibilidad para objetos que requieren varios estados.

- **PickThisUp:**

Esta función está dedicada a la recolección de objetos interactivos. Al implementar esta función, cualquier objeto puede ser recogido por el jugador, permitiendo así la integración con el sistema de inventario o de gestión de recursos. Su implementación es esencial para ítems, armas, monedas o cualquier elemento que el jugador pueda necesitar a lo largo del juego.

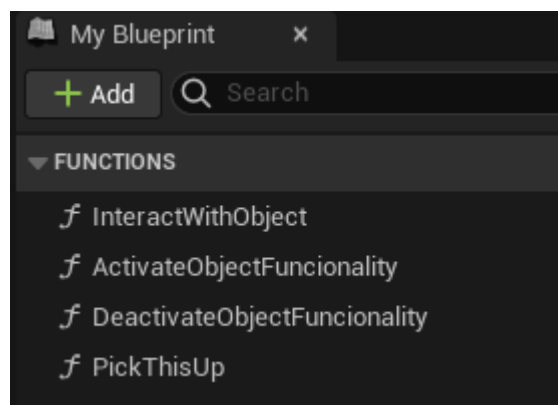


*Ilustración 64: Interfaz interacción objetos*

#### 4.5.5 GameMode

El **GameMode** define las reglas generales del juego, como quién es el jugador, qué HUD se debe usar, y qué personaje jugable se carga al inicio.

##### Configuración del Personaje y HUD

En el *Event Begin Play* del *GameMode*, se inicializa el personaje jugable (en este caso *FirstPersonCharacter*) y se asigna el HUD correcto para mostrar la información en pantalla. Esto asegura que, desde el principio, el jugador controle al personaje adecuado y vea la interfaz correspondiente.

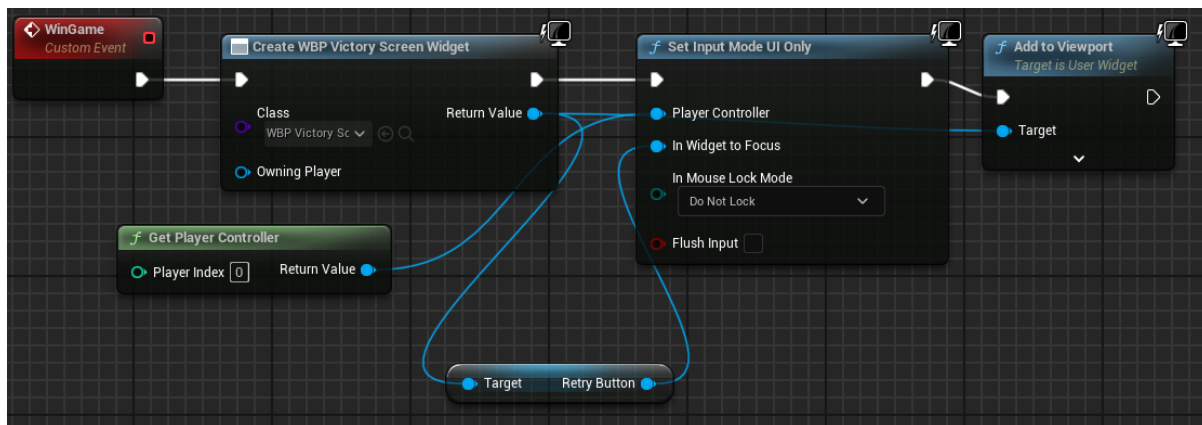


Ilustración 65: Blueprint Win Game de GameMode

#### 4.5.6 Blueprint del personaje

El *Character Blueprint* es el núcleo del comportamiento del personaje jugable. Aquí es donde se define todo, desde el movimiento hasta las acciones de combate, la salud, y las interacciones con el entorno.

El *Event Begin Play* juega un papel crucial en este *blueprint*, ya que es aquí donde se inicializan todas las variables y se configuran los sistemas del personaje.

- Asignación del *Input Mapping Context*: En el *Event Begin Play*, el *Input Mapping Context* se asigna al personaje, vinculando las teclas del jugador con los Input Actions.
- Inicialización de Variables:
  - Salud: En el *Begin Play*, inicializas la salud del personaje (por ejemplo, 100/100), para que cualquier daño o curación posterior se calcule correctamente.
  - Estado de Combate: Si el personaje tiene un estado de combate/no combate, esta variable se inicializa aquí para garantizar que el personaje esté en el estado correcto al inicio del juego.
- Vinculación con el HUD: También es en este punto donde el HUD se actualiza con la información correcta de las variables del personaje, como la salud. Si el personaje comienza con 100 puntos de salud, el HUD se actualiza en este momento para mostrar esta cantidad al jugador.

El movimiento del personaje está directamente relacionado con las entradas que provienen del *Input Mapping Context*. En el *Event Graph*, utilizas funciones como *Add Movement Input* junto con los vectores de dirección obtenidos a través de *Get Actor Forward Vector* y *Get Actor Right Vector*.

- Desplazamiento Adelante/Atrás e Izquierda/Derecha: Estas funciones procesan las entradas del jugador y mueven al personaje en la dirección correcta. Esto permite un control fluido y reactivo del personaje en el mundo del juego.

## **Ataques y Animaciones**

Los ataques con la mano izquierda y derecha están vinculados a las entradas del jugador a través de *IA\_LeftHandAttack* y *IA\_RightHandAttack*. Estas entradas activan funciones dentro del *Character Blueprint* que:

- Ejecutan las animaciones correspondientes.
- Calculan el daño a los enemigos.
- Aplican efectos visuales o de audio.

## **Temporizadores y Efectos Visuales**

El *Event Begin Play* también puede configurar temporizadores que controlan efectos visuales, como la sacudida de cámara cuando el personaje corre o realiza acciones intensas.

- *Walking Camera Shake*: A través de un temporizador, se controla cuándo activar y desactivar la sacudida de cámara, que mejora la sensación de inmersión y movimiento en el juego.

El *FirstPersonMesh* representa el modelo en 3D que ves cuando juegas en primera persona, como las manos y las armas. Este *mesh* también está vinculado a las animaciones del personaje, como los ataques o la recarga de armas.

## Vinculación con Animaciones

En el *Event Begin Play*, puedes asegurarte de que las animaciones iniciales estén configuradas correctamente. Por ejemplo, si el personaje comienza en una posición de "idle" o con un arma equipada, las animaciones correspondientes se configuran en este punto.

El *Event Begin Play* es uno de los eventos más importantes en tu proyecto, ya que sirve como punto de inicio para la configuración de todos los componentes del personaje y el juego. Aquí es donde:

1. Se asignan las entradas del jugador mediante el *Input Mapping Context*, lo que asegura que todas las acciones, como moverse, atacar y saltar, estén vinculadas correctamente a las teclas o botones.
2. Se inicializan las variables clave del personaje, como la salud y el estado de combate, asegurando que el personaje esté preparado desde el principio.
3. Se configura el HUD, que refleja las variables importantes del personaje (como la salud) y las muestra en pantalla.
4. Se vinculan los componentes y actores necesarios, como el *FirstPersonMesh*, las cámaras y las animaciones iniciales.
5. Se configuran efectos visuales y temporizadores, como la sacudida de cámara, que ayudan a mejorar la experiencia del jugador.

El proyecto de *Unreal Engine* está diseñado de manera estructurada, con el *Event Begin Play* actuando como el punto central donde todo el sistema se inicializa y comienza a funcionar. Desde la asignación de controles hasta la actualización del HUD y la inicialización de las variables de personaje, cada componente se conecta y funciona en armonía para ofrecer una experiencia fluida.

El *Character Blueprint*, el HUD, las interfaces, el *GameMode*, y el *FirstPersonMesh* están todos sincronizados para proporcionar una respuesta rápida y dinámica a las acciones del jugador, creando una experiencia inmersiva y bien diseñada.

Este enfoque asegura que todo el sistema funcione sin problemas, con cada pieza del proyecto contribuyendo al flujo de juego completo desde el momento en que el jugador inicia la partida.

## 4.6 Enemigo

### 4.6.1 AI Controller (AIController\_BaseEnemy)

El AI Controller es el cerebro que gobierna las acciones de los enemigos. El **AIController\_BaseEnemy** se encarga de gestionar el comportamiento del enemigo controlado mediante un árbol de comportamiento (*Behavior Tree*). Dentro de este *blueprint*, se observa lo siguiente:

- **Run Behavior Tree:** Al iniciar el evento *BeginPlay*, el controlador de IA carga y ejecuta un *Behavior Tree* asociado, que en este caso es el **BT\_SwordEnemy**. El *behavior tree* define las acciones y decisiones que tomará el enemigo durante el juego.
- **Blackboard:** El controlador también accede a un *blackboard* para almacenar información crucial para el comportamiento del enemigo. Por ejemplo, se observa cómo el controlador usa la función **Set Value as Object** para establecer al jugador como el objetivo a seguir dentro del *blackboard* bajo la clave **ActorToFollow**.

Este controlador tiene un componente esencial para la IA, que es el **BehaviorTreeComponent**, el cual se encarga de procesar el árbol de comportamiento asociado y las órdenes que derivan de él.

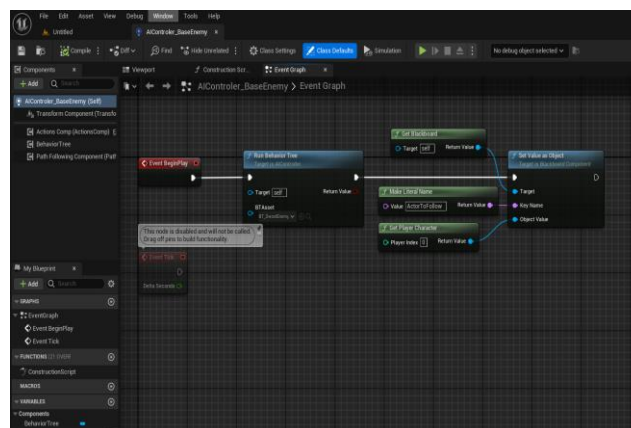


Ilustración 66: Blueprint AI controller del Base Enemy

#### 4.6.2 Behavior Tree (BT\_SwordEnemy)

El **Behavior Tree** es la estructura lógica que define cómo actúa el enemigo. Cada nodo en el árbol representa una tarea o una condición que el enemigo debe cumplir para avanzar al siguiente paso. Aquí se desglosan los principales componentes y condiciones del árbol de comportamiento:

- **Root Node:** El punto de partida para todas las decisiones de comportamiento del enemigo.
- **Selector:** Este nodo selecciona qué secuencia ejecutar en función de las condiciones del *blackboard*. Evalúa si el jugador está dentro del rango o si el enemigo debe patrullar.
- **Blackboard Based Conditions:** Estos nodos están basados en las condiciones almacenadas en el *blackboard*. Las claves más importantes son *ActorToFollow* (el jugador), *IsPlayerInRange* (si el jugador está dentro del rango de ataque), y *LastKnownPlayerLocation* (la última posición conocida del jugador). Estas condiciones determinan si el enemigo sigue atacando o si entra en estado de patrullaje.
- **Secuencias:**
  - **Follow and Attack:** Si el jugador está en rango, el enemigo se mueve hacia el jugador y lo ataca. En esta secuencia, se usan tareas como *BTT\_Attack\_Actor* (para iniciar el ataque) y *Move To* (para seguir al jugador).
  - **Patrol:** Si el jugador no está a la vista, el enemigo patrulla la zona en busca de actividad. La secuencia de patrullaje utiliza la tarea *BTT\_FindLocationToPatrol* para obtener una ubicación aleatoria en el área que el enemigo pueda recorrer.

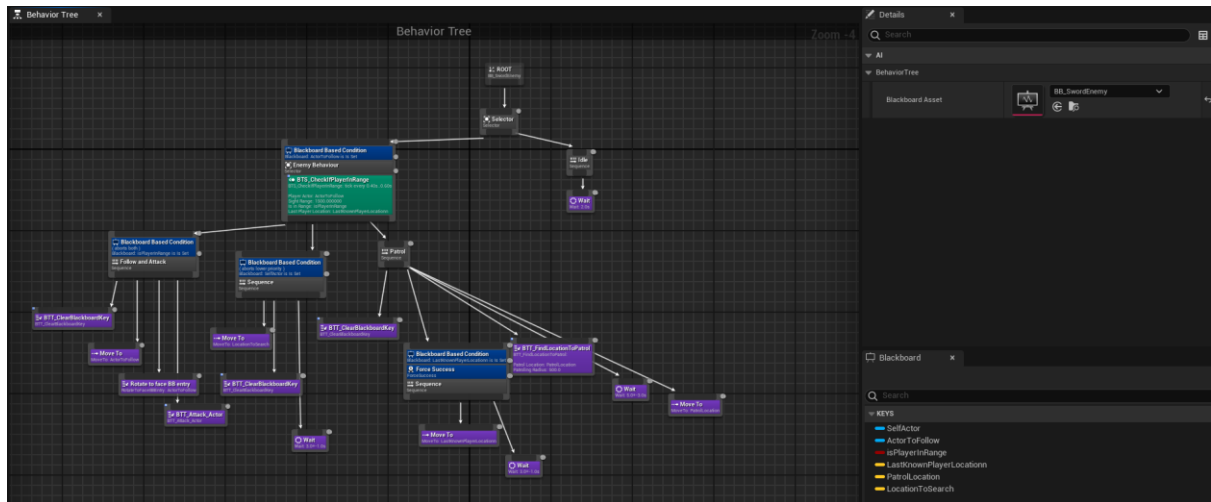


Ilustración 67: Behavior Tree Sword Enemy

#### 4.6.3 Blackboard (BB\_SwordEnemy)

El Blackboard es una especie de memoria de trabajo que usa el controlador de IA y el árbol de comportamiento para almacenar y recuperar información clave. En este caso, se utiliza para almacenar varios estados y objetivos para el enemigo.

- **Claves Principales:**

- **SelfActor:** El propio enemigo.
- **ActorToFollow:** El jugador o el objetivo al que el enemigo debe perseguir.
- **IsPlayerInRange:** Un valor booleano que indica si el jugador está dentro del rango de ataque.
- **LastKnownPlayerLocation:** La última ubicación conocida del jugador si salió de la línea de visión.
- **PatrolLocation:** Una posición en la que el enemigo patrullará si no tiene un objetivo claro.

Cada clave dentro del *blackboard* se actualiza dinámicamente dependiendo de las condiciones del juego, permitiendo que la IA cambie su comportamiento en función del entorno y las acciones del jugador.

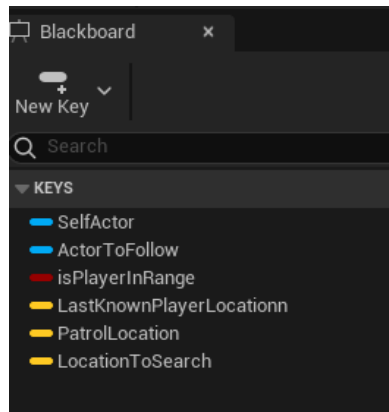


Ilustración 68: Blackboard Sword Enemy

#### 4.6.4 Tareas del Behavior Tree

Las tareas son las acciones individuales que el enemigo realiza dentro del *Behavior Tree*. Estas tareas son invocadas por los nodos de secuencia del *Behavior Tree* y gestionan la lógica específica de cada acción.

##### **BTS\_CheckIfPlayerInRange:**

Esta tarea es responsable de verificar si el jugador está dentro del rango de visión o de ataque del enemigo. Se utiliza una combinación de *traces* y coordenadas de ubicación.

- **Get Player Location:** La tarea obtiene la ubicación actual del jugador y la compara con la distancia de rango del enemigo.
- **Apply Line Trace by Channel:** Este nodo lanza un *line trace* para verificar si hay obstáculos entre el enemigo y el jugador, determinando si el jugador es visible o no.
- **Set Blackboard Value as Bool:** Dependiendo de los resultados del *trace*, esta tarea actualiza la clave `isPlayerInRange` del *blackboard*, lo que influye directamente en la próxima acción del enemigo (atacar o patrullar).

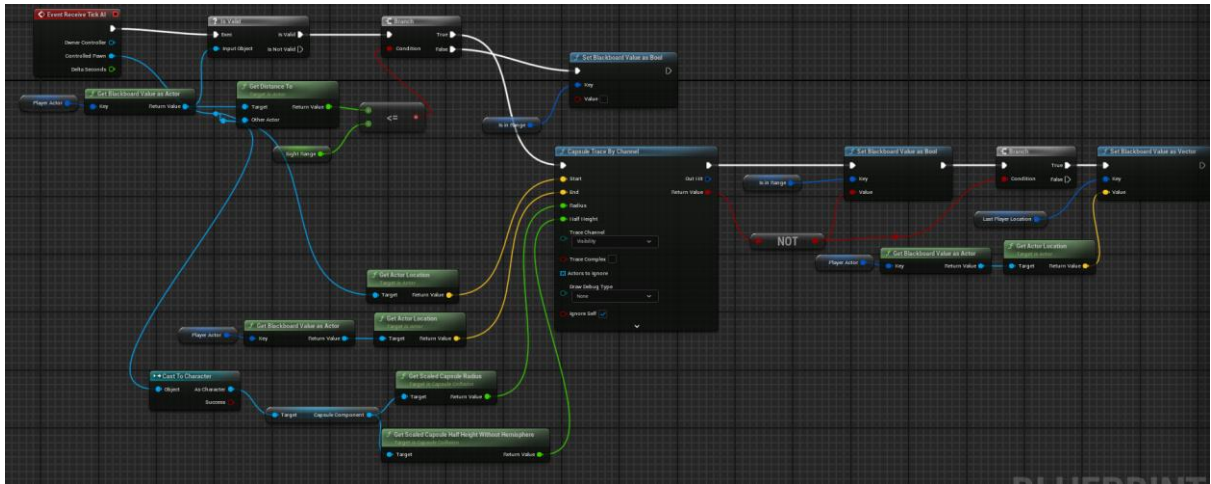


Ilustración 69: Blueprint CheckIfPlayerInRange

### BTT\_Attack\_Actor:

Esta tarea gestiona el comportamiento de ataque del enemigo. Se activa cuando el jugador está dentro del rango y la clave `IsPlayerInRange` es verdadera.

- **Send Enemy Attack:** Esta función ejecuta la animación de ataque, sincronizando la duración del ataque con la animación del enemigo.
- **Delay:** Añade una pausa temporal entre ataques para evitar que el enemigo ataque de forma continua y sin descanso.
- **Finish Execute:** Marca la tarea como completada para que el *Behavior Tree* continúe con la siguiente secuencia, ya sea seguir atacando o evaluar otras condiciones.

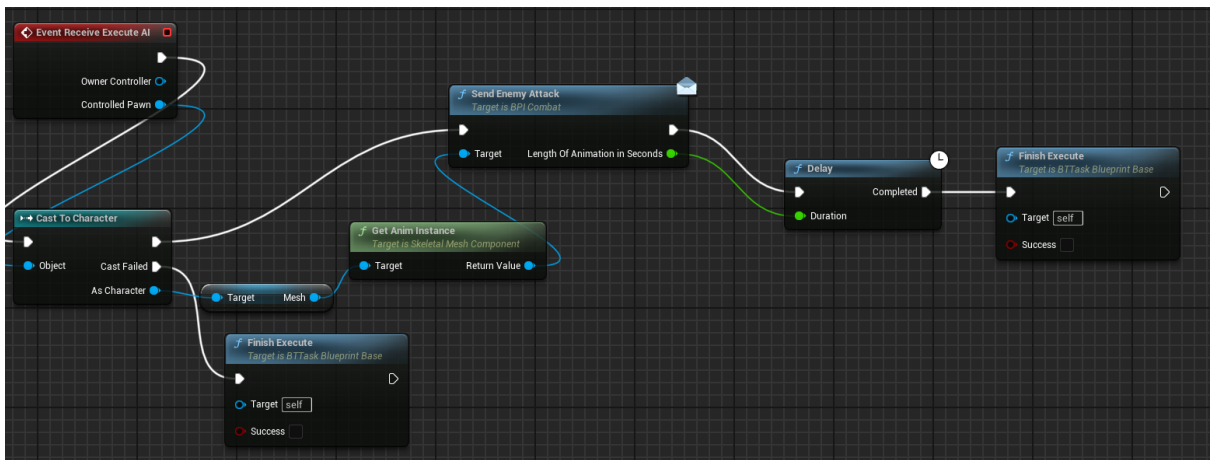


Ilustración 70: Blueprint Attack\_Actor

### BTT\_ClearBlackboardKey:

Esta tarea es simple pero crucial, ya que se encarga de limpiar valores del *blackboard*. Esto es necesario para reiniciar ciertas condiciones cuando el enemigo pierde al jugador de vista o cambia de estado (por ejemplo, dejar de seguir al jugador y pasar a patrullar).

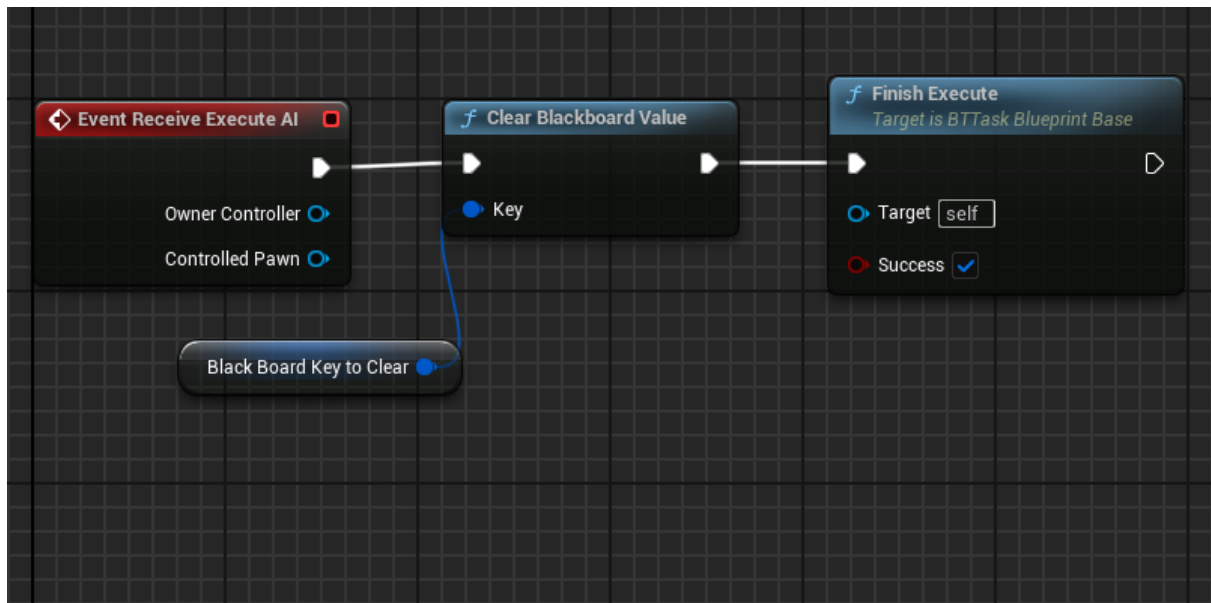


Ilustración 71: Blueprint ClearBlackboardKey

### BTT\_FindLocationToPatrol:

Esta tarea es activada cuando el enemigo entra en estado de patrullaje. Se encarga de encontrar una nueva ubicación dentro del radio de patrullaje establecido en el entorno.

- **GetRandomLocationInNavigableRadius:** Este nodo genera una ubicación aleatoria que el enemigo puede patrullar dentro de un área definida por un radio.
- **Set Blackboard Value as Vector:** Guarda esta nueva ubicación en la clave *PatrolLocation* del *blackboard*, que será utilizada por la secuencia de patrullaje para mover al enemigo a ese punto.

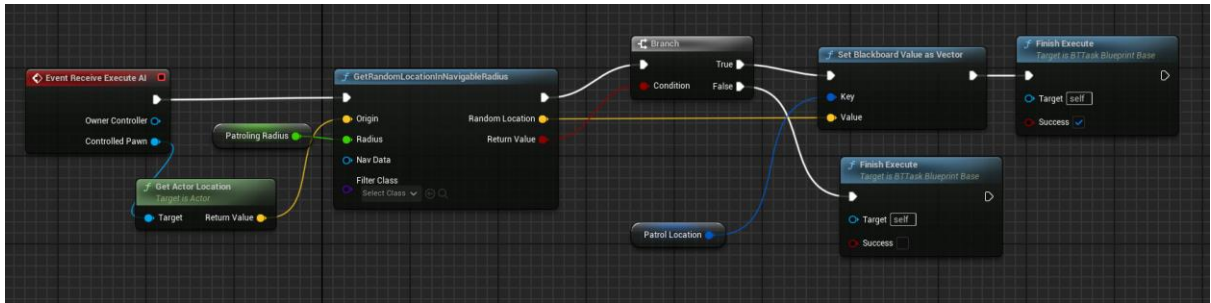


Ilustración 72: Blueprint FindLocationToPatrol

#### 4.6.5 Blueprints de Enemigos: BP\_Base\_Enemy y BP\_Sword\_Enemy

El **BP\_Base\_Enemy** actúa como clase base para todos los enemigos en el juego. Define las propiedades y comportamiento común de todos los enemigos, como recibir daño, morir y manejar colisiones.

- **Componente BPC\_CharacterInfo:** Este componente maneja la salud y la información vital del enemigo. El nodo **On Damage Received** actualiza la vida del enemigo cuando recibe un golpe, ejecutando animaciones de daño y revisando si el enemigo ha muerto.
- **On Death Event:** Cuando el enemigo muere, se activan varios nodos para desactivar colisiones, detener la simulación física y destruir al enemigo después de un breve retardo.

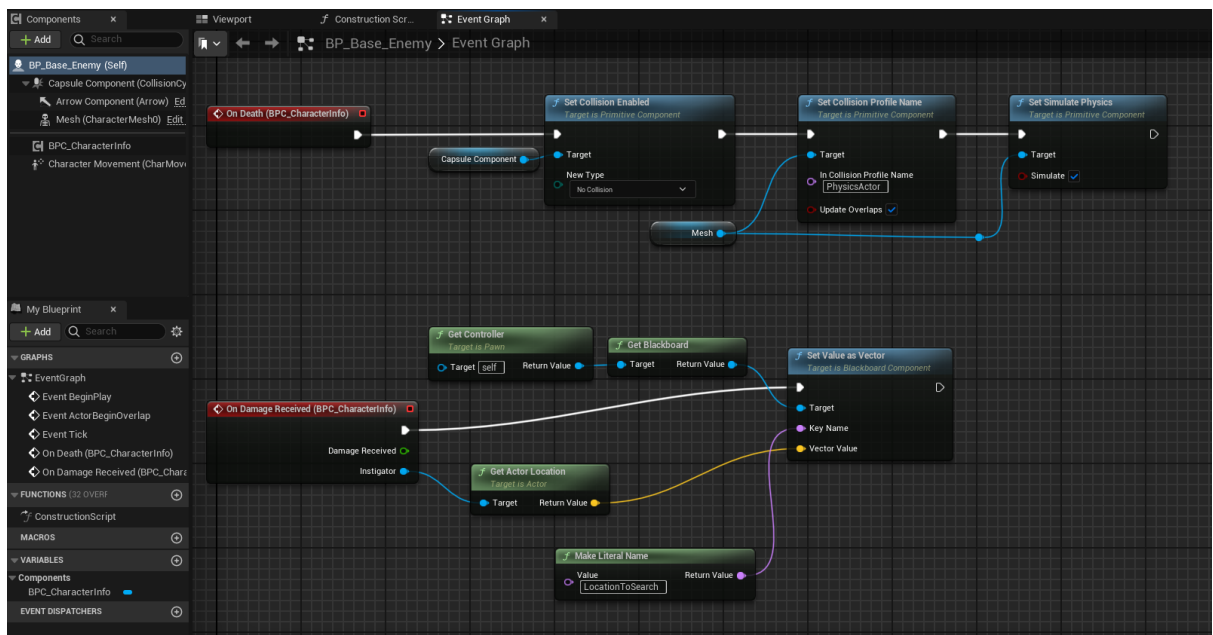


Ilustración 73: Blueprint Event Graph de BP\_Base\_Enemy

El **BP\_Sword\_Enemy** hereda del **BP\_Base\_Enemy**. Además de las funciones estándar del enemigo base, este *blueprint* maneja la lógica específica del enemigo armado con espada.

- **Componentes Específicos:**
  - **Weapon:** El enemigo tiene un componente de arma que se utiliza en su comportamiento de ataque.
- **On Damage Received:** Este evento activa una animación de daño específica para el enemigo cuando es golpeado. La animación puede incluir efectos de sangre o impactos visuales vinculados a su arma.

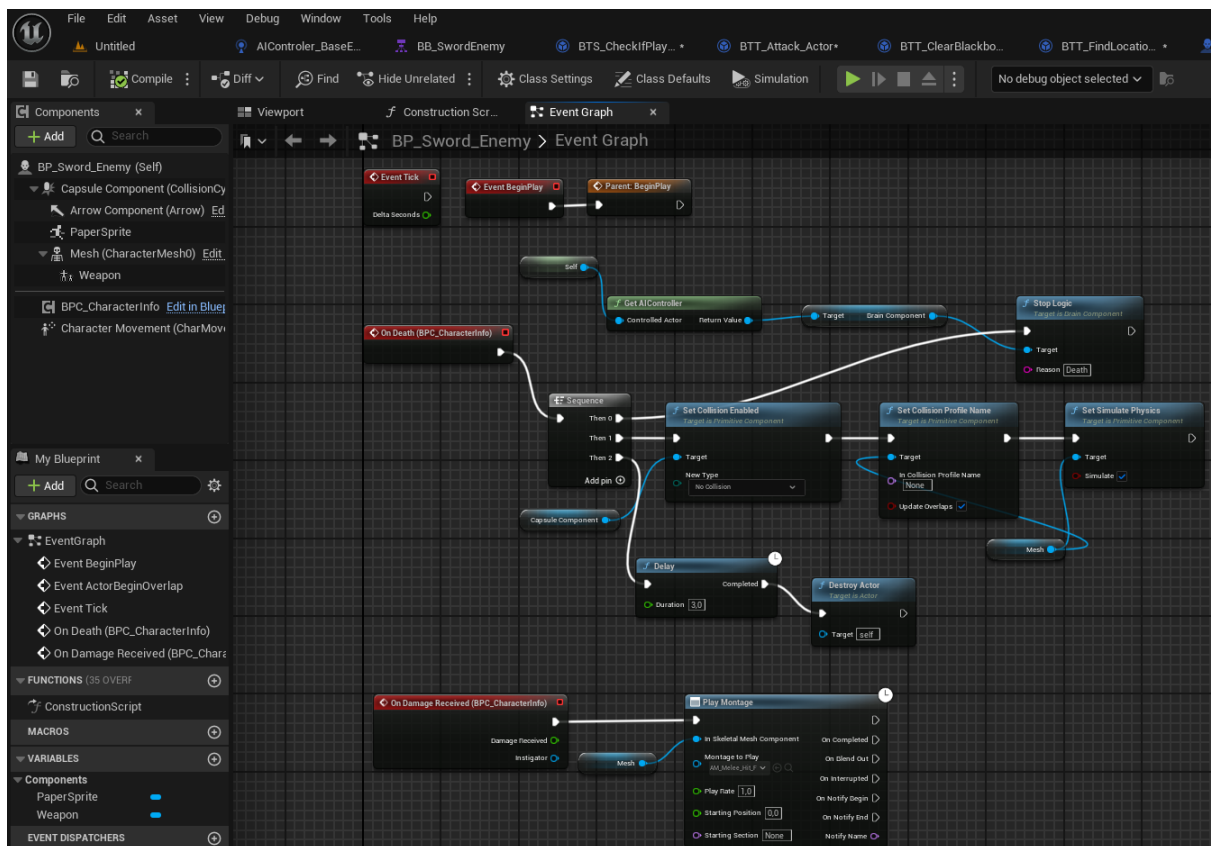


Ilustración 74: Blueprint Event Graph de BP\_Sword\_Enemy

## 4.7 NPC Convai

En la creación de los **NPCs** de Agora Island, se utilizó la herramienta **Convai**, la cual permitió desarrollar personajes interactivos con los que los jugadores pueden establecer diálogos tanto por chat escrito como mediante interacción por

voz a través del micrófono. Esta funcionalidad añade una nueva capa de inmersión al juego, ya que los jugadores pueden interactuar de manera natural con los personajes del entorno.

El proceso comenzó en la plataforma de Convai, donde se creó un personaje único al cual se le personalizó tanto su personalidad como su *backstory*. Estas características fueron diseñadas específicamente para ajustarse al contexto del mundo del juego y ofrecer diálogos que sean coherentes y enriquecedores para la narrativa. Gracias a Convai, fue posible ajustar la manera en que el *NPC* responde y se comporta en diversas situaciones, haciéndolo sentir más orgánico y realista en sus interacciones.

A lo largo del desarrollo, la integración de estos *NPCs* se realizó mediante Blueprints, lo que permitió conectar las funcionalidades del personaje con el sistema de diálogo en el juego. A continuación, explicaremos los distintos Blueprints utilizados para implementar las interacciones de los *NPCs* y cómo la herramienta Convai fue integrada en el ecosistema de "Agora Island", asegurando una interacción fluida y coherente con los jugadores.

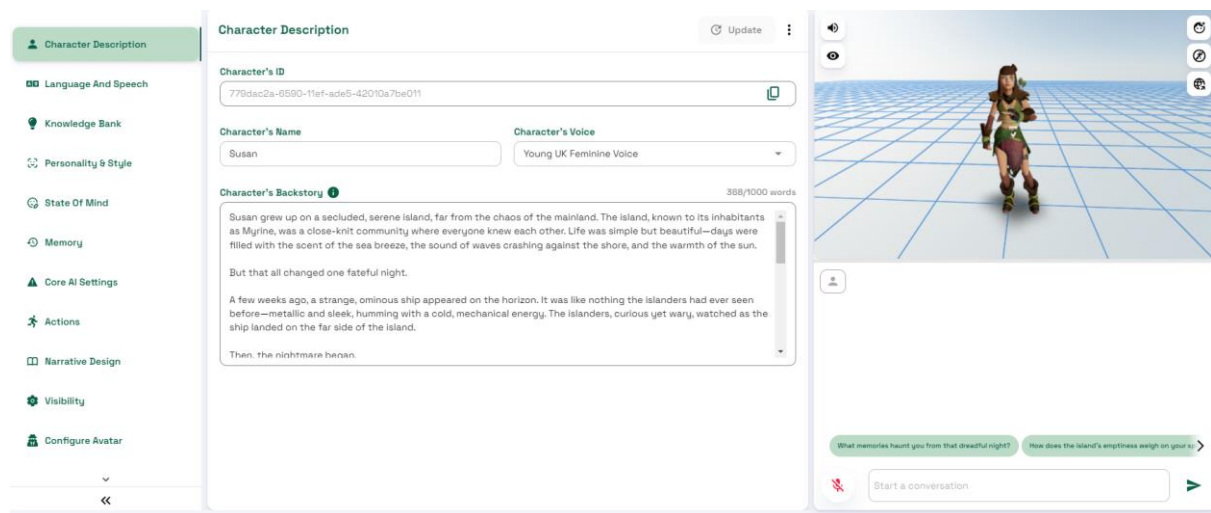


Ilustración 75: Creación NPC Convai

El siguiente *blueprint* en *Unreal Engine* está diseñado para gestionar el comportamiento de un *NPC* utilizando Convai, una herramienta de chat interactivo. A continuación, se explican los componentes, variables y el flujo de trabajo que se observa en el *blueprint*:

## Componentes y Variables:

- **ConvoSkeletalMesh:** Hace referencia a un "*Skeletal Mesh*", que representa el modelo 3D del *NPC*, animado mediante un esqueleto.
- **SkeletalMesh:** Otro componente de "*Skeletal Mesh*", que podría ser una copia o una versión configurada de manera distinta.
- **ConvaiPlayerSync:** Se utiliza para la sincronización de datos entre el *NPC* y el jugador a través del sistema Convai.

## Variables:

- **CharacterName:** Variable del tipo cadena (*string*) que contiene el nombre del *NPC*.
- **convoActive:** Variable booleana que indica si la interacción o conversación con el *NPC* está activa.

## Descripción del *Blueprint*:

1. **Event BeginPlay:** Este evento se dispara al comenzar el juego y se utiliza para inicializar los componentes y activar las funciones necesarias.
2. **ConvoIsActive = True (Set):** La primera acción del *blueprint* es activar la variable *ConvoIsActive*, configurándola como *True*. Esto significa que el *NPC* está disponible para interactuar con el jugador.
3. **ConvaiGetCharacterInfo:** Esta función obtiene información del *NPC*, como su ID, nombre y otros detalles relevantes (como idioma o avatar).
4. **Set Character Name:** Una vez obtenida la información, el nombre del personaje se asigna a la variable *CharacterName*, permitiendo que el sistema lo identifique correctamente durante la interacción.
5. **Condition check:** Se realiza una verificación para asegurar que la información del *NPC* es válida y que está listo para la interacción. Si la condición es verdadera, el flujo continúa.
6. **Player Setup:** Si la verificación es exitosa, se procede a la configuración del jugador, lo que indica que el *NPC* está listo para interactuar mediante el sistema de chat de Convai.
7. **Final Event:** El flujo finaliza con una función que completa la configuración del *NPC*, dejándolo preparado para interactuar con el jugador.

Como conclusión obtenemos que este *blueprint* establece los pasos necesarios para la configuración inicial de un *NPC*, permitiendo su interacción con el jugador a través del sistema de chat Convai.

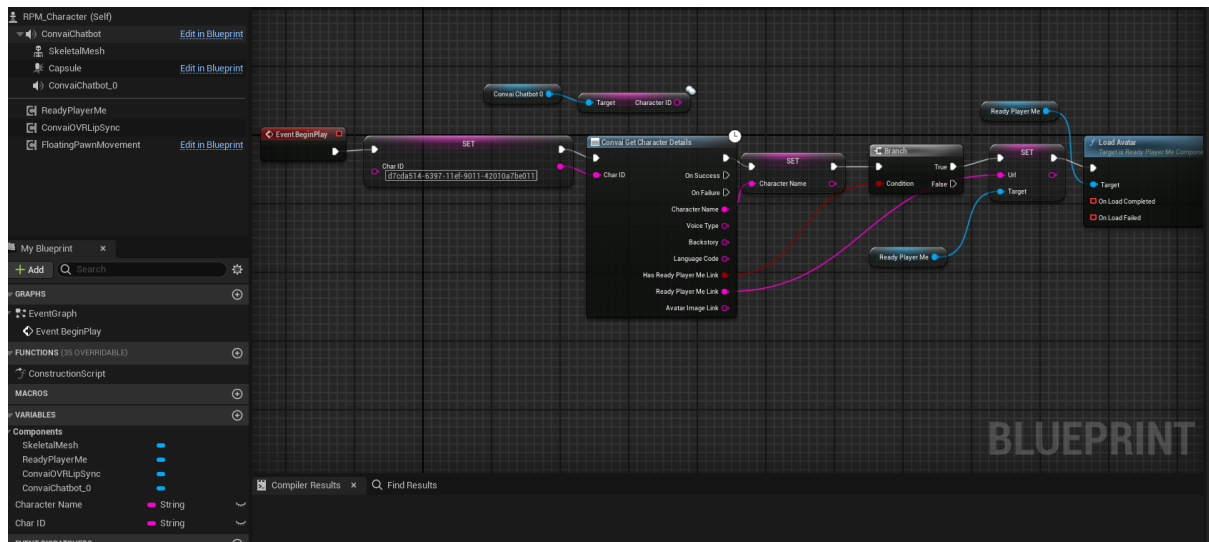


Ilustración 76: Blueprint Event Begin Play NPC Convai

## 4.8 Creación de los ataques del jugador

### 4.8.1 Espada

El sistema de la espada del jugador está desarrollado en un *blueprint* que gestiona todas las interacciones del arma con el entorno y los enemigos. Este *blueprint* maneja la detección de colisiones, la aplicación de daño, y la activación de efectos visuales y sonoros. Se estructura mediante un sistema de "*tracing*" que detecta impactos durante los ataques, junto con funciones que permiten aplicar daño de manera controlada y coherente con las acciones del jugador.

#### Variables del *Blueprint*

Las principales variables definidas en el *blueprint* de la espada incluyen las siguientes:

- **ShouldDealDamage (Boolean):** Esta variable controla si la espada está en condiciones de aplicar daño. Se activa únicamente cuando el personaje está realizando un ataque, asegurando que el daño solo se aplique durante las fases activas del combate. Esta variable es crucial para evitar

que se inflija daño cuando la espada no está en uso o cuando el personaje no está atacando.

- **PreviousSwordTraceStart/PreviousSwordTraceEnd (Vector):** Estas dos variables representan los puntos de inicio y fin de los trazos que la espada genera para detectar colisiones. Estos vectores se utilizan para calcular la trayectoria de la espada en tiempo real, permitiendo que el sistema detecte con precisión cualquier impacto con el entorno o los enemigos.
- **AttackVelocity (Vector):** La velocidad del ataque se guarda en esta variable y se usa para influir en la fuerza o magnitud del daño infligido. Un ataque más rápido o con mayor impulso puede aplicar más daño al objetivo, añadiendo realismo a la mecánica de combate.
- **SwordOwner (Actor):** Esta variable almacena una referencia al propietario de la espada, que en este caso es el personaje jugable. Esto permite que la lógica dentro del *blueprint* de la espada interactúe con otros componentes del personaje, como la gestión de estadísticas o la activación de efectos relacionados con el combate.
- **SystemTemplate (Sistema de Partículas):** Este sistema se encarga de manejar los efectos visuales como chispas, humo o sangre que se generan en el punto de impacto cuando la espada golpea una superficie o enemigo. El uso de este sistema de partículas incrementa la inmersión visual y la retroalimentación que recibe el jugador al realizar ataques.

### **Sistema de *Tracing* y Detección de Colisiones**

La detección de colisiones en este *blueprint* se gestiona a través de un sistema de *Line Trace*, que permite detectar si la espada impacta con un objeto durante el movimiento de ataque.

- **Line Trace by Channel:** Se ejecuta en cada fotograma durante un ataque, creando una línea invisible entre los puntos definidos por **PreviousSwordTraceStart** y **PreviousSwordTraceEnd**. Este sistema de *tracing* verifica si hay colisión con actores dentro de ese canal de detección (por ejemplo, enemigos o estructuras) y, de ser así, se activa el proceso de aplicación de daño.

- **Add Unique:** Este nodo asegura que la detección de colisiones sea eficiente, agregando cada objeto impactado a una lista única, lo que evita que el mismo objeto reciba múltiples golpes consecutivos en un solo ataque. Este sistema permite evitar errores como la aplicación de daño repetido innecesario durante un ataque continuo.

## Aplicación de Daño

Cuando la espada detecta una colisión válida a través del *tracing*, el sistema procede a la aplicación de daño al actor afectado. Esta lógica está diseñada para asegurar que el daño solo se aplique si la variable `ShouldDealDamage` está activa, lo cual previene la activación accidental de daño cuando la espada no está en una fase de ataque.

- **Apply Damage:** Este nodo maneja el daño infligido al enemigo una vez que se ha detectado una colisión. El cálculo de daño puede variar dependiendo de factores como la `AttackVelocity`, lo que permite que un ataque más rápido inflija mayor daño, añadiendo una capa de realismo a la mecánica de combate.
- **SwordOwner:** A través de esta referencia al actor propietario de la espada, el sistema de daño también puede actualizar las estadísticas del jugador, como el número de enemigos derrotados o activar habilidades especiales que se disparan al golpear con éxito a un enemigo.

## Efectos Visuales y Sonoros en los Impactos

Para mejorar la retroalimentación visual y sonora durante el combate, el *blueprint* activa efectos de partículas y sonidos en el momento del impacto con un objeto o enemigo.

- **Spawn Emitter at Location:** Este nodo genera efectos visuales como chispas o sangre en el punto exacto donde la espada impacta, proporcionando una sensación de impacto tangible al jugador. El sistema utiliza el *SystemTemplate* para elegir el efecto de partículas adecuado según el tipo de superficie impactada.

- **Efectos Sonoros:** Adicionalmente, se pueden activar sonidos específicos en el momento del impacto, reforzando la respuesta de las acciones del jugador mediante retroalimentación auditiva.

### **Interfaces para Gestión de Daño y Comunicación**

El *blueprint* de la espada utiliza interfaces para comunicarse de manera eficiente con otros actores del entorno, como los enemigos o el propio personaje jugador. Esto permite una interacción modular y flexible entre el arma y el resto de los sistemas de juego.

- **Interfaz de Daño:** A través de esta interfaz, la espada envía un mensaje a cualquier actor que implemente la interfaz para que reciba el daño calculado. Esto asegura que cualquier enemigo o estructura afectada por la espada pueda procesar el impacto sin necesidad de dependencias directas o referencias constantes entre *blueprints*.
- **Interfaz para Activación de Efectos:** Las interfaces también se utilizan para activar efectos visuales o de audio en otros actores, permitiendo que el sistema notifique al entorno cuando se ha producido un impacto. Esto garantiza que los efectos se reproduzcan correctamente en cada situación de combate.

### **Temporizadores y Control del Daño**

Para asegurar que el daño se aplique de manera controlada y realista, se han implementado temporizadores que limitan la frecuencia con la que la espada puede aplicar daño.

- **Timer:** El temporizador controla la ventana de tiempo durante la cual la espada puede aplicar daño tras un ataque. Esto evita que el jugador aplique daño varias veces durante el mismo golpe, ofreciendo una mecánica de combate justa y equilibrada.
- **Clear Timer:** Después de que el ataque ha concluido, el temporizador se borra, preparando el sistema para el próximo ataque del jugador. Este proceso asegura que los tiempos de recarga y el intervalo entre ataques se respeten de manera consistente, mejorando el ritmo de combate.

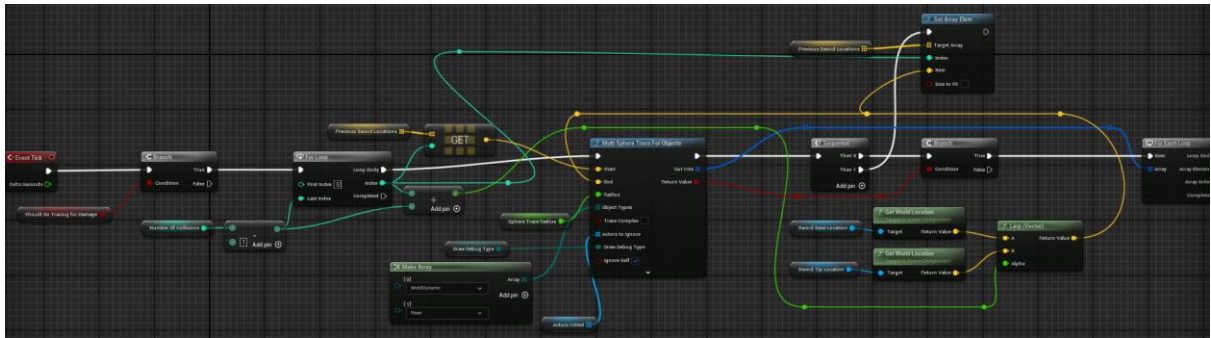


*Ilustración 77: Blueprint Event Tick BP\_Sword (Parte 1)*

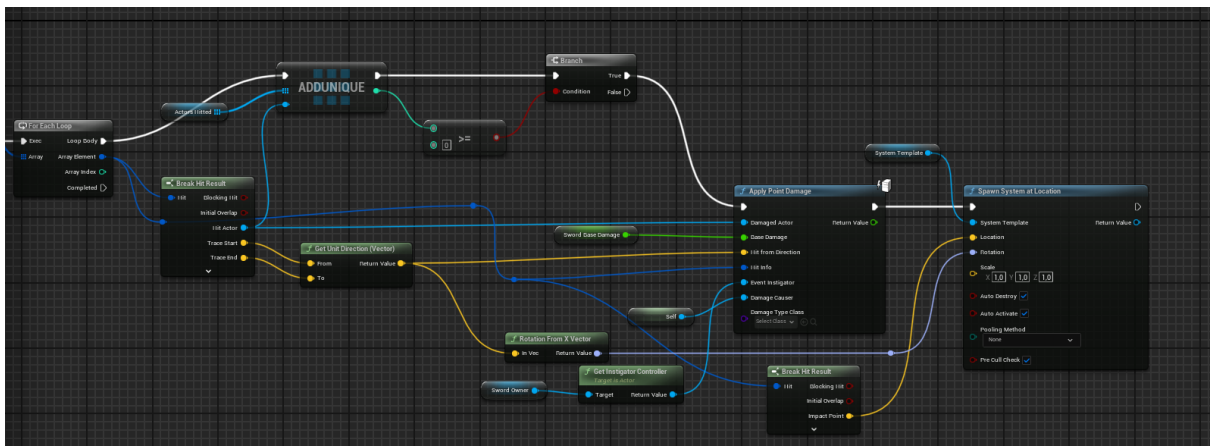


*Ilustración 78: Blueprint Event Tick BP\_Sword (Parte 2)*

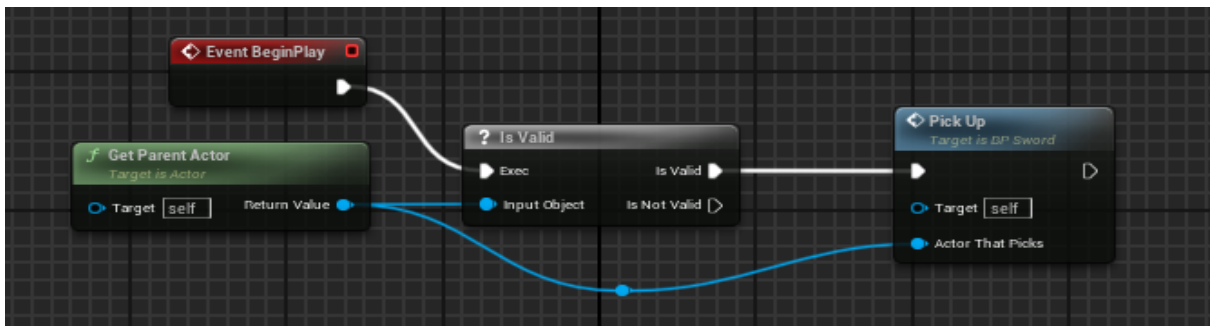


*Ilustración 79: Blueprint Event Begin Play BP\_Sword*

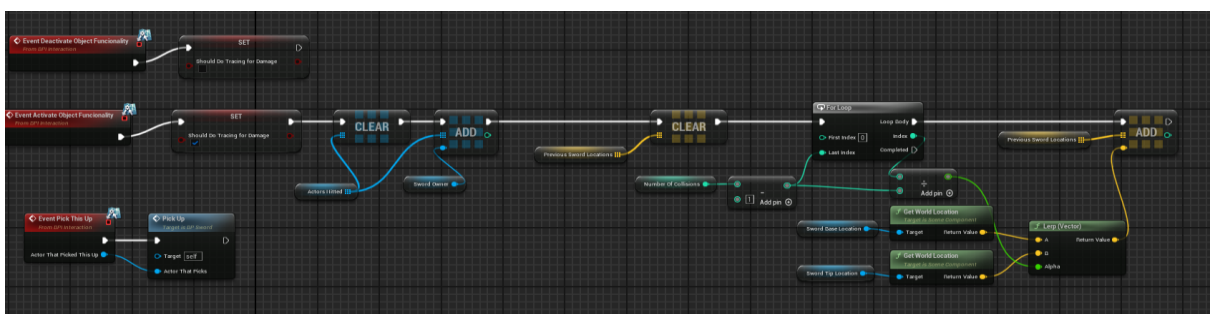


*Ilustración 80: Blueprint Funciones BP\_Sword*

## 4.8.2 Rayo

El **BP\_Projectile** es el *blueprint* base que controla el comportamiento general de un proyectil en el juego. Este tipo de *blueprint* maneja la lógica para proyectiles básicos como la velocidad, el daño, la colisión y los efectos visuales. En la jerarquía del sistema este *blueprint* actúa como una **clase padre** de la que heredan otros *blueprints* más específicos, como el **BP\_Magic\_Projectile**.

### Componentes

- **Sphere**: Es el componente de colisión principal que detecta cuándo el proyectil entra en contacto con otros actores. Este componente es crucial porque define el área de colisión, permitiendo que el sistema de daño se active cuando el proyectil toca un enemigo u objeto en el entorno.
- **ProjectileMovement**: Este componente es responsable de mover el proyectil a lo largo de su trayectoria. Controla la velocidad, la aceleración y la dirección del proyectil, asegurando que viaje de manera coherente en el espacio del juego.

### Variables

- **Projectile Damage (Float)**: Almacena el valor del daño que el proyectil aplicará cuando impacte contra un actor. Esta variable se pasa al sistema de daño para calcular la cantidad de vida que se debe reducir del actor objetivo.

### Eventos y Nodos

- **Event Begin Play**: Este nodo inicializa el comportamiento del proyectil. Dentro de este evento, se define que el proyectil debe ignorar al actor que lo disparó, lo cual evita que colisione con el personaje que lanzó el proyectil. También se ejecutan efectos visuales, como la sacudida de cámara al disparar el proyectil, mejorando la retroalimentación visual para el jugador.
- **Line Trace by Channel**: Este nodo realiza un trace desde la posición del proyectil para detectar si ha impactado algo durante su trayectoria. Al

detectar un impacto, activa la función **Apply Point Damage**, que aplica daño al actor impactado.

- **Apply Point Damage**: Este nodo maneja la lógica para aplicar daño al actor que ha sido impactado por el proyectil. Utiliza la variable **Projectile Damage** para determinar cuánto daño aplicar y se encarga de comunicarse con los actores que implementen una interfaz de daño.
- **Spawn System at Location**: Este nodo activa un sistema de partículas en el punto de impacto. Cuando el proyectil colisiona, genera efectos visuales como explosiones o chispas, proporcionando retroalimentación visual.

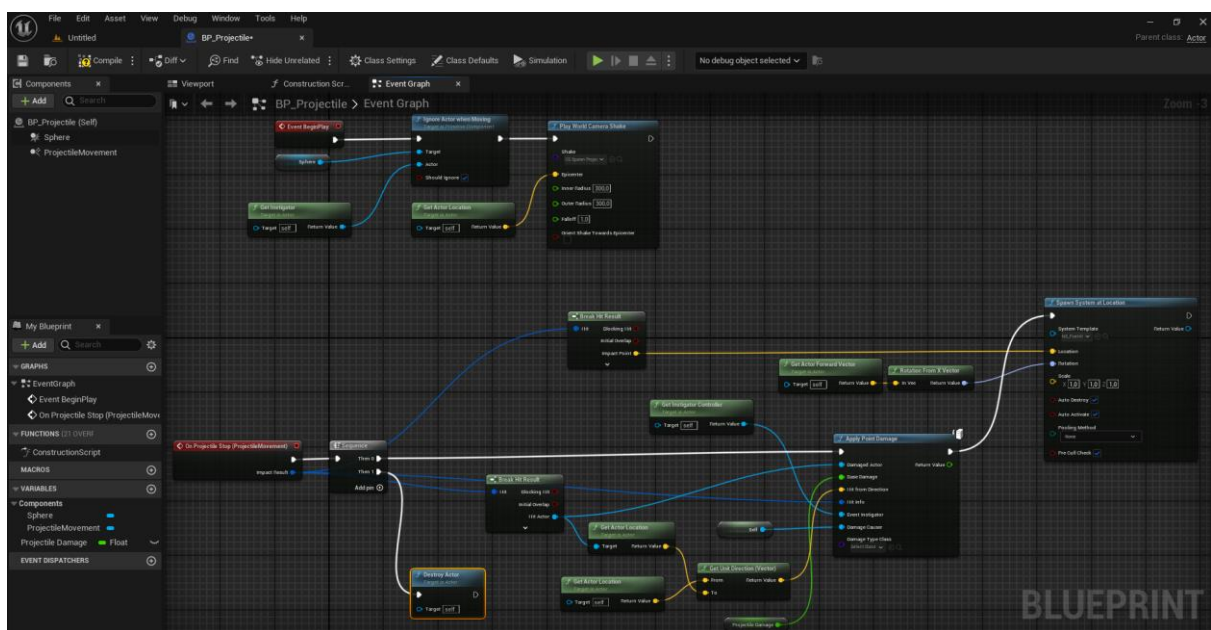


Ilustración 81: Blueprint BP\_Projectile

El **BP\_Magic\_Projectile** es una clase hija de **BP\_Projectile**, lo que significa que hereda la funcionalidad básica del proyectil, pero agrega o sobrescribe algunos comportamientos específicos. En este caso, este *blueprint* maneja las particularidades de un proyectil de magia o rayo.

## Componentes

- **NS\_Projectile**: Este componente es un sistema de partículas que representa el rayo o hechizo que se lanza en el juego. Añade un efecto visual a la trayectoria del proyectil, diferenciándolo de un proyectil normal como una bala o flecha.

- **Sphere:** Este es el mismo componente que vimos en **BP\_Projectile**, actuando como la base para la detección de colisiones.
- **ProjectileMovement:** Se hereda de **BP\_Projectile**, pero puede ser ajustado para que la velocidad o trayectoria del **BP\_Magic\_Projectile** sea diferente al del proyectil base.

## Eventos y Nodos

- **Event Begin Play:** Aquí se configura la lógica inicial para el proyectil mágico. Similar a **BP\_Projectile**, el **Magic\_Projectile** también debe ignorar al actor que lo dispara, utilizando el nodo **Ignore Actor when Moving** para evitar colisiones con el personaje que lanzó el hechizo.
- **Get Instigator:** Este nodo obtiene al actor que lanzó el proyectil. El instigador es importante para saber quién disparó el proyectil, de modo que el daño o los efectos visuales se atribuyan correctamente al jugador o al enemigo que lanzó el hechizo.

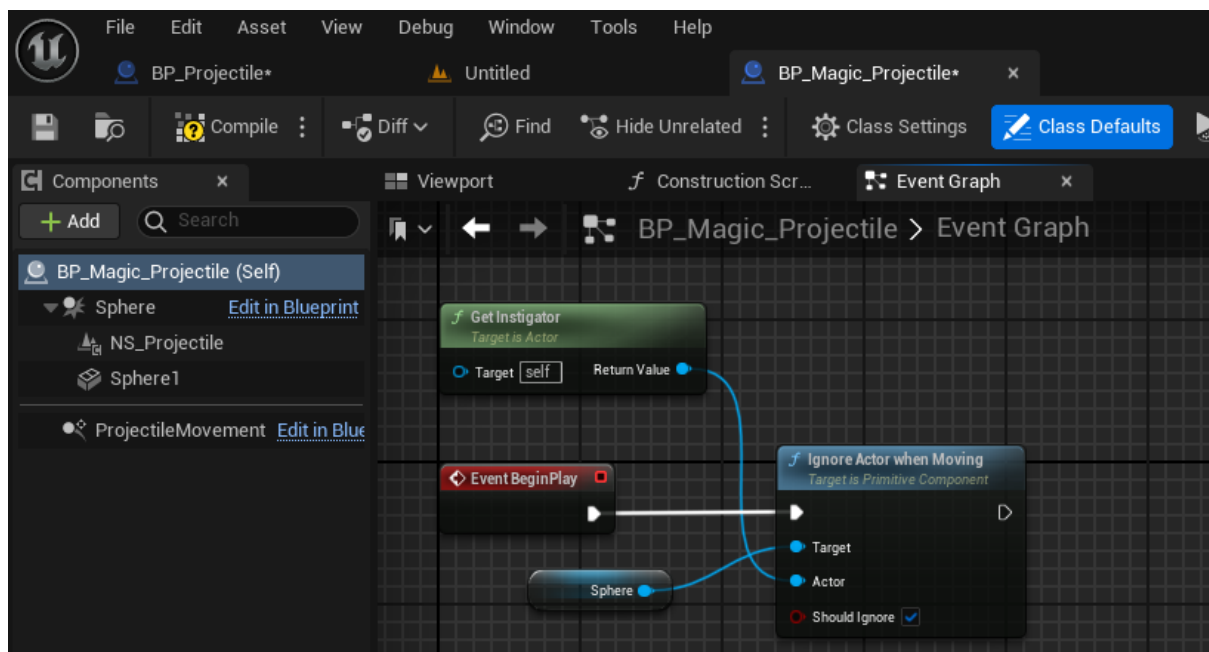


Ilustración 82: Blueprint BP\_Magic\_Projectile

El **BP\_Spell** actúa como el *blueprint* que gestiona el lanzamiento del hechizo o la activación de la magia. A diferencia de los anteriores, este *blueprint* gestiona la lógica de invocar o crear el proyectil mágico en el mundo del juego. Esto puede incluir la interacción del jugador para activar el hechizo y disparar el rayo.

## Componentes

- **Spellowner (Actor):** Esta variable almacena una referencia al actor que es el propietario del hechizo, normalmente el personaje jugador que lanza el hechizo. Esta referencia es importante porque el hechizo necesita saber quién lo ha activado, ya que la interacción entre el hechizo y otros objetos o actores dependerá del contexto de quién lo lanzó.

## Eventos y Nodos

- **Event Activate Object Functionality:** Este evento se activa cuando el hechizo es lanzado o invocado. Este nodo está conectado a una acción del jugador que dispara el hechizo, como presionar un botón para lanzar un ataque de rayo.
- **Cast to Pawn:** Este nodo se asegura de que el actor que está activando el hechizo sea un personaje (*Pawn*). Si la activación proviene de otro tipo de actor, el nodo fallará. Esto es útil para evitar que otros objetos no controlados por el jugador activen el hechizo.
- **Get Actor Transform:** Este nodo obtiene la posición y rotación actuales del jugador o del actor que está lanzando el hechizo. Esto asegura que el proyectil que se genera comience en la posición correcta, frente al jugador, en lugar de aparecer en una posición aleatoria del nivel.
- **SpawnActor BP\_Magic\_Projectile:** Este nodo es responsable de generar el proyectil mágico en el mundo del juego. Utiliza la clase *BP\_Magic\_Projectile* y la crea en la posición obtenida a través de *Get Actor Transform*. Además, el nodo *Collision Handling Override* se utiliza para ignorar colisiones en el momento de la creación del proyectil, asegurando que no se quede atascado o que no colisione con el jugador que lo lanzó.

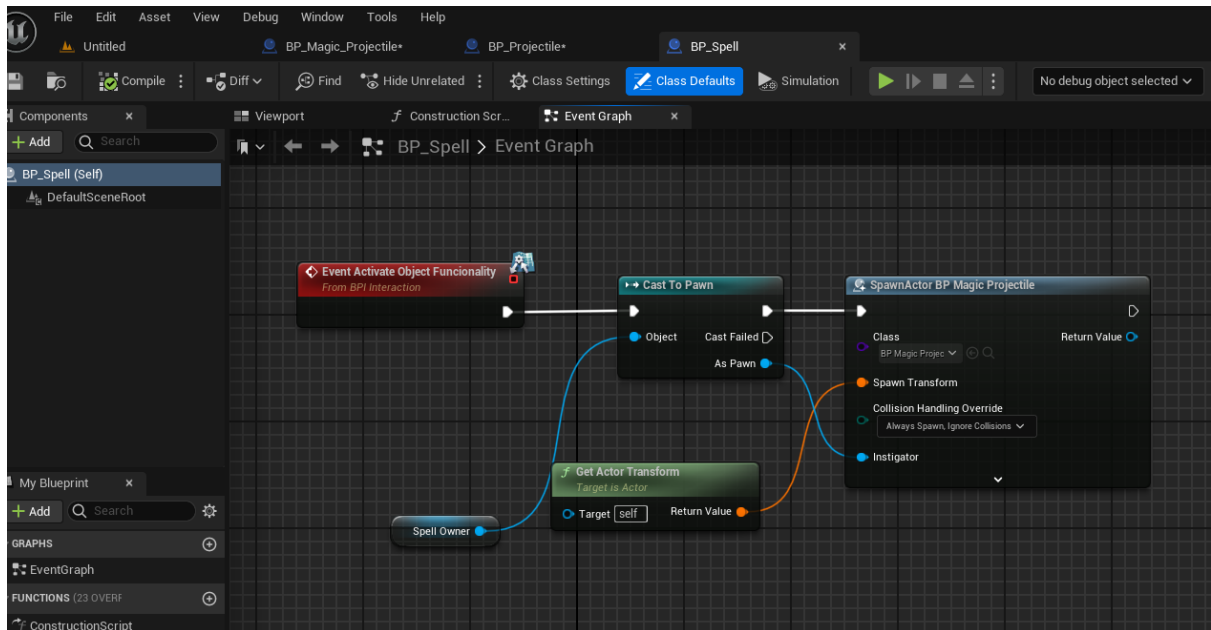


Ilustración 83: Blueprint BP\_Spell

Los tres *blueprints* se relacionan de la siguiente manera:

1. **BP\_Spell** es el *blueprint* encargado de activar el lanzamiento del hechizo o proyectil mágico. Al activarse, detecta al jugador que lanza el hechizo y usa **SpawnActor** para generar una instancia del **BP\_Magic\_Projectile** en el mundo del juego. En este momento, el sistema también asegura que el proyectil no colisione inmediatamente con el jugador que lo lanzó, utilizando la referencia al **Spellowner**.
2. **BP\_Magic\_Projectile**, siendo un hijo de **BP\_Projectile**, hereda la funcionalidad básica del comportamiento del proyectil. Sin embargo, añade características adicionales como efectos visuales de magia (partículas o sonidos) y se comporta de manera diferente en términos de velocidad o efectos al impactar. El **BP\_Magic\_Projectile** es el objeto que se mueve por el nivel una vez lanzado.
3. **BP\_Projectile** actúa como la base de comportamiento común de todos los proyectiles, gestionando las colisiones, trayectorias y la aplicación de daño. Define el movimiento a través de **ProjectileMovement** y utiliza *tracing* para detectar impactos con otros actores del nivel. Cuando el proyectil impacta, se ejecutan efectos visuales y se aplica daño a los actores afectados.

## 4.9 Efectos visuales

### 4.9.1 Texturas

Las texturas son la base visual de muchos de los efectos que se han implementado en el proyecto. A continuación, se detalla el propósito de las texturas y cómo se integran en los materiales y los sistemas de partículas.

Descripción de las Texturas:

- **T\_Fire**: Esta textura se utiliza para representar efectos de fuego o llamas. Es una textura base que se emplea en materiales relacionados con ataques o impactos de fuego, como el efecto **FireHit** en Niagara.
- **T\_FlareHigh** y **T\_FlareRadial**: Texturas para efectos de destello o explosiones de luz. Estas texturas se usan principalmente en los materiales de *flare* o explosiones rápidas, proporcionando un brillo radial, como en el caso de las explosiones de magia o fuego.
- **T\_Noise** y **T\_Noise02**: Estas texturas de ruido se usan para dar un efecto de distorsión o aleatoriedad.
- **T\_SparksFire**: Textura empleada en los efectos de chispas, particularmente útil para impactos con elementos de fuego o para simular chispas cuando el metal golpea una superficie.
- **T\_SwordGradient**: Se utiliza para crear el rastro que deja la espada cuando se balancea, lo que da una sensación de velocidad y poder durante un ataque.

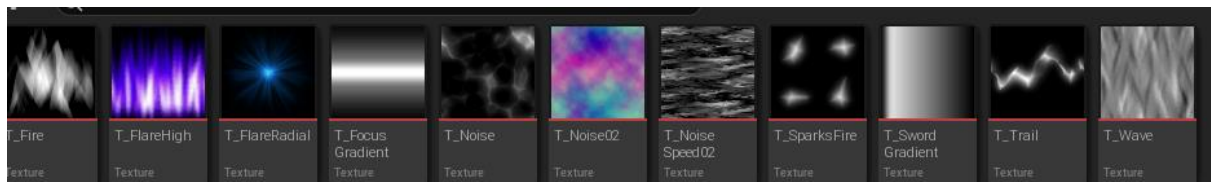


*Ilustración 84: Texturas utilizadas para Efectos visuales*

Estas texturas se combinan con los materiales para crear efectos visuales más complejos y detallados.

## 4.9.2 Materiales

Los materiales en *Unreal Engine* son el siguiente paso tras las texturas, donde se combinan múltiples propiedades para generar un efecto visual coherente. A continuación, describo los materiales y cómo interactúan con las texturas.

Descripción de los Materiales:

- **M\_EnemyHitScreen:** Este material es el encargado de generar un efecto visual en la pantalla del jugador cuando se golpea a un enemigo. En este caso, el material está configurado para generar un color rojo degradado, simulando el impacto y resaltando el golpe en la pantalla. Utiliza la textura **T\_Noise02** y una serie de multiplicaciones y *panning* para generar movimiento en la textura.
- **M\_FireNoise:** Este material toma la textura de fuego y ruido para simular un efecto de fuego en movimiento. A través de nodos de distorsión y un gradiente de opacidad, crea un efecto que simula el movimiento caótico de las llamas.
- **M\_SlashHit** y **M\_SwordTrace:** Estos materiales están diseñados para representar impactos de espada y rastros de ataques. **M\_SwordTrace** utiliza un gradiente para crear una estela de movimiento cuando el personaje realiza un ataque con la espada. Estos materiales también pueden tener transparencias y efectos de distorsión que mejoran la dinámica visual del golpe o corte.
- **M\_Magic\_Trail:** Este material está asociado a efectos visuales mágicos o proyectiles mágicos, proporcionando una estela visible detrás del proyectil. Utiliza gradientes de opacidad y texturas de ruido para dar una sensación etérea y cambiante.

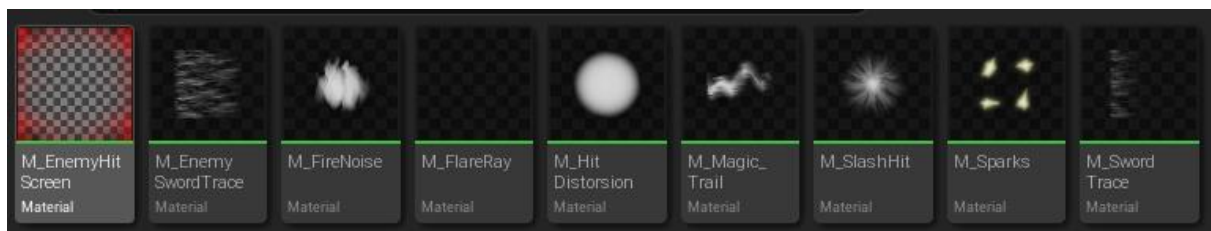


Ilustración 85: Materiales creados para el proyecto

## Efecto Visual: M\_EnergyHitScreen

El material `M_EnergyHitScreen` está diseñado para representar el daño causado a los enemigos en la pantalla del jugador, usando varios nodos para mezclar texturas y manipular el color. Este material utiliza `VectorToRadialValue` para convertir las coordenadas UV en un valor radial, lo que crea el efecto circular en el centro del golpe.

Nodos Principales:

- **VectorToRadialValue:** Este nodo convierte las coordenadas de textura en un valor radial, lo que permite crear un gradiente circular desde el centro del golpe.
- **Panner:** Controla el desplazamiento de la textura de ruido, dando la sensación de movimiento dentro del golpe, creando un impacto dinámico en la pantalla.
- **Texture Sample:** Las texturas base como `T_Noise02` y `T_FlareRadial` se usan en conjunto para añadir complejidad al efecto.
- **Multiplicaciones de Color:** Los valores de multiplicación combinan diferentes texturas y colores para obtener el tono final rojo oscuro que se muestra en la pantalla al golpear al enemigo.

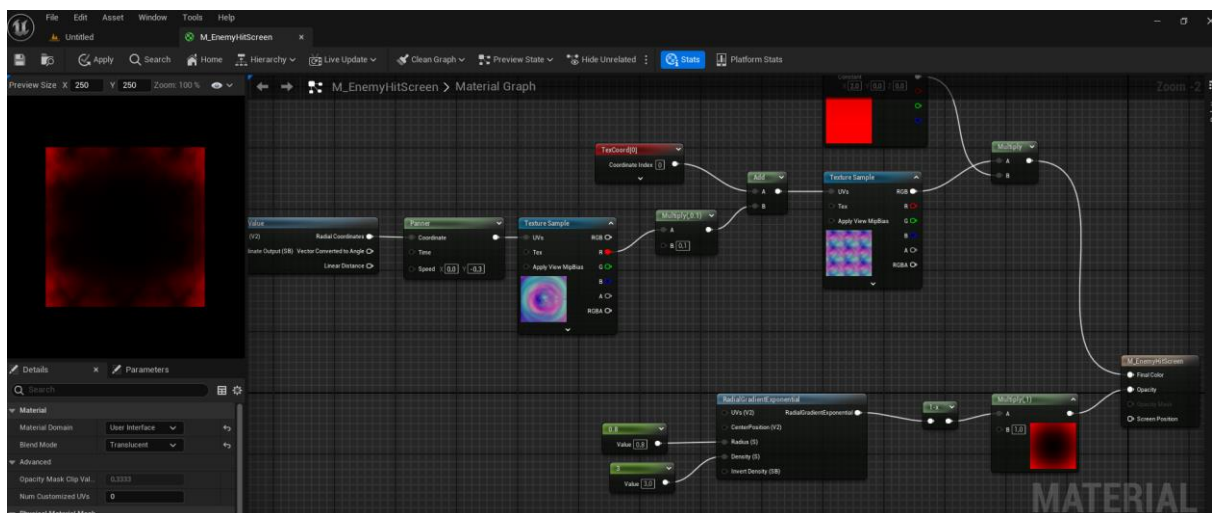


Ilustración 86: Blueprint `M_EnergyHitScreen`

### 4.9.3 Sistemas Niagara

Los sistemas **Niagara** son los que gestionan la simulación y renderizado de partículas en tiempo real. En las imágenes se observa el uso de varios sistemas Niagara para manejar diferentes tipos de efectos visuales como impactos de fuego, trazas de proyectiles y efectos de magia.

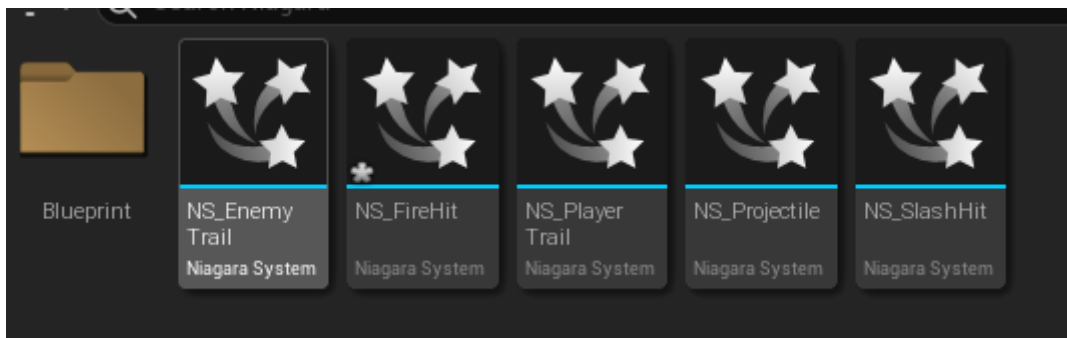


*Ilustración 87: Niagara System creados para el proyecto*

#### **NS\_FireHit:**

Este sistema Niagara es responsable de generar el efecto de impacto de fuego en el punto donde se realiza un ataque. Usa varios emisores de partículas para generar un efecto realista de chisporroteo y destellos de fuego cuando un proyectil o ataque golpea un objetivo.

- **SimpleSpriteBurst:** Este emisor genera un estallido instantáneo de partículas (en este caso, pequeñas llamas o destellos) al momento del impacto.
- **FireNoise:** Este emisor de partículas agrega una variabilidad en la dirección y el tamaño de las partículas de fuego, proporcionando un movimiento más natural y orgánico.

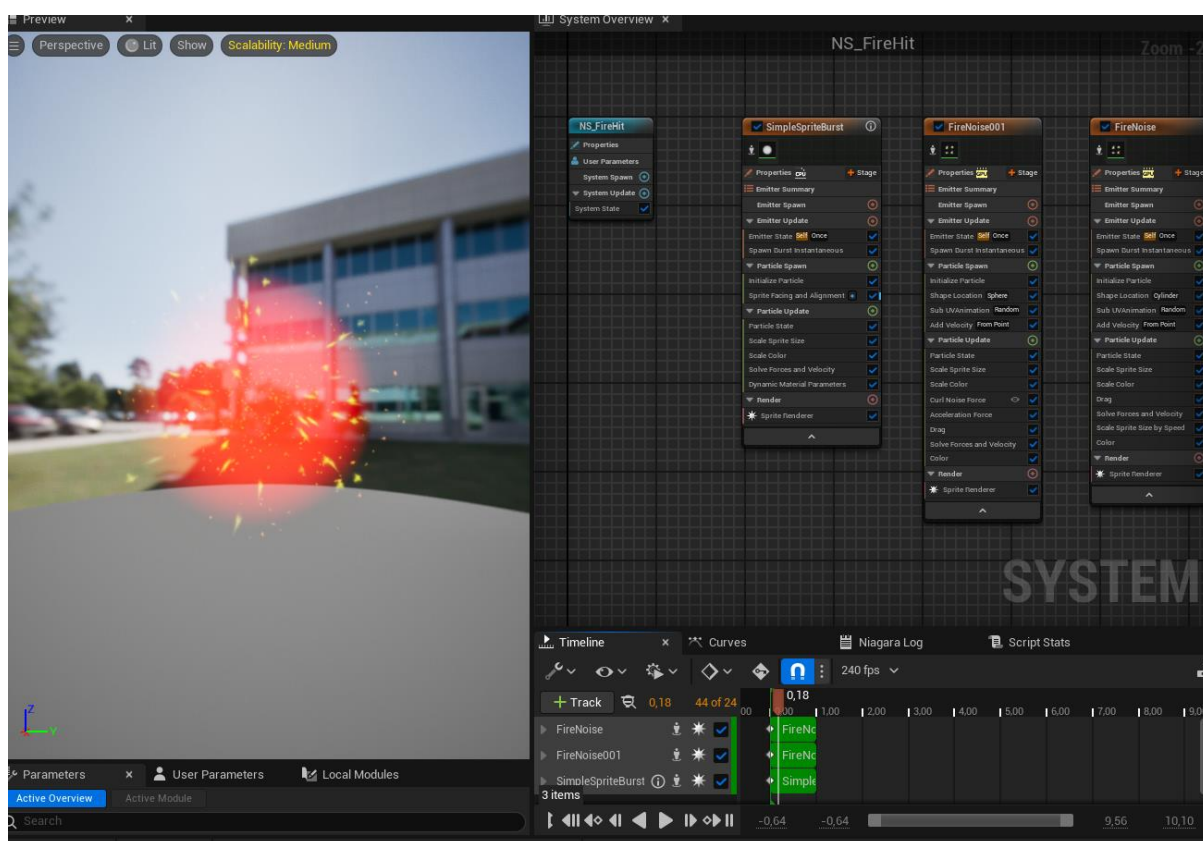


Ilustración 88: NS\_FireHit

#### NS\_Projectile y NS\_SlashHit:

Estos sistemas Niagara están diseñados para manejar los efectos visuales de proyectiles y cortes. **NS\_Projectile** genera la estela visible que sigue al proyectil, mientras que **NS\_SlashHit** genera el impacto visual cuando el arma entra en contacto con un enemigo o superficie.

- **RibbonEmitter**: Estos sistemas usan emisores de cintas (*ribbon emitters*) para crear los rastros que dejan los proyectiles o las armas en movimiento, dando una sensación de velocidad.

#### 4.9.4 Blueprints de Efectos Visuales

Finalmente, los blueprints proporcionan la lógica para implementar estos efectos visuales dentro del juego. En particular, se observan dos blueprints principales:

### BP\_VFXTrail:

Este *blueprint* está diseñado para generar un efecto de rastro visual cuando un objeto (como una espada o proyectil) se mueve rápidamente. Utiliza la lógica de eventos como **Received Notify Tick** para actualizar la posición de las partículas en cada *frame*.

- **Get Socket Location:** Obtiene la posición actual de un socket (un punto en el esqueleto del personaje o arma) para que el sistema de partículas siga el movimiento del objeto.
- **Set Vector Parameter:** Ajusta las posiciones inicial y final del efecto de rastro (como una estela de espada) para que las partículas se muevan de manera fluida y coherente con el objeto que las genera.

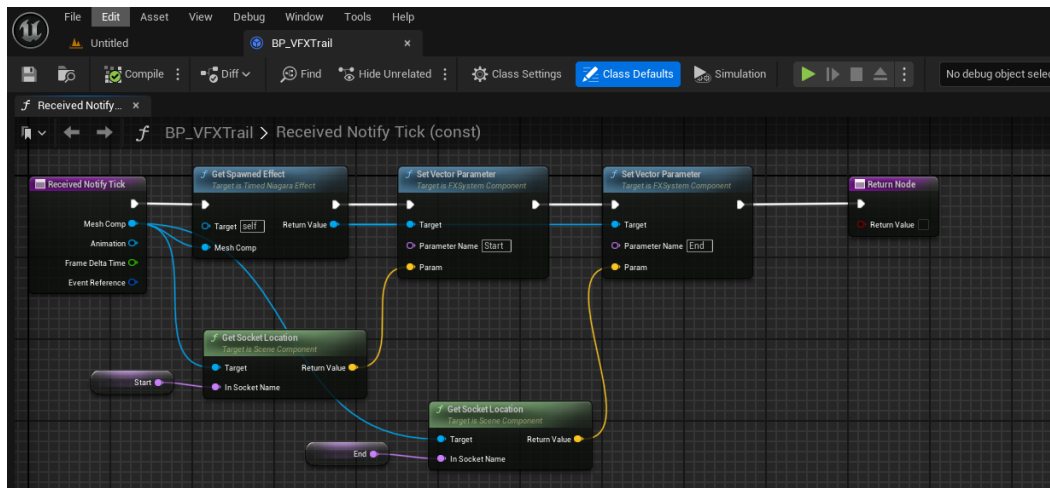


*Ilustración 89: Blueprint VFX Trail*

### Niagara Module Script:

Este script dentro de un sistema Niagara controla la posición y el comportamiento de las partículas a lo largo de su trayectoria. Utiliza parámetros vectoriales para determinar los puntos de inicio y fin de un efecto, como una estela o un impacto.

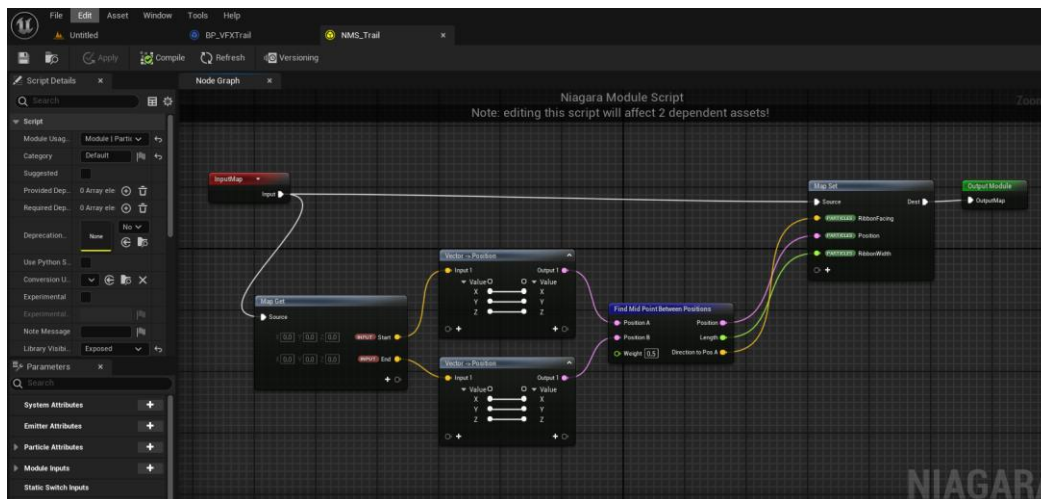


Ilustración 90: Blueprint Niagara Module Script

## 4.10 Animaciones

### 4.10.1 Animación personaje

El *blueprint* del personaje en primera persona (**First Person Character**) incluye un conjunto de acciones y eventos que definen cómo interactúa el jugador con el entorno y cómo se realizan las acciones dentro del juego.

- **Input Mapping Context e Input Action:** Estos *blueprints* gestionan las entradas del jugador, asignando funciones específicas a las teclas o botones. Esto permite realizar acciones como saltar, moverse, atacar, o utilizar armas de una forma sencilla y directa.
- **Event Begin Play:** En esta sección del *Blueprint*, se inician múltiples procesos cuando comienza el juego. Un ejemplo es la asignación de controladores de entrada o la activación de HUD para mostrar información en pantalla.
- **Mesh and Skeletal Components:** El componente visual del personaje está controlado por un esqueleto que responde a animaciones configuradas para el personaje principal. El *BP* del *Mesh* del personaje se enlaza con los eventos de acción, como ataques, que actualizan las animaciones y efectos visuales.
- **Interfaces:** Las interfaces permiten que los *Blueprints* se comuniquen entre sí. En este caso, se puede implementar una interfaz para gestionar la interacción entre el personaje y objetos como armas o habilidades.

#### 4.10.1.1 Animaciones (Main Character y Sword Enemy)

El proyecto incluye múltiples sistemas de animación para el personaje principal y el enemigo con espada.

- **Blend Spaces (BS\_Locomotion):** El *Blend Space* permite transiciones suaves entre diferentes animaciones basadas en variables como la velocidad y la dirección. Esto se utiliza principalmente para la locomoción del personaje, como correr o caminar.
- **Animation Blueprint (ABP\_TFG\_PersonCharacter, ABP\_Sword\_Enemy):** El *Animation Blueprint* se encarga de gestionar el estado de las animaciones del personaje y enemigo. Utiliza variables como "Speed" o "In Combat" para determinar qué animaciones deben reproducirse en cada momento.
- **Montajes de Animación (AM\_SwordEnemy\_Attack, AM\_MainCharacter\_SwordAttack):** Los montajes de animación permiten reproducir secuencias específicas de animación, como un ataque con espada. Estos montajes están sincronizados con la activación de colisiones y los efectos visuales para asegurar que el ataque se vea y se sienta fluido.

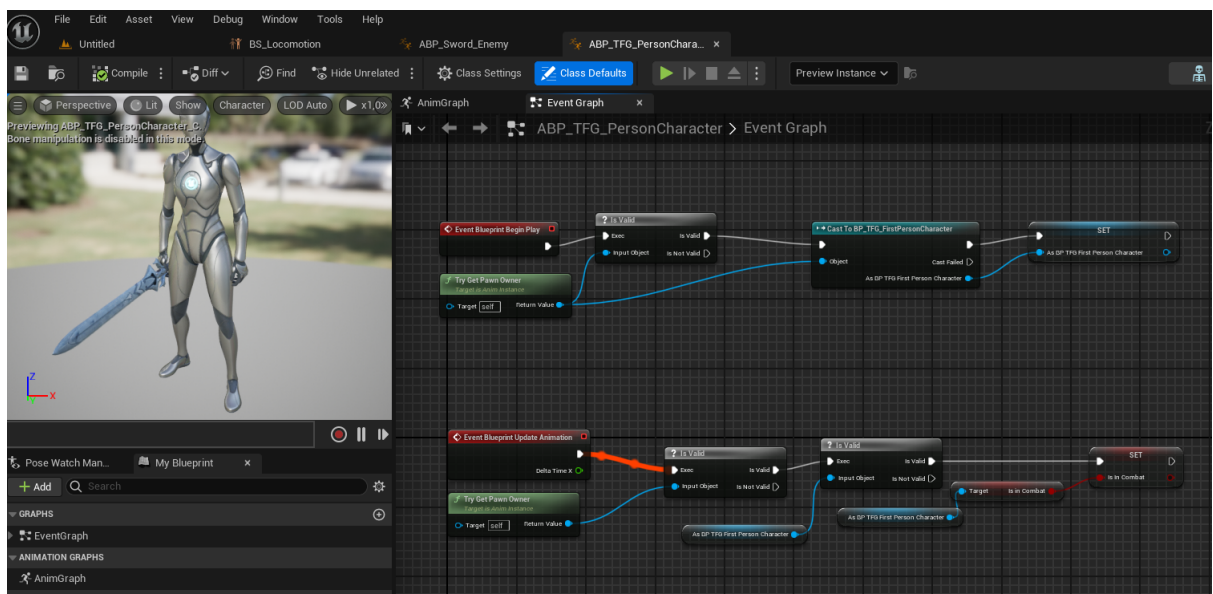


Ilustración 91: Animation Blueprint PersonCharacter (Parte 1)

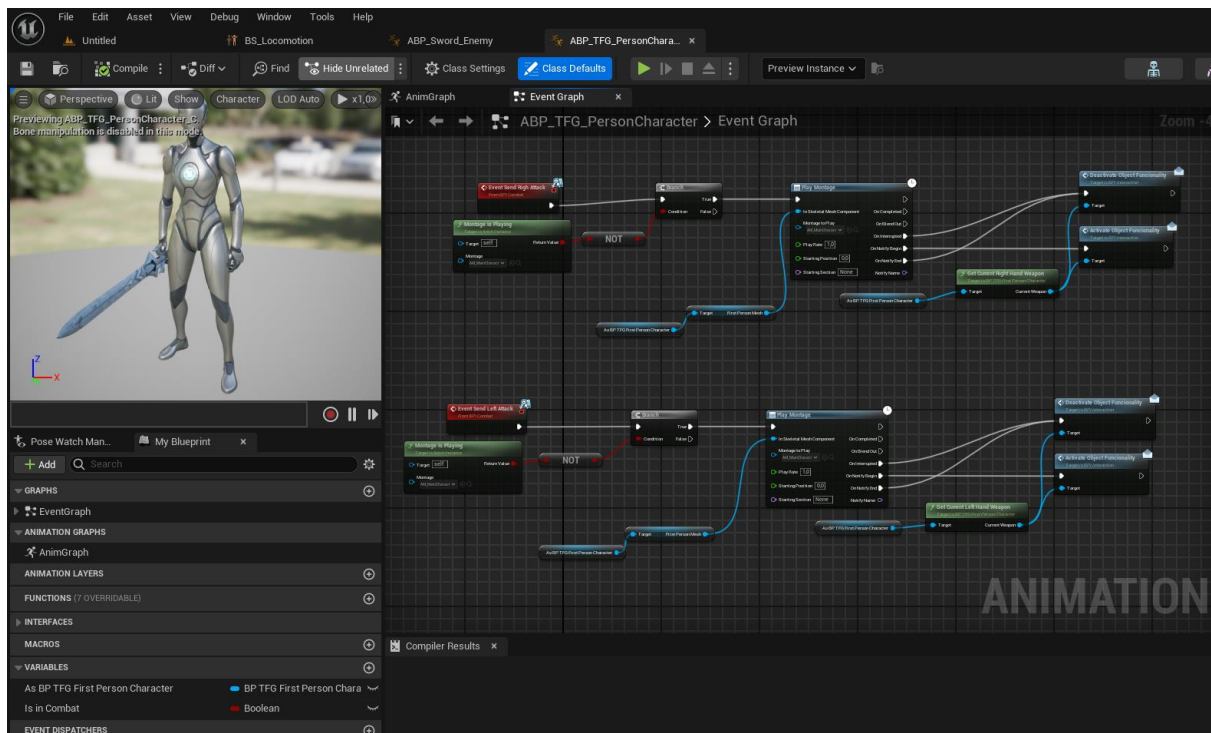


Ilustración 92: Animation Blueprint PersonCharacter (Parte 2)

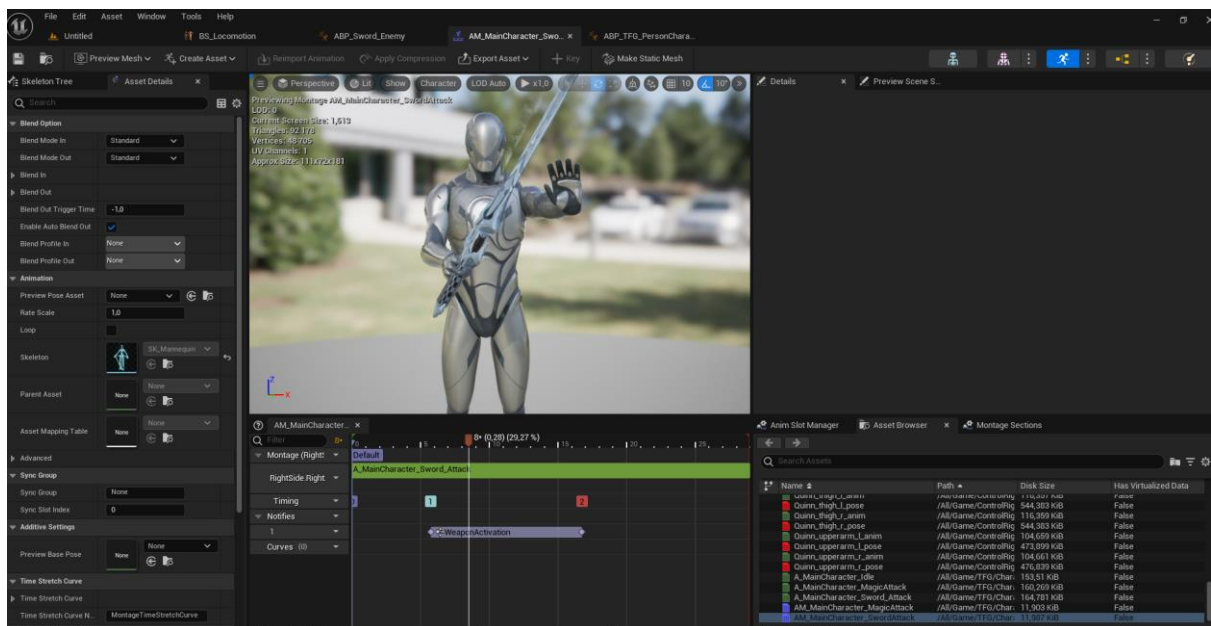


Ilustración 93: Animation Montage MainCharacter\_SwordAttack

## Level Sequences

Los *Level Sequences* permiten crear cinemáticas o secuencias controladas en el juego. Las animaciones del personaje principal pueden estar controladas mediante secuencias que reproducen ataques o escenas clave, lo que proporciona un alto grado de control sobre las interacciones visuales en momentos específicos del juego.

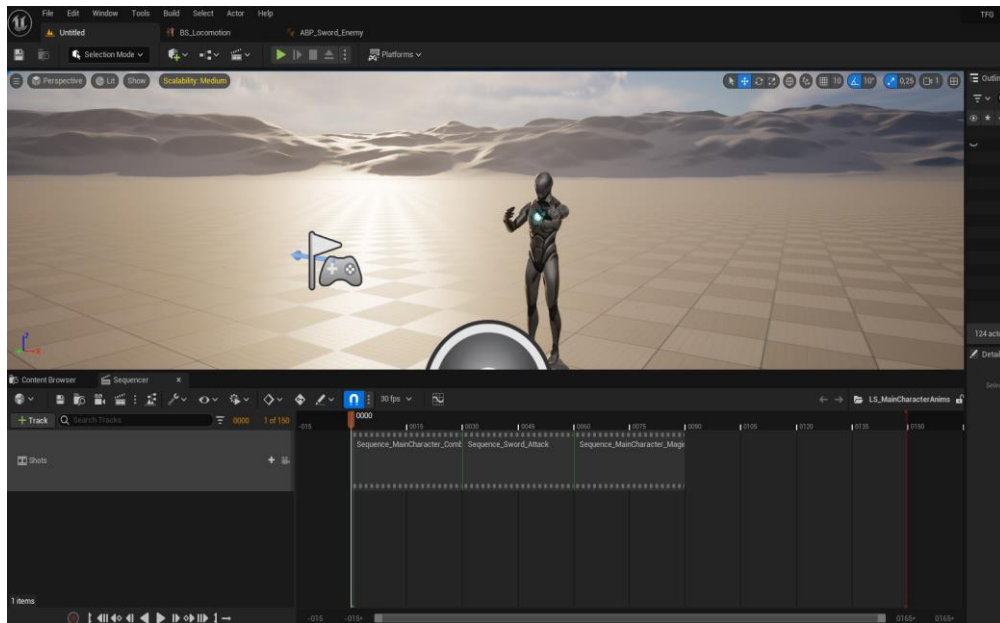


Ilustración 94: Level Sequence MainCharacterAnimation

La modularidad de los *Blueprints*, junto con las interfaces y los sistemas de comportamiento, facilita la expansión del proyecto. Esto permite implementar nuevas características o personajes adicionales sin necesidad de reescribir gran parte del código base. Las animaciones, efectos visuales y sistemas de partículas añaden una capa de inmersión que complementa el comportamiento de los personajes, ofreciendo una experiencia de combate visualmente rica y dinámica.

### 4.10.2 Animación enemigo

#### 4.10.2.1 Animación de Ataque (AM\_SwordEnemy\_Attack)

La animación de ataque de tu enemigo, representada en la imagen como *AM\_SwordEnemy\_Attack*, es una animación tipo *Montage*, la cual permite encadenar

varias animaciones para controlar mejor las acciones del personaje. Este *Montage* está diseñado para ataques cuerpo a cuerpo con una espada y tiene algunas características específicas que lo hacen adecuado para integrarlo con el sistema de combate del enemigo.

- **Montage Segmentado:** En el gráfico, se pueden observar dos segmentos de ataque cuerpo a cuerpo, *Melee\_Attack\_03\_A* y *Melee\_Attack\_02\_A*. Cada uno de estos segmentos es una acción de ataque que se puede reproducir durante la secuencia de combate. Estos ataques pueden activarse de manera secuencial o aleatoria en función del comportamiento del enemigo.
- **Notifies (WeaponActive):** La animación cuenta con *notifies*, en este caso *WeaponActive*, que es clave para sincronizar la activación del daño en el arma con el momento de la animación en que el golpe es efectivo. Esto significa que cuando se alcanza cierto punto en la animación, el *notify* activa el componente de daño del arma (el daño solo se inflige cuando la espada entra en contacto con el jugador, en el momento indicado).
- **Blend Options:** Las opciones de mezcla de la animación permiten que el enemigo pase de una animación a otra sin interrupciones bruscas. El **Blend Out** se controla para asegurarse de que la animación se mezcle suavemente cuando se termina o se interrumpe, de forma que las transiciones entre ataques y otros comportamientos (por ejemplo, moverse o recibir daño) sean fluidas.

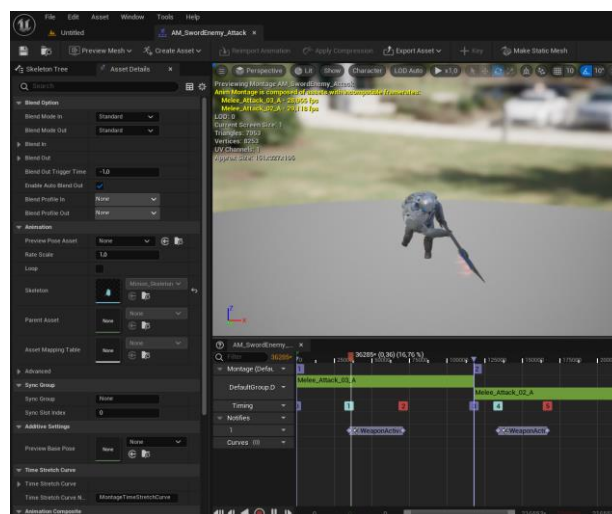


Ilustración 95: Animation Montage SwordEnemyAttack

#### 4.10.2.2 Blend Space de Locomoción (BS\_Locomotion)

El **Blend Space** es una herramienta clave para manejar las animaciones de locomoción del enemigo, permitiendo transiciones suaves entre distintos estados de movimiento (por ejemplo, caminar, correr o moverse en diferentes direcciones).

- **Ejes de Movimiento:** En el **BS\_Locomotion**, los dos ejes son **Dirección** y **Velocidad**. La dirección abarca el rango desde -1 a 1, donde -1 representa el movimiento hacia la izquierda y 1 representa el movimiento hacia la derecha. La velocidad va desde 0 (en reposo) hasta 2 (corriendo).
- **Interpolación de Animaciones:** El sistema de *Blend Space* permite que el enemigo cambie de una animación de caminata a una de carrera, o entre diferentes direcciones (adelante, atrás, izquierda, derecha), según los valores de dirección y velocidad. Este enfoque es esencial para que el movimiento del enemigo se vea natural y reactivo a las acciones del jugador.
- **Ejemplo en Acción:** Si el enemigo comienza a moverse en diagonal hacia adelante y a la derecha con velocidad máxima, el *Blend Space* calculará una interpolación entre las animaciones de caminar hacia adelante y hacia la derecha, ajustando la animación según la dirección y velocidad reales del personaje.

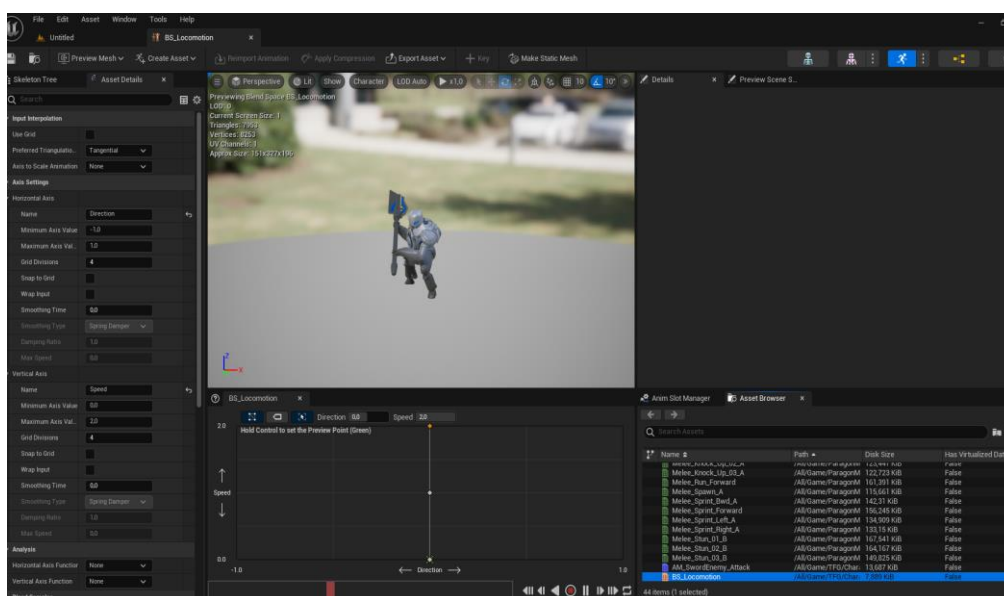


Ilustración 96: Blend Space Enemy

#### 4.10.2.3 Animation Blueprint (ABP\_Sword\_Enemy)

El **Animation Blueprint** (ABP) del enemigo es llamado **ABP\_Sword\_Enemy**, es la pieza central donde se gestiona la lógica que controla las animaciones. Este *blueprint* utiliza un *Anim Graph* para determinar qué animación reproducir en función del estado del personaje y del comportamiento controlado por la IA. En este caso, la estructura general del *blueprint* incluye capas de animaciones, *blend spaces*, y notificaciones clave para atacar, moverse y reaccionar.

##### Anim Graph: Control de Locomoción y Ataques

El **Anim Graph** en el *blueprint* tiene dos elementos principales que destacan: la locomoción y los ataques.

- **Blend Space de Locomoción:** En el gráfico se puede ver cómo se conecta el **BS\_Locomotion** con un nodo que gestiona el **LocoMotionPose**, proporcionando el movimiento base del enemigo en función de la velocidad y la dirección calculada por la IA. El enemigo cambiará de estado entre reposo, caminata o carrera en tiempo real.
- **Layered Blend per Bone:** Esta técnica permite mezclar animaciones en diferentes partes del cuerpo del personaje. En tu caso, la animación de ataque se mezcla con las animaciones de movimiento. Este nodo asegura que el enemigo pueda atacar mientras se mueve sin que las animaciones entren en conflicto entre sí. Esto se consigue aplicando la animación de ataque solo en los huesos superiores (por ejemplo, los brazos y la cabeza) mientras que el resto del cuerpo sigue moviéndose de acuerdo con la locomoción.
- **Notificaciones de Ataque:** Las notificaciones se utilizan para activar eventos durante las animaciones, como por ejemplo el daño de la espada en el momento correcto de la animación de ataque. Esto es gestionado en el *Anim Montage* pero reflejado en el *Anim Blueprint* para asegurar que el enemigo no cause daño fuera de los tiempos especificados.

## Variables y Estados de Animación

En el `ABP_Sword_Enemy`, se observan algunas variables clave que determinan qué animación debe reproducirse en todo momento:

- **Speed:** Controla la velocidad actual del enemigo, que influye directamente en la animación de locomoción.
- **BPBaseEnemyRef:** Una referencia al *blueprint* base del enemigo, que permite acceder a variables como salud, estado y la lógica de combate directamente desde el *blueprint* de animación.

Estas variables son actualizadas dinámicamente por la IA del enemigo, asegurando que las animaciones estén siempre sincronizadas con las acciones del enemigo.

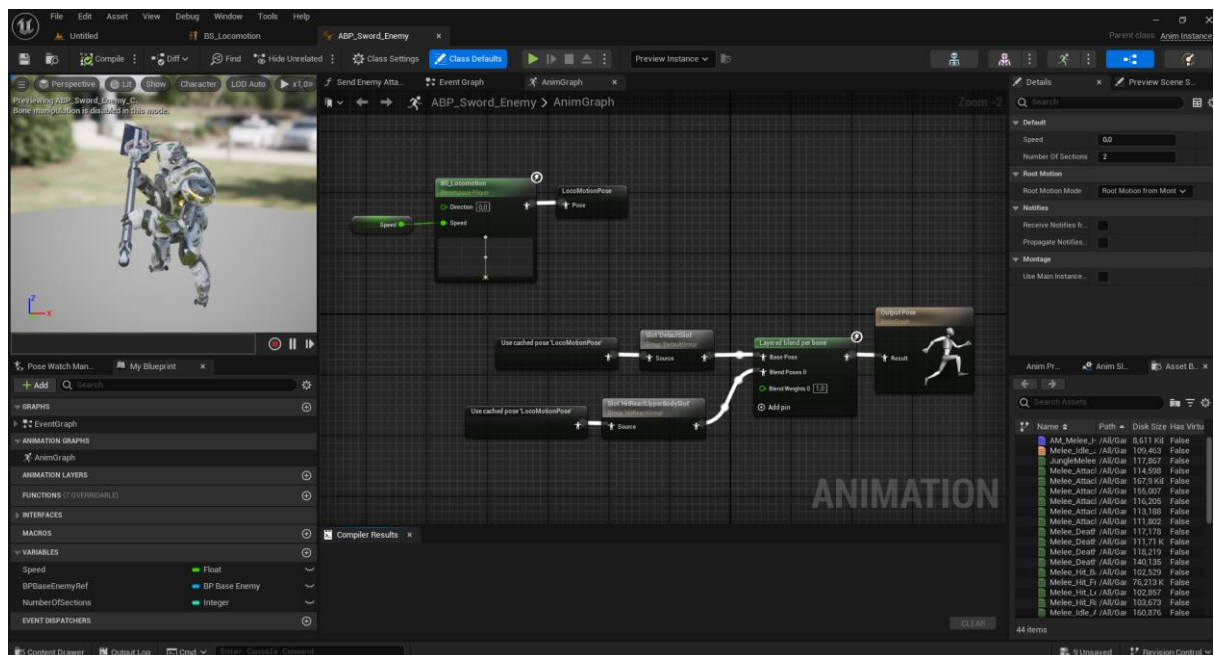


*Ilustración 97: Animation Blueprint Sword Enemy*

## 5.Pruebas

Una vez finalizado el desarrollo del juego, el siguiente paso crucial es llevar a cabo las pruebas. A diferencia de otros tipos de software, un videojuego no puede someterse a pruebas automáticas, ya que no se enfoca en la gestión de datos, sino en la interacción con personas y su experiencia durante el juego. Por esta razón, las pruebas se han ajustado y se han dividido en dos fases distintas.

Es importante destacar que la naturaleza de los videojuegos, donde cada usuario puede interactuar de manera única, hace que las pruebas sean un proceso más complejo y exhaustivo. Las primeras pruebas se enfocan en la funcionalidad del juego, asegurando que todos los elementos técnicos operen correctamente. La segunda fase, más subjetiva, evalúa la experiencia del jugador, revisando aspectos como la jugabilidad, la inmersión, y la respuesta emocional de los jugadores. Esto permite identificar problemas o mejoras necesarias antes del lanzamiento final.

### 5.1 Primera prueba (Alpha)

La primera prueba, conocida como la fase *Alpha*, se llevó a cabo utilizando la primera versión disponible del juego (Versión 1.0.0a). Durante esta etapa, se trabajó con un grupo reducido de cuatro jugadores experimentados. La decisión de limitar el número de participantes a cuatro se tomó para mantener un control más preciso sobre el *feedback* recibido. Involucrar a un mayor número de *testers* podría haber dificultado la gestión adecuada de los errores reportados y la identificación precisa de la versión que estaba probando cada usuario.

En esta fase inicial, es crucial detectar y corregir rápidamente los errores, ya que muchos de ellos pueden ser fallos graves que afecten la jugabilidad básica. Cada vez que se solucionaba una serie de errores, se distribuía una nueva versión del juego a los *testers*, continuando este proceso hasta que se logró una versión estable. A continuación, se detallan las versiones y los arreglos implementados en cada una:

#### 5.1.1 Versión 1.0.1

- Error solucionado: El guardado de partida no funcionaba.

- Error solucionado: Volumen del juego demasiado alto.
- Error solucionado: El mini mapa no se actualizaba al moverse el jugador.

### 5.1.2 Versión 1.0.2

- Error solucionado: Al comenzar la partida, el jugador aparecía debajo del suelo.
- Error solucionado: El botón de volver al juego desde el menú de pausa no funcionaba
- Error solucionado: Todos los títulos y botones tienen el mismo idioma y no contienen

Tras solucionar estos errores con la versión 1.0.2 los *testers* pudieron finalizar el juego en repetidas ocasiones sin encontrar errores o bugs, por lo que juego ya tenía una versión estable.

## 5.2 Segunda prueba (Beta)

El próximo paso es realizar una prueba a gran escala del juego. Dado que ya tenemos una versión estable, podemos lanzarlo para que sea evaluado. Hemos pedido a los *testers* que completen un cuestionario para recopilar sus opiniones sobre nuestro juego. Los resultados son los siguientes:

### 5.2.1 Perfil del jugador



Ilustración 98: Gráfico de cuánto juegan los usuarios

La mayoría de los participantes en la prueba son jugadores habituales, con tres de ellos jugando a videojuegos todos los días y dos varias veces a la semana. Esta tendencia sugiere que los *testers* estaban familiarizados con dinámicas de juego y controles, lo cual pudo influir positivamente en su capacidad para adaptarse y completar el videojuego propuesto. Esto es un factor a tener en cuenta al considerar el perfil de usuario objetivo para futuros desarrollos.



*Ilustración 99: Gráfico de plataforma de los usuarios*

El ordenador fue la plataforma más utilizada por los *testers* (4 jugadores), seguido por PlayStation (3 jugadores) y dispositivos móviles (3 jugadores). Este dato indica una preferencia notable por el ordenador, lo que sugiere que la optimización del juego para esta plataforma debe ser prioritaria. Sin embargo, el uso de consolas y dispositivos móviles también es significativo, por lo que se recomienda mantener y mejorar la compatibilidad multiplataforma del juego.

### 5.2.2 Experiencia de juego

A partir de los datos recogidos a través del formulario aplicado a los jugadores que probaron el videojuego desarrollado, se pueden extraer varias conclusiones relevantes respecto a su experiencia de juego:

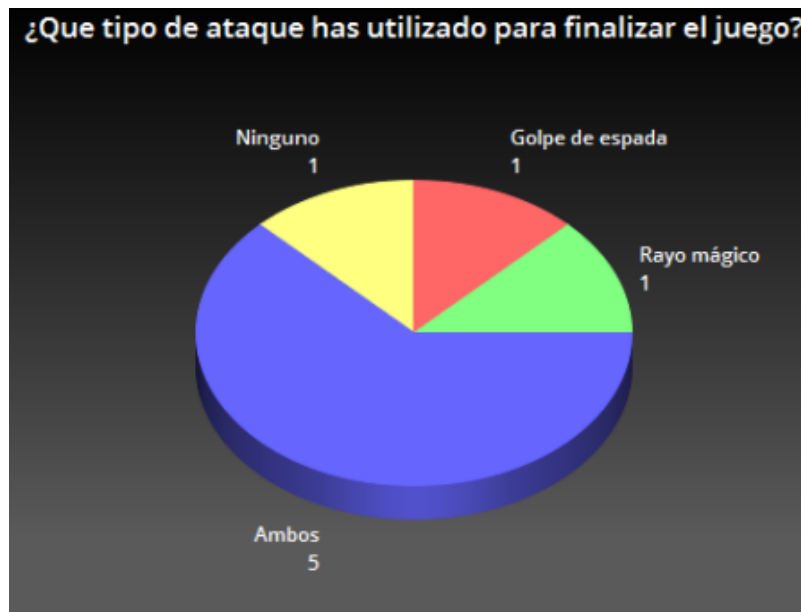


Ilustración 100: Gráfico ataques usados por el usuario

La mayoría de los jugadores (5 de 8) emplearon ambos tipos de ataques disponibles (espada y magia) para finalizar el juego. Este resultado sugiere que el diseño del sistema de combate es equilibrado y fomenta el uso de las diferentes opciones de ataque, evitando que el juego se vuelva repetitivo o limitado en cuanto a las estrategias disponibles para el jugador.



Ilustración 101: Gráfico vida suficiente

Seis de los *testers* consideraron que la cantidad de salud máxima disponible era adecuada para completar el juego, mientras que dos jugadores no compartieron esta opinión. Aunque la mayoría percibió la dificultad como justa, este hallazgo sugiere la posibilidad de revisar los picos de dificultad en ciertas

secciones del juego, o de implementar opciones de ajuste de dificultad para mejorar la accesibilidad y satisfacción de todos los usuarios.



Ilustración 102: Gráfico interacción NPCs

Cinco de los ocho jugadores interactuaron con todos los *NPCs* (Non-Playable Characters), lo que indica que la mayoría encontró estas interacciones significativas o necesarias para el progreso en el juego. No obstante, el hecho de que algunos jugadores interactuaran solo con algunos *NPCs* o incluso con ninguno, sugiere que la relevancia de estas interacciones podría no ser lo suficientemente clara. Se recomienda explorar maneras de destacar más la importancia de los *NPCs* dentro de la narrativa o mecánicas del juego.

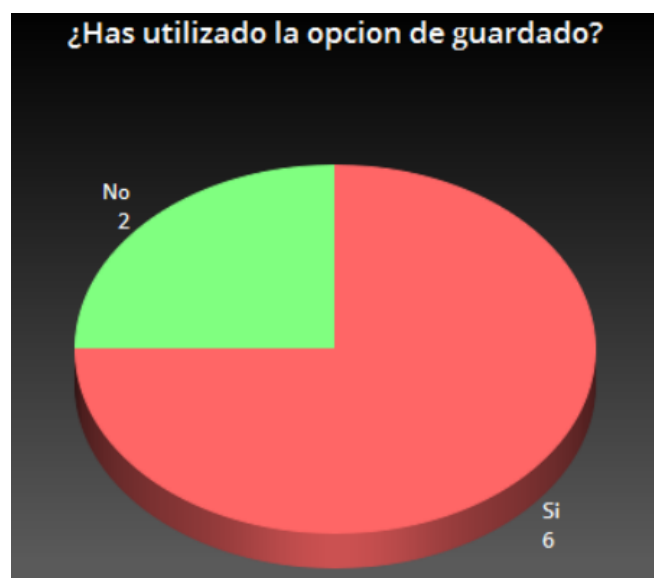


Ilustración 103: Gráfico guardado usuarios

Seis de los ocho *testers* utilizaron la opción de guardado, lo que evidencia que esta función es valorada por los usuarios. Sin embargo, el hecho de que dos jugadores no la utilizaran podría indicar que la opción de guardado no es lo suficientemente visible o que no se comunicó adecuadamente su importancia durante el juego. Esto sugiere que podrían implementarse mejoras en la interfaz o en la guía inicial del juego para asegurar que todos los jugadores comprendan y aprovechen esta característica.



*Ilustración 104: Gráfico finalización del juego*

De los ocho jugadores que probaron el videojuego, siete lograron completarlo. Esta alta tasa de finalización indica que el juego, en su estado actual, ofrece un desafío adecuado para la mayoría de los jugadores. Sin embargo, el hecho de que un jugador no lo haya terminado proporciona una oportunidad para revisar posibles puntos de fricción que podrían estar afectando negativamente la experiencia de algunos usuarios, especialmente aquellos menos experimentados.

Los resultados obtenidos reflejan que el videojuego desarrollado ofrece una experiencia de juego generalmente positiva y equilibrada para la mayoría de los jugadores, especialmente aquellos con cierta experiencia en videojuegos. No obstante, existen áreas clave para la mejora, como la comunicación de la importancia de las interacciones con *NPCs* y el uso de la opción de guardado, así como la consideración de la accesibilidad para jugadores con diferentes niveles de habilidad. Estos aspectos deberán ser considerados en futuras

iteraciones del desarrollo para optimizar la experiencia del usuario y aumentar la tasa de finalización del juego.

## 6. Conclusiones y líneas futuras

### 6.1 Conclusiones

A partir de los resultados obtenidos en las encuestas del apartado anterior, podemos concluir que se ha alcanzado de manera satisfactoria el objetivo principal del proyecto. No obstante, aunque el balance general es positivo, es importante señalar algunos problemas que surgieron durante el desarrollo. El más notable fue la falta de experiencia previa en el desarrollo de videojuegos, lo que obligó a retrasar el inicio del trabajo para adquirir los conocimientos necesarios a través de formación específica.

Además, surgieron desafíos particulares, como la implementación de inteligencia artificial para los enemigos y la creación de montajes para los golpes del personaje principal. Estos aspectos requirieron una considerable cantidad de tiempo y ajustes para asegurar que el juego cumpliera con los estándares deseados.

Cabe destacar que este proyecto ha sido una excelente oportunidad para aplicar de manera práctica los conocimientos adquiridos a lo largo de la carrera, en un contexto distinto al habitual, como es el desarrollo de videojuegos. Utilizar lo aprendido en un proyecto propio nos permite comprender mejor el valor de las herramientas y habilidades adquiridas durante el proceso formativo, y su utilidad en escenarios reales.

### 6.2 Líneas futuras

Una vez finalizado el proyecto existen dos alternativas a seguir:

- Continuar con el desarrollo de Agora Island, ampliando el mapa, añadiendo nuevos enemigos, nuevos ataques y habilidades. Además, podrían añadirse más personajes tanto jugables como para interactuar con ellos en el mapa.
- Iniciar un nuevo desarrollo desde cero, aprovechando ideas frescas y explorando una amplia gama de posibilidades en el diseño de videojuegos, puede abrir puertas a la creatividad y la innovación. Este enfoque permite experimentar con conceptos originales, implementar

mecánicas de juego innovadoras, y probar estilos artísticos únicos que podrían diferenciar al proyecto en un mercado saturado. Además, al comenzar un nuevo proyecto, se tiene la libertad de redefinir la visión y dirección del juego, tomando en cuenta las tendencias actuales y las preferencias cambiantes de los jugadores.

En cualquiera de los dos casos, la experiencia adquirida en desarrollos previos será un activo invaluable, permitiendo que el nuevo proyecto se lleve a cabo de manera más eficiente, con tiempos de producción reducidos y una calidad superior en el resultado final. Además, se podrá colaborar con ilustradores profesionales y músicos especializados para encargarse del arte y la banda sonora del juego. Esto no solo asegurará un nivel de excelencia en los aspectos visuales y auditivos, sino que también contribuirá a dotar al juego de una personalidad única y distintiva, haciendo que se destaque en un mercado competitivo y atraiga a una audiencia más amplia. La combinación de experiencia técnica y el talento artístico puede resultar en un producto que no solo cumpla, sino que supere las expectativas del público.



# Referencias

<https://www.atlassian.com/es/software/jira>

[https://es.wikipedia.org/wiki/The\\_Elder\\_Scrolls\\_V:\\_Skyrim](https://es.wikipedia.org/wiki/The_Elder_Scrolls_V:_Skyrim)

<https://es.wikipedia.org/wiki/Dishonored>

[https://es.wikipedia.org/wiki/Call\\_of\\_Duty:\\_Modern\\_Warfare\\_2](https://es.wikipedia.org/wiki/Call_of_Duty:_Modern_Warfare_2)

<https://www.canva.com/>

<https://app.leonardo.ai/>

<https://www.scrum.org/learning-series/what-is-scrum/>

[https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-0-documentation?application\\_version=5.0](https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-0-documentation?application_version=5.0)

<https://www.unfgames.com/create-an-island-in-ue5>

<https://www.unfgames.com/intro-materials-ue5>

<https://www.unfgames.com/introduction-unreal-engine-5-2-beginners-2023>

<https://www.unfgames.com/your-first-game-in-ue5>

<https://www.unfgames.com/intro-blueprints-ue5>

<https://www.turbosquid.com/>

<https://sketchfab.com/feed>

<https://convai.com/pipeline/create-character>



# Anexo I Game Design Document

## **Análisis de Juego**

Agora Island es un juego de acción en primera persona donde se explorará un mundo que ha sido atacado por robots. Tú objetivo será eliminarlos y devolver el cofre del tesoro a un pirata que se encuentra al inicio del mapa. La combinación de exploración combate y elementos interactivos ofrece una experiencia rica y emocionante.

## **Objetivo del proyecto**

El objetivo principal de *Agora Island* es ofrecer una experiencia de acción en primera persona enfocada en la exploración y el combate. Los jugadores deben enfrentarse a robots enemigos y descubrir áreas ocultas en la isla. El juego cuenta con una narrativa central que gira en torno al tesoro escondido y las naves espaciales presentes en el entorno. La exploración detallada y la interacción con el entorno serán clave para desentrañar la historia y alcanzar el final del juego. Éste enfoque incentiva a los jugadores a explorar cada rincón de la isla para descubrir todos sus secretos.

## **Género**

Juego de Acción en primera persona.

## **Plataforma**

PC

## **Audiencia Objetivo**

*Agora Island* está dirigido a jugadores que buscan una experiencia de acción en primera persona accesible y entretenida. Ideal para quienes disfrutan de la exploración y el combate sin complicaciones adicionales, como recolección de objetos o mejoras de habilidades. El juego ofrece un reto moderado, lo que lo hace adecuado tanto para jugadores casuales como para aquellos con poco tiempo para jugar. La narrativa y el entorno detallado recompensan la curiosidad y el deseo de descubrir secretos, proporcionando una experiencia gratificante sin frustraciones.

## **Historia y Personajes**

En *Agora Island*, los jugadores asumen el papel de un explorador que debe encontrar un tesoro escondido en una isla misteriosa, siguiendo las órdenes de un capitán pirata. La misión implica explorar la isla, enfrentarse a robots enemigos y superar diversos desafíos para cumplir con la tarea encomendada.

A lo largo del juego, el jugador interactúa con el capitán pirata y otros NPCs en las casas de la isla, quienes ofrecen pistas y ayudan a avanzar en la misión. La historia se enfoca en la búsqueda del tesoro y en cómo el entorno interactúa con la narrativa, ofreciendo una experiencia de juego rica y envolvente.

## **Descripción Gameplay**

*Agora Island* es un juego de acción en primera persona que se enfoca en la exploración y el combate en una isla misteriosa. Los jugadores asumen el rol de un explorador contratado por un capitán pirata para encontrar un tesoro escondido. La jugabilidad incluye explorar diversos escenarios naturales y enfrentarse a robots enemigos esparcidos por el mapa.

El juego ofrece una experiencia inmersiva en la que los jugadores interactúan con el entorno y superan desafíos mientras avanzan en la misión. *Agora Island* estará disponible exclusivamente para PC y proporciona una experiencia de un solo jugador centrada en la narrativa y la exploración, ideal para quienes buscan una aventura emocionante y detallada.

## **Experiencia del jugador**

Como jugador de *Agora Island*, explorarás una isla misteriosa llena de diversos escenarios naturales. Desde el inicio, tendrás acceso a una serie de mecánicas básicas que te permitirán moverte y combatir. A medida que avanzas en la isla, te enfrentarás a robots enemigos y descubrirás nuevas áreas, cada una con sus propios desafíos y secretos. Tu principal objetivo será cumplir la misión encomendada por el capitán pirata, que incluye superar obstáculos y enfrentar a enemigos para encontrar el tesoro escondido. La experiencia del jugador está

diseñada para ser inmersiva y emocionante, con un enfoque en la exploración y el descubrimiento de la narrativa que rodea a la isla.

#### Pautas para desarrollar el *Gameplay*

- **Entorno Interactivo:** El diseño del entorno debe permitir la interacción con diversos elementos naturales, como la vegetación, las piedras y el mar. Esto enriquece la exploración y el combate, creando una experiencia inmersiva.
- **Exploración Estructurada:** Aunque no se recojan objetos ni se mejoren habilidades, el mapa debe ofrecer áreas ocultas y desafíos que recompensen la curiosidad del jugador, incentivando la exploración sin sobrecargar al jugador con demasiada información de una vez.
- **Narrativa Integrada:** La historia del juego debe estar integrada en el entorno y los encuentros con *NPCs*, proporcionando contexto y motivación para la misión sin ser el foco principal del *gameplay*.
- **Interacción con *NPCs*:** Los *NPCs* en la isla deben ofrecer pistas y desafíos que ayuden a avanzar en la misión principal, añadiendo profundidad a la experiencia de juego sin complicar el objetivo principal.

#### Objetivo de juego y Recompensas

Objetivos de juego:

- **Cumplir la Misión del Capitán Pirata:** El principal objetivo es encontrar el tesoro escondido en la isla, como lo solicitó el capitán pirata. Este objetivo guía la exploración y el progreso en el juego.
- **Explorar el Entorno:** Recorrer diversos escenarios naturales de la isla, enfrentando a robots enemigos y descubriendo áreas ocultas. La exploración es clave para avanzar y completar la misión.
- **Superar Desafíos:** Enfrentarse a enemigos y superar obstáculos que aparecen a lo largo del juego. Cada desafío proporciona una oportunidad para mejorar la estrategia y habilidades de combate del jugador.

## Recompensas:

- **Progreso en la Historia:** Avanzar en la misión y descubrir elementos de la narrativa que revelan más sobre el tesoro y el entorno de la isla, proporcionando contexto y motivación.
- **Satisfacción Personal:** La recompensa principal es la satisfacción de completar la misión encomendada y desvelar todos los secretos de la isla. Este sentido de logro refuerza la motivación del jugador y la inmersión en el juego.

## Mecánicas de Juego

| Atributos del protagonista |                                                                                                                                                                                 |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Habilidades / Acciones     |                                                                                                                                                                                 |
| Salto                      |                                                                                                                                                                                 |
| Desplazamiento             |                                                                                                                                                                                 |
| Golpear con espada         |                                                                                                                                                                                 |
| Lanzar rayo                |                                                                                                                                                                                 |
| Recoger objeto             |                                                                                                                                                                                 |
| Dialogar con NPC           |                                                                                                                                                                                 |
| Modo de juego              |                                                                                                                                                                                 |
| <b>Modo historia</b>       | El objetivo de este modo es explorar la isla en la que te encuentras. Tienes que llegar hasta el centro de la isla en busca de un cofre mientras que te enfrentas con enemigos. |

Tabla 4: Mecánicas de juego

## Controles

| Botón Input               | Acción que realiza |
|---------------------------|--------------------|
| WASD                      | Moverse            |
| Barra espaciadora         | Salto              |
| Click derecho del ratón   | Golpe de espada    |
| Click izquierdo del ratón | Lanza un rayo      |

|             |                            |
|-------------|----------------------------|
| Botón F     | Recoger objeto             |
| Botón T     | Abrir micrófono de dialogo |
| Botón Enter | Abrir chat de dialogo      |

*Tabla 5: Controles de juego*

## **Estética del Juego e Interfaz de Usuario**

*Agora Island* presenta una estética que combina elementos naturales y tecnológicos para crear una atmósfera misteriosa y vibrante. El entorno de la isla está lleno de detalles visuales que reflejan su rica biodiversidad y tecnología alienígena, con colores vivos y contrastantes que destacan la belleza del paisaje mientras insinúan el peligro presente. La interfaz está diseñada para ser clara y funcional, mostrando solo la información esencial, como la salud del personaje, para mantener la inmersión en la exploración y el combate.

La interfaz de usuario será minimalista, con elementos gráficos que se integran sutilmente en el entorno del juego. Los menús proporcionarán detalles adicionales sobre la misión y el progreso sin interrumpir la experiencia de juego, manteniendo el enfoque en la narrativa y la interacción con el entorno. La combinación de estos elementos crea una experiencia visualmente atractiva y coherente con la atmósfera general del juego.



## Anexo II Manual de usuario

### Instalación

Para instalar un juego desarrollado en **Unreal Engine**, primero es necesario descargar el archivo ejecutable (.exe) o el archivo comprimido (.zip) que contiene el juego desde la página web del desarrollador o desde una plataforma de distribución de juegos.

Si el archivo descargado está comprimido, descomprímelo utilizando un programa como **WinRAR** o **7-Zip**. Una vez descomprimido, localiza el archivo ejecutable del juego dentro de la carpeta descomprimida. Haz doble clic en el archivo **.exe** para iniciar el juego sin necesidad de una instalación adicional, ya que los juegos en Unreal Engine generalmente se distribuyen como ejecutables autónomos que no requieren un proceso de instalación complejo.